Fast Ray-Tracing of Rectilinear Volume Data Using Distance Transforms

Milos Sramek and Arie Kaufman, *Fellow*, *IEEE*

Abstract—This paper discusses and experimentally compares distance-based acceleration algorithms for ray-tracing of volumetric data with an emphasis on the *Chessboard Distance (CD) voxel traversal*. The acceleration of this class of algorithms is achieved by skipping empty macro regions, which are defined for each background voxel of the volume. Background voxels are labeled in a preprocessing phase by a value, defining the macro region size, which is equal to the voxel distance to the nearest foreground voxel. The CD algorithm exploits the chessboard distance and defines the ray as a nonuniform sequence of samples positioned at voxel faces. This feature assures that no foreground voxels are missed during the scene traversal. Further, due to parallelepipedal shape of the macro region, it supports accelerated visualization of cubic, regular, and rectilinear grids. The CD algorithm is suitable for all modifications of the ray tracing/ray casting techniques being used in volume visualization and volume graphics. However, when used for rendering based on local surface interpolation, it also enables fast search of intersections between rays and the interpolated surface, further improving speed of the process.

Index Terms—Volume visualization, volume graphics, volume rendering, distance transforms, macro region, voxel traversal, speed up techniques, subvoxel precision.

1 Introduction

Popularity of the ray tracing technique is not only due to its ability to enhance spatial perception of the scene using such effects as transparency, mirroring, and shadow casting, but also for its versatility, which can benefit in such tasks as, for example, computing form-factors in radiosity [1]. Ray tracing and its simplified version, ray casting, also penetrated the area of volume visualization [2] since, in the framework of their basic scheme, both approaches enabled the implementation of various surface, as well as volume rendering techniques (for example, color compositing, reprojection, and MIP). Recently, a new domain, volume graphics [3], emerged, uniting both computer graphics and volume visualization, thus providing an environment for simultaneous modeling, manipulation, and rendering of objects defined both in symbolic (analytic) and discrete form. In volume graphics, analytic objects are scan-converted (voxelized) into a 3D raster of volumetric primitives [4], [5], [6], which is exactly the same data structure as the well-known voxel array used for representation of measured three-dimensional scalar data. From the perspective of traditional computer graphics, volume objects represent new, full-featured primitives, which can be treated either as standard objects with "hard" surfaces (Fig. 1) or as volume objects representing amorphous phenomena as fire, fog, or clouds.

- *M. Sramek is with the Commission for Scientific Visualization, Austrian Academy of Sciences, Sonnenfelsgasse 19/2, A-1010 Vienna, Austria. E-mail: milos.sramek@oeaw.ac.at.*
- *A.E. Kaufman is with the Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400. E-mail: ari@cs.sunysb.edu.*

Manuscript received 11 June 1999; accepted 29 Feb. 2000.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 110040.

The new volume graphics approach has not only required development of new optimized voxelization routines, but also has redefined the demands on speed, precision, and versatility of visualization algorithms, including that of ray tracing. In rendering volumetric objects, one takes advantage of the simplicity of the voxel data structure since only one object type, the voxel itself, should be processed instead of the numerous types used in computer graphics. This simplifies the ray-tracer and contributes to its acceleration. Of course, additional processing time for voxelization is necessary. However, object voxelization routines are usually simpler and faster than that for computation of ray-object intersections. Ray tracing of volumetric data is a task which is algorithmically similar to standard ray tracing of analytical objects if a space subdivision acceleration is used. In this case, the object space is similarly subdivided, either hierarchically or uniformly, into voxels [7], [8], which can be empty or can contain a list of contributing objects. The primary goal of the subdivision is to limit the number of ray-object intersection tests, which are themselves costly operations, by only performing tests with objects belonging to voxels pierced by a ray.

Ray traversal algorithms designed for the subdivision techniques can also be used for ray tracing volumetric data. However, one difference exists; namely, the voxel scene size. While the optimal subdivision for the subdivision techniques was found to be low (only hundreds of voxels [9]), data sets with significantly higher resolution are processed in visualization and volume graphics tasks. Therefore, applicability of these algorithms is only moderate and special techniques for acceleration of the volumetric ray tracing have been developed [10], [11], [12].

The distance-based acceleration techniques [11], [13], [14] for ray-tracing of voxel grids assume a preprocessing phase

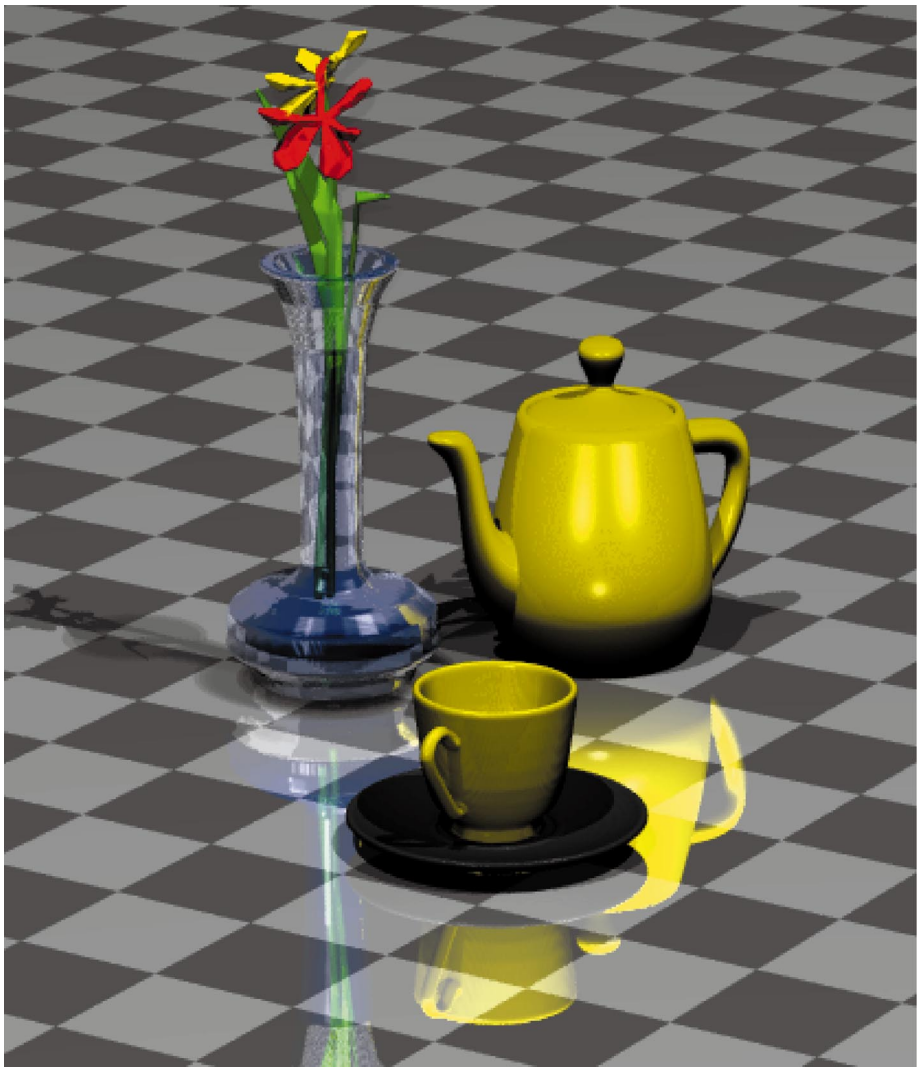


Fig. 1. A ray-traced scene, composed of several volumetric objects, voxelized from triangular and parametric models.

in which voxels not contributing to the final image are identified and the whole volume is classified into background and foreground. Further, to each background voxel a value is assigned, equal to its distance to the nearest foreground voxel. Different metrics can be used to define this distance [15]

$$\begin{aligned} D_1(\vec{r}) &= |p_x| + |p_y| + |p_z| && \text{city block distance} \\ D_2(\vec{r}) &= \sqrt{(p_x^2 + p_y^2 + p_z^2)} && \text{Euclidean distance} \\ D_\infty(\vec{r}) &= \max(|p_x|, |p_y|, |p_z|) && \text{chessboard distance,} \end{aligned} \quad (1)$$

where $\vec{r} = (p_x, p_y, p_z)$ is a vector between two voxels. Each background voxel thus defines around itself an empty macro region, with shape dependent on the corresponding metrics involved. These regions are skipped during the ray traversal, shortening the time spent for processing uninteresting regions of the scene.

This paper discusses different distance acceleration techniques, with an emphasis on the *Chessboard distance (CD) voxel traversal*. As its name indicates, it relies exclusively on the chessboard distance, defining cubic or parralelepipedal macro regions, with walls perpendicular to coordinate axes of the data set. Unlike the other macro region shapes, defined by the city-block and Euclidean distances, which are used in cubic grids, the CD macro regions can accelerate ray definition also in rectilinear grids with variable voxel distance along the coordinate axes (Fig. 2). We introduced the CD algorithm earlier in two different versions for cubic grids [14] and for rectilinear grids [16]. Here, we present a unified and extended version which works in Cartesian, regular, and rectilinear grids, together with additional experimental comparison of different distance based acceleration techniques in Cartesian grids.

The paper is organized in six sections: In Section 2, we define general concepts and ideas, which are used throughout the paper. In Section 3, ray acceleration techniques are discussed, with a special focus on distance based approaches. Section 4 presents a detailed description of the CD algorithm. In Section 5, results of experiments are presented, aimed at comparing different ray traversal techniques. Finally, Section 6 summarizes our results and advantages of the CD traversal technique.

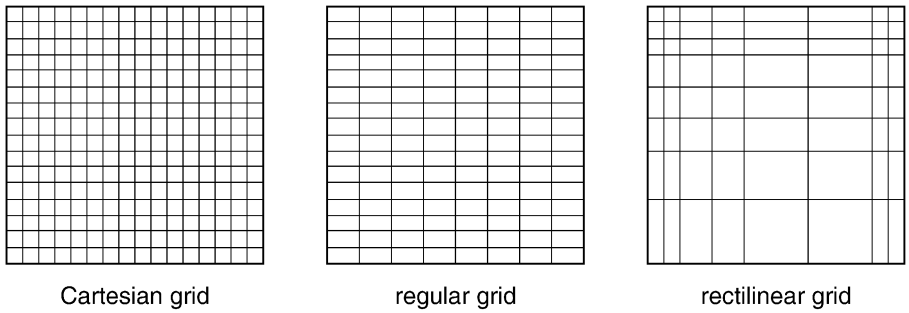


Fig. 2. Grid types.

2 Volume Visualization by Ray Tracing

The term *surface rendering* denotes a set of 3D data visualization techniques where only object surfaces contribute to the rendered image. One possibility is to build a *surface model*, that is, to define a set of patches approximating the surface [17]. These patches can then be rendered by some standard technique, usually with a hardware support. The *binary volume rendering* techniques represent an alternative. Although the image is still only contributed to by surfaces, no explicit surface model is defined. Instead, a trivariate implicit function $\mathcal{F} = \mathcal{F}(P, P_\sigma)$ is given, depending on voxel position P and data samples in a neighborhood P_σ. A continuous surface description can then be obtained by thresholding this function at a level T. The choice of the proper volume and gradient reconstruction functions is dependent on data properties and eventual preprocessing [18]. Recently, the reconstruction process has been analyzed from the point of view of optimal filter design by several authors [19], [20], [21], [22]. In general, they concluded that higher order filters should be preferred over the linear filters, which are traditionally used for reconstruction of the data and its gradient due to their simplicity. However, in the case of voxelized objects, we have overall control of the process and, thus, we can ensure such data properties, making utilization of first order filters possible [6].

In order to specify various surface properties (color, reflectivity, etc.) for different objects, an *object identifier* can be assigned to a voxel either directly during the scan conversion of an analytical object or as a result of *segmentation* in the case of scanned data [23], [24]. Voxels with no identifier assigned belong to *background* and can be ignored during the processing since they do not contribute to the rendered image.

Since the scene is defined within a 3D discrete raster, the ray should be represented as a *discrete ray*, that is, as an ordered sequence of voxels pierced by the ray, with the following properties:

1. To permit supersampling and recursivity, the ray should be able to start at any point outside or inside the scene, and with arbitrary direction,
2. To secure correct images, no object voxels along the ray should be missed. Therefore, the ray should fulfill the demands of 6-connectivity at least in the vicinity of an object [25].

The *probabilistic volume rendering* techniques present an alternative to surface approaches. Rather than segmenting the scene into objects and background, an *opacity* and *color* are assigned to each voxel, based on local properties of the data. The opacity reflects a measure by which the given voxel can contribute to the rendered image. A *culling* function can be defined, identifying the voxels which cannot contribute to the rendition and which can be discarded from consideration, similar to the background voxels in the surface rendering. Among others, techniques tracing primary rays (*ray casting*) through the scene were proposed, accumulating color and opacity of voxels or data samples obtained by interpolation along the ray.

Traversal of the ray voxels usually has more phases. Background voxels, surrounding the objects, are usually found first. Their traversal stops either when the ray leaves the scene or when the first object voxel is found. In the second case, the subsequent action depends on the visualization technique involved. In the case of binary volume rendering, a *hit-miss test* should be performed in order to know if the ray should continue further by the following object or background voxel (Fig. 3a, Ray B) or if a *ray-surface intersection* should be searched for (Fig. 3a, Ray A). The hit-miss test can be performed by evaluation of the interpolating function $\mathcal{F}$ at one or more points lying in the voxel and comparing the results with the threshold value. In order to detect the ray-surface intersection point exactly, a system of equations defined by the ray and the surface $\mathcal{F}(\mathcal{P}, \sigma) = T$ should be solved either analytically or numerically [14]. In the case of probabilistic volume rendering, the contribution of all object voxels has to be accumulated (Fig. 3b, rays A and B).

3 Macro-Region Based Voxel Traversal Algorithms

Not all voxels along the ray contribute to the rendered image with the same weight. Only some of them belong to the interesting objects or surfaces, while the others can be traversed rapidly or even totally skipped. This capability is called *space-leaping* [12] and exploits a kind of *coherence* inherent to the object and/or image space, as well as to a sequence of consecutive images.

The macro-region based voxel traversal algorithms exploit the object space coherence, that is, the tendency of object (background) voxels to occupy connected regions of the space. In this case, background voxels are gathered into cubic, parallelepipedal, or spherical macro-regions, which can be skipped in one step, thus reducing the number of

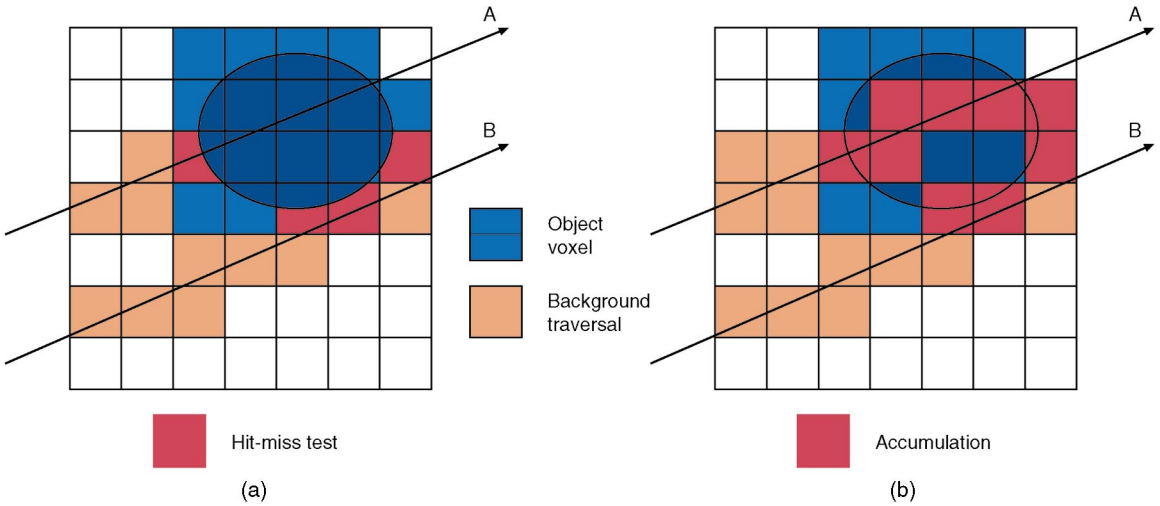


Fig. 3. Traversal of background and foreground voxels for (a) binary and (b) probabilistic volume rendering.

steps and, therefore, also the total rendering time. Various schemes for the macro-region definition are possible. Some of them are based on hierarchical encoding of the scene space [10], [26], others define the macro-regions directly in the original voxel scene [11], [12], [13], [14], [27].

Distance transforms convert a 2D (3D) binary image into an image, where each background pixel (voxel) is assigned a value (1) corresponding to its distance to the nearest object pixel (voxel). Although computing distances is, in principle, a global operation, algorithms were developed approximating the global distances by propagating distances between neighboring pixels [15]. Rules for propagation of the local distances defining different types of transforms are usually presented in a form of masks of different size. For each transform, a pair of masks is defined: one for a forward run, starting in the upper-left corner (2D case), and one for a backward pass, starting in the lower-right corner (Fig. 4). The constants in the cells are the local distances propagated over the image. First, all background pixels are assigned some "infinity" value. In both the forward and backward run, the new value of the actual pixel (distance 0 in the mask) is assigned a minimum of the sums of the image pixel values and the local distances defined by the mask.

The idea of exploiting distance transforms to accelerate background traversal was introduced by Zuiderveld et al. [11]. The proposed Ray Acceleration by Distance Coding (RADC) scheme works in two phases:

Preprocessing: The volume is segmented and distance information is added to background voxels by a 3D distance transform.

Rendering: As a basis for the ray traversal, the floating point 3D DDA algorithm, defining the ray as a sequence of equidistant samples, is used. The distance information permits skipping an appropriate number of samples and thus rapid traversal of the regions.

Since objects in volumetric data sets tend to congregate in the middle of the volume, rays usually skip the off-center parts rapidly and slow down in the object vicinity. For parallel projection, if the ray totally misses the object, the minimal distance along its path can be utilized for further acceleration. In such a case, this distance defines a region in the image plane, where it is not necessary to fire new rays because they all miss the object. The RADC algorithm works with various digital approximations of the ideal Euclidean distance (Fig. 5). Since different shapes of thus-defined free regions are not taken into account, highest speed is obtained with the chamfer distance, which is the best approximation of the Euclidean distance.

A similar technique was proposed by Cohen and Sheffer [13]. The authors call the free zones defined by the distance transform *proximity clouds*. Once a ray enters a cloud cell, it can safely skip the distance determined by the cell value. The algorithm is also based on the floating point 3D DDA algorithm. It differs from the RADC in that it takes the shape of the free zone into account: The step size depends not only on the distance value, but also on the type of distance and ray direction (this dependency is illustrated in Fig. 5). If the ray is defined by its direction vector $\vec{r} = (r_x, r_y, r_z)$ and the assigned distance is d, then the coordinate increment should be

$$\left(\frac{d\ p_x}{D(\vec{r})}, \frac{d\ p_y}{D(\vec{r})}, \frac{d\ p_z}{D(\vec{r})}\right), \tag{2}$$

where $D(\vec{r})$ is size of the projection vector in the corresponding metrics (Euclidean metrics is used in the case of chamfer distance):

$$D(\vec{r}) = \begin{cases} D_1(\vec{r}) & \text{for the city block distance} \\ D_2(\vec{r}) & \text{for the Euclidean distance} \\ D_\infty(\vec{r}) & \text{for the chessboard distance.} \end{cases} \tag{3}$$

Authors have shown that the average step for the city block distance can even be a small percentage longer than for the Euclidean distance. Another advantage of the city block distance is that its computation is easier than computation of the chamfer distance, which is usually used instead of the Euclidean distance.

Cohen and Sheffer's technique has two drawbacks [13]:

1. The distances are calculated for the cell centers, while the current location along the ray is not

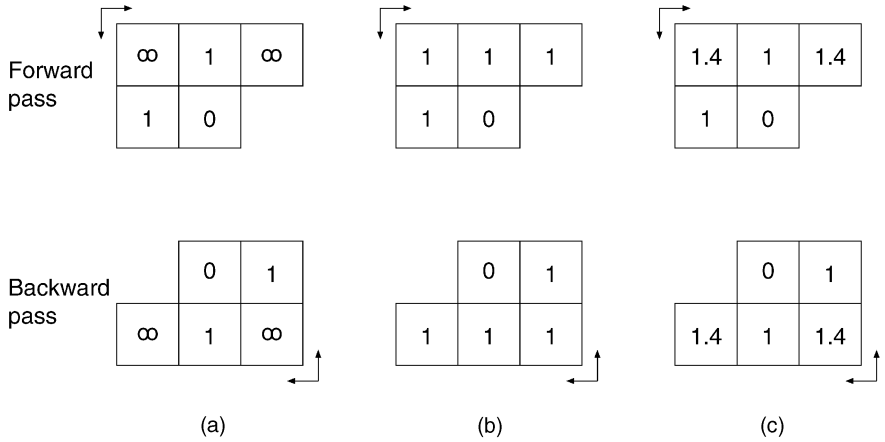


Fig. 4. Masks for computation of 2D distance transforms: (a) city block, (b) chessboard, and (c) chamfer distance mask, approximating the Euclidean distance.

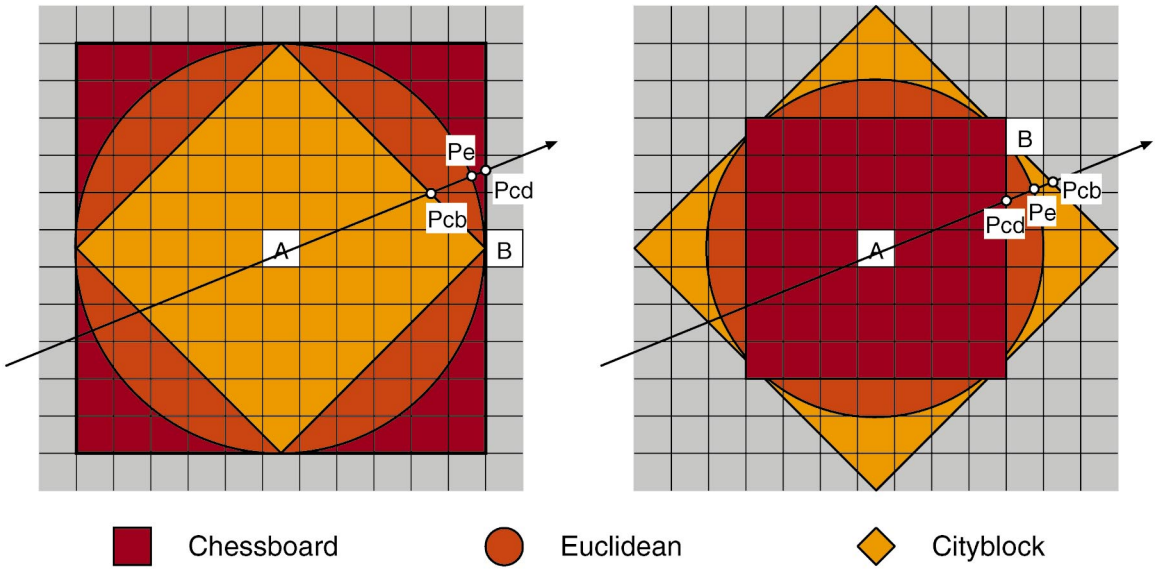


Fig. 5. Different shapes of free regions defined by different types of assigned distance. (A–background pixel, B–nearest object pixel). Pe, Pcb, and Pcd–next ray sample from A by a step defined with Euclidean, city block, and chessboard distance, respectively.

necessarily in the center. Therefore, to avoid skips beyond the free zone, the computed distance d must be decreased by 1.

2. The sequence of cells generated by the floating point 3D DDA algorithm does not fulfill the condition of 6-connectivity, which may cause a miss of some object voxels. Therefore, in the object vicinity, the algorithm is switched to an incremental cell traversal algorithm [9], [8] generating a 6-connected sequence.

A CD voxel traversal algorithm, introduced earlier [14], [16] and described in detail in this paper, relies exclusively on cubic macro-regions defined by the chessboard distance (CD). In comparison to the previous algorithms, it has several advantages:

1. A ray is defined as a nonuniform sequence of samples at its intersections with the macro-region faces (Fig. 6), which allows for utilization of the full size of the macro-region and thus overcomes the first drawback of the previous "proximity clouds" technique.
2. In the close vicinity of an object, where CD equals zero, the macro-regions are identical to single voxels. Since the samples are located on voxel faces, a 6-connected sequence of voxels is defined, overcoming the second drawback.
3. Simplicity of the cubic region geometry enables extension of the algorithm for regular and even rectilinear voxel grids.
4. Extension of the algorithm for nonsymmetric macro regions is possible.

4 Traversal of Cubic, Regular, and Rectilinear Grids by Cubic Macro-Regions

The essence of the proposed algorithm for definition of a discrete ray in a rectilinear grid resides in the construction of a secondary grid with the same dimensions as the original grid, but with cubic voxels. Cubic macro regions are then built for all voxels of the secondary grid by the standard distance transform technique. Each such macro region corresponds to a nonsymmetric paralelepipedal

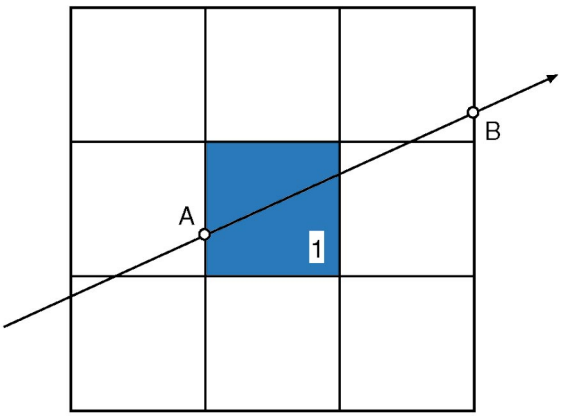


Fig. 6. Step over a macro-region defined by $\mathrm{CD} = 1$: A entry point, B exit point.

macro region in the original grid. The distance information of the secondary grid is then utilized to accelerate traversal in the original rectilinear grid.

Let $\mathbf{G}$ be a 3D grid of $(N_x+1)\times(N_y+1)\times(N_z+1)$ points p_{ijk}:

$$\mathbf{G} = \{p_{ijk} = (S_x^i, S_y^j, S_z^k) : \\ 0 \le i \le N_x, 0 \le j \le N_y, 0 \le k \le N_z\}, \tag{4}$$

where i, j, and k are integers,

$$S_\nu^m = \sum_{i=1}^{m} \Delta_\nu^{i-1}, \quad \nu = x, y \text{ or } z \tag{5}$$

are sample coordinates $(S_\nu^0 = 0)$ and $\Delta_\nu^0, \Delta_\nu^1, , \Delta_\nu^{N_\nu}$, $\nu = x, y$ or z, is the spacing between samples along each coordinate axis. In the case of a regular grid, (5) simplifies to

$$S_\nu^m = m\Delta_\nu, \quad \nu = x, y \text{ or } z, \tag{6}$$

where Δ_ν is spacing along the ν axis, and to

$$S_\nu^m = m, \tag{7}$$

if the grid is cubic, assuming unit step between samples.

Let a voxel be a tuple $V_{ijk} = (v_{ijk}, h_{ijk})$, where $v_{ijk} = (S_x^i, S_x^{i+1}) \times (S_y^j, S_y^{j+1}) \times (S_z^k, S_z^{k+1})$ is voxel volume and $h_{ijk} \in \{0, 1\}$ is its value. Grid points p_{ijk} are defined in voxel vertices and, therefore, the voxel scene $\mathbf{V}$:

$$\mathbf{V} = \{V_{ijk} : 0 \le i < N_x, 0 \le j < N_y, 0 \le k < N_z\}, \tag{8}$$

has one element less along each axis than $\mathbf{G}$. The voxel value $h = 1$ means that the voxel can contribute to the image: Either an object surface passes through its volume (binary volume rendering) or it can contribute with nonzero color and opacity in probabilistic volume rendering. The value $h = 0$ means that the voxel cannot contribute and, therefore, it can be skipped during the traversal without processing. We denote voxels with values equal to 0 (resp. 1) 0-voxels (resp. 1-voxels). Decision as to which value to assign to which voxels depends on the data and on selected interpolation rendering technique (Fig. 7).

Let the secondary voxel scene $\mathbf{V}'$ consist of the same number of unit cubic voxels along each axis and let the corresponding voxels have equal values:

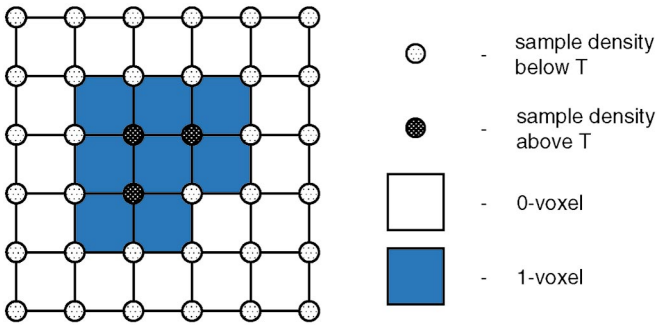


Fig. 7. Samples and voxels: The actual voxel value depends on the data and the selected interpolation technique. In this case, the continuous volume is reconstructed by trilinear interpolation and object surface is defined by its thresholding at level T. Therefore, all voxels, which have at least one vertex sample above the threshold T are assigned label 1.

$$h'_{ijk} = h_{ijk}. \tag{9}$$

If we now assign to each 0-voxel V'_{ijk} its chessboard distance n to the nearest 1-voxel, we define a cubic macro-region in $\mathbf{V}'$:

$$\mathcal{O}'_n(V'_{ijk}) = \{h'_{pqr} = 0 : i-n \le p \le i+n, \\ j-n \le q \le j+n, k-n \le r \le k+n\} \tag{10}$$

with its center in V'_{ijk} and with side size $2n+1$. A corresponding macro-region is defined in $\mathbf{V}$ around V_{ijk}:

$$\mathcal{O}_n(V_{ijk}) = \{h_{pqr} = 0 : S_x^{i-n} \le p \le S_x^{i+n}, \\ S_y^{j-n} \le q \le S_y^{j+n}, S_z^{k-n} \le r \le S_z^{k+n}\}, \tag{11}$$

which is, of course, generally not cubic and voxel V_{ijk} is not situated in its center.

The voxel scene $\mathbf{V}'$ corresponds to the form in which our data is stored in main memory or a file. We do not care about the grid spacing during the distance transform, which is done in exactly the same way as for the Cartesian scenes. The grid spacing is taken into account during the ray definition, as is shown in the next section.

4.1 The Algorithm

We know that there are no object voxels within $\mathcal{O}_n(V)$, so we can jump from V directly to the first voxel outside of $\mathcal{O}_n(V)$. The traversal speed is thus increased by reducing the number of visited voxels (Fig. 8). We assume that the direction vector has only nonnegative coordinates. Generalization to all possible directions is done by proper initialization of some variables.

Let us imagine that the ray has reached voxel V with coordinates $\mathcal{V} = (v_x, v_y, v_z)$ and with assigned chessboard distance n, at an entry point $\mathcal{P} = (p_x, p_y, p_z)$ positioned at one of its faces. It is necessary to point out that $\mathcal{V}$ represents the position of the voxel in the secondary cubic grid and, therefore, also in the data set, and $\mathcal{P}$ represents the position in the coordinate system of the original rectilinear grid. It is now necessary to find the nearest intersection of the ray $\mathcal{X} = \mathcal{P} + t\vec{r}$ with the planes:

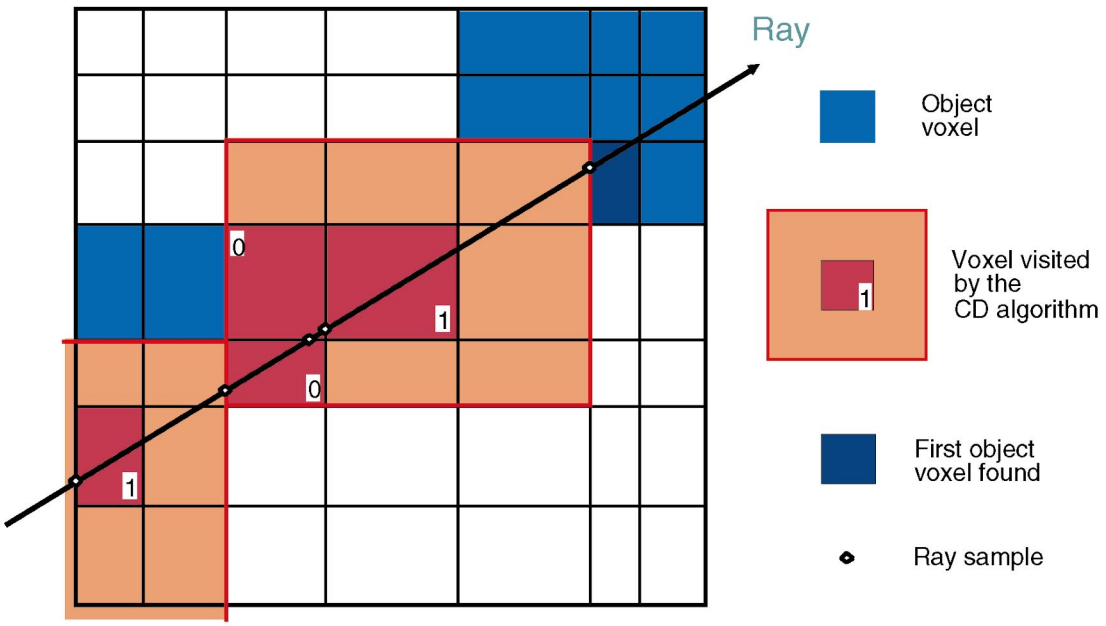


Fig. 8. Rectilinear grid traversal by cubic macro-regions.

$$\begin{aligned} x = S_x^{v_x+n} &\Rightarrow t_x = \tfrac{S_x^{v_x+n}-p_x}{r_x} \\ y = S_y^{v_y+n} &\Rightarrow t_y = \tfrac{S_y^{v_y+n}-p_y}{r_y} \\ z = S_z^{v_z+n} &\Rightarrow t_z = \tfrac{S_z^{v_z+n}-p_z}{r_z}. \end{aligned} \quad (12)$$

The nearest intersection is defined by $t = \min(t_x, t_y, t_z)$.

Due to both the symmetry of the algorithm with respect to all three coordinate axes and the possibility given by the C language macro preprocessor of manipulating source code symbols, the algorithm can be coded in a compact form, represented by the following two macros (Figs. 11 and 12):

MasterStep updates the sample position and voxel coordinate for that axis ν, for which $t_\nu = \min(t_x, t_y, t_z)$. Versions **MasterStep_1** and **MasterStep_n** for single voxel and macro region traversal (Fig. 12) differ only in the step size, which updates the coordinate value. The remaining two lines, testing if the voxel is within the scene bounds and updating the point coordinate p_x, are the same.

SlaveStep updates the variables for the remaining two axes. Now, the macros differ significantly. In the single voxel step (**SlaveStep_1**) the new ray point is positioned on a wall of the same voxel as the previous point. Therefore, it is sufficient to update only the point coordinate, while the voxel coordinate remains the same. The situation is different for the macro-region step (**SlaveStep_n**):

1. The ray may leave the scene in the y direction; therefore, the new point coordinate should be compared with the scene bounding box, and
2. the y voxel coordinate should be updated since the ray may skip several voxels along the y direction. In the case of a rectilinear grid it is not possible to compute the new Y coordinate directly from the point coordinate p_y. Therefore, function **Locate** finds this coordinate by binary search in the array S_y of the grid point coordinates. Values v_y and $v_y + n$ define lower and upper bounds for this search. However, in the case of Cartesian and regular grids, the search can be replaced by direct computation.

The algorithm can be easily extended to arbitrary rays by mirroring the scene along the axes with negative projection vector coordinates:

```
if(r_ν < 0) then
    p_ν ← S_ν^{N_ν} − p_ν
    S_ν^i ← S_ν^{N_ν} − S_ν^i
endif
```

and by replacement of coordinate increments by decrements in Fig. 12. This inversion should be taken into account in addressing both the 3D attribute and data arrays.

A faster version of the algorithm can be obtained by replacing the sample coordinate p_ν by $\frac{p_\nu}{r_\nu}$, by which we remove the divisions in (12) and multiplications in **S**lave-**S**tep macros (Fig. 12). This change should be applied also to the S_ν arrays. Of course, special cases of rays perpendicular to coordinate axes should be treated separately.

4.2 Modifications of the CD Traversal Algorithm

4.2.1 CD Traversal with Anisotropic Macro Regions

Let us imagine the following situation: The ray is passing in close vicinity to an object. The step size (and, therefore, also its speed) decreases first, then it reaches its minimum and, finally, it increases again. In the last phase, the factor limiting the step size is the distance to the object already passed. This drawback can be overcome by introduction of *anisotropic* macro regions:

$$\begin{aligned} \mathcal{O}^n(\mathcal{V}) = \{h_{pqr} = 0 : v_x - n_x^- \le p \le v_x + n_x^+, \\ v_y - n_y^- \le q \le v_y + n_y^+, v_z - n_z^- \le r \le v_z + n_z^+\}, \end{aligned} \quad (13)$$

where

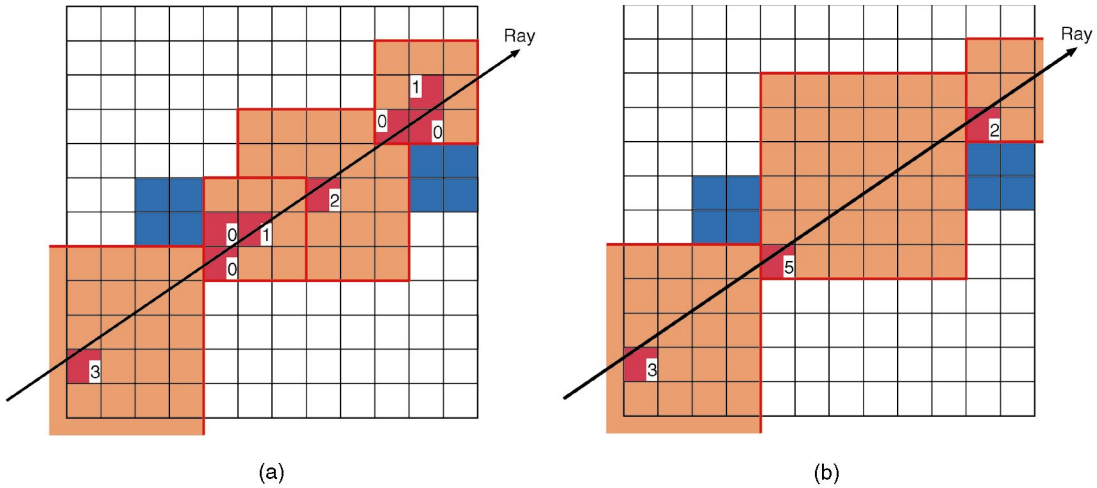


Fig. 9. Comparison of CD traversal with (a) isotropic, and (b) anisotropic macro regions. See legend in Fig. 8.

$$n_\nu^- = \begin{cases} n & \text{if } r_\nu < 0 \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

and

$$n_\nu^+ = \begin{cases} n & \text{if } r_\nu \geq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

We see that, in this case, the actual voxel lies in one of the macro region vertices. Of course, instead of one symmetric macro region, eight anisotropic macro regions should be assigned to each background voxel. Which of them is loaded to accelerate the traversal depends on the projection vector parameters. Both CD voxel traversal approaches with symmetric and anisotropic macro regions are compared in Fig. 9. We see that the additional acceleration is obtained by increasing the speed of the rays passing in close vicinity of objects.

The anisotropic macro regions can be obtained by a minor change of the original algorithm computing the chessboard distance. Now, instead of two passes, only one is necessary. In order to get all eight combinations, each run should start from a different vertex of the scene with appropriate mask (Fig. 10).

4.2.2 *Template-Based CD Voxel Traversal*

The idea of template-based voxel traversal, introduced in [28], is based on an observation that, under special viewing conditions (parallel projection, rays starting on a *base-plane*), each ray follows the same sequence of voxels with respect to the starting voxel. Therefore, it is not necessary to generate the sequence for each ray separately. A special structure, a *ray-template*, stores all necessary information about the sequence. Since the base-plane is perpendicular to one of the coordinate axes, a distorted intermediate image is obtained, which should subsequently be warped to correct dimensions.

The proposed template-based CD voxel traversal algorithm is based on 26-connected templates [29]. Again, in the preprocessing phase, chessboard distance is assigned to all background voxels. Since the macro region represents a sphere in D_∞ metrics, which corresponds to the 26-connectedness of the ray template, knowing its size n enables us to skip n voxels of the template to reach the first voxel outside the macro region. The ray-template can be implemented as an array of voxels, therefore, skipping the whole macro region requires only one operation: increment of the actual template voxel coordinate by n.

4.3 Implementation

We implemented the algorithm on an HP720 workstation equipped with 32 MB of main memory. The first test data set was originally cubic. Therefore, we initially converted it to rectilinear, by summing several voxels in the top-bottom direction to get a typical CT data set: The voxel was $1 \times 1 \times 2$ in the lower facial part of the skull and $1 \times 1 \times 4$ in the cranial part. The size of the data set thus reduced from $175 \times 235 \times 225$ to $175 \times 235 \times 94$. Fig. 13a shows a distorted image when assuming cubic voxels. The distortion is removed in Fig. 13b, where proper voxel dimension were taken into account.

The second data set was obtained by a CT tomograph. Since the physicians were interested in the central part of the skull, it was scanned with 0.91 mm per slice, while the top part was scanned with resolution 6 times lower. Fig. 14 shows the distorted and correct renditions again.

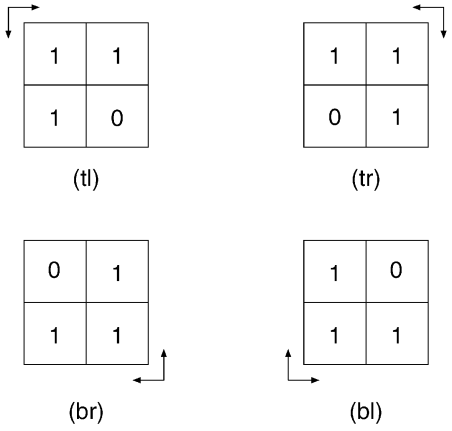


Fig. 10. Anisotropic CD traversal (2D case): CD computation masks. (tl)—top-bottom left-right run, (tr)—top-bottom right-left run, (br)—bottom-top right-left run, (bl)—bottom-top left-right run.

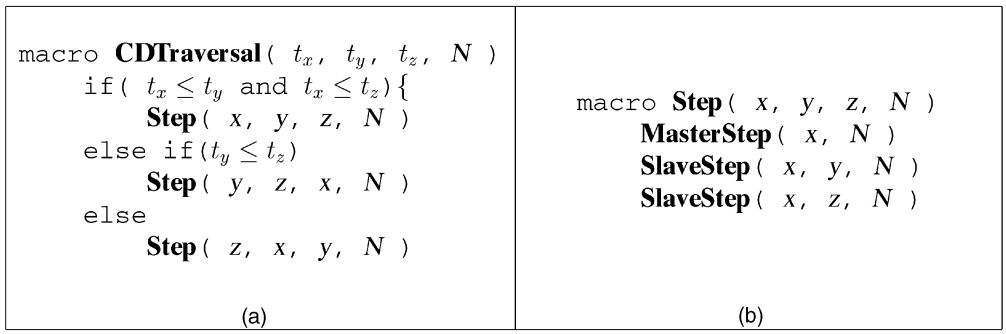


Fig. 11. Traversal of rectilinear grids by cubic macro-regions: (a) grid traversal and (b) step over a macro region. N stands for size of the macro region and $t_\nu, \nu = x, y \; or \; z$ is defined by (12).

5 Experimental Comparison of Different Ray Generators

Previous sections were devoted to description of several algorithms generating the ray as a sequence of voxels or samples. This description, however, does not enable us to decide about efficiency of this or that algorithm; this question can be answered only on behalf of an experimental evaluation. A performance comparison of various algorithms from the data published by their authors is impossible since different test scenes, as well as hardware platforms and operating systems, have usually been used for their implementation. Therefore, we designed our own tests, aimed at answering the following two questions: 1) Which is the efficiency of various distance-based ray traversal techniques, and 2) how do different techniques influence detection of ray-surface intersection point.

5.1 Comparison of Hierarchical and Distance Acceleration Techniques

In this subsection, we compare different ray traversal schemes from the point of view of background traversal efficiency, that is, that phase of ray traversal which is terminated either by detection of the first object voxel or by leaving the scene. For this purpose, an experiment was set up, based on rendering of phantom scenes with randomly positioned spheres of various size and number.

Each scene, built up of $128 \times 128 \times 128$ voxels, was subdivided into $N \times N \times N$ subregions (N = 2-21). Within each subregion, a voxelized sphere was randomly placed (8-9,261 spheres), with such size that the total volume of all spheres was identical for all N (Fig. 15). Thus, we obtained scenes with approximately an equal number of object voxels, but with different surface-to-volume ratio (SVR), which was computed as a ratio of the number of object surface voxels (having at least one background voxel in its 26-neighborhood) and the total number of object voxels. For the less complex scenes with big spheres, this ratio is low, while, for the scenes with a large number of small spheres, it approaches unity (nearly all object voxels are also surface voxels).

We compared three different classes of ray generators:

1. As a representative of techniques, visiting all voxels along the ray path, we selected Cleary and Wyvill's algorithm [8] for fast voxel traversal (FVT). It generates a 6-connected sequence of voxels pierced by the ray in a uniformly subdivided scene.
2. Hierarchical approaches are exemplified by the Spackman and Willis's SMART (Spatial Measure for Accelerated Ray Tracing) oct-tree traversal [26], which works on an oct-tree represented as a breadth first list.
3. The CD voxel traversal algorithm version for Cartesian grids typifies the distance techniques.

In the experiment, only parallel primary rays were traced until the first object voxel was found, with no subsequent shading. Results of the experiment are depicted in Fig. 16, where y axis values represent the pure traversal time necessary to render a 250×250 image and x axis defines scene complexity.

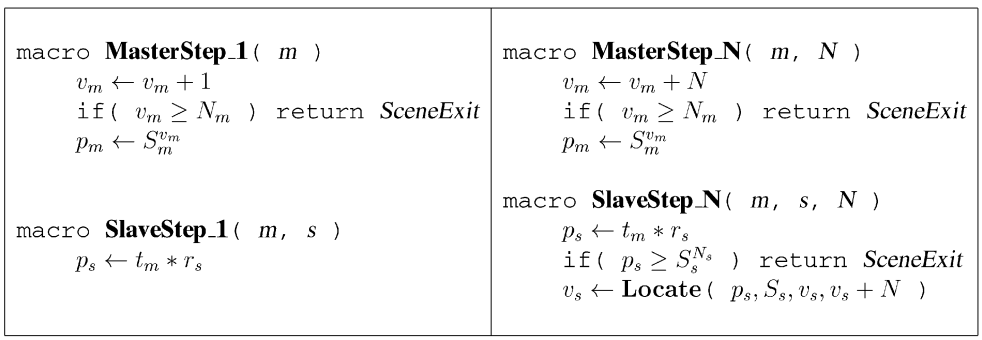
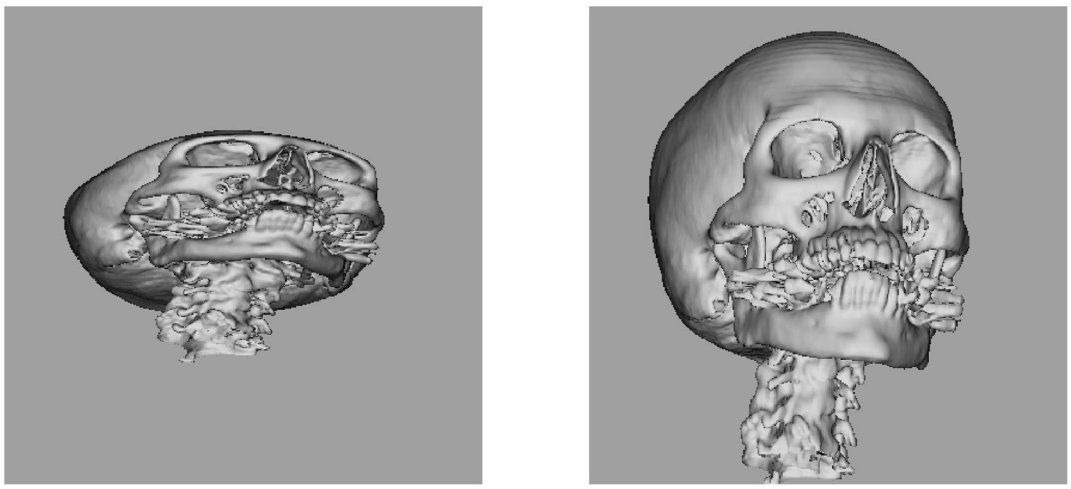
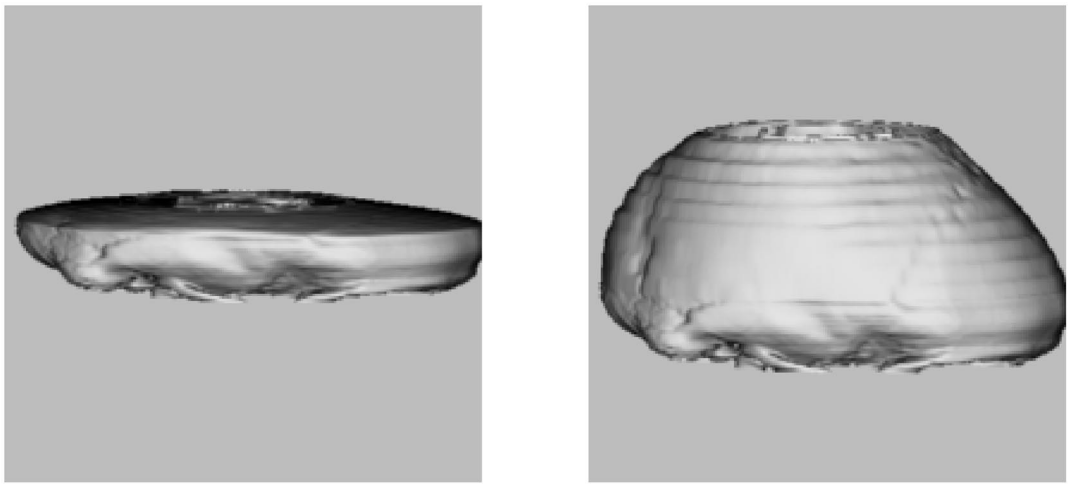


Fig. 12. Update of control variables for one voxel step (left) and steps over macro regions with size N (right). m denotes the driving axis and s one of the dependent axes. (Variables are defined in Sections 4 and 4.1.)



(a) (b)

Fig. 13. Data set 1: (a) cubic voxel assumed, (b) voxel dimensions taken into account.



(a) (b)

Fig. 14. Data set 2: (a) cubic voxel assumed, (b) voxel dimensions taken into account.

TABLE 1
Comparison of Background Traversal and Rendering Times for the Single Voxel and Macro-Region Versions of the Algorithm (Data Set 1)

version	background traversal time [s]	rendering time [s]
single voxel	56.1	67.2
macro-region	18.5	29.1

If we render two scenes with an equal number of occupied voxels, but with different distribution in space, surface is usually found earlier in the more complex one. This tendency is illustrated in Fig. 16 by the FVT curve. In this case, the shorter total traversed distance reduces render time since this is not influenced by a variable traversal step. Both the CD and the SMART curves are the result of two contradictory tendencies: Smaller steps in densely populated scenes tend to increase the traversal time, while the shorter traversed distance decreases it. The first tendency prevails on the left side of the graph, where the time increases with increasing scene complexity; while the second tendency prevails on the opposite side.

Fig. 17 depicts traversal cost maps for the CD, FVT, and SMART algorithms (images (c) and (d) are enhanced because they were originally too dark). The brighter the value, the more traversal steps were necessary to find the object surface. We see that the greatest difference between FVT cost, on one hand, and CD and SMART cost, on the other hand, is in regions where rays totally miss the object. We can also see a blocky structure in the SMART algorithm cost map which is caused by geometric properties of the oct-tree structure. The CD algorithm cost map is free of this structure, which shows that the CD macro regions adapt better to the object shape than the oct-tree structure. Together with the complex oct-tree traversal algorithm, this

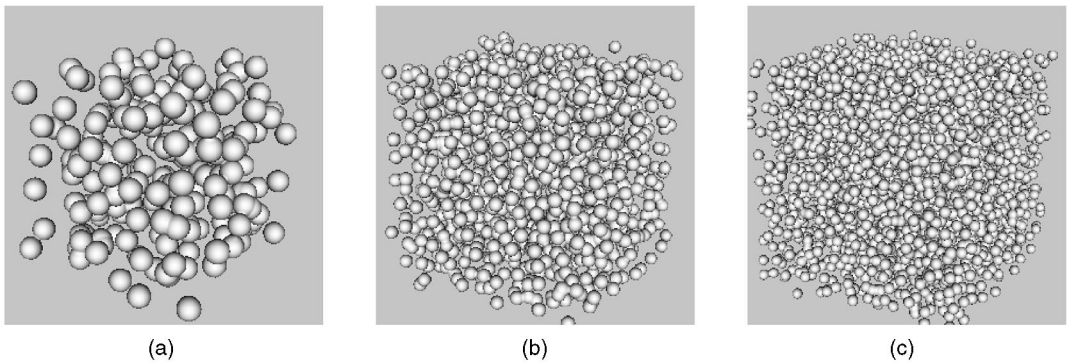


Fig. 15. Example of test scenes with different complexity: (a) 216 spheres, $SVR = 0.549$; (b) 1,331 spheres, $SVR = 0.794$; and (c) 2,744 spheres, $SVR = 0.891$.

is the reason for longer traversal times of the SMART algorithm in comparison to that obtained for the CD algorithm (Fig. 16).

On the basis of this experiment, we see that the distance-based algorithms can significantly accelerate the background traversal phase of rendering by ray tracing. For a simple scene, with one or a few objects in its center, the acceleration can reach values of around 10 in comparison to the FVT. Although this ratio decreases with growing scene complexity, the CD algorithm is faster than FVT even for such complex scenes, as for example, that in Fig. 15b with 1,331 simulated objects. In the scenes with extremely high complexity (Fig. 15c), where mean length of a traversed ray is only a few voxels, their speed is approximately equal.

5.2 Comparison of Different Distance-Based Techniques

Cohen and Sheffer [13] theoretically compared performance of the DDA voxel traversal algorithms based on Euclidean (which was approximated by the chamfer distance) and city block distance. They showed that, although the city block distance was only a rough approximation of the Euclidean distance to the nearest object surface point, it allowed even better acceleration than the true Euclidean distance (they estimated ratio of mean step length for both distances to be $\bar{s}_b/\bar{s}_e = 1.0316$). Similar results were obtained also for the chessboard distance ($\bar{s}_c/\bar{s}_e = 1.0145$ [23]).

We experimentally verified these theoretical conclusions by rendering the same phantom scenes again. The graph in Fig. 18a shows dependency of the mean step length on scene complexity. In this case, in order to obtain a correct distance comparison, the DDA algorithm is not switched to FVT in the object vicinity (see Section 3 for why it should be). Although this introduces rendering errors and, therefore, for each distance type used, it gives different total traversed length, it enables correct comparison of the mean DDA step lengths. We can see that the DDA algorithm mean step is almost nearly independent of distance type used for all scenes.

Fig. 18a also allows us to compare the DDA with the CD algorithm. We can see that, over the whole range of scenes, the step of the CD is approximately 1 voxel unit longer. This

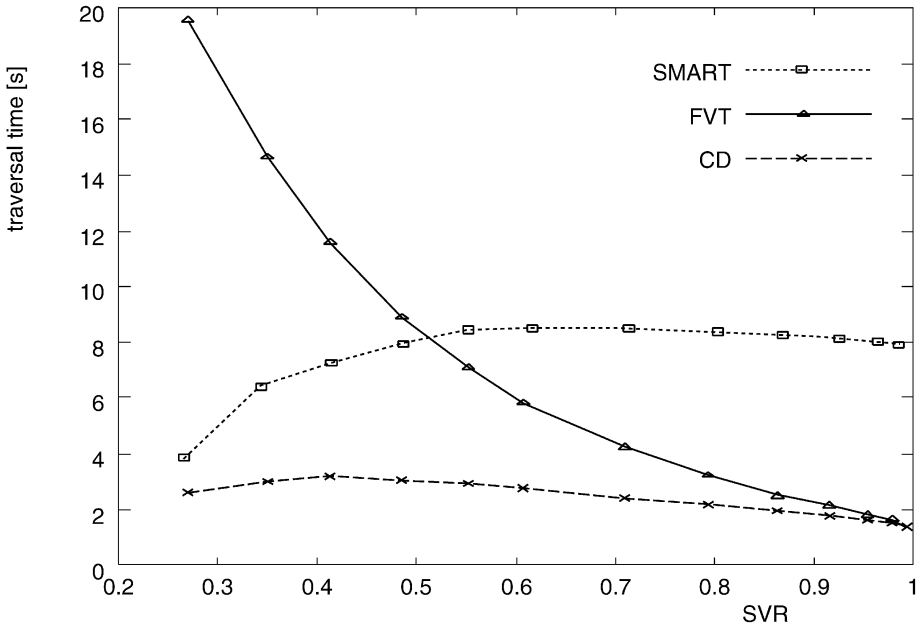


Fig. 16. Dependency of the background traversal time on scene complexity.

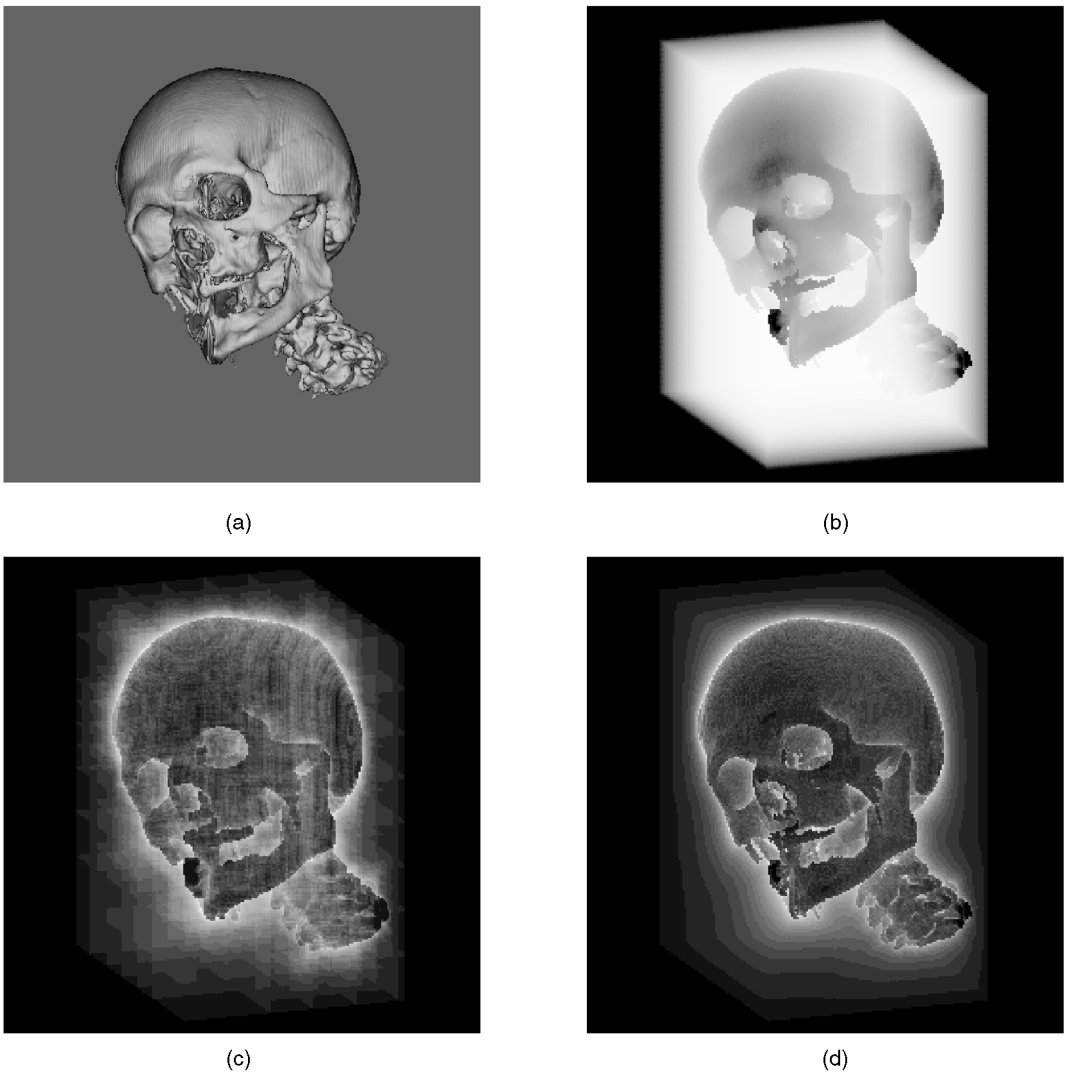


Fig. 17. Comparison of efficiency of different ray generators: (a) shaded surface, (b) FVT algorithm cost, (c) SMART algorithm cost, and (d) CD algorithm cost. (Images (c) and (d) are enhanced because they were originally too dark.)

is the consequence of the precision of this algorithm (again, see Section 3), which makes utilization of the full macro region size possible.

Fig. 18b shows results of a similar experiment, but, in this case, the DDA traversal is switched to FVT in the object vicinity. We see that the mean step in the DDA case is shorter now since some fraction of the total distance is traversed by FVT with short steps. The DDA with city block distance is more effective than with chessboard and chamfer distances. This can be explained by larger city block distance values at voxels near an object surface due to which the algorithm is switched to the precise FVT with a shorter step later (Fig. 19). This was verified by counting the steps of the FVT algorithm after switching. Their number was actually 1.4-2.3 times higher (depending on the scene complexity) for the chessboard and chamfer distances than for the city block distance. Due to the switch to FVT, the earlier mentioned difference between the chessboard and chamfer distances in complex scenes vanishes.

The experiment enabled us to compare two parameters which were dependent not on scene complexity, but exclusively on the algorithm itself: *initialization time* and *mean step cost* (Table 2). The largest overhead is added by the CD algorithm due to a quite complex initialization of its decision variables.

The last graph (Fig. 20) answers a question, how does the switch from the CD (DDA) to FVT algorithm in the vicinity of an object influence the rendering time. The higher DDA and CD step cost indicate a possibility that, in the case of a lower assigned distance, the performance of the FVT can be better. Fig. 20 shows the results obtained by switching from the CD algorithm to FVT at different distances (no switch, switch at distance 1, 2, and 3). Similar results were obtained also for the DDA with all three distance types. We see that switching degrades the algorithmic performance because of the overhead necessary for conversion from CD to FVT traversal. Moreover, if no surface is found, a switch back from FVT to CD is also necessary. Therefore, it is useful to switch only from DDA to FVT, but only because of the rendering errors, which can occur due to the possible miss of some object voxels.

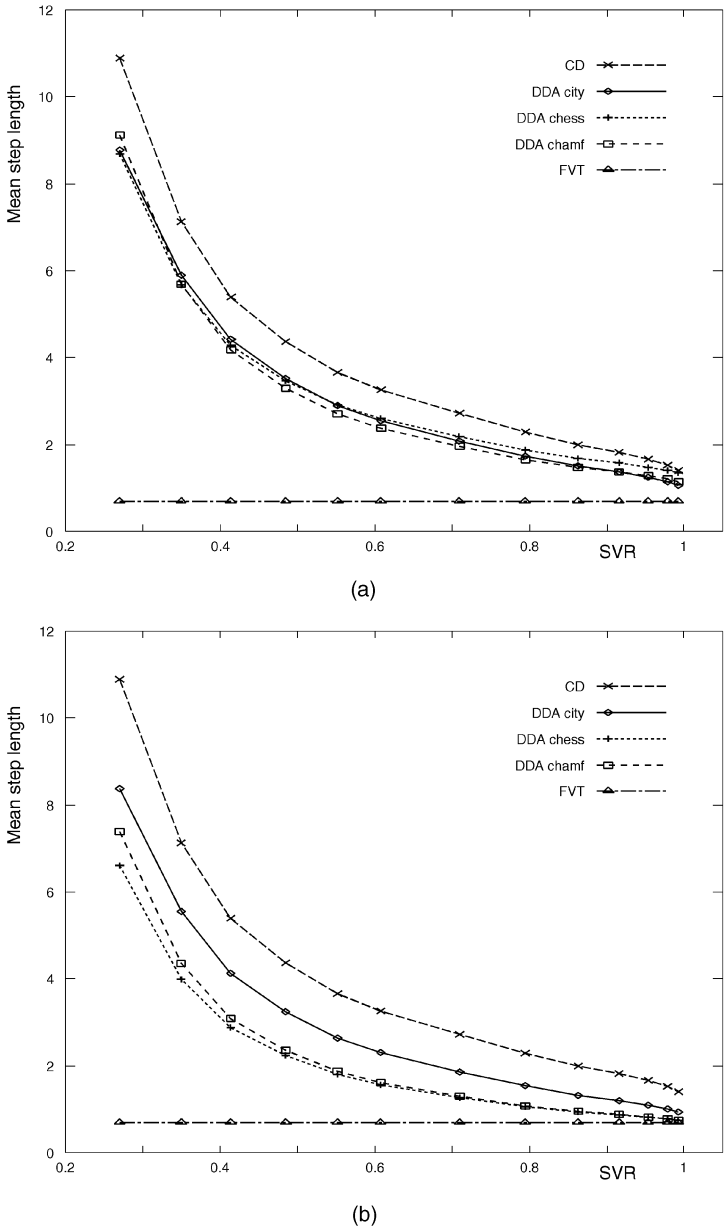


Fig. 18. DDA, CD, and FVT voxel traversal mean step vs. scene complexity. (a) DDA algorithm with no switch to FVT, (b) DDA with switch to FVT in the object vicinity.

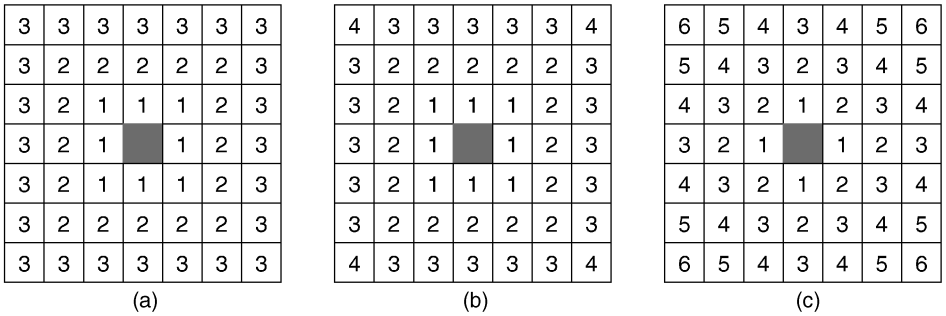


Fig. 19. (a) chessboard, (b) chamfer, and (c) city block distance in the vicinity of an object pixel (a 2D cut of a volume).

5.3 Hit-Miss Test and Surface Detection Comparison

As we have already mentioned in the previous sections, choice of the background traversal algorithm can influence the hit-miss test and ray-surface intersection computation. From this point of view, we compared two possibilities for the discrete line definition, based on

TABLE 2
Scene Independent Parameters for the Tested Algorithms

algorithm	Distance type	Initialization time [s]	Step cost
CD	chessboard	0.54	2.15
FVT	—	0.21	1.00
DDA	city block	0.27	1.53
DDA	chessboard	0.30	1.52
DDA	chamfer	0.48	1.50

Step cost is expressed relative to the FVT algorithm.

1. nonuniform samples, lying on faces of pierced voxels (represented by the CD algorithm) and
2. equidistant samples along the ray (represented by the 3D DDA).

For comparison purposes, a scene with a voxelized sphere was generated and rendered with such parameters that all rays hit the surface.

The first case of a nonuniform sequence of samples at voxel faces is depicted in Fig. 21a. Once an object voxel is reached, it is necessary to evaluate the interpolating function $\mathcal{F}$ at the sample points (Fig. 21a, points 2, 3) until its value exceeds the threshold T (point 3). In the final step, the exact position of the intersection should be searched for between the last two samples (points 2 and 3).

It should be pointed out that:

1. It is necessary to evaluate $\mathcal{F}$ only at samples on faces shared by two object voxels since those lying between an object and background voxel (Fig. 21a, point 1) will always give values below the threshold T,
2. For the samples lying on voxel faces, the trilinear interpolation function degenerates to bilinear, which can be computed faster, and
3. Since we assume continuous gradient of $\mathcal{F}$ within the voxel, a numerical root finding method based on derivatives can be involved (for example, Newton-Raphson).

However, the situation is quite different with equidistant DDA samples (case 2). The complete trilinear function $\mathcal{F}$ should be evaluated often for more points (Fig. 21b, points 1, 2, 3) and bisection should be used due to a possible discontinuity of its derivatives at voxel faces. Some overhead is also added since the search process works within more than one voxel.

Results of this experiment are summarized in Fig. 22. In addition to the two already mentioned iterative root finding schemes (`newton, bisect`) a simple noniterative intersection estimation by linear interpolation between the two $\mathcal{F}$ values above and under the threshold T was added. The rendering times are expressed with respect to the best result reached with the iterative approach.

The background traversal, although with 3D complexity, represents the shortest time interval since much more time is spent for the hit-miss test and the surface position detection. The `bisect` surface detection is slower for the 3D DDA than for the CD (more voxels involved), which is even slower than `newt` surface detection (derivatives). The 3D DDA hit-miss test also needs more time, due to more samples and trilinear interpolations.

We can see that the approach with uniform ray sampling needs nearly three times more time than that with samples on voxel faces. The intersection estimation (`int`) is, of course, the fastest, although not as precise. However, its precision is high enough for applications with just primary rays, as is usually the case for visualization of scanned objects.

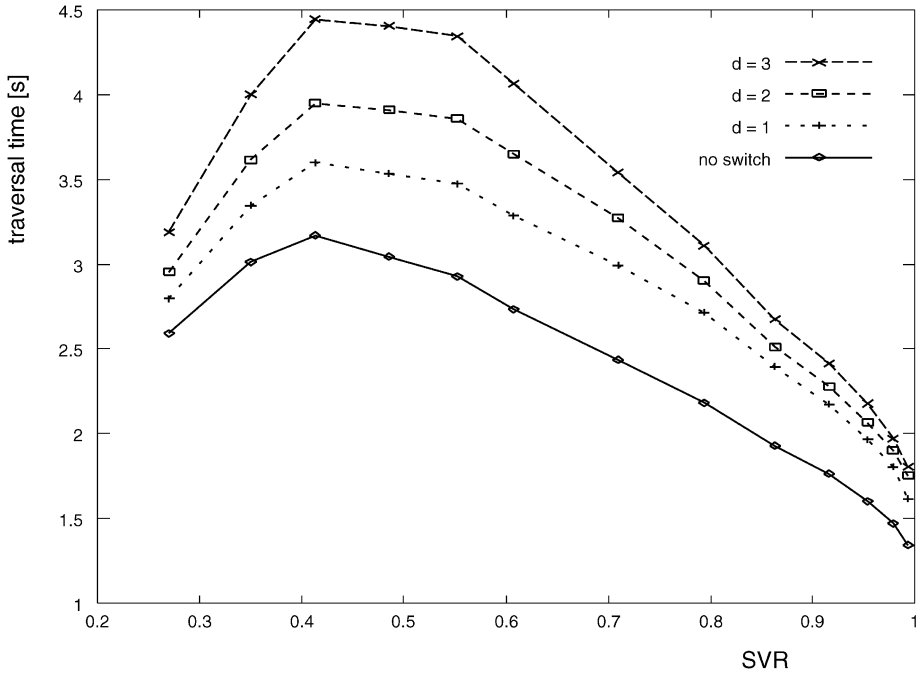


Fig. 20. CD algorithm switched to FVT.

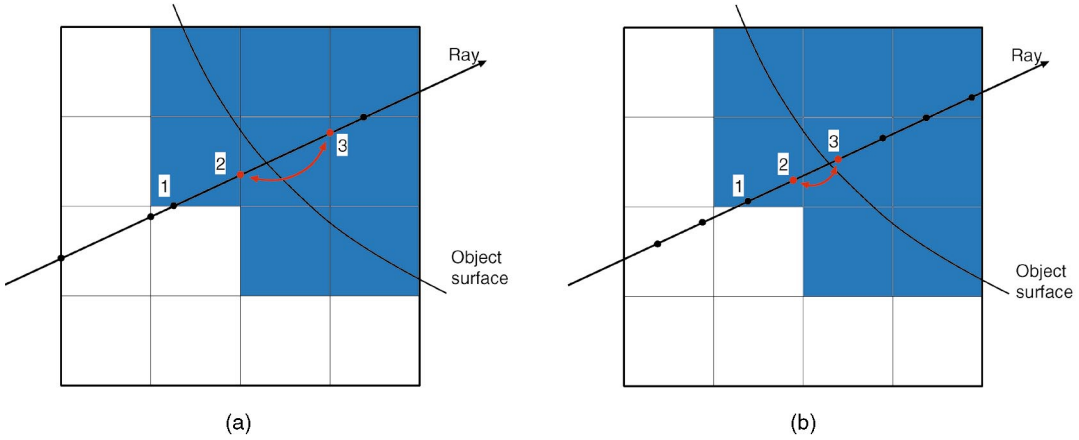


Fig. 21. Surface point detection: (a) CD and (b) 3D DDA algorithms.

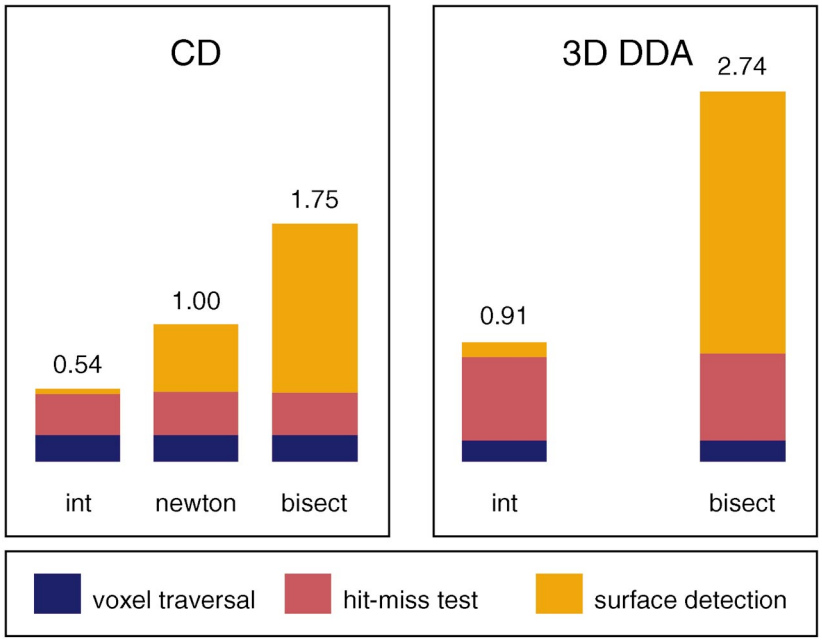


Fig. 22. Relative background traversal and surface detection times for two different ray generators.

6 Conclusion

We presented the *Chessboard Distance Voxel Traversal Algorithm* aimed at accelerating visualization of voxel grids by ray tracing or ray casting. Its most important features are:

- The algorithm assumes segmented data sets and proper initialization of all background voxels by their chessboard distance to the nearest foreground voxel.
- The speed up is gained by fast traversal of background areas of the volume utilizing macro regions, defined around each background voxel by means of the assigned chessboard distance value. Since the macro regions are empty, they can be skipped in a single long leap.
- No additional storage is necessary in the case of segmented and labeled data since the macro regions are defined by a single integer, stored within the otherwise empty background voxel.
- The distances are view independent and thus should be computed only once, independently of the viewing parameters.
- The algorithm is not connected with any special visualization technique. It can be used to speed of rendering with surface reconstruction by interpolation, as well as volume oriented techniques with accumulation.
- The algorithm enables us to define rays with arbitrary direction defined in floating point precision, starting within or outside of the voxel volume.
- The algorithm supports fast ray-surface interpolation detection.

In addition, we performed several experimental comparisons, aimed at speed analysis of different voxel traversal acceleration techniques. We have shown that both hierarchical and distance-based voxel traversal techniques prevail in performance over the single voxel step approaches introducing up to a 10-fold speed up of the background traversal. Furthermore, we have shown that

both analyzed distance approaches (3D DDA and CD) outperform the oct-tree approach (SMART). Mutual comparison of both distance approaches have shown that they traverse the background regions with approximately equal speed. However, the CD algorithm supports faster detection of ray-surface intersections, which results in a significant speed up of the whole rendering process.

Finally, the most important difference of the CD algorithm, in comparison to the 3D DDA techniques, resides in its ability to work with rectilinear volumes, further improving the visualization speed. Thus, the algorithm makes possible visualization without the necessity to resample to cubic voxels, as, for example, in the case of CT scans with variable slice thickness. It further provides us with a possibility of representing volume areas with lack of details in lower resolution, thus saving space and adding additional speed up.

ACKNOWLEDGMENTS

This work has been supported by US Office of Naval Research grant N000149710402, US National Research Laboratories grant N00014961G015, US National Science Foundation grant MIP9527694 and VEGA 2/6017/99 (Slovak Republic). The data sets were provided by the University of North Carolina and Dr. Serge Weis from the University of Munich, Germany.

REFERENCES

[1] J.R. Wallace, K.A. Elmquist, and E.A. Haines, "A Ray Tracing Algorithm for Progressive Radiosity," *Proc. SIGGRAPH '89,* pp. 315-324, July 1989.
[2] L.M. Sobierajski and A. Kaufman, "Volumetric Ray Tracing," *Proc. IEEE Symp. Volume Visualization,* pp. 11-18, 1994.
[3] A. Kaufman, D. Cohen, and R. Yagel, "Volume Graphics," *Computer,* vol. 26, no. 7, pp. 51-64, July 1993.
[4] A. Kaufman, "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes," *Proc. SIGGRAPH '87,* pp. 171-179, July 1987.
[5] S.W. Wang and A. Kaufman, "Volume-Sampled 3D Modeling," *IEEE Computer Graphics and Applications,* vol. 14, no. 5, pp. 26-32, Sept. 1994.
[6] M. Sramek and A. Kaufman, "Alias-Free Voxelization of Geometric Objects," *IEEE Trans. Visualization and Computer Graphics,* vol. 5, no. 3, pp. 251-266, 1999.
[7] A. Fujimoto, T. Tanaka, and K. Iwata, "Arts: Accelerated Ray-Tracing System," *IEEE Computer Graphics and Applications,* vol. 6, no. 4, pp. 16-26, 1986.
[8] J.C. Cleary and G. Wyvill, "Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision," *The Visual Computer,* vol. 4, no. 2, pp. 65-83, July 1988.
[9] J. Amanatides and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," *Proc. EUROGRAPHICS '87,* pp. 3-10, 1987.
[10] M. Levoy, "Efficient Ray Tracing of Volume Data," *ACM Trans. Computer Graphics,* vol. 9, no. 3, pp. 245-261, 1990.
[11] K.J. Zuiderveld, A.H.J. Koning, and M.A. Viergever, "Acceleration of Ray-Casting Using 3D Distance Transforms," *Visualization in Biomedical Computing II, Proc. SPIE 1808,* pp. 324-335, 1992.
[12] R. Yagel and Z. Shi, "Accelerating Volume Animation by Space-Leaping," *Proc. Visualization '93,* pp. 62-84, 1993.
[13] D. Cohen and Z. Sheffer, "Proximity Clouds—An Acceleration Technique for 3D Grid Traversal," *The Visual Computer,* vol. 10, no. 11, pp. 27-38, Nov. 1994.
[14] M. Sramek, "Fast Surface Rendering from Raster Data by Voxel Traversal Using Chessboard Distance," *Proc. Visualization '94,* pp. 188-195, 1994.
[15] G. Borgefors, "Distance Transformations in Digital Images," *Computer Vision, Graphics, and Image Processing,* vol. 34, no. 3, pp. 344-371, 1986.
[16] M. Sramek, "Fast Ray-Tracing of Rectilinear Volume Data," *Proc. Virtual Environments and Scientific Visualization '96,* pp. 201-210, 1996.
[17] W.E. Lorensen and H.E. Cline, "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm," *Proc. SIGGRAPH '87,* pp. 163-169, 1987.
[18] I. Bajla, I. Holländer, "Nonlinear Filtering of Magnetic Resonance Tomograms by Geometry-Driven Diffusion," *Machine Vision and Applications,* no. 10, pp. 243-255, 1998.
[19] S.R. Marschner and R.J. Lobb, "An Evaluation of Reconstruction Filters for Volume Rendering," *Proc. Visualization '94,* pp. 100-107, Oct. 1994.
[20] M.J. Bentum, B.B.A. Lichtenbelt, and T. Malzbender, "Frequency Analysis of Gradient Estimators in Volume Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 2, no. 3, pp. 242-254, Sept. 1996.
[21] I. Carlbom, "Optimal Filter Design for Volume Reconstruction and Visualization," *Proc. Visualization '93,* pp. 54-61, 1993.
[22] T. Möller, R. Machiraju, K. Mueller, and R. Yagel, "Evaluation and Design of Filters Using a Taylor Series Expansion," *IEEE Trans. Visualization and Computer Graphics,* vol. 3, no. 2, pp. 184-199, Apr.-June 1997.
[23] M. Sramek, *Visualization of Volumetric Data by Ray Tracing.* Austria: Austrian Computer Society, 1998.
[24] U. Tiede, T. Schiemann, and K.H. Höhne, "High Quality Rendering of Attributed Volume Data," *Proc. Visualization '98,* pp. 255-262, 1998.
[25] R. Yagel, D. Cohen, and A. Kaufman, "Discrete Ray Tracing," *IEEE Computer Graphics and Applications,* vol. 12, no. 5, pp. 19-28, Sept. 1992.
[26] J. Spackman and P. Willis, "The SMART Navigation of a Ray through an Oct-Tree," *Computers & Graphics,* vol. 15, no. 2, pp. 185-194, 1991.
[27] O. Devillers, "The Macro-Regions: An Efficient Space Subdivision Structure for Ray Tracing," *Proc. Eurographics '89,* pp. 27-38, 1989.
[28] R. Yagel and A. Kaufman, "Template-Based Volume Viewing," *Proc. Eurographics '92,* pp. C153-C167, 1992.
[29] I. Holländer and M. Sramek, "An Interactive Tool for Manipulation and Presentation of 3D Tomographic Data," *Proc. CAR '93 Computer Assisted Radiology,* pp. 278-383, 1993.

Milos Sramek received his Ing (MSc) degree from the Faculty of Electrical Engineering, Slovak Technical University, Bratislava, Slovakia, in 1982. Since 1988, he has been a member of the research team at the Institute of Measurement Science, Slovak Academy of Sciences, Bratislava, Slovakia. From 1991 to 1997, he simultaneously worked at the Institute of Information Processing of the Austrian Academy of Sciences in Vienna, Austria, on different projects in the area of medical imaging and remote sensing. In 1996, he completed and defended his PhD thesis "Visualization of Volumetric Data by Ray Tracing" at the Vienna University of Technology, Austria. From 1997 to 1999, he worked in a postdoctoral position at the State University of New York at Stony Brook. Since 1999, he has been with the Commission for Scientific Visualization of the Austrian Academy of Sciences. Vienna, Austria. His research interests include image processing, 2D and 3D data segmentation and visualization, and voxelization and volume rendering algorithms.

Arie E. Kaufman received a BS in mathematics and physics from the Hebrew University of Jerusalem in 1969, an MS in computer science from the Weizmann Institute of Science, Rehovot, in 1973, and a PhD in computer science from the Ben-Gurion University, Israel, in 1977. He is the director of the Center for Visual Computing (CVC), a leading professor and chair of the Computer Science Department, and leading professor of Radiology at the State University of New York at Stony Brook. He was the founding editor-in-chief of the *IEEE Transaction on Visualization and Computer Graphics* (TVCG), 1995-1998. Dr. Kaufman has been the cochair for multiple Eurographics/Siggraph Graphics Hardware Workshops, the papers or program cochair for the IEEE Visualization '90-'94 and ACM Volume Visualization Symposium '92, '94, '98, and the cofounder and member of the steering committee of the IEEE Visualization conference series. He has previously chaired and is currently a director of the IEEE Computer Society Technical Committee on Visualization and Computer Graphics. He is the recipient of a 1995 IEEE Outstanding Contribution Award, the 1996 IEEE Computer Society's Golden Core Member, 1998 IEEE Fellow, 1998 ACM Service Award, and 1999 IEEE Computer Society's Meritorious Service Award. Dr. Kaufman has conducted research and consulted for about 30 years specializing in volume visualization; graphics architectures, algorithms, and languages; virtual reality; user interfaces; and multimedia. For more information see http://www.cs.sunysb.edu/~ari.