MIT-SHM *(MIT Shared Memory Extension)* facilitates memory-sharing for certain data on one machine between the server and client. **Input extension** *(X11 Input Extension Protocol Specification* and *X11 Input Extension Library Specification)* is the standard mechanism for the definition and use of additional input devices. Support for 3-D graphics introduced in the current release X11R5 is implemented as an extension of the X server as well. The **X3D-PEX** *(PHIGS extension to X)* extension is defined in the draft X standard.

Some vendors provide their own extensions of the X server. The MIT X Consortium controls the registration and standardization of these extensions. Currently, there are more than 40 different extensions of the X server registered. Some of them are limited to one platform; others are widely available. A well-known frequent extension is support for the **Display PostScript** language *(Adobe-DPS-Extension)*. Special hardware and software of Silicon Graphics workstations can be used by means of the **SGI-XGL** extension.

References

[1] Asente, P. – Swick, R.R.: *X Window System Toolkit. The Complete Programmer's Guide and Specification.* X Version 11, Release 4. Digital Press, Bedford, MA, 1990.

[2] Barkakati, N.: *UNIX Desktop Guide to OPEN LOOK.* SAMS, Carmel, IN, 1992.

[3] Limpouch, A.: *X Window System - programování aplikací.* Grada, Praha, 1993.

[4] McCormack, J. – Asente, P.: An Overview of the X Toolkit. *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software* (Banff, Oct. 17–19, 1988), ACM Press, 1988, pp. 46–55.

[5] Mikes, S.: *X Window System Program Design and Development.* Addison-Wesley, Reading, MA, 1992.

[6] Scheifler, R.W. – Gettys, J.: *X Window System. The Complete Reference to Xlib, X Protocol, ICCM, XLFD.* X Version 11, Release 4. Digital Press, Bedford, MA, 1990.

[7] Young, D.A.: *X Window Systems: Programming and Applications with Xt.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[8] Peterson, Ch.D.: *Athena Widget Set - C Language Interface.* X Window System. X Version 11, Release 5. MIT, Cambridge, MA, 1991.

[9] Rosenthal, D.: *Inter-Client Communication Conventions Manual.* Version 1.1. MIT X Consortium Standard. X Version 11, Release 5. MIT, Cambridge, MA, 1991.

[10] Flowers, J.: *X Logical Font Description Conventions.* Version 1.4. MIT X Consortium Standard. X Version 11, Release 5. MIT, Cambridge, MA, 1991.

[11] Gettys, J. – Scheifler, R.W.: *Xlib - C Language Interface.* MIT X Consortium Standard. X Version 11, Release 5. First Revision. MIT, Cambridge, MA, 1991.

[12] McCormack, J. – Asente, P. – Swick, R.R.: *X Toolkit Intrinsics - C Language Interface.* X Window System. X Version 11, Release 5. First Revision. MIT, Cambridge, MA, 1991.

An Algorithm for Fast Voxel Scene Traversal

Miloš Šrámek*
Institute of Measurement Science
Slovak Academy of Sciences,
Dúbravská cesta 9, 842 19 Bratislava, ČSFR

Abstract

Together with improvement of tomographic methods of scanning 3D data, residing in an increase of distinguishing ability of the scanners, the quality of final 3D reconstruction of the data comes into foreground. An inevitable condition for visualization of small details, comparable with the voxel size is application of such algorithms, which enables to define a surface of the scanned object on subvoxel level. Coming out from paper published at previous year of *Winter School on Computer Graphics and CAD Systems* we present an algorithm for fast traversal of the voxel scene, which enables to find a nearest intersection of a ray with the object surface defined in such a way. High speed of the algorithm utilizes object coherency of the scene, which means that voxels belonging to the object are usually grouped in connected regions.

The algorithm consists from two steps: preprocessing and scene traversal. In the preprocessing phase we assign to each background voxel of the segmented scene a value equal to its chessboard distance from the nearest object voxel, which defines a cubic macro region with no object voxels inside. While generating a discrete ray, we can jump over this region, which results in up to 6-fold speed up of the traversal. The algorithm has no additional demands on memory, since the distance is stored in originally "empty" background voxels.

1 Introduction

Together with improvement of tomographic methods of scanning 3D data, residing in an increase of distinguishing ability of the scanners, the quality of final 3D reconstruction of the data comes into foreground. An inevitable condition for visualization of small details, comparable with the voxel size is application of such algorithm, which enables to define a surface of the scanned object on subvoxel level. Traditionally, triangulation methods[LC87] are used, which approximate object surface defined within a neighborhood of 8 voxels(cell)

by up to 4 triangles. In the first phase a triangular model of the surface is constructed (up to 10^6 triangles, according to complexity of the object), which is subsequently rendered by standard tools. Without a hardware accelerator, rendering of such amount of triangles can be very slow.

Different approach is direct surface rendering based on ray casting which assumes:

- defining an object surface approximation within a cell,
- finding the cell, where the ray-surface intersection can be found and
- exact computation of the intersection position.

In the fist part of the paper we shall deal with a question of suitable surface approximation function and in the second we shall describe an algorithm for fast background voxel traversal, which is aimed to find first such cell, where intersection ray-surface can be found.

2 Definitions

Let 3D image $\mathcal{P}$ be a set of $K \times L \times M$ values, which represent samples of some measured property in vertices of regular unit grid:

$$\mathcal{P} = \{p_{ijk} \in \mathcal{R} : 0 \leq i < K, 0 \leq j < L, 0 \leq k < M\},$$

where i, j and k are integers.

Let us segment $\mathcal{P}$ into n subsets u_l, such that

$$\bigcup_l u_l = \mathcal{S} \quad \wedge \quad \bigcap_l u_l = \emptyset$$

Let u_l be l^{th} object and l its identifier. Let $\mathcal{I} = \{0, 1, \ldots n-1\}$ be a set of object identifiers and let $l \in \mathcal{I}$. Binary image $\mathcal{B}$ is the set

$$\mathcal{B} = \{b_{ijk} \in \mathcal{I} : 0 \leq i < K, 0 \leq j < L, 0 \leq k < M\},$$

Without any loss of generality we can assume, that border values of $\mathcal{B}$, i.e. such, for which at least one of coordinates i, j or k is either equal 0 or $K-1$, $L-1$ and $M-1$ respectively, belongs to the 0^{th} object.

Voxel let be a tuple $V_{ijk} = (v_{ijk}, h_{ijk})$, where v_{ijk} is voxel volume

$$v_{ijk} = < i, i+1) \times < j, j+1) \times < k, k+1)$$

and h_{ijk} is its value, $h_{ijk} \in \{0, 1\}$. Point $[i, j, k]$ is voxel vertex and similarly point $[i + 0.5, j + 0.5, k + 0.5]$ is voxel center. Voxel value h_{ijk} will have the following meaning:

$h_{ijk} = 0$ means certainty that none of the object surfaces pass through this voxel and

$h_{ijk} = 1$ means, that we are not sure with this.

The voxel value depends on the binary image $\mathcal{B}$ and on a choice of the surface approximation. Voxel with value equal to 0(1) we denote 0-voxel(1-voxel).

The set $\mathcal{S}(p, q, r) = \{V_{ijk} : 0 \leq i < p, 0 \leq j < q, 0 \leq k < r\}$ we denote voxel scene and the set of points

$$D = \{(i, j, k) : 0 \leq i < p, 0 \leq j < q, 0 \leq k < r\}$$

voxel scene domain.

The scene $\mathcal{S}^C(M, N, O)$ will be a center representation of image $\mathcal{P}(\mathcal{B})$, if the sample $p_{ijk}(b_{ijk})$ will be assigned to the voxel V_{ijk} center (Fig. 1a). Analogously, if the sample will be assigned to voxel vertex, the vertex representation will be the scene $\mathcal{S}^V(M-1, N-1, O-1)$(Fig. 1b). It is obvious that size of scene $\mathcal{S}^V$ along each axis is 1 voxel smaller than size of $\mathcal{S}^C$.

Let us the voxels with corresponding $b_{ijk} = 0$ denote background voxels (in both representations) and the rest foreground voxels.

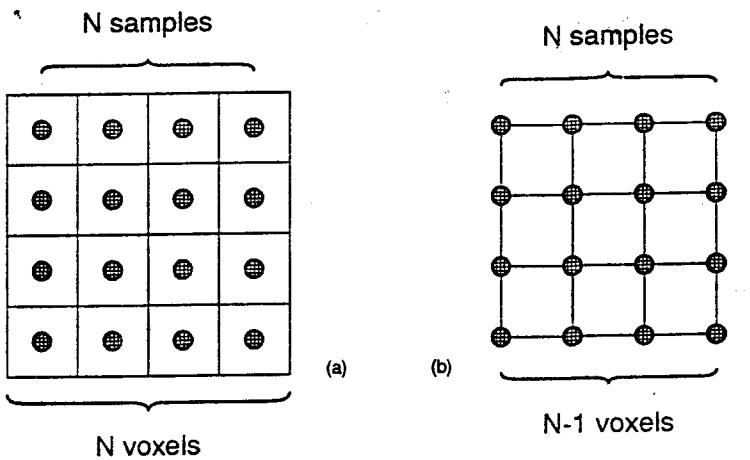

Fig. 1: (a) center and (b) vertex image representation

3 Definition of surface point by ray casting

Let the surface S_l of the l^{th} object u_l is defined by means of interpolating function $\mathcal{F}_l$ and a threshold value t_l:

$$S_l = \{\mathcal{F}_l(x,y,z) = t_l : (x,y,z) \in \bigcup_{ijk} v_{ijk}\}$$

where

$$\mathcal{F}_l(x,y,z) = \sum_{ijk} F_l^{ijk}(x,y,z)$$

and

$$F_l^{ijk}(x,y,z) = \begin{cases} F_l(x,y,z,\sigma_{ijk}) & \text{ak } [x,y,z] \in v_{ijk} \\ 0 & \text{ak } [x,y,z] \notin v_{ijk} \end{cases}$$

σ_{ijk} represents values p_{ijk} or b_{ijk} in some neighborhood of voxel V_{ijk}. In the case of p_{ijk} we speak about gray level interpolating function and we suppose, that the object l was segmented by simple thresholding with threshold t_l. In the case of some other segmentation method[Š93] we come out of the values b_{ijk} in such a way that to samples $b_{ijk} = l$ we assign value 1 and to samples with $b_{ijk} \neq l$ we assign value 0. In this case we speak about binary interpolating function and as threshold we use value $t_l = 0.5$ for all objects.

The exact position of the object surface and ray we get by solving a system given by equations of ray and the surface:

$$\begin{aligned} X &= A + t\vec{u} \\ F(x,y,z,\sigma_{ijk}) &= t_l \end{aligned} \qquad (1)$$

where A is the eye position and $\vec{u}$ is the ray direction vector.

It may happen, that the given voxel is intersected by surfaces of more than one objects. In this case it is necessary to compute the intersections with all of them and to take the nearest to the eye point into account.

In the following we shall describe some possibilities for definition of the function F_l in dependency on σ_{ijk}.

3.1 0-order interpolation in center representation

We shall use this kind of interpolation in the case of fast, less precise rendering of 3D image. Let us define $F_l(x,y,z,\sigma_{ijk})$ within the voxel volume as

$$F_l(x,y,z,\sigma_{ijk}) = \begin{cases} 1 & \text{ak } b_{ijk} = l \\ 0 & \text{ak } b_{ijk} \neq l \end{cases}$$

which means that the binary interpolating function is within the voxel volume constant. The voxel value h_{ijk} will be in this case initialized by $min(1, b_{ijk})$ (Fig. 2a). To detect the surface by means of ray casting consists in finding the first voxel of the discrete ray with nonzero value with no further surface point computation.

3.2 Trilinear interpolation in center representation

Trilinear interpolation function is defined as

$$F(x,y,z,\sigma_{ijk}) = T(x,y,z,v_0,v_1,\ldots,v_7)$$

where

$$T(x,y,z,v_0,v_1,\ldots,v_7) = t_{xyz}xyz + t_{xy}xy + t_{xz}xz + t_{yz}yz + t_x x + t_y y + t_z z + t$$

Coefficients $t \ldots t_{xyz}$ depend on values $v_0, v_1, \ldots, v_7$, which are positioned in the voxel V_{ijk} vertices. Since in the center representation we suppose that the samples of the scanned field are situated in voxel centers, $v_0, v_1, \ldots, v_7$ must be computed[Š92] as a mean of values of those voxels, which share the given vertex. E.g. for vertex $[i,j,k]$ of voxel V_{ijk} it will be

$$v_0 = (s_{i-1j-1k-1} + s_{ij-1k-1} + s_{i-1jk-1} + s_{i-1j-1k} + s_{ijk-1} + s_{ij-1k} + s_{i-1jk} + s_{ijk})/8$$

Due to the mean computation this approach gives smooth surfaces. As it has been shown in [Š92], such surfaces are also shifted into the object, which may completely "smooth out" object details comparable with voxel size.

Another consequence of the mean value computation is that the surface can be detected not only within object voxels but also within background voxels in their vicinity. Therefore, in this case we shall assign value 0 only to those background voxels of the scene $\mathcal{S}^C$ not 26-adjacent to an object voxel. To all other voxels we shall assign 1(Fig. 2b).

Due to its smoothing ability we shall prefer this kind of interpolation for objects segmented by some other method as the thresholding.

3.3 Trilinear interpolation in vertex representation

The interpolating function is in this case defined as in the previous section, only the values $v_0, v_1, \ldots, v_7$ are directly given by samples $s_{ijk} \ldots s_{i+1j+1k+1}$ positioned in the voxel vertices. We shall assign value zero to background voxels and 1 to foreground voxels. This kind of interpolation is suitable for scenes with details on voxel level.

4 Discrete ray traversal

In the previous sections we came to the conclusion, that, as the consequence of the segmentation and choice of interpolation method, we partition the scene into 0-voxels with no

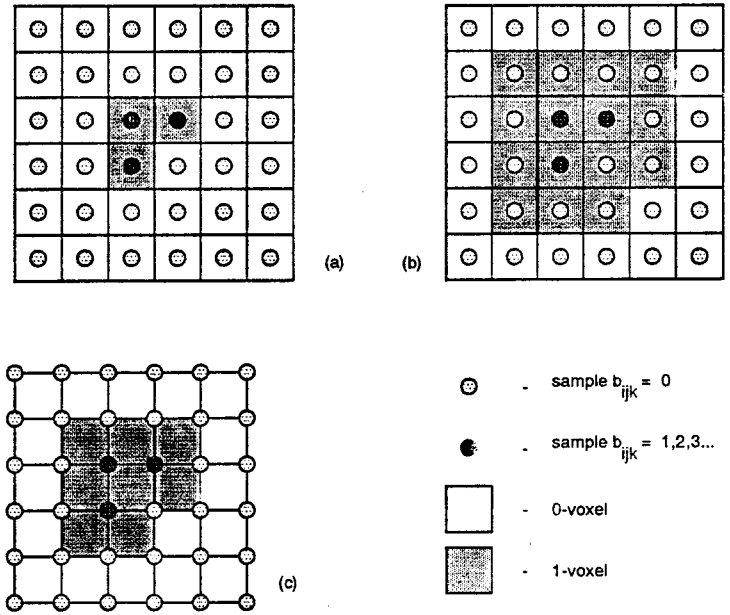

Fig. 2: Scene initialization: (a) $\mathcal{S}^C$ for 0 order interpolation, (b) $\mathcal{S}^C$ for trilinear interpolation (c) $\mathcal{S}^V$ for trilinear interpolation

object surface and 1-voxels, where the ray-surface intersection can be found. Let us define the discrete ray as an ordered sequence of voxels V_i of the scene $S^C(S^V)$, which are pierced by a the ray with equation $X = A + t\vec{r}$.

The goal of the discrete ray traversal algorithm is to find its first 1-voxel. We shall proceed with the following voxel of the only in such case, when no ray-surface intersection is found within the volume of the previous one. If we add to the demand of correctness of the surface point detection also the demand of high speed, the discrete ray traversal algorithm should have following properties:

1. it should pass the region of background voxels usually surrounding the object as fast as possible, i.e. it should utilize the object scene coherency[OM87] and
2. it must not miss any object voxel, i.e. at least in the vicinity of the object the voxels of the ray should fulfill the condition of 6-adjacency[YCK92].

In the algorithm design we shall start from the assumption that to each 0-voxel V_{ijk} its chessboard (CD) distance[Bor86] to the nearest object voxels is assigned. Thus a cubic macro region is created

$$\mathcal{O}^n(i,j,k) = \{h_{pqr} = 0 : i-n \leq p \leq j+n, i-n \leq q \leq j+n, k-n \leq r \leq k+n\}$$

with center in V_{ijk} and with side size $2n+1$. Since CD is independent on projection parameters, we can assign it during a preprocessing phase.

Since we know that there are no object voxels within $\mathcal{O}^n(i,j,k)$, we can jump from V_{ijk} directly to the first voxel outside of $\mathcal{O}^n(i,j,k)$ pierced by the ray. The traversal speed up is thus achieved by bringing the number of visited voxels down. Detailed description of the algorithm is in [Š94], therefore we shall limit ourselves only on brief description of its main loop, i.e. of steps between macro regions and on description of traversal of single macro region. Since the algorithm is symmetric with respect of all axes, we shall present only relations for the x axis. Further, we shall assume that the direction vector has only nonnegative coordinates. Generalization to all possible directions can be done by proper initialization of some variables.

4.1 Scene traversal

Let us imagine that we have reached the voxel V_{ijk} with assigned $CD = n$ and ray entry point $\hat{p} = (p_x, p_y, p_z)$, the coordinates of which are expressed with respect to $[i,j,k]$. Let entry (exit) face be that through which the ray enters (exits) the voxel.

To find the first ray voxel outside of $\mathcal{O}^n(i,j,k)$ it is necessary to find the nearest intersection of the line $X = \hat{p} + s\vec{r}$ with planes $x = n$, $y = n$ and $z = n$. Let us suppose that the entry face is of type X, i.e. that it is perpendicular to x axis. On Fig. 3 we can see that for each n exists such threshold value $t_y^x(n)$ of the input coordinate p_y that (2D case)

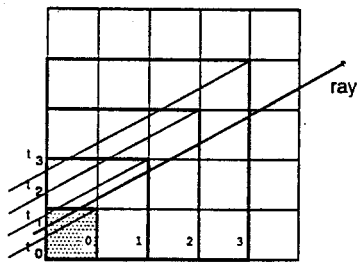

Fig. 3: Entry point coordinate thresholds

1. if $p_y < t_y^x(n)$ then exit face is also of X type and
2. if $p_y > t_y^x(n)$ then it is of Y type.

which in 3D case holds also for p_z. Upper index denotes the type of the entry face. It can be shown that

$$t_y^x(n) = n\frac{r_x - r_y}{r_x}$$

The basic scheme of the algorithm is on Fig. 4 left and scheme for the computation of exit face type on the right side. The expression $p_y \geq t_y^x(n) + p_z\frac{r_y}{r_z}$ enables to distinguish between types Y and Z when it is not X.

```
while(in scene and object not found )
      n = GetMacroRegionSize();
      if(faceType == X)
            macroRegionStep(x,y,z,n);
      else if(faceType == Y)
            macroRegionStep(y,z,x,n);
      else if(faceType == Z)
            macroRegionStep(z,x,y,n);
end while
```

```
macroRegionStep(x,y,z,n)
{
if(p_y ≤ t_y^x(n) and p_z ≤ t_z^x(n))
      NoChangeMacroRegionStep(x,y,z,n);
else if(p_y ≥ t_y^x(n) + p_z r_y/r_z)
      ChangeMacroRegionStep(x,y,z,n);
else
      ChangeMacroRegionStep(x,z,y,n);
}
```

Fig. 4: CD discrete ray traversal algorithm

4.2 Macro region traversal

In this part we indicate calculation of the next voxel position and the coordinates of the ray intersection with it. As it results from Fig. 4b, one has to distinguish between 2 kinds of steps across the macro region:

1. without face type change (NoChangeMacroRegionStep, Fig. 5a) a
2. with face type change (ChangeMacroRegionStep, Fig. 5b).

The first case is a step across a slab defined by two parallel planes with distance n. If we denote $d_y^x(n) = n * \frac{r_y}{r_x}$ then for the next voxel holds:

$$\begin{aligned} i' &= i + n \\ j' &= j + int(d_y^x(n)),\ \ p_y' = fract(d_y^x(n)) \\ k' &= k + int(d_z^x(n)),\ \ p_z' = fract(d_z^x(n)) \end{aligned}$$

The second case is analogous, only the intersection with plane $x = 0$ must be calculated first(Fig. 5b):

$$p_y = \frac{r_y}{r_x} * p_x$$

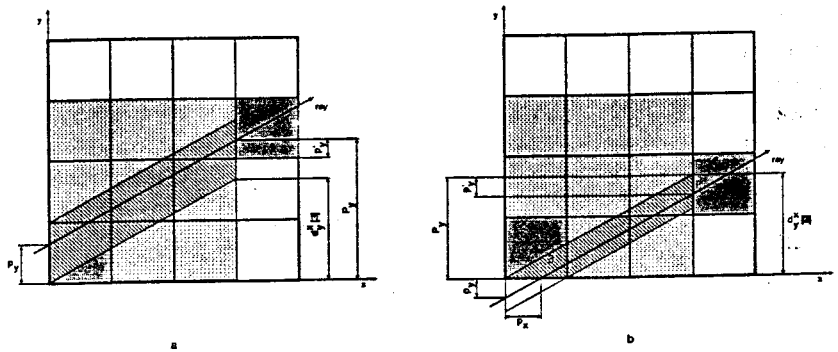

Fig. 5: Macro region traversal: (a) entry and exit face of the same type, (b) with different types

4.3 Summary

In the previous section we have proposed the algorithm for fast discrete ray traversal based on cubic macro regions. We have shown that the control variable ot the traversal can be the position of the entry point of the ray into the voxel, which is advantageous in coherence with exact computation of the ray-surface intersection point. Further speed up of the traversal can be achieved by

1. implementation with fixed point arithmetics,

2. tabulation of arrays t and d. This version is suitable only for parallel projection and traversal of primary rays only and

3. in the case when it is not necessary to know the exact position of the ray entry point (e.g. for 0 order interpolation), by a new control variable

$$(\tilde{p}_x, \tilde{p}_y, \tilde{p}_z) = (\frac{p_x}{r_x}, \frac{p_y}{r_y}, \frac{p_z}{r_z}),$$

which takes off some divisions.

5 Implementation and results

In this section we would like to present results of our comparison of the CD traversal algorithm performance with some algorithms known in the literature, namely Cleary's and Wyvill's algorithm for fast voxel traversal (FVT) [CW88] and Spackman's and Willis's SMART (Spatial Measure for Accelerated Ray Tracing) navigation oct-tree traversal [SW91].

The first one generates a sequence of voxels in a uniformly subdivided scene that are pierced by the ray. The decision, as to which voxel will be the next in the sequence, is controlled by three decision variables dx, dy and dz, which record the total distance along the ray from some common point to its last crossing with X, Y and Z type voxel faces. If, say, dx is the smallest one, than the next crossing will be with the X type face and therefore the next voxel will have the x coordinate incremented. This approach results in an efficient incremental algorithm that has only a few operations for each loop and might be implemented in integer arithmetic.

The second chosen approach works on the oct-tree represented as a breadth first list. Two decision vectors control the progress of the ray through the oct-tree:

- H_{SMART} vector, which navigates iterative horizontal steps from sibling to sibling by examination of its component signs, and
- V_{SMART} vector, which controls recursive vertical steps from parent to child by midpoint comparison of its components and a comparison variable.

Both decision vectors are maintained in an incremental way and also may be implemented in integer arithmetic.

Performance of three CD traversal algorithm versions has been studied:

CD1, the basic version,

CD2, the basic version with tabulated arrays t and d with respect to macro-region size, and

CD3, as CD2 with $(\tilde{p}_x, \tilde{p}_y, \tilde{p}_z)$ control variable.

The algorithms have been implemented in C language on Decstation 5000/200 UNIX workstation equipped with 48 MB of main memory.

5.1 Phantom study

To compare these algorithms and to obtain information about their behavior over a large variety of scenes with various complexities, a computer experiment was set up, based on rendering of scene phantoms: randomly positioned spheres of various size and number.

The notion of the spatial coherency leads to the following idea: The "more" coherent scene contains a few larger objects, while the "less" coherent scene contains a lot of small objects. Therefore we can say that the scene with a single spherical object has the highest spatial coherency. Since the sphere has the smallest surface-to-volume ratio (SVR), we propose this ratio to be a measure of the scene coherency for the purpose of the algorithm comparison.

The sequence of scene phantoms for algorithm comparison purpose has been generated in the following manner. The scene built up of 128x128x128 voxels was subdivided into NxNxN subregions ($N = 1 - 12$). Within each subregion a voxelized sphere was randomly placed (1-1728 spheres), such that total volume of all spheres is identical for all N. Thus we obtain scenes with equal number of object voxels but with different SVR.

In the experiment, only parallel primary rays were traced until the first object voxel was found, with no subsequent shading. The results of the experiment are depicted in Fig.6(a). The y axis values represent the pure traversal time measured as a difference between user time (in the sense of UNIX `time` command) of such program version, which stops computation for each ray in that moment, when the first object voxel is found, and a version, which stops just before the traversal starts. This approach enables to eliminate the influence of various initialization and data loading conditions of various algorithms.

The total traversal time can be expressed as

$$T = t.n \tag{2}$$

where t is one step time and n is number of traversal steps necessary to render the scene. If we express t as $c.t_0$, where t_0 is one step time for some reference algorithm (e.g. the fastest one, in our case FVT) and n as a ratio of total distance traversed and mean step size $\frac{d}{d_0}$, we obtain

$$T = \frac{c}{d_0}\, d.t_0 \tag{3}$$

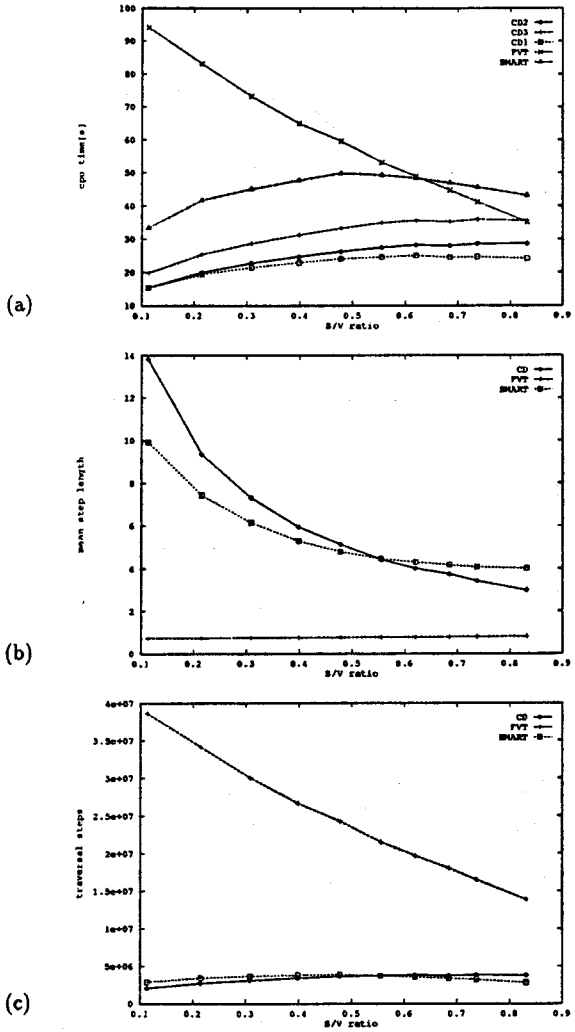

Fig. 6: (a) CPU time, (b) mean step size and (c) number of steps vs. surface-to-volume ratio for rendering of 5 arbitrary views of phantom scenes with different object coherency.

While the term $d.t_0$ is a constant for the given scene, the algorithm efficiency is given by the proportion of the relative one step cost c and mean step size d_0.

The experiment has shown that the cost c for CD1, CD2 and CD3 versions was nearly constant for the whole range of SVR with values 3.8, 4.2 and 5.2 respectively. Slight increase within an interval of 5% can be explained by the growing proportion of face chance steps with higher computational demands. The value of c for the SMART algorithm was in the range from 6.9 to 7.9. This increase is caused by higher number of vertical oct-tree traversal steps in respect to horizontal steps for the more complex scene.

The dependence of the mean step length d_0 on SVR is depicted in Fig.6(b). While the FVT algorithm step is determined only by voxel geometry and therefore is constant, the CD and SMART mean distances are also affected by distribution of object voxels in the scene. The more compact scene, with small number of larger objects, results in a greater traversal step mean length, and therefore in shorter total traversal time.

The relative efficiency of two algorithms is given in (3) by the ratio of relative cost and mean step size. In the case of CD and FVT algorithms, this ratio is more favorable for all of the CD versions for nearly the whole range of SVR. Of course, with increasing complexity of the scene the relatively higher step cost of CD begins to prevail and FVT achieves higher performance than CD.

Fig.6(c) plots the number of traversal steps vs. SVR. The increasing complexity of the scene results in smaller total traversed distance as well as in lower number of steps for FVT algorithm, which is the reason for shorter rendering time. On the other hand, decrease of mean step size for the CD and SMART algorithms counteracts the smaller traversed distance. As a consequence, the number of traversal steps as well as rendering time increase.

Both CD and SMART algorithms adapt their step to the actual object distribution, and therefore exhibit similar behavior with respect to SVR. Although for the more complex scenes, the oct-tree structure results in the larger mean traversal step (Fig.6b), the lower cost of the single step makes CD predominant over the whole SVR range.

6 Conclusion

In our contribution we have shown (a) how to define an approximating function to an object surface within a voxel and (b) we have proposed CD algorithm for fast discrete ray traversal which is based on knowledge of chessboard distance of the background voxels from the nearest object voxel. As the computer experiment has shown, one step of such algorithm can be up to 15 times longer than for traditional one voxel step algorithms. In spite of the fact, that complexity of one step of CD algorithm is higher, it results in up to 6-fold speed up.

Another possibility to shorten the scene rendering time granted by the CD algorithm, but

we did not deal with, is minimization of number of rays totally missing the object. The minimal CD of voxels such ray passes through can be used to estimate a distance in which it is sufficient to generate next ray without missing any object voxel.

References

[Bor86] Gunilla Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34(3):344–371, 1986.

[CW88] John C. Cleary and Geoff Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, July 1988.

[LC87] W.E. Lorensen and H.E. Cline. Marching cubes: A high-resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.

[OM87] Matasaka Ohta and Mamoru Maekawa. Ray coherence theorem and constant time ray tracing algorithm. In T. Kunii, editor, *Computer Graphics 1987 – Proceedings of CG International '87*, pages 303–314. Springer–Verlag, 1987.

[SW91] John Spackman and Philip Willis. The smart navigation of a ray through an oct-tree. *Comput.&Graphics*, 15(2):185–194, 1991.

[Šr92] Miloš Šrámek. Ray tracing volume data with subvoxel precision. In Václav Skála, editor, *Proceedings of the Winter School on Computer Graphics and CAD systems*, pages 47–65, Plzeň, Technical University, ČSFR, 1992.

[Šr93] Miloš Šrámek. Interactive segmentation of tissues for medical imaging. In Václav Hlaváč and Tomáš Pajdla, editors, *Czech Pattern Recognition Workshop '93*, pages 164–171, Temešvár u Písku, Czech Republic, November 4^{th}-6^{th}, 1993.

[Šr94] Miloš Šrámek. Cubic macro-regions for fast voxel traversal. In *GKPO'94*, Poland, June 1994. accepted.

[YCK92] Roni Yagel, Daniel Cohen, and Arie Kaufman. Discrete ray tracing. *IEEE Computer Graphics and Applications*, 12(5):19–28, September 1992.

Generating Plants for Computer Graphics

Bedrich Benes CTU, Fac. of Electrical Eng.,
Dept. of Computer Science
Karlovo nám. 13, 121 35 Praha 2
BENES@CSLAB.FELK.CVUT¿CZ

keywords: Tree, L-system, Rendering, Computer Graphics

Computer Graphics focuses its activites mostly on the modelling of the real world. It means in practice to find a model of the modelled object which can be interpreted by means of computer graphics. There has been a lot of progress in this research. We have fundamental models of light of reflecting, and are able to simulate real surfaces (e.g. an orange, a metal, a wood and so on). To visualise these models we can use on of two principal methods: **ray tracing** or **radiosity** [ZARA92].

There are several open problems in Comupter Graphics. One of these is the general problem of time. These methods are really strongly time consuming. This time depends on the modeled scene. The time needed for visualisation increase with the number of modeled objects and can be strongly decreased with a smart algorithm or a good data representation or speed hardware.

Lindenmayer [LIND68] showed a structure of a parallel string rewriting system for the growth of living organisms in 1968. This method describes the geometry of a tree, but it was used for these purposes for the first time in 1984 in the work of Aono and Smith [SMIT84]. This work continues in the work of Lindenmayer and Pruzsinkiewicz at present. A focus of these works lies in the topology of trees, a geometry of these objects is showed in works of [BLOO85], [OPPE86], [KAWA82].

The generation of topology is based on string rewriting systems (called by Lindemayer [LIND68] L-systems).

Let V [PRUS86] equal a set called alphabet whose members are symbols. Let V^* equal a set of words over V and V^+ is a set of non-empty words over V.

The L-system is a triple [LIND68] $G = (V, \omega, P)$, where

- V is an alphabet