Fast volume render techniques for interactive analysis

Noordmans, H.J.; Smeulders, A.W.M.; van der Voort, H.T.M.

DOI
10.1007/s003710050109

Publication date
1997

Published in
The Visual Computer

The Visual Computer

Fast volume render techniques for interactive analysis

H.J. Noordmans,
A.W.M. Smeulders,
H.T.M. van der Voort

Department of Biological Information Technology,
University of Amsterdam, Kruislaan 403,
1098 SJ Amsterdam, The Netherlands
e-mail: herke@wins.uva.nl

Without graphics hardware, interactive volume rendering is almost impossible with the current generation of computers and software. We describe the implementation of a volume renderer for interactive analysis of confocal images. We propose several techniques to accelerate the rendering of grey-value volumes. We propose to illuminate the volume selectively with ray templates to get a proper shadow cue in the shortest feasible time. In the viewing phase, rendering is distinctively accelerated for four user interactions: (1) a total change by successive adaptive refinement, (2) an unknown change in the view with this refinement strategy combined with suspended interpolation, (3) a known change in the view by recalculating only that part and (4) a view translation by recalculating the uncovered part.

Key words: Volume rendering – Interaction – Successive adaptive refinement – Confocal microscopy

Correspondence to: H.J. Noordmans

1 Introduction

Volume rendering is increasingly important for the analysis of volume data. It gives a clearer view of the total structure, as opposed to the detailed information one gets from viewing the volume through slices. If the volume rendering is fast enough, it can be used to interactively analyse the volume. It may then be used to explore the volume data qualitatively or to guide quantitative measurements. In exploring the volume, the renderer must react quickly to changes in the colour and transparency of the volume data, in the view angle or in the position of the view, et cetera. For quantitative measurements, the renderer must show measurement results quickly and visualize 3D objects that guide the measurements (e.g. 3D probe). In both cases, high demands are made on the volume renderer. It should produce views with sufficient quality in the shortest feasible time. A possible solution is to do rendering calculations in hardware or across multiple processors, which considerably speeds up volume rendering (Lorensen and Cline 1987; Yoo et al. 1992; State et al. 1995). For the case in which this kind of hardware is not available, algorithms have been developed to accelerate volume rendering in software (Bergman et al. 1986; Levoy 1990a, b, c; Montani and Scopigno 1990; Nielson and Hamann 1990; Shu and Lin 1991). Although these techniques reduce the rendering time considerably, they are hardly sufficient to provide interactive responses.

In this paper, we discuss the software implementation of a volume renderer to be used for the interactive analysis of confocal images. These images are recorded by a confocal laser scanning microscope (CSLM) (Wilson and Sheppard 1984). A specimen, stained with fluorescent dyes, is placed under the microscope. The fluorescent molecules excited by dye-specific monochromatic laser light, emit light with a dye-specific larger wavelength. The light is captured with sensitive detectors and converted into an electronic signal. The volume is scanned step by step by the CSLM to determine the amount of fluorescent material of each volume element. This results in a volume image where each voxel value represents the amount of fluorescence detected. By staining the specimen with more than one dye, several structures can be recorded at the same time. This results in a set of volumes that should be combined by the renderer during display.

2 Interactive rendering of confocal images

Confocal images make specific demands on a volume renderer. As the images often contain a large amount of noise, it is difficult to ascertain whether a voxel belongs to the object of interest or to the background (Kaufman et al. 1990). This means that we cannot rely on volume-rendering techniques that segment the volume data before display (Levoy 1988). The renderer should be able to visualize the raw, unprocessed volume image, the result of a segmentation step, or both at the same time. This gives two kinds of images: *grey-value images* where the value of a voxel represents the amount of fluorescent material present on the location, and a *label image* where the value of a voxel represents the type of material on the same location (binary images are defined as label images with two labels). The raw, unprocessed image is stored as a grey-value image; the result of a segmentation step is stored as a label image. The image types give different interpretations of the value of a voxel. With label images, a voxel belongs to one object: and the value at a subvoxel position equals that of the closest voxel. With grey-value images, the voxel value acts as a density measure; the interpolation is required to determine the value at a subvoxel position. Both image types should be visualized appropriately by the volume renderer.

Another demand is that the renderer should be able to render 3D objects to be used for true interactive measurements. For accurate manipulation and measurement, the position and size of the object, together with its relation to other volume objects, should be as clear as possible. There are two ways to render the objects: as surface objects mixed with the volume data in the rendering stage (Levoy 1990b; Ebert and Parent 1990), or as a group of voxels that constitute the object. The latter approach may create sampling artefacts, but results in a far simpler volume renderer than the first approach. We have chosen this approach to visualize 3D objects.

The volume renderer should generate views with an adequate number of visualization cues to faithfully visualize the volume data and the 3D objects. In the analysis of confocal images, we consider occlusion and shadowing two essential cues (Voort et al. 1993), as both enhance the spatial relation between volume objects. An optimal effective shadow cue can be obtained when the shadow of one object is visible on another object. This hampers the use of fast illumination methods where the illumination value is based only on the light direction and not on the amount of material in front of the light source (Drebin et al. 1988; Yoo et al. 1992).

For an interactive response, the renderer should deliver a proper response time. This requires that, in addition to a fast rendering algorithm, the underlying light-matter interaction model should be computationally efficient. Two extremes of the scattering equation lead to fast algorithms: *high-albedo* and *low-albedo* scattering (Kajiya 1984). In the first case, light is scattered so often that all voxels are illuminated equally. In the second case, light is scattered only once; the rendering process can be split into an illumination phase and a viewing phase (Kajiya 1984). Because we consider shadows essential for depth perception, we opt for the low-albedo extreme.

To offer flexibility in visualizing the volume data, the scatter model is based on fluorescence. In the simulated fluorescence process (SFP) (Messerli et al. 1993; Voort et al. 1993), we model an amount of fluorescent material in each voxel, the amount being proportional to the voxel value. In the illumination phase, we simulate a light source emiting photons that excite the fluorescent molecules to a higher energy level. These molecules re-emit part of the energy as photons that arrive at the final view. To get the low-albedo extreme, the emitted light is not able to re-excite other fluorescent molecules (nonoverlapping excitation and emission spectra). For an effective shadow and occlusion cue, a photon may be absorbed by any material it passes. As with fluorescence, the emitted wavelength is larger than the excitation wavelength, which enables materials to absorb light in the illumination phase differently than light in the viewing phase. With label images, each label denotes one type of fluorescent material, and each material is excited by a specific wavelength and emits a specific wavelength. With N materials, each material is described by $2N+1$ parameters: one scatter efficiency, N absorption constants for each wavelength in the illumination phase, and N absorption constants for each wavelength in the viewing phase. The large number of parameters offer the user much flexibility in visualizing the volume data.

3 Evaluation of the literature

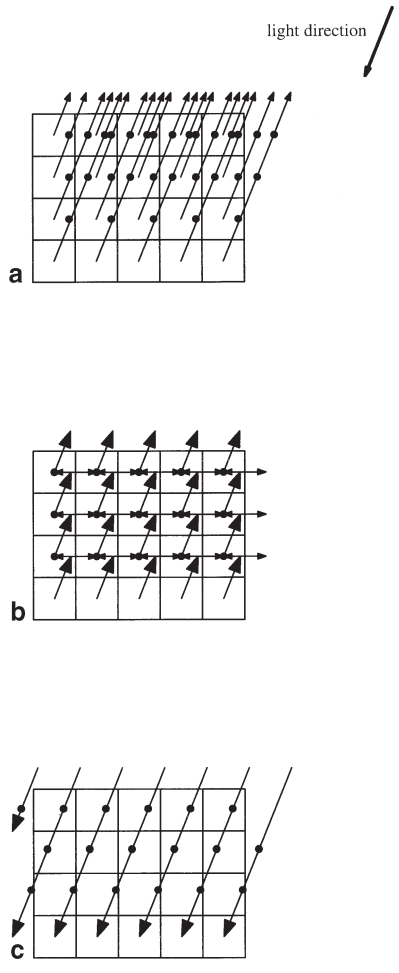
For the implementation of the volume renderer, we have first investigated whether existing techniques are sufficient to produce views at an interactive rate. First, we discuss techniques for the illumination phase, then those in the viewing phase.

3.1 Illumination phase

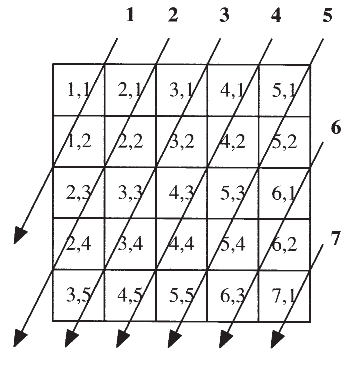
In the illumination phase, we simulate a light source illuminating the volume with size $n \times n \times n$ from a certain position (Fig. 1). The light source emits light that is scattered or absorbed by material inside the voxels. The usual tactic for calculating the amount of light reaching the material is to calculate the amount of absorbing material between the actual voxel and the light source. The most accurate way is to sample back from each voxel to the light source (Fig. 1a) (Kajiya 1984). The total number of samples is relative to $n^3 \times n$, giving an $\mathbb{O}(n^4)$ illumination algorithm. The high computational complexity can be reduced to $\mathbb{O}(n^3)$ by calculating the illumination value of a voxel from the illumination value of voxels that are one layer closer to the light source (Ebert and Parent 1990) (Fig. 1b). As Fig. 2 illustrates, the bilinear interpolation causes a strong and perceptually inexplicable blur of shadows. The complexity of $\mathbb{O}(n^4)$ can also be simplified by reducing the voxel size in the shadow buffer (Ebert and Parent 1990; Levoy 1990d). When reduction of factor r is applied in each direction, the computational complexity diminishes to $\mathbb{O}(n^4/r^4)$. The shadow becomes somewhat broader, but the calculation time is shorter.
Another approach is to trace along a set of rays from the light source into the volume, where each ray hands over an amount of light to each voxel it passes (Fig. 1c). Illumination is optimal $[\mathbb{O}(n^3)]$ when each voxel is hit only once (Voort et al. 1993); however, with a low hit density, rippled illumination patterns arise at strongly absorbing boundaries because not all surface voxels are directly illuminated by the light source (Fig. 3). Comparing both strategies, we see that there is a clear trade-off between quality and speed.

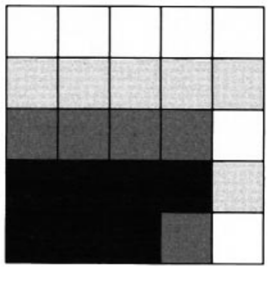
3.2 Viewing phase

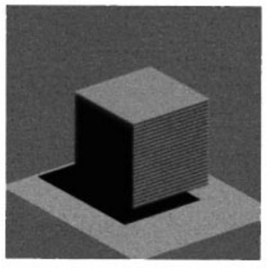
There are many techniques to speed up the rendering process. An overview is given in Table 1. *Adaptive subdivision* is a technique to reduce the number of rays in image regions with less detail. This technique is often implemented in conjunction with *adaptive refinement*. The first update shows only regions with high detail, while other regions are interpolated. More detail is shown in subsequent refresh iterations. *Adaptive ray termination* is a technique in which sampling along a ray is broken off when the opacity along the ray exceeds a threshold. *Voxel group projection* projects similar voxels simultaneously. *Presence acceleration* skips empty voxels in sampling along a ray. *Adaptive sampling* decreases the number of samples in homogeneous regions. *Template based viewing* speeds up rendering of parallel views by using fixed ray templates. *Shear-warp projection* efficiently projects the volume data on the view. *Local volume update* speeds up rendering of varying volume data by casting rays only for parts where the volume has changed. *View movement* shifts, rather than recalculates, the view when the view is translated. *Blur prevention* prevents unnecessary image blurring if the volume changes, but it is difficult to predict where this happens.
The techniques can be sorted in two categories: those that cast rays into the volume to calculate how much light reaches the view and voxel methods that project the light of voxels onto the view. Some methods also accelerate the rendering of unsegmented volumes, while others only accelerate the rendering of segmented volumes.
To compare the performance of various techniques, we estimate the acceleration factor expressed as lower and upper bounds, as for many techniques the performance depends on the configuration of the volume data. For each technique, we estimate the difference between the final view and the view rendered without the acceleration technique.
Looking at Table 1, we see that some techniques result in large image errors. These techniques clearly trade quality for speed. We chose not to



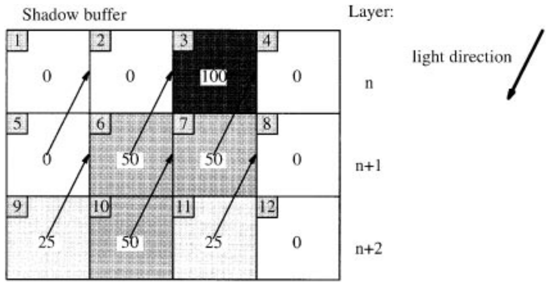
1







3



2

Fig. 1a–d. Three techniques to illuminate the volume: **a** sample from voxel to light source to calculate the amount of absorbing material; **b** use light information at previous layer to calculate illumination value; **c** follow a bundle of rays from light source through the volume; the light ray gives off part of the intensity to the voxels it passes

Fig. 2. Illumination artefacts with back interpolation: layer *n* is processed first, then layer $n + 1$. The value of voxel 7 is based on interpolating the values of voxels 3 and 4. In repeating the procedure for the next layers, the shadow gets broader

Fig. 3a–c. Origin of ripples in illuminating the volume with a bundle of light rays: **a** seven illumination rays – each voxel describes which ray hits the voxel and in which order; **b** resulting illumination buffer with ripple pattern at right side; **c** effect on rendering a 50×50×50 cube

Table 1: Properties of volume render acceleration techniques

Technique	Voxel/ray[a]	Segmentation[b]	Acceleration factor	Image error (percentage)[c]	Ease of implementation[d]
Adaptive subdivision (Levoy 1990a, Shu and Lin 1991, Akimoto et al. 1991)	r	No	5–10	3–10	0
Adaptive refinement (Levoy 1990a, this paper)	r	No	5–50	30–>0	0
Adaptive ray termination (Levoy 1990b)	v/r	No	1–100	0–1	+
Voxel group projection (Laur and Hanrahan 1991)	v	Yes	10–100	10	0
Presence acceleration (Zuiderveld et al. 1992; Danskin and Hanrahan 1992)	v	Yes	3–10	0	+
Adaptive sampling (Danskin and Hanrahan 1992)	r	No	2–3	3–10	–
(Lacroute and Levoy 1994)	v	No	2–3	3–10	–
Template based viewing (Yagel and Kaufman 1992)	r	Yes	5	10[e]	0
(this paper)	r	No	2	0–1	–
Shear-warp projection (Lacroute and Levoy 1994)	v	No	2–3	5	0
	v	Yes	1–2	5	–
Local volume update (Shen 1994; this paper)	r	No	3–500	0–1	+
Blur prevention (this paper)	r	No	1–5	0	0
View movement (this paper)	v/r	No	1–100	0	+

[a] Image rendered by (v) projecting voxels; (r), casting rays
[b] Segmentation of data set required before acceleration technique can take effect
[c] RGB difference, black and white differ 100% (see Eq. 1)
[d] – difficult, 0 normal, + easy
[e] 0% error for binary volumes

use them, as we want to offer the user a clear view with no missing details. Nonetheless, we accept a low-quality first view update as long as the final view shows all details. A second conclusion we draw from Table 1 is that, without hardware, one technique alone is not sufficient to obtain interactive volume rendering. In addition, the performance of some techniques depends so much on the configuration of the volume data that a constant acceleration is not guaranteed. We have tried to alleviate this problem by combining acceleration techniques. As there are more acceleration techniques available for ray casting, our renderer uses the following techniques: adaptive refinement, adaptive subdivision, adaptive ray termination, template based viewing, local volume update, view movement and blur prevention.

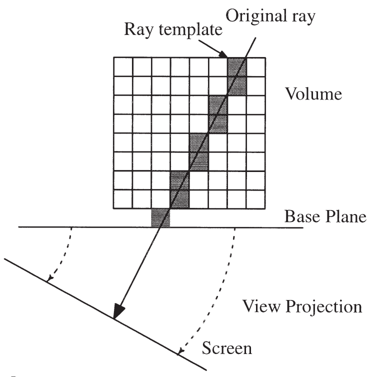
4 Volume-rendering accelerations for interactive analysis

In contrast to rendering single views, interactive visualization renders views of a large coherence. Renderers capable of recognizing the coherence can save time if there is no need to recalculate the parts. The first point is to let the renderer recognize whether the illumination phase is indeed necessary. In practice, the illumination phase only needs to be recalculated upon a change of the light source, a change in the volume data, or a changed absorption of fluorophores in the illumination phase. Other changes, such as colour or absorption of fluorophores in the viewing phase, do not require new illumination calculations.

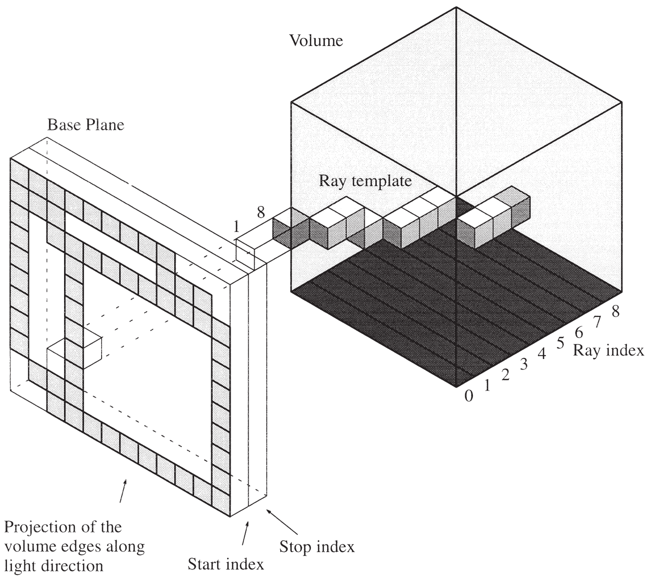
The second point is that, with many user interactions, only part of the view changes. Examples are changes in colour of a part of the volume data, manipulation (movement or rotation) of a 3D object or translation of the view. A renderer that recognizes the coherence in these situations will accelerate the viewing phase considerably.

4.1 Illumination phase

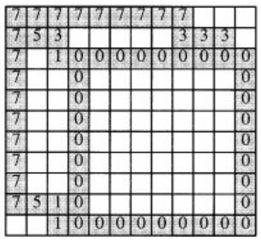
For the illumination of the confocal image, we have to choose between two illumination techniques: (1) start from a voxel and calculate how much light it receives or (2) follow light rays through the volume. The computational complexi-



4

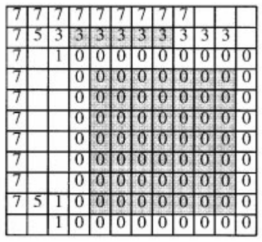


5



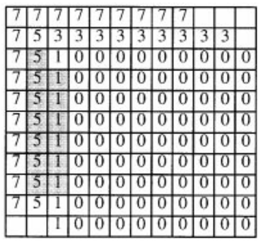
7	7	7	7	7	7	7	7	7			
7	5	3						3	3	3	
7		1	0	0	0	0	0	0	0	0	0
7			0								0
7			0								0
7			0								0
7			0								0
7			0								0
7			0								0
7	5	1	0								0
		1	0	0	0	0	0	0	0	0	0

6a



7	7	7	7	7	7	7	7	7			
7	5	3	3	3	3	3	3	3	3	3	
7		1	0	0	0	0	0	0	0	0	0
7			0	0	0	0	0	0	0	0	0
7			0	0	0	0	0	0	0	0	0
7			0	0	0	0	0	0	0	0	0
7			0	0	0	0	0	0	0	0	0
7			0	0	0	0	0	0	0	0	0
7			0	0	0	0	0	0	0	0	0
7	5	1	0	0	0	0	0	0	0	0	0
		1	0	0	0	0	0	0	0	0	0

6b



7	7	7	7	7	7	7	7	7			
7	5	3	3	3	3	3	3	3	3	3	
7	5	1	0	0	0	0	0	0	0	0	0
7	5	1	0	0	0	0	0	0	0	0	0
7	5	1	0	0	0	0	0	0	0	0	0
7	5	1	0	0	0	0	0	0	0	0	0
7	5	1	0	0	0	0	0	0	0	0	0
7	5	1	0	0	0	0	0	0	0	0	0
7	5	1	0	0	0	0	0	0	0	0	0
7	5	1	0	0	0	0	0	0	0	0	0
		1	0	0	0	0	0	0	0	0	0

6c

Fig. 4. Template-based viewing of binary volumes

Fig. 5. Illumination with ray templates. The base plane contains start and stop indices of individual rays

Fig. 6a–c. Calculation of start indices of the base plane of Fig. 5: **a** calculate indices for rays passing the volume edges; **b** fill horizontal strips. When strip starts and ends with the same edge index, all rays of the strip enter the volume at the same index; **c** idem for vertical strips

ties are $\mathcal{O}(n^4)$ and $\mathcal{O}(n^3)$. In both cases, the illumination is accelerated if a small loss in quality is allowed by reducing the size of the light buffer (inverse of shadow buffer) by a factor r. To reduce ripple patterns caused by the second approach, voxels are illuminated with more than one ray by increasing the ray density by a factor ρ. With these modifications, the complexities become $\mathcal{O}\left(\frac{n^4}{r^4}\right)$ and $\mathcal{O}\left(\rho\frac{n^3}{r^3}\right)$. The ratio between the complexities of these two methods is $\frac{n}{r\rho}$, which means that, for confocal images in general, $n = 100$, boiling down to an image of one million voxels. In most images, strong absorbing boundaries are rare, and setting ρ to 1 provides sufficient quality. Only in extreme cases must the ray density be increased. The ratio then diminishes to $\frac{100}{r}$. We see that r should be very large for back sampling to be superior to ray illumination. Therefore, we prefer to illuminate the data set with a bundle of rays.

For a fast and uniform illumination of the data set, we use the same sampling technique as in template-based viewing (Fig. 4). This means that all light rays are assumed to run parallel, which corresponds to a light source at infinity. The rays with an intermediate distance of one voxel provide uniform illumination. The face of the volume of which the normal vector has the minimal angle between the light rays and is closest to the light source is the principal face. By illuminating the volume, we follow every ray from the base plane (parallel to the principal face) until the ray leaves the volume or has lost all intensity. With template rays, each voxel is illuminated once. To reduce the ripple pattern, the hit density r is increased by reilluminating the volume with ray templates that have been calculated with different initial offsets.

The computational complexity of template-based illumination is $\mathbb{O}(t_s n^3 + t_i n^2)$, where t_s denotes the time per sample for an illumination calculation and t_i denotes the time needed to calculate at which sample a ray enters or leaves the volume. If $t_s << t_i$, for instance, when the illumination stops after a few samples due to a strong absorbing volume object, the illumination phase can be accelerated, not when entrance and exit points of a ray are calculated on the fly, but when they are calculated beforehand. When considering the entrance points of the rays passing through the edges of the volume, we see that, when starting from the same sample position, the rays in the same strip start from the same position (Figs. 5, 6a). Thus, only the entrance and exit positions have to be calculated for rays through the edges of the volume, and the indices of the other rays have to be filled in by looking at the indices of the edge rays (Fig. 6b–c). This gives a computational complexity of $\mathbb{O}(t_s n^3 + t_f n^2 + t_i 8n)$, where the filling time t_f is far lower than t_i.

When only a part of the volume changes, as is the case with interactive manipulation, not all illumination rays need to be recalculated. These rays are determined by projecting the affected volume onto the base plane. An extra plane parallel to the base plane (refresh surface) stores the rays that need to be recalculated. We chose to represent the affected volume by the closest encompassing block, as the projection is easy to calculate. After all affected volumes have been projected on the refresh surface, the surface is traversed to see which illumination rays need to be recalculated. As the illumination angle has not changed since the previous situation, we can use the same entrance and exit points. This means no calculation for this situation ($t_s = t_i = 0$) is required. If the fraction of recalculated rays is given by f, the computational complexity becomes $\mathbb{O}(f t_s n^3)$.

4.2 Viewing phase

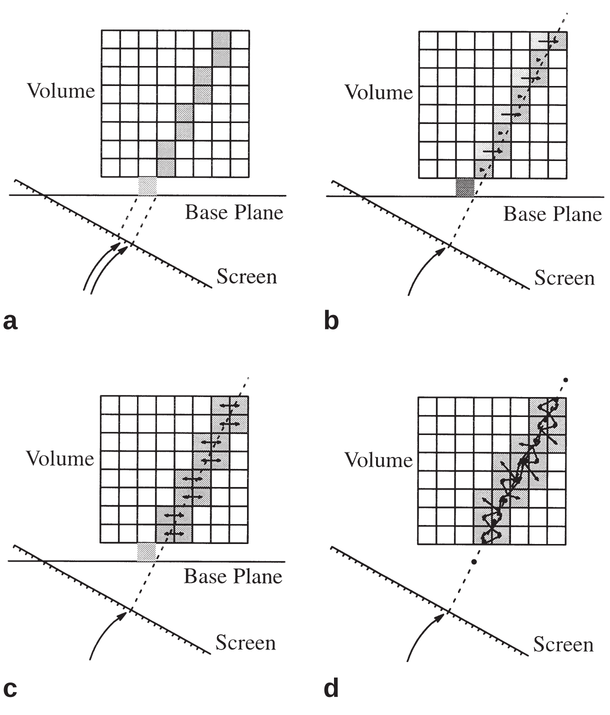
In the viewing phase we combine several acceleration techniques from Table 1. Some have already been described in the literature; others are new. First, we describe how template-based viewing is adapted to visualize grey-value volumes. Secondly, we describe an adaptive refresh scheme from which more profit is gained from the coherence between views in interactive visualization.

We have modified template-based viewing on several points for a more appropriate visualization of grey-value volumes. The first modification is *not* calculating the total view on the base plane and then projecting it on the screen, but rather starting from the screen and determining which rays have to be traced. In this way we avoid tracing invisible rays that fall outside the view, as well as a projection of the total view if only a part changes. The modification also makes template-based viewing suitable for acceleration by adaptive refinement techniques.

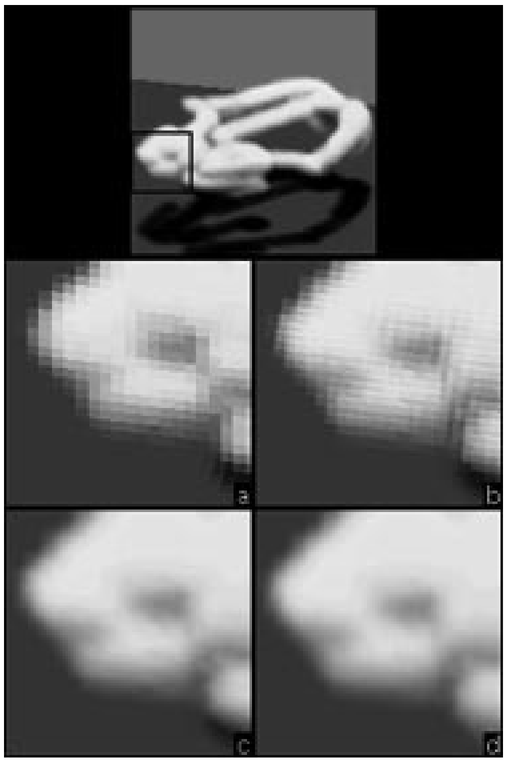
The second modification is interpolating along the ray because, with the original implementation, grey-value volumes are rendered with poor quality. We considered four interpolation strategies: (1) interpolation in template-based viewing, (2) nearest-neighbour interpolation, (3) bilinear interpolation, and (4) trilinear interpolation. The interpolation strategies are illustrated in Fig. 7; the corresponding qualities of a zoom operation for rendering a grey-valued volume are shown in Fig. 8.

(1) With the interpolation strategy used in the original implementation of template-based viewing, samples are restricted to one ray template. Although the rendering time is short (0.2 s on a 86 SPECfp92 machine), the corresponding quality is poor.

(2) Template-based viewing can be improved by letting the template alter its shape, if the template ray does not exactly intersect the view pixel at the centre. This is implemented as follows. In addition to the volume position, we also store the difference with the theoretical ray for each sample posi-



7



8

Fig. 7a–d. Interpolation strategy: **a** template-based viewing; **b** nearest-neighbour interpolation; **c** bilinear interpolation; **d** trilinear interpolation

Fig. 8. Rendering quality of zooming in on a grey-value volume with the interpolation strategies of Fig. 7 (volume 80×70×16; views 128×128): **a** template-based viewing; **b** nearest-neighbor interpolation; **c** bilinear interpolation; **d** trilinear interpolation

tion of the template. The template is chosen below the theoretical ray in each direction, in such a way that the difference is always positive and maximally one voxel. For each view pixel, we select the template ray that starts below this view pixel. The template is traversed, and at each sample position, the differences are added up. If the sum exceeds 0.5, the sample position is moved to a higher position. This resulting interpolation strategy corresponds to nearest-neighbour interpolation. Compared to the strategy 1, the rendering time is much longer (1.0 s), and the quality is only slightly better.

(3) A second improvement is to alter the shape of the template, and to use the remaining difference for bilinear interpolation. The rendering is approximately twice as long as with the previous strategy (1.9 s), but the quality is considerably improved.

(4) We compared the three interpolation strategies described with digital differential analyser (DDA) sampling in combination with trilinear interpolation. The sampling step has been chosen to be the size of one voxel. The third interpolation strategy can be considered as a special case of strategy 4, where the sampling step has been chosen in such a way that each sample lies on a plane through the voxel centres. In Fig. 7c, the sampling step is 1.4 times the size of a voxel. With trilinear interpolation, the rendering time is 2.5 times longer than with strategy 3 (5.0 s), but the increase in quality is hardly visible.

Comparing the four interpolation strategies, we clearly see a trade-off between rendering speed and rendering quality. The first interpolation strategy is by far the fastest and should be used when each view pixel maps to a different ray template,

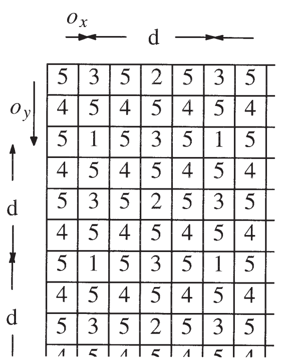


Fig. 9. Pixel calculation sequence for refresh surface. First pass, calculate value of number 1 pixels, raster distance $d = 4$, offset from image origin (o_x, o_y). Second pass, calculate the values of number 2 pixels based on the value of four closest number 1 pixels. If their colour variance exceeds a threshold, the pixel is calculated else interpolated. The process is repeated for the other pixels. Number 3 pixels are based on the value of the four closest number 1 and 2 pixels; number 4 pixels on 1, 2 and 3 pixels; and number 5 pixels on 1, 2, 3 and 4 pixels

or if the transparency of the volume object is high and sharp transitions hardly occur. In other cases, the bilinear interpolation strategy should be used, as it offers the best quality with a minimum of calculation time.

In the next step, we developed an adaptive scheme to make more use of the coherence between subsequent views. First, we converted the refresh scheme of Akimoto et al. (1991) to enable successive refinement. In the original implementation, the view is built in several passes (Fig. 9). In the first pass, the exact colour is determined by tracing each pixel on a raster with spacing d. In subsequent passes, the colour variation of the neighbouring pixels v, is compared to a threshold a. If $v > a$, the colour is determined from ray tracing; otherwise, by interpolating the pixel colours of the four neighbouring pixels. For successive refinement, it is necessary to store the variation values in a refresh surface to avoid redundant variation calculations. A value in the refresh surface, a refresh value ρ, is coded in an integer with the following meaning:

$\rho > 0$: $v = \rho$, colour interpolated

$\rho < 0$: $v = -\rho$, colour from ray tracing

$\rho = \infty$: variation should be determined from neighbour pixels.

The refreshing is started by setting all refresh values to ∞.

After the first iteration, the view is refined by reducing the variation threshold a or spacing d. Each time, all pixels are accessed in the same order. If the variation value of an interpolated pixel is higher than a, the colour of the pixel is determined from ray casting, and the refresh value is negated. It broadcasts its change to the pixels that depend on this pixel. If the value is lower, the pixel colour does not change. Updates end when all pixels have been ray traced. The final error therefore is 0%.

With adaptive refinement, views are rendered with more and more detail. The first update can be speeded up by using a high variation threshold, but has the disadvantage of low quality. A better quality can be obtained by recognizing that, for user interactions, large parts of the view usually remain the same. We distinguish four types of user interaction:

- Change of total view, e.g. change in view angle or rendering quality
- Change of an unknown part of the view, e.g. the changing part may be difficult to calculate (change in colour or absorbing capacities of part of the volume data)
- Change of a localized part of the view, e.g. movement of 3D cursor
- Translations of the view, e.g. during close inspection.

We have accelerated the adaptive refinement scheme to accommodate these interactions. The first method of acceleration is blur prevention, which avoids unnecessary blurring in areas where the volume data do not change. When a new pixel colour is calculated, it is compared to the old colour. If

$$\frac{R_0 - R_1 + G_0 - G_1 + B_0 - B_1}{3} < 1.5\%, \qquad (1)$$

the difference is considered unnoticeable. In that case, the broadcast to pixels depending on this pixel does not take place. For such a pixel, if the colour variation is lower than the variation threshold and all neighbouring pixels did not broadcast, interpolation is skipped. The result of the blur prevention is not a faster rendering time, but a better quality within the same amount of time.

The second acceleration, local volume update, is mostly used when data change in the volume, e.g. in moving a 3D object. Then, the view changes at several positions: the old and the new positions of the object and its shadow. We mounted a table underneath the volume data with an absorbing layer to get a more effective shadow cue and a second view from reflection. We allow an artefact and do not re-render the shadow on other volume objects as these shadows can be relatively large, but play only a minor role in the actual perception of the object. All changed parts of the view are marked on the refresh surface by making the refresh values positive. After all parts have been drawn, the pixels that need to be refreshed are calculated or interpolated with the described adaptive refinement scheme. The speed-up is proportional to the area of the changed parts divided by the total view area.

The third acceleration, view movement, is based on the strong correlation between two views when translating one with respect to the other. In that case, we round off a subvoxel translation to integer pixel coordinates and shift the view together with the refresh surface. The offsets of the raster (o_x, o_y) change so that the same view positions lie on the raster. After translating the view, a new part is uncovered by setting the refresh values to ∞. The speed-up is proportional to the area of the uncovered part divided by the total view area.

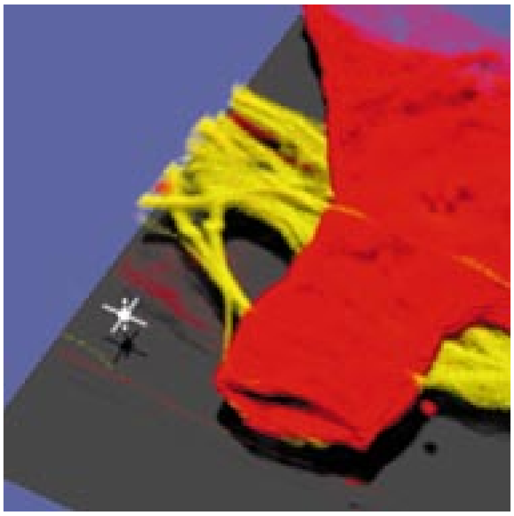
The degree to which a user experiences interactiveness depends on the loop that controls the adaptive refinement process. In our implementation, the adaptive refinement process is controlled by the spacing d and variation threshold a. After each refinement, the value of d or a is decreased. As d can only take a limited number of values, we prefer d to depend on a and to control the refinement process through variable a. Before each refinement sequence, a starts with the value a_0 and decreases after each iteration by Δa. Both a_0 and Δa are adjusted thereafter: a_0 after the first response, Δa after the next refinements. It appears that this control loop is insufficient for the routine to come back at regular times, especially if d decreases from 2 to 1 and suddenly all remaining rays are cast. We decided to allow the refinement routine to return also if the number of cast rays exceeds a predefined maximum. By means of flags, the routine remembers which rays have been traced so that the refinement can easily be continued in the next call.

With these modifications, we expect the renderer to generate views with higher quality in the same amount of time, as it effectively uses the information from the previous view. We shall illustrate this with the interactive analysis of a confocal image of a blood vessel and nerves of a cat retina.

5 Interaction times

To illustrate how the modifications of the adaptive refinement scheme affect the rendering time and view quality, we interactively analyse the confocal image of a blood vessel and nerves of a cat retina (Fig. 10) with the help of the white 3D probe in the bottom left corner. The image has been recorded with two fluorescent dyes: one labels the blood vessel, the other, the nerves. Each labelling results in a grey-value volume image, while the cursor is scan-converted into another grey-value image (geometry volume). Thus three volumes are passed in rendering the view: vessel, nerve and geometry volume. As the view has a size of 256×256 pixels, the quality offered by the interpolation techniques shown in Fig. 7a and b is insufficient because several pixels would map to the same view template. Therefore, we prefer the sampling quality as visualized in Fig. 7c. From the time to render Fig. 7c, we estimate that Fig. 10 is rendered in 53 s ($4\times$ larger view, $3\times$ more volumes, $2\times$ more samples along a ray). In practice, the rendering time appears to be far shorter (20 s) due to the large extent of empty space around the volume.

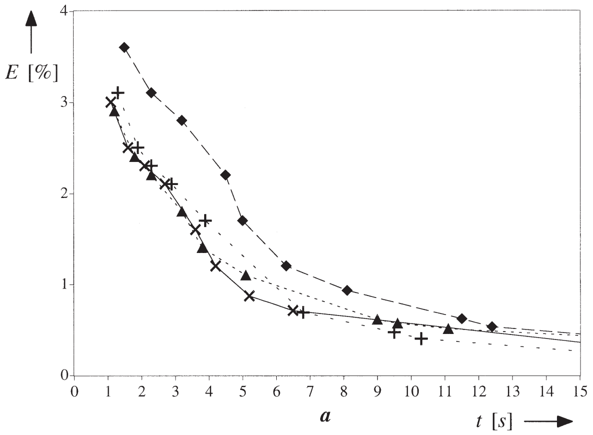
Starting from the situation of Fig. 10, we simulate the following interactions: a slight change in view angle, darkening the colour of the nerves, moving a 3D cursor and translating the view. For all interactions, we do not need to reilluminate the volume (only a small part of the volume is reilluminated when the cursor is moved). Each time, the view is calculated by two implementations: the first one where the rendering of a view starts from scratch after a change, and the second where we have implemented our modifications. As a measure of quality, we calculate the mean difference $<E>$ between all pixel values of the current view and the pixel values of the final view. The timings are performed on a SGI Indy equipped with a R4400SC CPU running at 150 MHz (86 SPECfp92, 82SPECint 92) with 64 MB main memory.



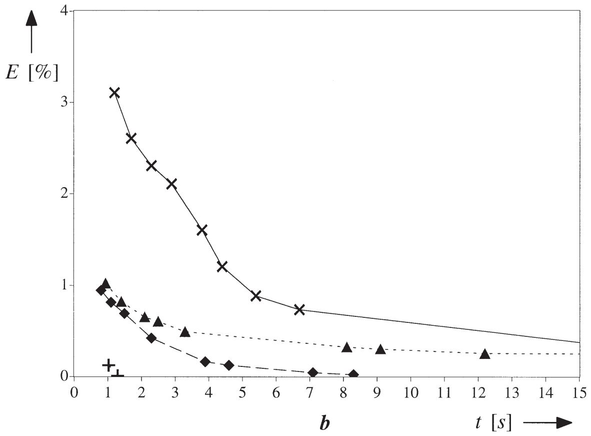
10

Fig. 10. Status before user interaction. Three grey-valued volumes are rendered at the same time: vessel volume (red), nerve volume (yellow), geometry volume with cursor (white). Volume size 167×153×33; voxel size 0.16×0.16×0.2 μm^3; view 256×256. Confocal image data published with courtesy of the authors (J.M. Messerli et al. 1993)

Fig. 11a, b. View error *E* (difference with final situation) as a function of update time for the four user interactions of Fig. 12: × changing view angle; ▲ darken nerves; + move 3D cursor; ◆ 33% horizontal translation. Each marker corresponds with one view update



11a



11b

The difference for each view update with the two implementations is shown in Fig. 11. Each marker corresponds to one view update. If the rendering starts from scratch, the difference for the first three interactions similarly decreases with time (Fig. 11a). The difference decreases more slowly for the view translation, as a larger part of the volume is visible, which requires more volume traces. If we apply our modifications, we get the graphs of Fig. 11b. The first updates are shown in Fig. 12. When comparing Fig. 11a and b, we see that, for a change in view, the corresponding graphs are similar. The reason is that almost all pixels get a new colour upon a change in view angle. For the change in nerve intensity, the difference *E* is lower for the second implementation because there are no unnecessary distracting interpolations. In moving the 3D cursor, the difference remains small, as only a small part of the view is affected. Finally, for translating the view, the difference for the second implementation is about three times less, as only 33% of the view needs to be re-rendered.

We conclude that the second implementation produces views with an equal or better quality in the same time as the first implementation without our modifications. The speed-up may vary from 0 (change in view angle) to 1500% (movement of a 3D cursor).

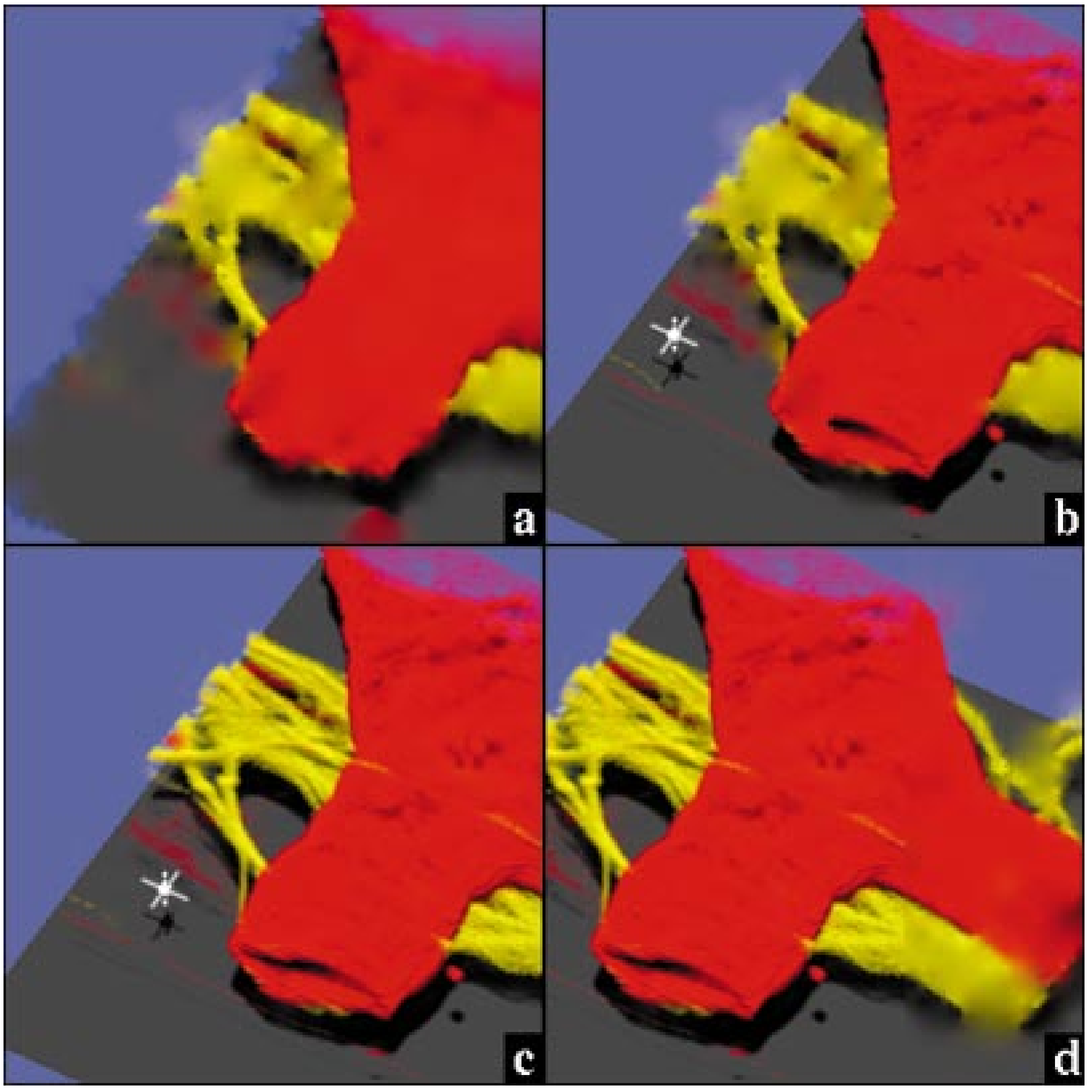


Fig. 12a–d. First response with improved rendering techniques after four types of user interactions: **a** change view angle; **b** darken nerves; **c** move 3D cursor; **d** 33% horizontal translation

6 Conclusions

Several techniques have been studied to obtain interactive rates for a volume renderer to be used for analysing confocal images. Selective illumination, only when and where it is necessary, increases the interactive response by several orders of magnitude. Template-based illumination has been introduced for an illumination that is faster than tracing back from each voxel to the light source. At the same time, the method gives no broadening effects as with interpolating the illumination value from the previous layer. The method can easily be adapted to increase the number of rays that illuminates a voxel. This enables a smooth control between speed and quality.
For the viewing phase, we adapted template-based viewing for a grey-value volume to obtain a 2.5 times faster viewing method with the same quality as in case of DDA sampling combined with trilinear interpolation. Further speed-up is obtained by taking more advantage of the coherence between

two views. In suspending superfluous interpolation and only redrawing the changed parts, we can accelerate the rendering process up to 1500%.
All the techniques together give a first response within 1 s for average-sized confocal images on current workstations. As the speed of computers increases, we expect a further reduction in response time in the near future.

Acknowledgement. This research project was supported by a grant from the Technology Foundation (STW), Utrecht, The Netherlands, Project AWI00.2282.

References

Akimoto T, Mase K, Suenaga Y (1991) Pixel-selected ray tracing. IEEE Comput Graph Appl 11:14–22
Bergman L, Fuchs H, Grant E (1986) Image rendering by adaptive refinement. Comput Graph 20:29–37
Danskin J, Hanrahan P (1992) Fast algorithms for volume ray tracing. Proceedings of 1992 Workshop on Volume Visualization. N.A., pp 91–98.
Drebin R, Carpenter L, Hanrahan P (1988) Volume rendering. Comput Graph 22:65–74
Ebert DS, Parent RE (1990) Rendering and animation of gaseous phenomena by combining fast volume and scanline A-buffer techniques. Comput Graph 24:357–366
Kajiya JT (1984) Ray tracing volume densities. Comput Graph 18:165–173
Kaufman A, Yagel R, Bakalash R, Spector I (1990) Volume visualization in cell biology. Visualization '90, IEEE Computer Society Press, Los Alamitos, pp 160–167
Lacroute P, Levoy M (1994) Fast volume rendering using shear-warp factorization of the viewing transformation. Siggraph'94 Orlando, ACM Press, New York, pp 451–458
Laur D, Hanrahan P (1991) Hierarchical splatting: a progressive refinement algorithm for volume rendering. Comput Graph 25:285–288
Levoy M (1988) Display of surfaces from volume data. IEEE Comput Graph Appl 8:29–37
Levoy M (1990a) Volume rendering by adaptive refinement. Visual Comput 6:2–7
Levoy M (1990b) A hybrid ray tracer for rendering polygon and volume data. IEEE Comput Graph Appl 10:33–40
Levoy M (1990c) Efficient ray tracing of volume data. ACM Trans Graph 9:245–261
Levoy M (1990d) Gaze-directed volume rendering. Comput Graph 24:217–223
Lorensen WE, Cline HE (1987) Marching cubes: a high resolution 3D surface construction algorithm. Comput Graph 21:163–169
Messerli JM, Voort HTM van der, Rungger-Brändle E, Perriard JC (1993) Three-dimensional visualization of multi-channel volume data: the amSFP algorithm. Cytometry 14:725–735
Montani C, Scopigno R (1990) Rendering volumetric data using the STICKS representation scheme. Comput Graph 24:87–93
Nielson GM, Hamann B (1990) Techniques for the interactive visualization of volumetric data. Visualization '90, IEEE Computer Society Press, Los Alamitos, pp 45–49
Shen HW, Johnson CR (1994) Differential volume rendering: A fast volume visualization technique for flow animation. (cont.) Visualization '94, IEEE Computer Society Press, Los Alamitos, pp 180–187
Shu R, Liu A (1991) A fast ray casting algorithm using adaptive isotriangular subdivision. Visualization '91, IEEE Computer Society Press, Los Alamitos, pp 232–238
State A, McAllister J, Neumann U, Chen H, Cullip TJ, Chen DT, Fuchs H (1995) Interactive volume visualization on a heterogeneous message-passing multicomputer. 1995 Symposium on Interactive 3D Graphics Monterey, ACM Press, New York, pp 69–74
Voort HTM van der, Noordmans HJ, Messerli JM, Smeulders AWM (1993) Physically realistic volume visualization for interactive analysis. Proceedings of Eurographics, Aire-la-Ville, Eurographics, Paris, pp 295–306
Westover L (1990) Footprint evaluation for volume rendering. Comput Graph 24:367–376
Wilson T, Sheppard CJR (1984) Theory and practice of scanning optical microscopy. Academic Press, London
Yagel R, Kaufman A (1992) Template-based volume viewing. Eurographics 11:153–167
Yoo TS, Neumann U, Fuchs H, Pizer SM (1992) Direct visualization of volume data. IEEE Comput Graph Appl 11:63–71
Zuiderveld KJ, Koning AHJ, Viergever MA (1992) Acceleration of ray-casting using 3D distance transforms. Proceedings of Visualization in Biomedical Computing 1992, Chapel Hill, N.C., pp 324–335.

HERKE JAN NOORDMANS is a PhD student at the University of Amsterdam (BioCentrum Amsterdam, Institute for Molecular Cell Biology) and has a great interest in intelligent interactive image analysis systems. He studied applied physics at the University of Twente, Enschede, The Netherlands, and graduated in 1992. His PhD thesis (June 1997) is entitled *Interactive analysis of 3D microscope images.*

ARNOLD W.M. SMEULDERS has been a Professor of Computer Science on multimedia information systems since 1993, and a Rear Professor of Biology on 3D microscopic imaging at the University of Amsterdam. Since 1977, he has been active in image processing and computer vision. He has published well over 125 papers. Initially interested in accurate and precise measurement from digital images, his current interest is in image databases and intelligent interactive image analysis systems. Another field of interest is method engineering and system engineering aspects of image processing.

HANS T.M. VAN DER VOORT studied physics at the University of Amsterdam. After receiving his MS in 1981, he became active in the field of confocal microscopy. His work included the design of image processing systems, theory of microscopic image formation, volume visualization and image restoration. He received his PhD in 1989 with a thesis entitled *Three-dimensional image formation and processing in the confocal microscopes*. His special interests lie in image restoration and interactive volume exploration. In 1994, he founded his own software company, *Scientific Volume Imaging B.V.*