

Modelling and Rendering Graphics Scenes Composed of Multiple Volumetric Datasets

Adrian LEU and Min Chen
Department of Computer Science
University of Wales Swansea
Singleton Park, Swansea SA2 8PP, United Kingdom
{csadrian, m.chen}@swansea.ac.uk

Abstract. This paper presents a method for modelling graphics scenes consisting of multiple volumetric objects. A two-level hierarchical representation is employed, which enables the reduction of the overall storage consumption as well as rendering time. With this approach, different objects can be derived from the same volumetric dataset, and 2D images can be trivially integrated into a scene. The paper also describes an efficient algorithm for rendering such scenes on ordinary workstations, and addresses issues concerning memory requirements and disk swapping.

Keywords. Volume graphics, volume rendering, modelling, multiple-volume scene representation.

1. Introduction

The advances in volume visualisation [1-2] over the past decade, coupled with the rapid development of computer power, suggested that volume visualisation may be developed into *volume graphics* [3-4] as a major sub-field in computer graphics. Volume-based techniques provide support for modelling objects built from sampled as well as computed data, visualising their surfaces as well as internal structures, and rendering solid models as well as amorphous phenomena. It is such features that make volume graphics potentially more attractive than traditional surface-based graphics and allow it to reach beyond the scope of visualisation applications.

One of main obstacles in the development of volume-based applications has been the size of volume datasets. However, the problem is becoming alleviative due to the rapid reduction in the cost of computer hardware. This very much reminds us an old scenario in image space where the development of raster graphics superseded that of vector graphics. Considering the advances made in, and the advantages demonstrated by, volume visualisation, we have to ask ourselves whether the same scenario would repeat in object space. The recent ONR volume visualisation workshop [4] highlighted the potential of volume graphics, and the technical problems that are yet to be resolved. This paper addresses the problem of modelling and rendering complex scenes that are composed of multiple volumetric datasets.

The foundation of volume graphics has been laid by application driven research in the past decade, namely the visualisation of large, volumetric datasets of solid or amorphous nature. The work of Herman and Liu [5] on the display of organs from computed tomograms, Blinn's on the display of clouds and dusty surfaces [6] and Kajiya's on rendering volumetric

densities [7] led to the development of *volume rendering*, a term which was invented by Drebin, Carpenter and Hanrahan in [8] that also contains some important ideas including the use of local gradients, image warping and digital compositing of slices in a volume. The ray casting algorithms for volume rendering by Levoy [9] and Sabella [10] have generated much interest in the field. Since then, many software and hardware solutions were devised to deal with large datasets that often required a huge amount of processing power and storage space. Solutions to meet those requirements include hardware coding of rendering algorithms (among which the work by Kaufman et al., Höhne et al. and Gunther et al. [11] being the most advanced to day), and algorithms for viewing transformation [12], spatial arrangements of voxels [13], irregular grids [14] and other data representations (such as wavelet, Fourier, Hartley or compression) [15].

A few modelling schemes for organising information in a volumetric dataset were proposed. Some of them were developed for specific applications (such as the generalised voxel-model for the VOXEL-MAN atlas by Höhne et al. [16]), others for specific hardware implementations (such as the Cube-3 machine by Pfister and Kaufman [17]).

A majority of the solutions to day have attempted to provide real-time frame rates for visualising large, single, static or dynamic datasets obtained from either simulation or digitisation. Apart from [16] and [18] little attention has been given to the problems arising from rendering a complex volumetric scene. To deal with multiple datasets, most approaches have involved a pre-processing stage where objects are voxelised and integrated into a single volumetric dataset, which is then rendered using a conventional rendering algorithm [19]. This suffers from a number of drawbacks. (a) For a relatively complex scene, the resulting dataset can be extremely large, often containing many ‘blank’ voxels, and this leads to excessive space and time consumption in both voxelisation and rendering. (b) Individual objects need to be resized and transformed in order to conform to a single regular grid and its uniform resolution. For previously voxelised datasets, the re-sampling process usually results in degeneration of data quality. (c) It is also difficult to associate individual objects with different physical properties such as colour and reflection coefficients.

In this paper, we propose a two-level scene representation scheme in which objects are modelled separately from their underlying volumetric datasets. The scheme facilitates multi-volume scenes, object-based properties, data sharing and image integration, while reducing the space requirement extensively. We name this representation scheme as TROVE (Two-level Representation Of Volumetric Environments). Along with TROVE, we also present a rendering method that is primarily a combination of the surface-based ray tracing algorithm and the ray casting algorithm for direct volume rendering [20]. A modification is made to a brute force approach to reduce the amount of disk swapping in a situation that a whole scene cannot entirely reside in the memory. The rest of the paper consists of two main sections that address the modelling and rendering issues concerning TROVE respectively, which are followed by our concluding remarks.

2. Modelling Multi-Volume Scenes

2.1 Volumetric Objects

Volumetric datasets often take the form of raw data obtained from digitisation and simulation. They have also been used as a means for modelling surface-based objects such as implicit surfaces [21] and CSG models [22]. The most commonly used volume representation is a 3D regular grid where each voxel (i.e. grid point) is associated with one or a set of scalar values.

The data is normally assumed to be continuous within the volume, and values at non-voxel points are estimated through interpolation. Most raw volume datasets do not contain colour or opacity information, which is usually defined based on the voxel values using mapping functions.

For instance, given a raw volumetric dataset $V = \{v_{x,y,z} \mid x=0,1,\dots,N_x; y=0,1,\dots,N_y; z=0,1,\dots,N_z\}$, we can construct two such mapping functions, $\mathcal{C}(v) \in \mathbf{R}^3$ and $\mathcal{O}(v) \in \mathbf{R}$, for defining the RGB colour and opacity of each voxel. The tuple $\langle V, \mathcal{C}, \mathcal{O} \rangle$ in fact specifies a simple object, for which V defines its geometry and \mathcal{C} , together with \mathcal{O} , define its very basic physical characteristics. Complex objects can also be specified by giving additional mapping functions, which may be used for defining reflection coefficients, extra colour elements (such as in the 7-colour model [23]), and other physical characteristics. Without undermining any generality, we shall concentrate only on \mathcal{C} and \mathcal{O} in the following discussions unless otherwise stated.

A volumetric dataset can also be associated with an interpolation function \mathcal{I} used to define the values of non-voxel points. Although linear interpolation is the most widely-used method in volume applications, quadratic, or other non-linear functions are sometimes employed to achieve smoother interpolation [24]. With \mathcal{I} , we can extend the tuple to $\langle V, \mathcal{C}, \mathcal{O}, \mathcal{I} \rangle$.

For an object represented by such a tuple, the regular grid of V , where voxels are uniformly positioned, determines a local coordinate system. In order to position the object in a graphics scene, we associate the tuple with a transformation \mathcal{T} , that facilitates the placement of voxels in a world coordinate system. This new tuple $\langle V, \mathcal{C}, \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ is defined as the representation of a *volumetric object* in TROVE.

It is necessary to stress the difference between a volumetric object and its underlying volumetric dataset. When a volumetric dataset V is associated with two different sets of $\mathcal{C}, \mathcal{O}, \mathcal{I}, \mathcal{T}$, there are two different volumetric objects. Figure 1 shows a scene that consists of fifteen objects derived from the same volume dataset, each with varying $\mathcal{C}, \mathcal{O}, \mathcal{I}$ or \mathcal{T} . All the objects, including the cubes, the floor and the different heads or skulls, were rendered from the computer tomography (CT) scan of a head.



Figure 1. A scene consisting of fifteen objects sharing the same dataset.

2.2 Two-Level Scene Representation

The scene shown in Figure 1 consists of fifteen different objects built upon the same volumetric dataset which is a CT scan of 256x256x113 voxels. The dataset itself occupies about 16MB space. If the scene were to be represented by a single volume, each object would have to be transformed, and colour and opacity values would have to be mapped onto every voxel, resulting in a minimum of 240 MB data. Taking the blank voxels into account, the total space requirement would be much more. It would be impossible to accommodate scenes with a large number of objects, such as fractal-based scenes. Such a level of the storage requirement can be sufficiently reduced if a two-level scene representation scheme is introduced.

Consider a scene composed of n objects, $\langle V_1, C_1, O_1, J_1, T_1 \rangle, \langle V_2, C_2, O_2, J_2, T_2 \rangle, \dots, \langle V_n, C_n, O_n, J_n, T_n \rangle$. Assume that some objects shared the same volumetric datasets, and there are m distinguishable datasets in the scene, V'_1, V'_2, \dots, V'_m . We organise the data into two levels, namely the *voxel* level and the *object* levels, as illustrated in Figure 2. The voxel level contains all the raw volumetric datasets V'_1, V'_2, \dots, V'_m , each of which maintains its original resolution and local coordinate system. At the object level, each object is described by a set of functions including C_i, O_i, J_i and T_i , together with a link pointing to the corresponding dataset at the voxel level.

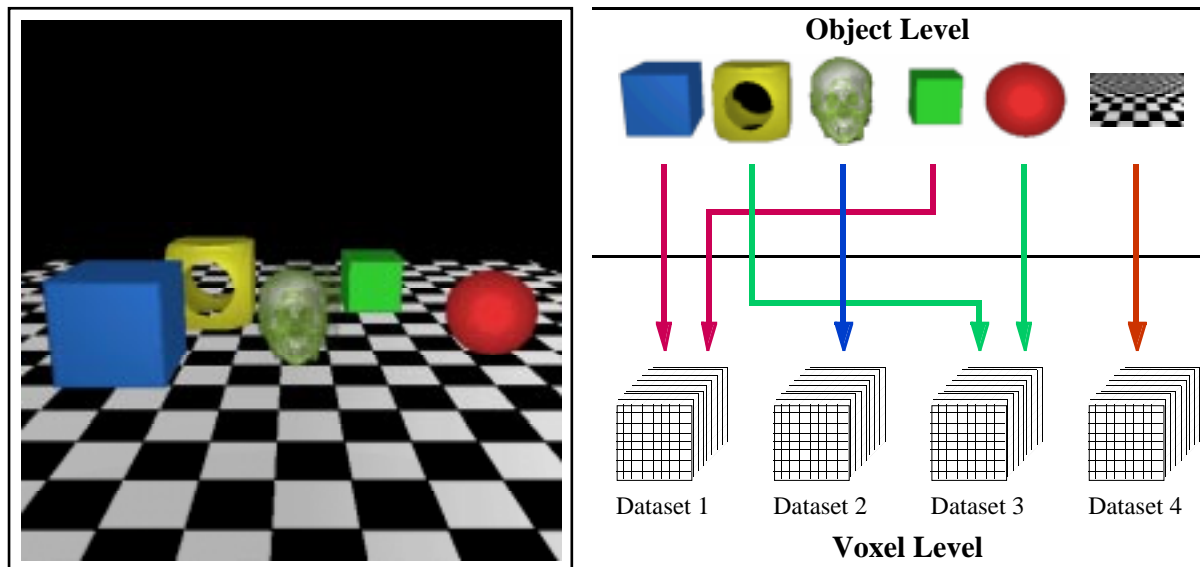


Figure 2. The data hierarchy in a TROVE representation.

This two-level hierarchy forms the basis of the TROVE scheme, and it offers the following advantages in comparison with the method of integrating all objects into a single volumetric dataset:

- By maintaining the original volumetric datasets, the scheme eliminates all ‘blank’ voxels except for those within the original datasets.
- It allows objects to share voxel-level data that normally takes much more space than various associated functions.
- It offers a rendering algorithm the flexibility of computing colour and opacity values at different rendering stages, the decision of which is normally influenced by the trade off between space and speed.

- It naturally provides each object with a bounding box, which will play an important role in the rendering process to be described later.

These advantages and some others will become more apparent when we further our discussions in the following sections.

2.3 The Specification of V , C , O , I and T

2.3.1 Volumetric Dataset

Volumetric datasets are mostly defined in the form of a 3D regular grid where each grid point is associated with some information. Typically, a CT dataset that is obtained using an X-ray scanning device contains a collection of scalar values representing the density of the sampled points in the volume. There are many other digitising devices that generate such datasets directly, including MRI, electronic sensing and surface measuring. In finite element analysis, voxel values may represent various simulated physical properties such as temperature, velocity, pressure, etc. Voxelisation of surface representations is also common process in computer graphics. For example, range data obtained from different camera positions is often voxelised first in order to be merged together.

Most existing sampling devices do not generate colour information directly in the volumetric form, and instead, colours are usually sampled as 2D images which are then texture-mapped onto objects during rendering. Such a process may seem a little clumsy in volume-based modelling and rendering. Volumetric datasets are a direct extension of 2D images, and naturally voxels can be defined with colour information. Many existing volume visualisation techniques are capable of rendering such datasets without involving texture mapping. It is therefore necessary for a volume-based representation scheme to accommodate datasets consisting of colour or other information. Arguably, it is possible for a digitising device to be able to generate colour as well as geometric information, or at least it is not difficult to map textures directly onto volumetric data. In scientific visualisation, it is also not uncommon for using multiple colour channels to represent different physical properties (e.g. red for temperature, green for pressure and blue for velocity).

To provide sufficient flexibility, the TROVE scheme allows voxel-level data to be specified through a combination of channels chosen from:

- *voxel channel* (V) — A raw dataset upon which the colour and opacity mapping functions are defined. The data is normally generated directly by a digitisation, simulation or voxelisation process. The channel accepts data with a depth of 1, 8, 16 or 32 bits.
- *bit channel* (V^1) — This channel can be used to specify the colour of a volumetric object directly as a binary scalar field. A typical use of this channel is the modelling of monochrome images as volumetric objects.
- *grey channel* (V^8) — This channel can be used for specifying directly the colour of a volumetric object whose colour range is restricted to 256 grey scales.
- *red channel* (V^R), *green channel* (V^G), *blue channel* (V^B) — These three channels are normally used together for specifying the colour of a volumetric object directly instead of through a colour mapping function.
- *alpha channel* (V^α) — This channel is used to define raw opacity data.

There are also several additional channels specified in TROVE, including:

- *three additional colour channels and a beta channel* — These channels enable the specification of the 7-colour model [23] which normally leads to more realistic colour combination in volume rendering. This model offers
- *an interpolation channel* — This channel is designed to support a more sophisticated rendering engine that could handle different interpolation functions associated to different parts of a dataset.
- *four user-definable channels* — These channels are reserved for future extension. For example, they may be used to define additional voxel channels upon which more complicated mapping functions can be defined.

In most applications, only one or a few channels are needed for each volumetric dataset. The flexibility of using different channels enables objects from different sources to be mixed together into a scene easily. Figure 3 shows such a scene with four objects, including:

- a sphere which is specified using only a voxel channel, and whose colours and opacity are defined through mapping functions at the object level,
- a floor which is specified through a bit channel,
- and two background planes both of which are specified through red, green and blue channels.

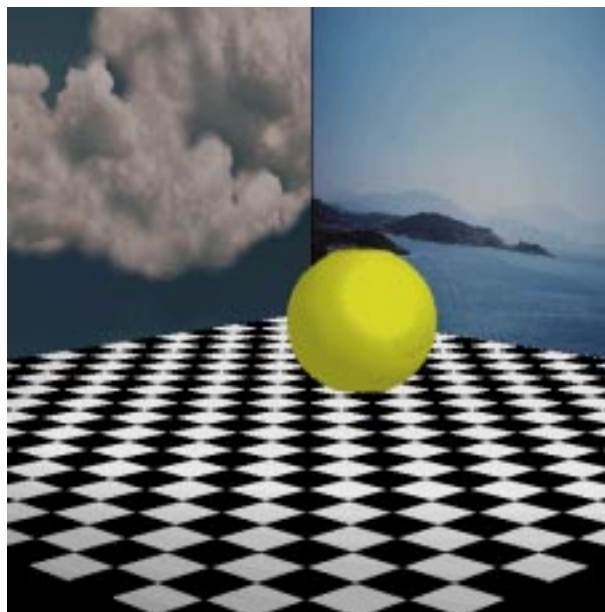


Figure 3. A scene consisting of four volumetric datasets each defined through a different set of channels.

The figure includes a sphere defined through the voxel channel, a chess-board floor defined through the bit channel and two photo images defined through the red, green and blue channels.

Figure 3 also demonstrates an important feature of the TROVE scheme, that is, images can be trivially integrated into graphics scenes. Each image is basically a volumetric dataset with one of its dimension equal to 1. Moreover, the concept of translucency can also be applied to imaging data in the same way as volumetric data, allowing the concept of layers that is widely available in image processing software to be implemented in a graphics system. In terms of both modelling and rendering, this mechanism is more cost-effective than traditional texture-mapping in surface-based graphics.

2.3.2 Colour Mapping

When colour information is not available at the voxel level, for example, through \mathbf{V}^R , \mathbf{V}^G and \mathbf{V}^B , it is necessary to define a *colour mapping function* \mathbf{e} . Constructing a look-up-table is the most commonly-used method in volume rendering. Given a volume \mathbf{V} , a colour look up table divides $(-\infty, \infty)$ into a set of s domains, $(-\infty, \mathbf{d}_1)$, $[\mathbf{d}_1, \mathbf{d}_2)$, ..., $[\mathbf{d}_{s-1}, \infty)$, each associated with a colour. \mathbf{e} is therefore defined upon this look-up-table as:

$$\mathbf{e}(\mathbf{v}) = \begin{cases} \mathbf{c}_1 & -\infty < \mathbf{v} < \mathbf{d}_1, \\ \mathbf{c}_2 & \mathbf{d}_1 \leq \mathbf{v} < \mathbf{d}_2, \\ \dots & \dots \\ \mathbf{c}_s & \mathbf{d}_{s-1} \leq \mathbf{v} \leq \infty. \end{cases} \quad [1]$$

where $\mathbf{c}_i \in \mathbf{R}^3$ is a 3-tuple representing red, green and blue values. In many cases, it is thereby desirable to represent \mathbf{e} using three polynomial functions for the red, green and blue components respectively. For example, given a voxel value \mathbf{v} , its colour can be obtained as:

$$\begin{aligned} \mathbf{e}^R(\mathbf{v}) &= \mathbf{r}_t \mathbf{v}^t + \mathbf{r}_{t-1} \mathbf{v}^{t-1} + \dots + \mathbf{r}_1 \mathbf{v} + \mathbf{r}_0; \\ \mathbf{e}^G(\mathbf{v}) &= \mathbf{g}_t \mathbf{v}^t + \mathbf{g}_{t-1} \mathbf{v}^{t-1} + \dots + \mathbf{g}_1 \mathbf{v} + \mathbf{g}_0; \\ \mathbf{e}^B(\mathbf{v}) &= \mathbf{b}_t \mathbf{v}^t + \mathbf{b}_{t-1} \mathbf{v}^{t-1} + \dots + \mathbf{b}_1 \mathbf{v} + \mathbf{b}_0. \end{aligned} \quad [2]$$

where $\{\mathbf{r}_i, \mathbf{g}_i, \mathbf{b}_i, 0 \leq i \leq t\}$ are pre-defined coefficients for the mapping functions. Relatively low order functions, such as cubic or quadratic ones, are normally quite adequate in most applications. Thus, this form of mapping functions usually requires less space than a look-up-table, and can be processed faster.

With the TROVE scheme, users are allowed to specify one of following types of colour mapping functions for each object:

- *raw* — using the colour information defined at the voxel level. The commonly-used channel combinations include $(\mathbf{V}^R, \mathbf{V}^G, \mathbf{V}^B)$, (\mathbf{V}^8) , and (\mathbf{V}^1) .
- *look-up-table* — which is defined by an integer value s , together with $s-1$ real values $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_{s-1}$.
- *general polynomial* — which is defined by an integer value t , together with $3(t+1)$ real values $\mathbf{r}_0, \mathbf{r}_1, \dots, \mathbf{r}_t; \mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_t; \mathbf{b}_0, \mathbf{b}_1, \dots, \mathbf{b}_t$.
- *constant, linear, quadratic and cubic* —TROVE allows these four special forms of polynomial functions to be defined with their own type identifiers.

Figure 4 shows an example scene consisting of four different objects and a chess floor. The spheres and the cubes were rendered using different colour maps, namely a look-up table for the two layers sphere and the yellow cube, a linear map for the green sphere and a quadratic map for the red cube.

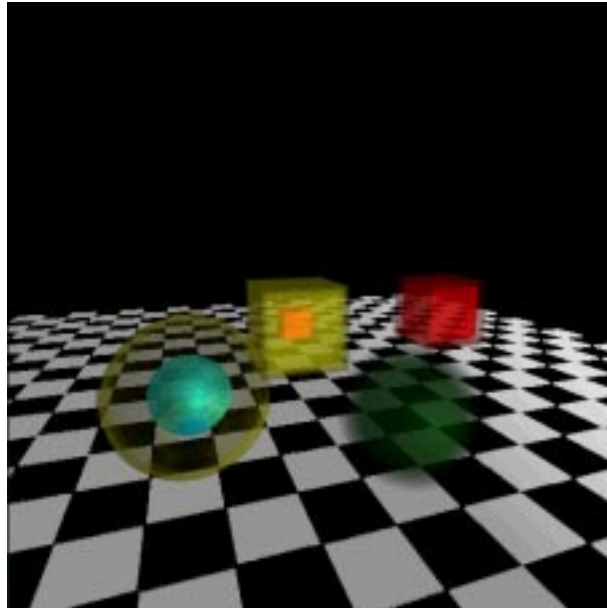


Figure 4. Objects using different colour mapping functions.

2.3.3 Opacity Mapping

Similar to colour mapping functions, users may specify an opacity mapping in a number of forms, including

- *raw* — using the opacity information defined at the voxel level through channel (\mathbf{V}^α).
- *look-up-table* — which is defined by an integer value s , together with $s-1$ real values $\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_{s-1}$.
- *general polynomial* — which is defined by an integer value t , together with $(t+1)$ real values $\alpha_0, \alpha_1, \dots, \alpha_t$. For a given voxel value \mathbf{v} , its opacity is computed as $\mathcal{O}(\mathbf{v}) = \alpha_t \mathbf{v}^t + \alpha_{t-1} \mathbf{v}^{t-1} + \dots + \alpha_1 \mathbf{v} + \alpha_0$.
- *constant, linear, quadratic* and *cubic* — four special forms of polynomial functions defined with their own type identifiers.

The concept of opacity plays an important role in volume-based graphics. Not only does it influence the colour of an object, but more importantly it defines the geometric appearance of the object. While voxel values in a volumetric dataset define the underlying geometry of the volume, it is opacity values that determine the visible geometry of an object built upon the dataset. As illustrated in Figure 5, objects, which are built from the same spherical volumetric dataset but with different opacity mapping functions, appears with very distinctive geometric features. The red and green spheres are defined from the same dataset, but their opacities are defined through a look-up table and a linear polynomial function, respectively. The two cubes have the same original dataset and have been defined with a look-up table opacity maps.

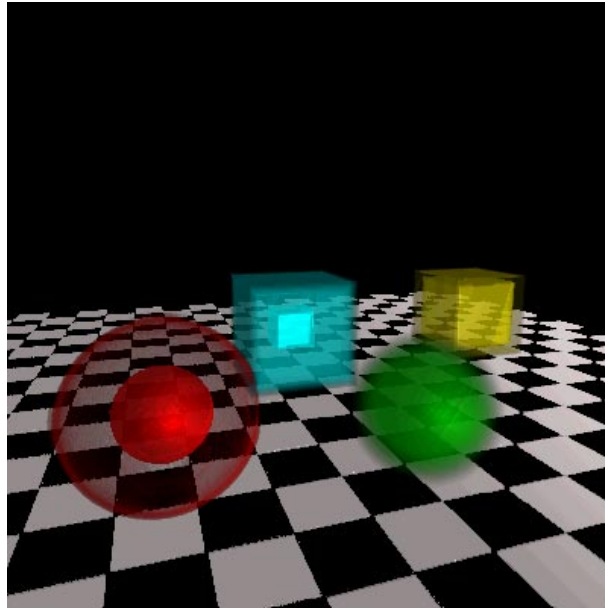


Figure 5. Objects using different opacity mapping functions.

2.3.4 Interpolation

In a volumetric dataset, only the values at the grid points are explicitly defined. Values of non-grid points are determined through interpolation. So far linear interpolation has dominated most volume rendering applications[24], though the use of non-linear interpolation has also been reported in the literature [25]. Moreover, many non-linear functions used are application-dependent, and they were defined based on the underlying simulation process which generated the datasets. It is generally difficult for a modelling scheme to facilitate all those functions. The TROVE scheme allows a selection of interpolation functions of certain generality.

Given a volumetric dataset $\mathbf{V}=\{v_{x,y,z} \mid x=0,1,\dots,N_x; y=0,1,\dots,N_y; z=0,1,\dots,N_z\}$, and an arbitrary non-grid point (u, v, w) within the volume, an interpolation function $\mathcal{I}(\mathbf{V}, u, v, w) \in \mathbf{R}$ determines the value at the point. Let $x \leq u \leq x+1$, $y \leq v \leq y+1$, $z \leq w \leq z+1$. As shown in Figure 6, such a point will be bounded by a small cube cornered by eight grid points.

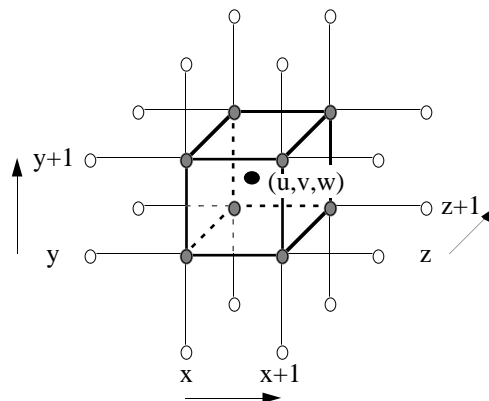


Figure 6. A non-grid point (u, v, w) surrounded by eight grid points.

The TROVE scheme allows the specification of four basic interpolation functions, which are:

- nearest neighbour — \mathcal{J} returns the value of the nearest grid point.
- linear interpolation — \mathcal{J} interpolates tri-linearly the values of the eight neighbouring grid points, as shown in grey in Figure 6.
- non-linear interpolation — \mathcal{J} interpolates the values of the eight neighbouring grid points by applying a non-linear function to the x, y, z directions respectively in a manner similar to tri-linear interpolation. It is a one dimensional quadratic function in the form of

$$\mathcal{H}(t) = v_{\text{start}} - (\omega_2 t^2 + \omega_1 t + \omega_0)(v_{\text{start}} - v_{\text{end}}) \quad [3]$$

where $0 \leq t \leq 1$ and ω_i ($i=0,1,2$) are user-definable coefficients. This feature is particularly useful for providing anti-aliasing to low resolution or low depth volumetric data, such as bitmap images.

- cubic interpolation — this assumes that the small cubic volume surrounded by the eight grid points is a cubic field

$$\begin{aligned} \mathcal{Q}(x,y,z) = & \lambda_1 x^3 + \lambda_2 y^3 + \lambda_3 z^3 + \lambda_4 x^2 y + \lambda_5 x^2 z + \\ & \lambda_6 y^2 x + \lambda_7 y^2 z + \lambda_8 z^2 x + \lambda_9 z^2 y + \lambda_{10} xyz \\ & + \lambda_{11} x^2 + \lambda_{12} y^2 + \lambda_{13} z^2 + \lambda_{14} xy + \lambda_{15} yz \\ & + \lambda_{16} zx + \lambda_{17} x + \lambda_{18} y + \lambda_{19} z + \lambda_{20}. \end{aligned} \quad [4]$$

where λ_i ($i=1,2, \dots,20$) are determined by the values of the eight grid points plus another 12 points chosen from the 24 extended neighbouring points (only 16 extended neighbouring are shown in Figure 6). Special treatment is necessary for boundary conditions and for low resolutions.

2.3.5 Transformation

Like most graphics modelling environments, each scene in TROVE has a world coordinate system. The position of each object in a scene is specified through a transformation function \mathcal{T} which transforms local *volume coordinates* (VC) defined within a volumetric dataset to *world coordinates* (WC) via *normalised volume coordinates* (NVC). Such a transformation may facilitate a combination of geometrical transformations including translation, rotation, scaling, distortion and clipping.

Each object is considered to be bounded by a hexahedral box with quadrilateral faces. As shown in Figure 7, it is not necessary for the adjacent faces or edges to be orthogonal, nor for the size of object to correlate with the dimensions of the corresponding volumetric dataset. In TROVE, a volumetric dataset in VC is first mapped onto the unit cube in NVC, where an orthogonal box is specified by its x, y, z-extends $\in [0, 1]$. The box is then mapped onto the bounding box of an object through the eight corner points.

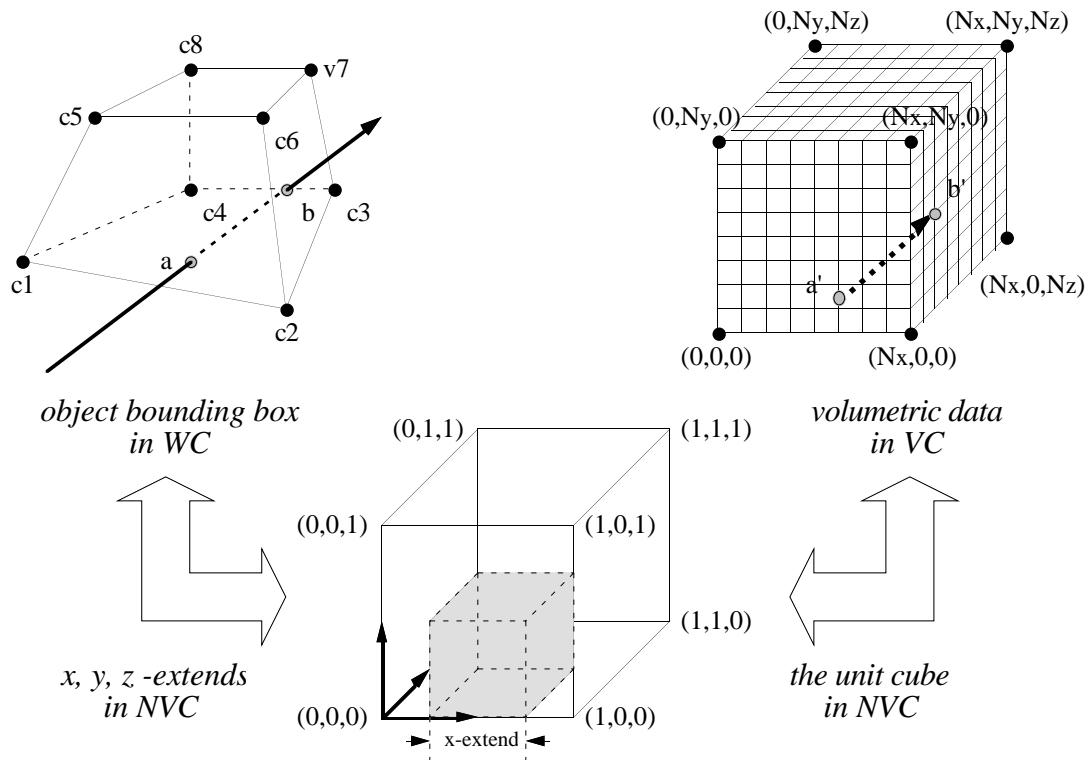


Figure 7. The transformation between local coordinates and world coordinates.

The problem of positioning the object thus becomes the problem of specifying the eight corners of its bounding box in WC and its extends in NVC. In TROVE, the bounding box of an object can be defined by either

- giving a 4×4 transformation matrix which is to be applied to the corners of a unit cube in WC,
- or specifying the positions of the eight corners in world coordinates directly,

The former is suitable for simple transformations, while the latter is easy to experiment through trial-and-error.

A reverse transformation will be needed during a rendering process. For example, consider a ray traced through the scene that hits an object as shown in Figure 7. Assume that the ray enters the bounding box at point a and then leaves from b . We need to convert a and b to the local volumetric coordinates for the ray to be traced through the interior of the dataset. It is not difficult to see that this is just a standard texture mapping problem. These bounding boxes also assist with the rendering process by facilitating fast intersection tests and reducing the overall computational costs, which will be discussed in Section 3.

The main advantage of allowing distorted bounding boxes is to enable a wider range of objects to share to same volumetric dataset. This further reduces the space requirements of complex scenes. For example, it is not difficult to build all those heads shown in Figure 8 from a simple cubic volumetric dataset. With the same CT scan used in Figure 1 two heads were obtained through rotation from the original dataset, the blue and green one respectively. The pink object was obtained through scaling along the x -axis and the yellow one through deformation of the bounding box.

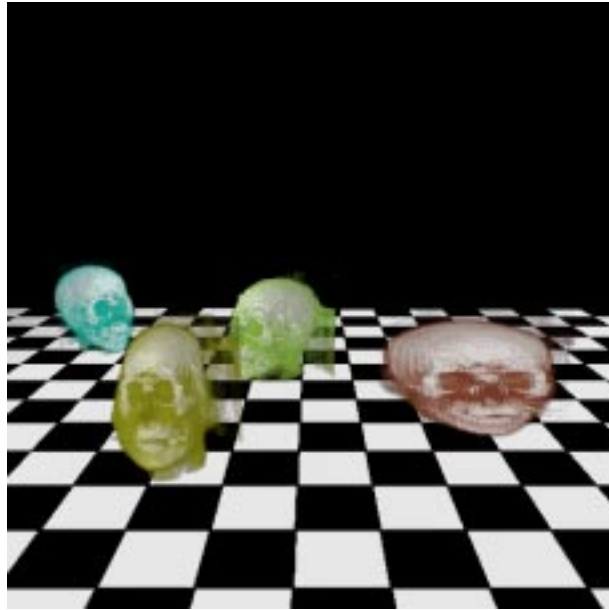


Figure 8. A set of objects built upon the same volumetric dataset with different transformations (rotation, scaling and distortion).

3. Rendering Multi-Volume Scenes

A multi-volume scene may be rendered in a number of ways. For example, surfaces can be first constructed, and then rendered with appropriate textures [26]. Voxels can also be splatted onto an image plane as coloured footprints [27], with the aid of a hidden surface removal algorithm, typically the z-buffer or depth sorting method. We have chosen ray casting as the primary rendering method for TROVE. While surface reconstruction suffers from the inability to render amorphous structures, and splatting has difficulties to produce high quality surface for solid models, ray casting is able to handle both types of objects. Both surface reconstruction and splatting involves the processing of every voxel in the scene, while ray casting deals with only those voxels which rays pass through. Therefore ray casting is likely to be in a much lower order of complexity in the average case.

Our ray casting algorithm is a combination of traditional surface-based ray tracing and volume-based ray casting widely used in direct volume rendering. The former is performed at the object level of TROVE, and the latter at the voxel level.

3.1 Basic Ray Casting Algorithm

Consider a ray that is cast into a scene as illustrated in Figure 9, passing through a pixel in the image plane and intersecting with several volumetric objects. In TROVE, objects are bounded by hexahedral boxes, so intersections can be computed by intersecting the ray with faces of those bounding boxes. For each object that the ray interests, we obtain a pair of intersection points $\langle t_{near}, t_{far} \rangle$ represented by their distances to the starting point of the ray. Assuming that the volumetric objects do not intersect with each other, we can easily determine the order of intersections by examining these intersection points.

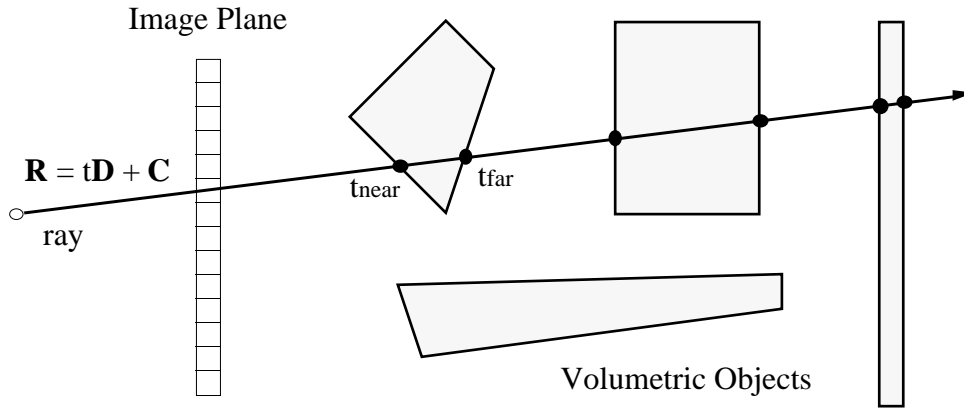


Figure 9. A ray intersections with several volumetric objects.

Let the ray enter the first intersected object through $\mathbf{a} = t_{near} \mathbf{D} + \mathbf{C}$ and leave at $\mathbf{b} = t_{far} \mathbf{D} + \mathbf{C}$. As discussed in Section 2.3.5, \mathbf{a} and \mathbf{b} are then transformed to local coordinates of the corresponding volumetric dataset, resulting \mathbf{a}' and \mathbf{b}' as shown in Figure 7. The volume-based ray casting algorithm then takes over and samples through the dataset at a regular interval. At each sampling point, the ray accumulates both colour and opacity as:

$$\mathbf{colour}_{init} = \text{nil}$$

$$\mathbf{opacity}_{init} = 0$$

$$\text{loop: } \begin{aligned} \mathbf{colour}_{next} &= \mathbf{colour}_{cur} + \mathbf{colour}_{sample} \cdot \mathbf{opacity}_{sample} \cdot (1 - \mathbf{opacity}_{cur}) \\ \mathbf{opacity}_{next} &= \mathbf{opacity}_{cur} + \mathbf{opacity}_{sample} \cdot (1 - \mathbf{opacity}_{cur}) \end{aligned} \quad [5]$$

$$\mathbf{colour}_{final} = \mathbf{colour}_{cur} + \mathbf{colour}_{background} \cdot (1 - \mathbf{opacity}_{cur})$$

If it accumulates enough opacity, the ray terminates and set the appropriate pixel in the image plane to the accumulated colour. If not enough opacity has been accumulated, the ray will be cast further into the scene, carrying with it the accumulated colour and opacity. For each ray, we maintain a list of possible intersections along the ray, in the form of a tuple $\langle \mathbf{O}, t_{near}, t_{far} \rangle$, organised in ascending order. A basic algorithm is outlined below :

Algorithm 1:

```

1  for each ray R[j] do
2    ordered_list := [];
3    for each volume object O[i] do
4      if R[j] intersects O[i] at t_near and t_far then
5        Insert(<O[i], t_near, t_far>, ordered_list);
6      endif
7    endfor
8    R[j].colour := nil;
9    R[j].opacity := 0;
10   while R[j].opacity < 1 and ordered_list != [] do
11     Remove(<O[k], t_near, t_far>, ordered_list);
12     Transform(<O[k], t_near, t_far>, <V[l], a', b'>);
13     VolumeRayCasting(R[j], <V[l], a', b'>);
14   endwhile
15   if R[j].opacity != 1.0 then
16     AccumulateColour(R[j], BACKGROUND, 1.0);
17   endif
18 endfor

```

The `Insert()` procedure inserts an object identifier and the associated intersection points into the ordered list if the corresponding volumetric object was hit by the ray, and the `Remove()` procedure removes the nearest object from the list. After $\langle t_{near}, t_{far} \rangle$ is transformed to \mathbf{a}' and \mathbf{b}' in local coordinates of the corresponding volumetric dataset, the procedure `VolumeRayCasting()` traces that dataset by sampling along the line between \mathbf{a}' and \mathbf{b}' .

Figure 10 shows an example scene consisting of a set of objects of varying colour and translucency. It demonstrates that the ray casting process has successfully captured the surfaces of the opaque objects as well as the internal structures of those translucent.

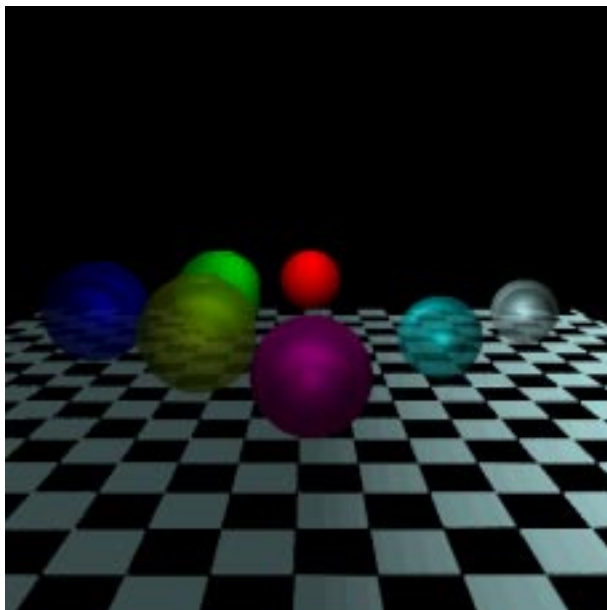


Figure 10. An example scene that demonstrates that ray casting has captured the surfaces of the opaque objects and the internal structures of those translucent

3.2 Major Computational Details

3.2.1 Intersection Computation

For each ray, the algorithm described above first builds an intersection list. This process requires only data at the object level, and involves up to six ray-quadrilateral tests and six inclusion tests. Considering the normals \mathbf{N}_k ($k=1,\dots,6$) of the six planar faces of a box pointing outside the object. These normals may relate to the ray direction \mathbf{D} in three ways:

(a) $\mathbf{D} \cdot \mathbf{N}_k = 0$ — the face is parallel to the ray and it is ignored.

(b) $\mathbf{D} \cdot \mathbf{N}_k < 0$ — we compute a potential t_{near} intersection using the ray-quadrilateral test. If there is an intersection, we record the distance from the ray starting position \mathbf{C} to the intersection point as $t_{near,k}$.

(c) $\mathbf{D} \cdot \mathbf{N}_k > 0$ — similar to (b), we compute $t_{far,k}$ if there is an intersection.

The $\langle t_{near}, t_{far} \rangle$ is determined as

$$t_{near} = \text{maximum}(\text{all } t_{near,k} \text{'s}) \text{ and } t_{far} = \text{minimum}(\text{all } t_{far,k} \text{'s}).$$

The ray does not intersect the box if no intersection points were found.

3.2.2 Volume Ray Casting

During volume ray casting, we assume the ‘blank’ space between any two consecutive objects intersected by a ray does not contain any opaque material. The backward ray casting from the viewing position is employed to take advantage of possible early termination of rays. Equation (5) is applied throughout the sampling process for pairs of $\langle t_{near}, t_{far} \rangle$ along each ray in the increasing order of t_{near} until an early termination or all intersections being sampled. If the accumulated opacity value is less than 1.0, the default background colour is mixed into the accumulated colour with an appropriate proportion, resulting in the final pixel colour.

The sampling distance is determined by a parameter associated to each object. It is a positive value defined in world coordinates, and the value is converted to the local volumetric coordinates prior to the rendering. When the parameter is defined as 0, the local distance 1 will be used. This equips the TROVE scheme with some degree of flexibility in controlling the sampling density.

3.3 Dealing with Disk Swapping

The main problem encountered while implementing this brute-force approach is the heavy disk swapping of voxel level data at line 12 in the pseudo-code given in Section 3.1. Most general purpose workstations nowadays have 32~128MB real memory but much more virtual memory. As a typical volumetric dataset requires about 8~32 MB, for a relatively complex scene, it is difficult for the real memory to accommodate all datasets during rendering. Every time a ray leaves an object and enters another, it is most likely for virtual-real memory swapping to occur. As most volumetric datasets would contain some transparent or translucent voxels, the possibility for a ray to be traced through several datasets is high. In this kind of situations, the succeeding datasets may have to be swapped into real memory from the disk while those already in real memory may have to be swapped out. More likely the same scenario would repeat for a group of nearby rays. Experiments carried out to quantify the cost of such swapping indicated that the time used for swapping and context switch is likely to be much more than that for rendering when a scene involves 10 or more volumetric datasets [28].

We noticed in the brute-force algorithm that the voxel level data is required only by the `VolumeRayCasting()` procedure. The intersection calculation needs just the data at the object level, the size of which is a lot smaller than that at the voxel level. This suggests that the swapping time could be reduced in many situations by grouping all `VolumeRayCasting()` calls for the same dataset together. The two level data hierarchy of TROVE facilitates the separation of the volume rendering from the rest of the computation. This leads to an alternative approach that is outlined below:

Algorithm 2:

```
1  for each ray R[j] do
2      ordered_list[j] := [];
3      for each volume object O[i] do
4          if R[j] intersects O[i] at t_near and t_far then
5               $\langle V[1], a', b' \rangle = \text{Transform}(\langle O[i], t_{near}, t_{far} \rangle);$ 
6              Insert( $\langle O[i], t_{near}, t_{far}, V[1], a', b', -, - \rangle,$ 
                    ordered_list[j]);
7          endif
8      endfor
9  end for
```

```

10 Sort tuple ids in all ordered_lists according to V into a list;
11 for each dataset V[k] do
12     LoadVolume(V[k]);
13     for each tuple <O[i], t_near, t_far, V[k], a', b', -, -> do
14         VolumeRayCasting(colour, opacity, <V[k], a', b'>);
15         Modify the tuple by adding colour and opacity;
16     endfor
17     UnloadVolume(V[k]);
18 endfor

19 for each ray R[j] do
20     R[j].colour := nil;
21     R[j].opacity := 0;
22     while R[j].opacity < 1 and ordered_list [] do
23         Remove(<-, -, -, -, -, -, colour, opacity>, ordered_list);
24         AccumulateColour(R[j], colour, opacity);
25     endwhile
26     if R[j].opacity != 1.0 then
27         AccumulateColour(R[j], BACKGROUND, 1.0);
28     endif
29 endfor

```

There are three phases in the approach, namely ‘intersection computation (lines 1-9)’, ‘volume rendering (lines 10-18)’ and ‘image composition (lines 19-29)’. During the volume rendering stage, all intersections corresponding to the same dataset (not just the same object) are grouped together and intermediate colour and opacity values are obtained for each interaction by sampling through the dataset. Only one swapping is necessary as far as the whole dataset can fit into the real memory. This reduces drastically the amount of disk swapping experienced with the brute-force algorithm. Even when a dataset cannot fully reside in the real memory due to either the size of the dataset or the competition for space from other running processes, the saving on swapping time is still remarkable. There is no extra cost for intersection computation and image composition. The major trade-off includes the cost of unnecessary voxel-level sampling which could otherwise be avoided through the early termination of some rays, and the extra space required for storing intermediate colour and opacity values.

In order to compare the performance of these two algorithms and to quantify the advantage of data sharing, we have measured the timings for rendering a scene similar to the one in Figure 1 using both algorithms. The testing scene contains 15 objects which are constructed from 1 and 15 volumetric datasets respectively, each with dimensions 81x81x81. The resolution of the final images is 512x512.

The algorithms were run on a DEC Alpha 2100 Server, with a 275 MHz clock and 128M memory, and the results are recorded in Table 1. During the tests there were other computation jobs in the environment, which may affect the timing marginally.

The timings presented are given in seconds and broken down to those for :

- volume rendering, which include ray casting and transformation calculations from world coordinates to local;
- intersection computation, which includes calculations for ray-box intersections and transformations from local coordinates to world coordinates;
- image composition;
- voxel-level data loading and swapping;

- other computation, which includes initialisation, object-level data input, image saving, preprocessing of normals and transformation matrices, and other system time (such as context switch, but excluding voxel-level data swapping).

ALGORITHM	Algorithm 1	Algorithm 2	Algorithm 1	Algorithm 2
No. objects / volumes	15/15	15/15	15/1	15/1
Volume rendering	261.76	241.95	248.03	244.35
Intersection computation	19.30	18.33	18.32	17.83
Image composition	-	1.10	-	1.00
Data loading and swapping	6670.28	2.55	0.28	0.30
Other computation	279.11	127.22	39.68	33.24
Total time	7230.45	391.15	306.31	296.72
Speedup (Alg1/Alg2)	-	18.49	-	1.03

Table 1. Breakdown timings (in seconds) for ray-casting a multi-volume scene with different data composition and using different algorithms.

From Table 1, we can easily observe a significant speed improvement of Algorithm 2 over Algorithm 1 in the case where a scene consists of several volumetric datasets, mainly through the reduction of disk swapping. Data sharing also has an important role to play in reducing the number of volumetric datasets, and subsequently the time used for data loading and swapping. Algorithm 2 also has shown a small improvement over Algorithm 1 in the one-dataset case, due to the reduction of occasional swapping between object-level and voxel-level data which happens to Algorithm 1 only.

4. Conclusions

We have described a modelling scheme, namely TROVE, for building complex scenes composed of multiple volumetric datasets, together with a rendering algorithm that is a combination of traditional ray tracing and volume ray casting. The scheme offers a two-level hierarchical organisation of graphical data, allowing a high degree of data sharing and space reduction. The hierarchy also separates data of different computational nature (i.e. intersection and volume sampling), which facilitates an improved rendering algorithm that reduces a huge amount of disk swapping, leading to the reduction of about 95% rendering time in our example. The results presented throughout the paper also demonstrates the scheme’s capability and flexibility in modelling various solid and amorphous objects and thereby the potential scope of volume graphics.

We are carrying out further research into the introduction of reflective and refractive rays, and the computation of shadows. Work on “*Constructive Volume Geometry*” is also being carried out in a more theoretic manner in order to model objects with overlapped bounding boxes consistently [29]. Although there are still many technical problems yet to be solved before Jim Kajiya’s prediction in 1991, “... in 10 years, all rendering will be volume rendering”, can be realised, we strongly believe that volume-based research is ready to reach beyond the scope of visualisation applications, and to play a much more important role in computer graphics. There are more treasure troves to be found in volume graphics.

References

- [1] Stytz, M. R., Frieder, G. and Frieder, O., "Three-dimensional medical imaging: algorithms and computer systems", *ACM Computing Surveys*, 23(4), 1991 421-499.
- [2] Yagel, R., Kaufman, A. And Zhang, Q., "Realistic volume imaging", *Proceedings of Visualisation 1991*, San Diego, October 1991, 226-231.
- [3] Kaufman, A., Cohen, D. and Yagel, R., "Volume graphics", *IEEE Computer*, 26(7), July 1993, 51-64.
- [4] "Volume graphics", *ONR Volume Visualisation Workshop Reports*, June 1996
- [5] Herman, G. T. and Liu, H. K., "Three Dimensional Display of Human Organs from Computer Tomograms", *Computer Graphics and Image Processing*, 9, January 1982, 1-21.
- [6] Blinn, J., "Light Reflection for the Simulation of Clouds and Dusty Surfaces", *SIGGRAPH Computer Graphics*, 1982, 21-29.
- [7] Kajiya, J. T., "Ray Tracing Volume Densities", *SIGGRAPH Computer Graphics*, 3(1), July 1984, 165-174.
- [8] Drebin, R. A., Carpenter, L. And Hanrahan, P., "Volume Rendering", *SIGGRAPH Computer Graphics*, 22(4), August 1988, 65-74.
- [9] Levoy, M., "Display of Surfaces from Volume Data", *IEEE Computer Graphics and Applications*, 8(3), May 1988, 29-37.
- [10] Sabella, P., "A Rendering Algorithm for Visualising 3D Scalar Fields", *SIGGRAPH Computer Graphics*, 22(4), August 1988, 51-58.
- [11] Hesser, J., Manner, R., Knittel, G., Strasser, W., Pfister, H. and Kaufman, A., "Three Architectures for Volume Rendering", *Computer Graphics Forum*, 3(14), August 1995, 111-122.
- [12] Lacroute, P., and Levoy, M., "Fast Volume Rendering Using a Shear-Warp Factorisation of the Viewing Transformation", *Proceedings SIGGRAPH '94 Conference*, July 1994, 451-457.
- [13] Wilhelms, J. And Van Gelder, A., "Octrees for Faster Isosurface Generation", *ACM Transactions on Graphics*, 11(3), July 1992, 201-227.
- [14] Prakash, C. E. And Manohar, S., "Volume Rendering of Unstructured Grids - A Voxelisation Approach", *Computers and Graphics*, 19(5), September/October 1995, 711-726.
- [15] Lippert, L. and Gross, M. H., "Fast Wavelet Based Volume Rendering by Accumulation of Transparent Texture Maps", *Computer Graphics Forum*, 3(14), August 1995, 431-443.
- [16] Hohne, K.H., Bomans, M., Pommert, A., Riemer, M., Schiers, C., Tiede, U., Wiebecke, G., "3D-Visualisation of Tomographic Volume Data using the Generalised Voxel-Model", *Visual Computer*, 6, January 1990, 28-36.
- [17] Pfister, H. And Kaufman A., "Real-time Architecture for High-resolution Volume Visualisation", *Proceedings of the 8th Eurographics Hardware Workshop*, 12(3), Barcelona, Spain , September 1993, 72-80.
- [18] Wilhelms, J. P., van Gelder, A., Tarantino, P., Gibbs, J., "Hierarchical and Parallelisable

- Direct Volume Rendering for Irregular and Multiple Grids”, *Proceedings of IEEE Visualisation '96*, October/November 1996, 65 -73.
- [19] Yagel, R., Cohen, D. and Kaufman, A., “Discrete ray tracing”, *IEEE Computer Graphics and Applications*, 12(5), (September 1992), 19-28.
- [20] Levoy, M., “Efficient ray tracing of volume data”, *ACM Transaction on Graphics*, 9(3), February 1990, 245-261.
- [21] Blinn, J. F., “A generalization of algebraic surface drawing”, *ACM Transactions on Graphics*, 1(3), July 1982, 235-256.
- [22] Requicha, A. A. G., “Representations for rigid solids: theory, methods, and systems”, *ACM Computing Surveys*, 12(4), December 1980, 437-464.
- [23] Oddy, R. J. and Willis, P., J., “A physically based colour model”, *Computer Graphics Forum*, 10(2), 1991, 121-127 .
- [24] Parrott, R.W., Stytz, M.R., Amburn, P., Robinson, D., “Towards Statistically Optimal Interpolation for 3-D Medical Imaging”, *IEEE Engineering in Medicine and Biology*, 12, September 1993, 49-59.
- [25] Moller, T., Machiraju R., Mueller K. and Yagel R., “Classification and local error estimation of interpolation and derivative filters for volume rendering”, *Proceedings-Symposium on Volume Visualization*, San Francisco, California, October 28-29, 1996, 71-77.
- [26] Van Gelder, A. and Kwansik, K., “Direct volume rendering with shading via three-dimensional textures”, *Proceedings-Symposium on Volume Visualization*, San Francisco, California, October 28-29, 1996, 23-30
- [27] Westover, L., “Footprint evaluation for volume rendering”, *Computer Graphics*, 24(4), August 1990, 367-376.
- [28] Chen, M. and Leu, A., “Parallel multi-volume rendering on distributed memory architectures”, *Proceedings of the First Eurographics Conference on Parallel Graphics and Visualisation*, Bristol, U.K., September 28-29, 1996, 253-257 .
- [29] Chen, M. and Tucker, J. V., “Constructive volume geometry: an algebraic specification”, *CS-Report 25/97*, Department of Computer Science, University of Wales Swansea, 1997.