Volume 19 (2000), number 4 pp. 281–293 COMPUTER GRAPHICS forum

Constructive Volume Geometry

Min Chen and John V. Tucker

Department of Computer Science, University of Wales Swansea, Swansea SA2 8PP, United Kingdom
m.chen@swansea.ac.uk, j.v.tucker@swansea.ac.uk

Abstract

We present an algebraic framework, called Constructive Volume Geometry (CVG), for modelling complex spatial objects using combinational operations. By utilising scalar fields as fundamental building blocks, CVG provides high-level algebraic representations of objects that are defined mathematically or built upon sampled or simulated datasets. It models amorphous phenomena as well as solid objects, and describes the interior as well as the exterior of objects. We also describe a hierarchical representation scheme for CVG, and a direct rendering method with a new approach for consistent sampling. The work has demonstrated the feasibility of combining a variety of graphics data types in a coherent modelling scheme.

Keywords: constructive volume geometry, scalar fields, volume visualisation, volume data types, volume modelling, constructive solid geometry.

1. Introduction

The majority of the existing schemes for modelling graphics objects deal with the geometrical specification of solids and surfaces[1, 2, 3]. Many of them, such as Constructive Solid Geometry (CSG)[4], have a sound theoretical foundation, and are well supported by commercial modelling tools. However, the primary deficiencies of these schemes include their inability to encapsulate the internal description of an object, and the difficulty in maintaining "well-defined" representations.

A scalar field $F(p)$ offers a true 3D representation that defines data for every point p in Euclidean space $\mathbb{E}^3$, and conceptually, it contains more geometrical information than a surface function $G(p) = 0$ that models only those points on a surface. In particular, one can effectively use one or a group of scalar fields to specify the exterior of an object (e.g. blobby model[5]), the interior of an object (e.g. colour volume[6] and solid texture[7]), and an amorphous phenomenon (e.g. clouds[8]).

Another true 3D representation is a *volume dataset*, where an object is no longer treated simply as a solid bounded by a surface but as a mass of discrete points with heterogeneous properties. Such data types are commonly used in medical imaging[9] and scientific visualisation[10], and are obtainable from many digital scanning devices (such as CT, MRI and CCD TV) and computational processes (such as finite element analysis). Two important factors that justify the popularity of these data types are the simplicity of their data structures, and their ability to represent the interior of an object and amorphous phenomena. Mathematically, they are close to the raw data produced by modern scientific instruments and simulations.

The work presented in this paper aims at providing a new approach to the constructive modelling of a variety of graphics objects with a focus on true 3D representations. We formulate an algebraic framework where objects defined by mathematical scalar fields can be combined with objects built from volumetric datasets, using algebraic operations. Thus, we call this algebraic framework *Constructive Volume Geometry (CVG)*.

The CVG methodology is based on designing algebras of spatial objects in the context of computer graphics. A CVG algebra consists of a set of spatial objects and some operations on these objects. By specifying the operations of a CVG algebra and applying them to some spatial objects, more complex spatial objects are created. Syntax for these operations are contained in the signature of the CVG algebra and compositions of operations are denoted by CVG terms over the signature. We will give an

© The Eurographics Association and Blackwell Publishers Ltd 2000. Published by Blackwell Publishers, 108 Cowley Road, Oxford OX4 1JF, UK and 350 Main Street, Malden, MA 02148, USA.

abstract account of the general framework, illustrating it with CVG algebras for several classes of objects.

It is common to consider that CSG for solid modelling is based on the specification of a "volume" of points in 3D space. However, CSG defines only the geometry of an object, and treats all internal points homogeneously. In this paper, we will show that CSG is embedded in one of the simplest CVG classes, and we will highlight the novelty of CVG and its merits over CSG.

Algebraic methods have been applied to the specification of solid objects, CSG operations, texture mapping and so forth[11]. Our CVG methodology is guided by the algebraic theory of data types[12, 13]. In this paper, we have employed and adapted only a few simple algebraic concepts (i.e. signature, algebra and term). The paper is self-contained and reader need not be familiar with the general algebraic theory of data types.

In Section 2, we briefly examine the existing methods relevant to CVG, and describe the problems which motivated this work. We then present the core theory of CVG, in particular the concepts of spatial objects and their composition in Section 3. Example object classes are given to illustrate the concepts and operations. In Section 4, we discuss the implementation of a data representation scheme for CVG, and in Section 5 we describe a direct rendering method that was used to generate the figures in this paper, and address a few technical issues. This is followed by our concluding remarks in Section 6.

2. Background

2.1. Related Work

Geometric modelling plays a key role in computer graphics and computer aided design (CAD). Since interactive computer graphics became viable more than three decades ago, great efforts have been made in this field, resulting in a variety of modelling schemes. Here we use the term *scheme* to denote a specific data type and its associated methods for creating graphics objects.

Many existing modelling schemes were designed to support *solid modelling*, a discipline that has a strong industrial relevance and encompasses a body of theory, techniques and systems focused on "informationally complete" representations of solids[2]. Those which "made waves" include boundary representations (b-reps), constructive solid geometry (CSG), sweeping, parametric surfaces, cell decomposition, spatial occupancy enumeration (SOE), octrees and binary space-partitioning trees[14]. In particular, *Constructive Solid Geometry (CSG)*[4] allows complicated objects to be represented as combinations of various ordered *union*, *intersection* and *difference* operations on simpler solid objects, which may be bounded primitives or halfspaces. It is supported by Boolean algebra and a set of well-understood regularised set operators. Boundary representations and CSG are the most extensively used schemes in computer graphics and CAD. They have a range of versions, and are often used in conjunction with each other and other schemes, leading to many hybrid schemes. Ray tracing has been found to be an efficient way for rendering CSG objects[15].

For the past two decades, we have also witnessed the rapid popularisation of a number of modelling schemes that are capable of representing non-solid objects, or objects whose boundary cannot easily be described explicitly. They include implicit surfaces, volume datasets, particle systems, fractals and grammar-based models. There are also many special-purpose schemes, such as ocean-wave models, cloud models and cloth models, many of which are physically-based[14].

In particular, *implicit surfaces* facilitate the representation of "blobby models"[5] and "soft objects"[16] through scalar fields, and the composition of more complicated objects using combinational field operations. Their elegance lies in the mapping from implicit functions in the real domain to surface-based objects primarily in the binary domain. In addition to their built-in composition capability, the concept of CSG was also applied to implicit surfaces[17]. The recent work on the modelling and rendering of implicit surfaces[18, 19] further demonstrates the capability of mathematical scalar fields realised through complex combinational composition.

With *volume datasets*, an object is represented by a "volume" that is commonly defined by a 3D grid of points, where each grid point is associated with a value. The past decade has witnessed significant advances in *volume visualisation*, driven mainly by applications such as medical imaging and scientific computation. The work in this field has produced a collection of volume rendering methods that enable 3D information in a volumetric dataset to be selectively rendered into 2D images. These methods fall into three main classes, namely surface reconstruction[20, 18], ray casting[21, 22] and forward projection[23, 24]. Previous work on volume modelling[25] include hierarchical volume representation[26, 27], multi-volume modelling and rendering[28], and frequency-domain modelling and rendering[29]. Work has also been carried out on interactive data modification through procedural tools and operators. Examples are volume sculpting[30], interactive sculpting[31], and the haptic interaction model[32]. A noticeable development in recent years is the realisation of CSG with volume representations through voxelisation[33, 34, 35].

These developments suggest that volume-based techniques may have the potential to match and overtake surface-based techniques in computer graphics[36, 37]. We believe that the key to achieving this is a conceptual

© The Eurographics Association and Blackwell Publishers Ltd 2000

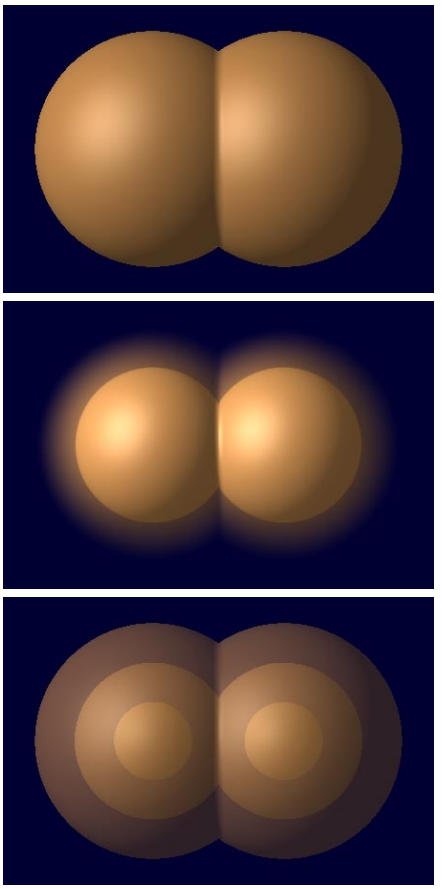


Figure 1: *The union of spatial objects defined by different opacity fields. (a) The union of two spatial objects in the* $\mathbf{O}(\Sigma_{Boolean\text{-}opacity})$ *class. The two opacity fields are centered at* $(0.5, 0, 0)$ *and* $(-0.5, 0, 0)$ *respectively. Both are defined as* $O(p) = 1$ *if the radius* $r \leq 1$*, and* $O(p) = 0$ *otherwise. (b) The union of two spatial objects in the* $\mathbf{O}(\Sigma_{opacity})$ *class. Both opacity fields are defined as:* $O(p) = 1$ *if the radius* $r < 0.6$*,* $O(p) = 0.9 - r$ *if* $0.6 \leq r \leq 0.9$*, and* $O(p) = 0$ *otherwise. The object is rendered as an amorphous phenomenon. (c) The union of two spatial objects in the* $\mathbf{O}(\Sigma_{opacity})$ *class. Both opacity fields are defined as:* $O(p) = 1 - r$ *if the radius* $r \leq 1$*, and* $O(p) = 0$ *otherwise. The image shows iso-surfaces defined by* $O(p) = 0.1$*,* 0.4 *and* 0.7 *respectively.*

unification of volume data types, both discrete and continuous.

2.2. Motivation

Although it would be unnecessary and impractical, and perhaps even impossible, to seek unification of a large number of modelling schemes into one super-scheme, it is desirable to have a general purpose scheme which

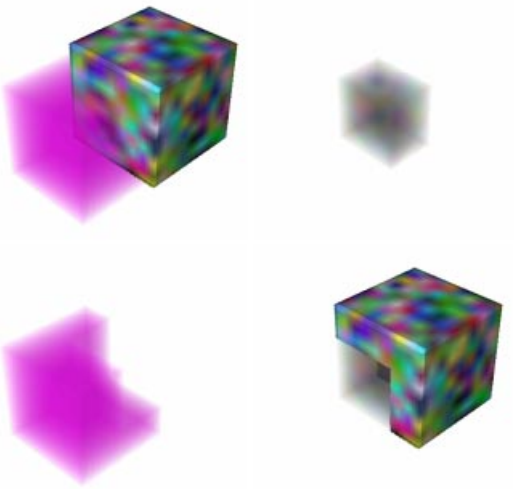


Figure 2: *Applying basic CVG operations to two cubic spatial objects in the* $\mathbf{O}(\Sigma_{4\text{-}colours})$ *class. Clockwise from top left:* $\boxed{\cup}(\mathbf{o}_1, \mathbf{o}_2)$*,* $\boxed{\cap}(\mathbf{o}_1, \mathbf{o}_2)$*,* $\boxed{-}(\mathbf{o}_2, \mathbf{o}_1)$ *and* $\boxed{-}(\mathbf{o}_1, \mathbf{o}_2)$*, where* $\mathbf{o}_1$ *is of opacity* 0.5 *and is associated with a constant colour in a cubic domain, while* $\mathbf{o}_2$*, is of opacity* 1*, and its colour fields are interpolated scalar fields defined upon some random volumetric datasets.*

exhibits a number of important features, such as descriptive power, true 3D representation, constructive geometry, mathematical rigour and finite describability.

The descriptive power of volume datasets is generally recognised. They are true 3D representations, which allow geometrical and physical properties to be defined on every point in a given 3D volume. Most applications employ only very simple data structures in the form of 3D arrays, which facilitate fast hardware and parallel implementations. Despite these features, it must be recognised that the conventional volume-based modelling method is not as sophisticated as those of other schemes. It lacks (i) a high-level method that allows complex objects to be built from simple ones; and (ii) a mature mathematical specification that supports combinational operations on volumetric datasets. Programmers tend to regard volumetric representations as trivial 3D arrays, and manipulate them with "arbitrary" procedural operations.

On the other hand, schemes such as CSG and implicit surfaces have a well-studied mathematical foundation, and are equipped with constructive operations. Despite their important roles in computer graphics and CAD, these schemes are unable to specify internal structures and to model amorphous phenomena. These drawbacks are increasingly becoming noticeable and problematic in modern graphics.

Built upon previous work on CSG, implicit surfaces

© The Eurographics Association and Blackwell Publishers Ltd 2000

and volume visualisation, we propose *Constructive Volume Geometry (CVG)* as a general purpose framework where objects of a true 3D nature can be specified, manipulated, combined, rendered and animated in a consistent manner.

3. Constructive Volume Geometry

In this section, we outline the algebraic framework of Constructive Volume Geometry (CVG), and the general concepts that facilitate combinational operations on objects built upon scalar fields and volumetric datasets.

3.1. Scalar Fields and Spatial Objects

Let $\mathbb{R}$ denote the set of all real numbers, and $\mathbb{E}^3$ denote 3D Euclidean space. A *scalar field* is a function $F : \mathbb{E}^3 \to \mathbb{R}$.

A *spatial object* is a tuple, $\mathbf{o} = (O, A_1, \ldots, A_k)$, of scalar fields defined in $\mathbb{E}^3$, including an *opacity* field $O : \mathbb{E}^3 \to [0,1]$ specifying the "visibility" of every point p in $\mathbb{E}^3$, and possibly other *attribute* fields $A_1, \ldots, A_k : \mathbb{E}^3 \to \mathbb{R}, k \geq 0$.

The opacity field, which specifies the "visible geometry" of a spatial object, is the most elementary field in CVG. Given an opacity field $O : \mathbb{E}^3 \to [0,1]$, a point $p \in \mathbb{E}^3$ is said to be *opaque* if $O(p) = 1$, transparent if $O(p) = 0$, and *translucent* or *semi-transparent* otherwise. Any point which is not transparent is potentially visible to a rendering algorithm.

Attribute fields are used to define other properties of a spatial object, such as colours, reflection coefficients and so on. In addition, we may employ attribute fields for representing non-graphical properties (e.g. a magnetic field or a distance field). When the specification of "invisible" geometry is necessary, an explicit geometry field may also be introduced. For example, to achieve bump mapping, one may use an additional field to influence normal calculation. A field can also be built from one or more other fields using appropriate mapping functions. Without losing generality, we consider only a "flat" tuple representation in this work. This does not, by any means, prohibit the inclusion of vector or tensor fields in CVG for specific applications.

Different applications may define spatial objects with different attributes, and it is necessary to maintain the inter-operability of objects in a consistent manner. We therefore utilise *algebraic signatures*[12, 38] for naming and identifying scalar fields. A *spatial object signature* Σ is essentially a collection of names for space, attributes and scalar fields. We use $\mathbf{O}(\Sigma)$ to denote the set of all spatial objects with signature Σ. We conveniently call $\mathbf{O}(\Sigma)$ a *spatial object class*, or *class* for short. Spatial objects in the same class are thus inter-operable[38].

Note that we do not require a scalar field to be a continuous function. This is consistent with the spatial occupancy of objects in real life, and continuity is not needed in the modelling scheme. However, continuity is an interesting property intimately connected with computability theory. We have also deliberately avoided the introduction of the isosurface concept which has been extensively used in areas such as implicit surface and volume visualisation. This is simply because we are interested in developing a scheme that models objects in their three dimensional entirety, and that is independent of specific rendering algorithms or applications.

3.2. Operations on Spatial Objects

A *CVG algebra* is simply a spatial object class and a family of operations $\Phi_1, \Phi_2, \ldots, \Phi_m$. Given a CVG algebra, we may choose some spatial objects and apply a sequence of operations to create a complex spatial object in the algebra. The composition of such a sequence of operations is represented by an algebraic expression called a *CVG term*. A CVG term over a CVG algebra is recursively defined by:

$$t ::= \mathbf{o}_1 \mid \cdots \mid \mathbf{o}_n \mid \Phi_1(t_1, \ldots, t_{n_1}) \mid \cdots \mid \Phi_m(t_1, \ldots, t_{n_m}),$$

where $\mathbf{o}_i \in \mathbf{O}(\Sigma)$ and the t_j's are CVG terms.

With the definitions in 3.1, operations defined upon spatial objects can be decomposed into simple arithmetic operations on scalars through a series of operational decompositions. Let $\mathbf{o}_1 = (O_1, A_{1,1}, \ldots, A_{1,k})$ and $\mathbf{o}_2 = (O_2, A_{2,1}, \ldots, A_{2,k})$ be two objects of the same signature Σ. For example, we may define a general binary operation

$$\Phi : \mathbf{O}(\Sigma) \times \mathbf{O}(\Sigma) \to \mathbf{O}(\Sigma)$$

as the composition of any set of 2(k+1)-ary operations $\mathbf{G}_0, \mathbf{G}_1, \ldots, \mathbf{G}_k$ on scalar fields by:

$$\begin{aligned}\Phi(\mathbf{o}_1, \mathbf{o}_2) = (&\mathbf{G}_0(O_1, A_{1,1}, \ldots, A_{1,k}; O_2, A_{2,1}, \ldots, A_{2,k}),\\ &\mathbf{G}_1(O_1, A_{1,1}, \ldots, A_{1,k}; O_2, A_{2,1}, \ldots, A_{2,k}),\\ &\vdots\\ &\mathbf{G}_k(O_1, A_{1,1}, \ldots, A_{1,k}; O_2, A_{2,1}, \ldots, A_{2,k})).\end{aligned}$$

In many applications it is adequate to define Φ using only binary operations on the corresponding attribute fields, such as $\mathbf{G}_i(A_{1,i}, A_{2,i})$; and sometimes allowing operations involving opacity fields in addition addition to the corresponding attribute fields, such as $\mathbf{G}_i(O_1, A_{1,i}, O_2, A_{2,i})$.

In the basic version of CVG, each scalar field operation $\mathbf{G}$ is a pointwise extension of a scalar operation $\mathbf{G}$, that is, $\mathbf{G}$ is defined by applying $\mathbf{g}$ to scalars at every point in $\mathbb{E}^3$. Note that the extension is uniquely determined: for operations $\mathbf{g}$ and $\mathbf{h}$ on scalars with their

© The Eurographics Association and Blackwell Publishers Ltd 2000

pointwise extensions $\mathbf{G}$ and $\mathbf{H}$ to operations on scalar fields,

$$\mathbf{g} = \mathbf{h} \Longleftrightarrow \mathbf{G} = \mathbf{H}.$$

Furthermore, standard properties and laws of operations on scalars also hold for the corresponding operations on scalar fields[38].

3.3. Classes

3.3.1. Opacity Only Class

One of the simplest classes in CVG may contain only an opacity field $O : \mathbb{E}^3 \rightarrow [0,1]$ that determines the visibility of $\mathbb{E}^3$. Let Σ_{opacity} be the signature of this class, and let $\boxed{\cup}$, $\boxed{\cap}$ and $\boxed{-}$ be its three basic combinational operations which are defined by:

$$\begin{aligned} \textit{union} &: \boxed{\cup}(\mathbf{o}_1, \mathbf{o}_2) = \mathbf{MAX}(O_1, O_2) \\ \textit{intersection} &: \boxed{\cap}(\mathbf{o}_1, \mathbf{o}_2) = \mathbf{MIN}(O_1, O_2) \\ \textit{difference} &: \boxed{-}(\mathbf{o}_1, \mathbf{o}_2) = \mathbf{SUB}_{01}(O_1, O_2) \end{aligned}$$

where **MAX**, **MIN** and $\mathbf{SUB}_{01}$ are the pointwise extensions[39] of scalar operators **max**, **min** and $\mathbf{sub}_{01}$ defined in Appendix A.

As mentioned in 3.1, the opacity field implicitly defines the "visible geometry" of an object. This geometrical feature becomes more obvious if we substitute the interval [0, 1] by the Boolean domain $\mathbb{B} = \{0,1\} \subset [0,1]$ in $\mathbf{O}(\Sigma_{\text{opacity}})$. In this context, operations $\boxed{\cup}$, $\boxed{\cap}$ and $\boxed{-}$ are essentially operations on point sets defined by the Boolean scalar fields, and they are equivalent to those in CSG (Constructive Solid Geometry)[4]. It has been shown in our study[38] that CSG based on union $\cup$, intersection $\cap$ and difference $-$ is embedded in the corresponding CVG class $\mathbf{O}(\Sigma_{\text{Boolean-opacity}})$ based on $\boxed{\cup}$, $\boxed{\cap}$ and $\boxed{-}$; that is, through an injective homomorphic mapping from a CSG algebra to a CVG algebra, the well known CSG laws can be deduced from the CVG laws in $\mathbf{O}(\Sigma_{\text{Boolean-opacity}})$. In practice, $\mathbf{O}(\Sigma_{\text{Boolean-opacity}})$ possesses the same modelling capability as the CSG Method, and Figure 1(a) shows a union of two spatial objects defined with Boolean scalar fields.

$\mathbf{O}(\Sigma_{\text{opacity}})$, however, exhibits far more descriptive power than $\mathbf{O}(\Sigma_{\text{Boolean-opacity}})$ with its real domain scalar fields as demonstrated in Figures 1(b) and 1(c). CSG is not capable of modelling any amorphous phenomenon such as Figure 1(b), while it would require 6 objects with CSG to define Figure 1(c).

3.3.2. Common Graphics Classes

It is intended for CVG not to fix a set of object classes or a set of operations for each class, but to provide an algebraic framework for defining a variety of operations appropriate to the geometrical, graphical and physical properties defined in a computer graphics problem. Examples of classes for common graphics applications are:

- $\mathbf{O}(\Sigma_{\text{4-colours}})$ — where each spatial object is defined with an opacity field and three colour fields: $O, R, G, B : \mathbb{E}^3 \rightarrow [0,1]$.
- $\mathbf{O}(\Sigma_{\text{visualisation}})$ — which is designed for data visualisation, and has a data field $D : \mathbb{E}^3 \rightarrow \mathbb{R}$ in addition to $O, R, G, B : \mathbb{E}^3 \rightarrow [0,1]$. The opacity and colour fields may be defined by mapping from D.
- $\mathbf{O}(\Sigma_{\text{Phong}})$ — which models the basic object properties used in the Phong illumination model, and it contains an opacity field and four attribute fields defining the coefficients for ambient, diffuse and specular reflections and specular reflection exponent.
- $\mathbf{O}(\Sigma_{\text{7-colours}})$ — which is based on an idealised physical analogy[40], where the colour of an object is separated into the part from pigmented particles, and that from a homogeneous medium. In addition to six colour fields, an additional field defines the opacity of the material and the proportional effect of the two parts of colour.

Consider $\mathbf{O}(\Sigma_{\text{4-colours}})$ as an example. Its underlying concept was originally formulated for describing images[41], and was later adapted in the form of $\mathbf{O}(\Sigma_{\text{visualisation}})$ for volume visualisation[21]. The examples of some commonly used operations in $\mathbf{O}(\Sigma_{\text{4-colours}})$ include:

- *union*: $\boxed{\cup}(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{MAX}(O_1, O_2),$ $\mathbf{SELECT}(O_1, R_1, O_2, R_2), \mathbf{SELECT}(O_1, G_1, O_2, G_2),$ $\mathbf{SELECT}(O_1, B_1, O_2, B_2))$;
- *intersection*: $\boxed{\cap}(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{MIN}(O_1, O_2),$ $\mathbf{SELECT}(O_1, R_1, O_2, R_2), \mathbf{SELECT}(O_1, G_1, O_2, G_2),$ $\mathbf{SELECT}(O_1, B_1, O_2, B_2))$;
- *difference*: $\boxed{-}(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{SUB}_{01}(O_1, O_2), R_1, G_1, B_1)$;
- *blend*: $\boxed{\oplus}(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{ADD}_{01}(O_1, O_2),$ $\mathbf{MIX}_{01}(O_1, R_1, O_2, R_2), \mathbf{MIX}_{01}(O_1, G_1, O_2, G_2),$ $\mathbf{MIX}_{01}(O_1, B_1, O_2, B_2))$;
- *cap*: $\blacksquare(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{CAP}_{01}(O_1, O_2), \mathbf{CAP}_{01}(R_1, R_2),$ $\mathbf{CAP}_{01}(G_1, G_2), \mathbf{CAP}_{01}(B_1, B_2))$;
- *trim*: $\blacksquare(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{TRIM}_{01}(O_1, O_2), \mathbf{TRIM}_{01}(R_1, R_2),$ $\mathbf{TRIM}_{01}(G_1, G_2), \mathbf{TRIM}_{01}(B_1, B_2))$.

The scalar field operations in the above list can easily be derived from scalar operations by pointwise extension as given in Appendix A. Other operations on scalar fields can also be introduced. For example, one may introduce pointwise extensions of arithmetic operators for building implicit surfaces. Figure 2 (see p. 283) shows the results of applying CVG operations to a translucent pink object $\mathbf{o}_1$ and a randomly-coloured solid object $\mathbf{o}_2$. Figure 3 illustrates the effect of combinational operations to the interior of the objects.

© The Eurographics Association and Blackwell Publishers Ltd 2000

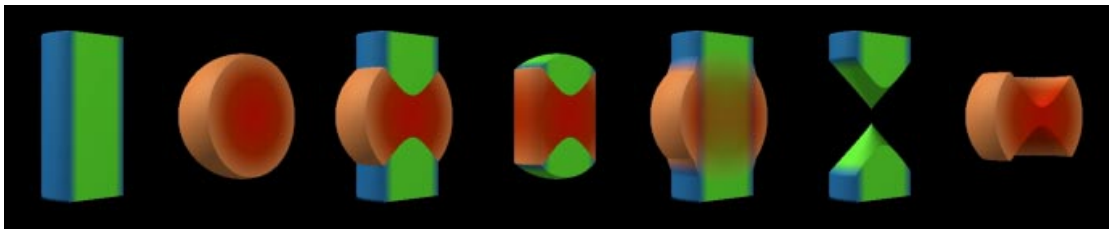


Figure 3: *The effect of applying combinational operations to the interior of objects. From left to right, they are cylinder* $\mathbf{c}$*, sphere* $\mathbf{s}$*,* $\boxed{\cup}(\mathbf{c},\mathbf{s})$*,* $\boxed{\cap}(\mathbf{c},\mathbf{s})$*,* $\boxed{\oplus}(\mathbf{c},\mathbf{s})$*,* $\boxed{-}(\mathbf{c},\mathbf{s})$ *and* $\boxed{-}(\mathbf{s},\mathbf{c})$*. The geometry (i.e. opacity) and colour of both cylinder and sphere are specified using simple scalar fields. For instance, the opacity field of the sphere is defined as* $(1-r)$ *where* r *is the radius. Only the iso-surface* $\tau = 0.1$ *of each object is rendered. Each object is also subtracted by an additional spatial object to reveal its internal structure.*

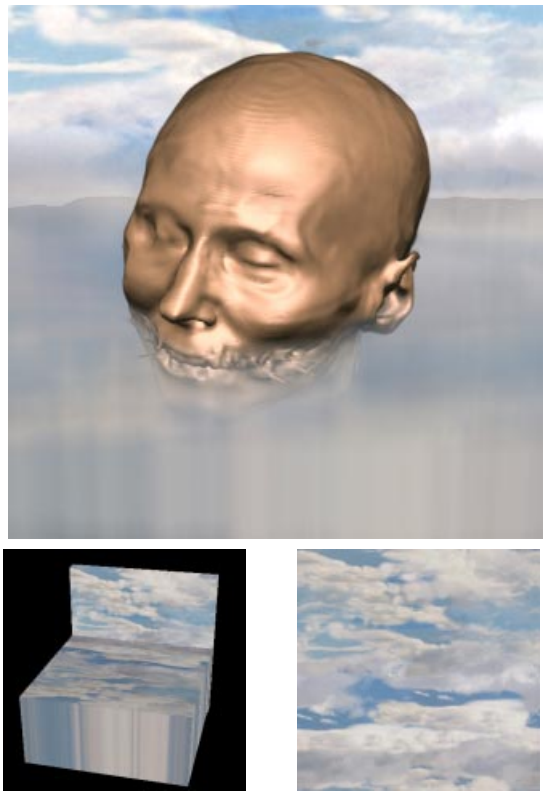


Figure 4: *Man in the sky. This scene is constructed from three interpolated volume objects, namely* **head**, **sky** *and* **clouds**. *The opacity and colour fields of* **head** *are built from a CT dataset (from the University of North Carolina). Both* **sky** *and* **clouds** *are constructed from an image (bottom left) with appropriate transformations (bottom right), and their colour fields are defined using the original RGB colours of the image. Object* **sky** *is placed at the background of the scene and is completely opaque, while* **clouds** *that simulates the clouds surrounding the CT head is modelled by setting its opacity at each voxel in proportion to the red component of the corresponding image pixel.*

With the scalar operations in Appendix A, and other additional ones if necessary, more complicated signatures may be formulated to model the composition of spatial objects based on physics or empirical simulation. For instance, $\mathbf{O}(\Sigma_{\text{visualisation}})$ may be extended to include more data fields for multivariate visualisation, and $\mathbf{O}(\Sigma_{\text{Phong}})$ may be extended to include refraction properties. Our theory of CVG has provided a mathematically sound framework for such extensions, together with a mechanism, to be outlined below, for dealing with fields which cannot be described by simple functions.

3.4. Volume Objects

A scalar field can be defined by any mathematical function in $[\mathbb{E}^3 \to \mathbb{R}]$ as shown above. However, most graphics objects in practice are contained in a finite region in $\mathbb{E}^3$, and many are represented by data specified on a discrete point set. To facilitate constructive operations on such objects, it is necessary to introduce briefly the concepts of volume objects and interpolated volume objects.

A scalar field $F : \mathbb{E}^3 \to \mathbb{R}$ is *bounded* if there exists a bounded set $X \subset \mathbb{E}^3$ such that

$$p \in \mathbb{E}^3 - X \qquad \text{implies} \qquad F(p) = c,$$

where c is a predefined constant in $\mathbb{R}$. We say F is bounded by X. For a bounded opacity field, we choose $c = 0$.

For a spatial object whose opacity and attributes are defined by a finite set P of points, a practical method to define its scalar fields is to derive a bounding set X from P and to interpolate the values over X from the values at the points in P.

Given a finite set $P = \{p_1, p_2, \ldots, p_n \mid p_i \in \mathbb{E}^3\}$ of distinct points, we will call the convex hull $\mathscr{V}(P)$ of the point set P the *volume* of P, and $p_1, p_2, \ldots, p_n$ voxels.

When each voxel p_i is associated with a known scalar value v_i, and the value at every other point in $\mathscr{V}(P)$ can

© The Eurographics Association and Blackwell Publishers Ltd 2000

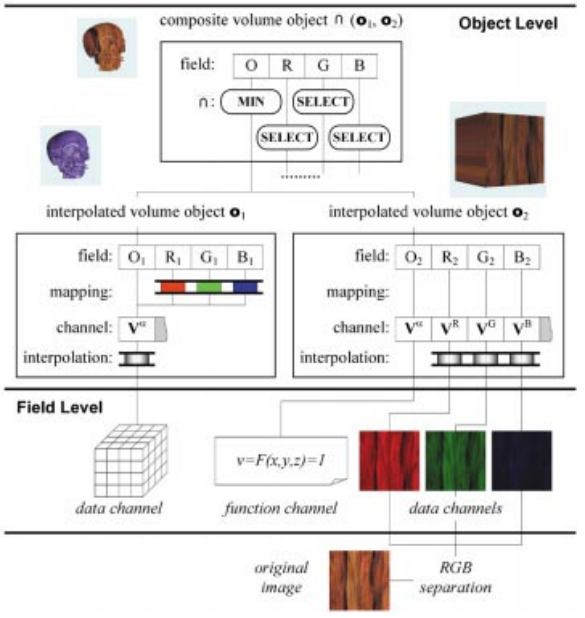


Figure 5: *A data representation scheme for CVG*

Figure 6: *With CVG, discrete and continuous data types can easily be integrated into a single scene.*

be uniquely determined by an interpolation function I defined upon the known scalar values, an *interpolated scalar field* F can be defined in $\mathbb{E}^3$ by:

$$F(p) = \begin{cases} I(p,(p_1,v_1),\ldots,(p_n,v_n)) & p \in \mathscr{V}(P), \\ c & p \notin \mathscr{V}(P). \end{cases}$$

The most typical interpolated scalar field would be

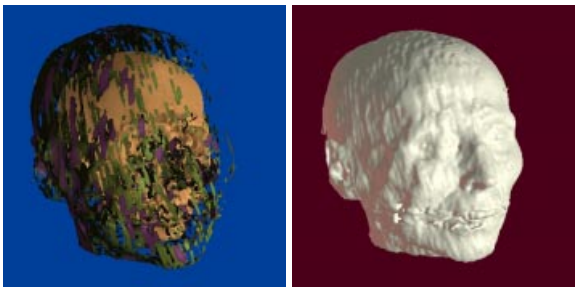


Figure 7: *CVG operations are used to synthesise expressive images with 3D strokes defined by scalar fields.*

Figure 8: *A CVG scene rendered as two isosurfaces, one translucent and one opaque, using direct surface rendering.*

the data obtained through computed tomography (CT), where voxels are organised in the form of a regular 3D grid and each voxel is associated with a density value. Tri-linear interpolation is usually used to determine the unknown values in $\mathscr{V}(P)$, and this, together with the grid of voxels, defines an interpolated scalar field. For a non-regular volume with scattered voxels, 3D Delaunay tetrahedralisation[42] may be applied to $\mathscr{V}(P)$, and unknown values in each tetrahedron are then determined by bary-centric interpolation, which is commonly used in finite element analysis.

A spatial object $\mathbf{o} \in \mathbf{O}(\Sigma)$ is a *volume object* if there is a bounded set $X \subset \mathbb{E}^3$ such that the opacity field of $\mathbf{o}$ is bounded by X. In other words, it is certain that $\mathbf{o}$ is not visible in $\mathbb{E}^3 - X$, though it might also contain "invisible geometry" in X.

© The Eurographics Association and Blackwell Publishers Ltd 2000

An *interpolated volume object* is a volume object with at least one interpolated scalar field. In practice, when a volume object is made of a mixture of mathematical and interpolated scalar fields, it is usually the convex hulls of the interpolated scalar fields which determine the bounded set of the opacity field.

In volume visualisation, for instance, an interpolated scalar field D may be defined upon a CT dataset. By defining a few simple mapping functions, we obtain an opacity field O and three colour fields R, G and B. These fields form a spatial object $\mathbf{o} \in \mathbf{O}(\Sigma_{\text{visualisation}})$ as defined in 3.3. $\mathbf{o}$ is an interpolated volume object, since $\mathbf{o}$ is derived from P as:

$$O(p) = \begin{cases} M(I_D(p,(p_1,v_1),\ldots,(p_n,v_n))) & p \in \mathscr{V}(P), \\ 0 & p \notin \mathscr{V}(P); \end{cases}$$

where M is an opacity mapping function.

We also notice that a 2D image is an interpolated volume object if one associates it with an opacity field bounded by the convex hull of the image. Figure 4 shows an example of applying the CVG operations to volume objects built from images and volumetric datasets. This figure also demonstrates the capability of CVG in modelling both solid and amorphous objects.

4. Specifying CVG Scenes

The previous section presented the basic theoretical concepts for Constructive Volume Geometry (CVG) that enable a complex spatial object to be constructed from a set of simple objects using a CVG term. In this section, we briefly describe the practical considerations for modelling CVG scenes. A detailed discussion on the development of a CVG modelling environment can be found in the authors' recent publication[43].

As with many other algebraic data types in computer science, a CVG term can be represented by a CVG tree, where non-terminal nodes represent CVG operators and terminal nodes represent spatial objects whose scalar fields are either interpolated scalar fields or defined mathematically.

As shown in Figure 5, we organise the data of a CVG tree into two levels, namely the *field level* and *object level*. The field level contains the specification of all scalar fields, including those defined mathematically and those built upon discrete datasets. At the object level, a spatial object is represented by a CVG tree that corresponds to a CVG term. The root represents the final composite volume object. A composite geometrical transformation matrix may be defined at each node, and a set of mapping functions at each terminal node.

To facilitate efficient processing of a CVG term, bounding boxes are also assigned to the spatial objects at the terminal nodes. Such a bounding box is called an *H-box* as it is defined by a hexahedral box with quadrilateral faces. This in effect makes all spatial objects into volume objects. We also pre-process and store the x-, y-, z-extents of each sub-tree as a bounding box (called a *B-box*) at the corresponding node. These two types of boxes enable a substantial reduction in the amount of sampling required during processes such as rendering and voxelisation.

We utilise three coordinate systems for scene specification, and they are:

- *World Coordinate System (WCS)* — a theoretically unbounded 3D domain where all spatial objects in a scene are positioned.
- *Normalised Volume Coordinate System (NVCS)* — a unit cubic domain which is used to standardise the transformation between an H-box (defined in WCS) and a local field coordinate system.
- *Field Coordinate System (FCS)* — a bounded 3D domain $[x_1,x_2]\times[y_1,y_2]\times[z_1,z_2](x_1<x_2, y_1<y_2, z_1<z_2)$ which defines a bounded set as discussed in 3.4. For instance, given a volume dataset, one would typically define the domain as $[0,N_x-1]\times[0,N_y-1]\times[0,N_z-1]$ that corresponds to a specific volumetric dataset, where N_x, N_y and N_z are the dimensions of the dataset.

In our implementation, we have included a set of built-in scalar fields, such as spherical, cylindrical, parabolic and hyperbolic fields, and random volumetric datasets. There are also a range of mapping and interpolation functions which have added an extra degree of flexibility to the specification of CVG scenes. Figure 6 shows a scene composed of spatial objects defined with discrete and continuous data types. In the scene, for example, the stools are defined by scalar fields of one sheet hyperboloids. The head and skulls are interpolated volume objects constructed from the same CT dataset with different opacity mapping. Each head or skull is combined with the corresponding stool through a union operation before colour mapping is applied. An image, as a background, is treated simply as an interpolated volume object. The "planet" is an intersection of the CT dataset and a spherical field, and the latter is also used to define the atmosphere of the "planet".

We have also experimented with the synthesis of expressive images with relatively complex CVG operations. Figure 7 shows two such examples, where the 3D strokes in the image on the left are defined by cylindrical scalar fields randomly placed in a domain under the influence of some fields. The composite object is then intersected with the CT dataset, before it is combined with the skull. For the image on the right we subtract the 3D strokes from the CT dataset using the difference operation to simulate carving actions in sculpting.

© The Eurographics Association and Blackwell Publishers Ltd 2000

The two-level representation of a CVG tree also facilitates a degree of data sharing and separates data to suit different computational processes. For Figure 6, for instance, we need to store the CT dataset only once at the field level as the space requirement for such data is considered to be relatively expensive.

5. Direct Rendering CVG Scenes

Given a CVG scene represented by a CVG term, the goal of a rendering process is to synthesise a 2D image representing a view of the scene. As demonstrated by previous work on voxelising CSG objects[33, 34, 35, 36], it is possible to voxelise a CVG scene into a volume buffer, which can then be rendered using a volume renderer. For some scenes, it may also be possible to extract a set of surface representations from a CVG term, and to synthesise an image using a surface renderer. However, as discussed earlier, it would be more desirable to render a CVG term directly by processing individual components as independently as possible. In this section, we describe our approach to the direct rendering of CVG scenes and discuss a few technical issues in implementation.

5.1. Sampling

Ray marching[44] has been used to evaluate fields in previous work on implicit surfaces[45] and volume rendering[21]. Although the L-G surface technique[46] has become a preferable approach in the area of implicit surfaces, we have adapted the ray marching mechanism for its generality and its ability to deal with both solid objects and amorphous phenomena.

The basic mechanism is to sample at regular intervals along each ray cast from the view position. At each sampling point p, we recursively determine if p is inside the current CVG subtree, until we reach a terminal node. If p is inside the H-box at the node, appropriate opacity and attribute values at p are obtained and returned to the parent node. At each non-terminal node, a CVG operation is applied to the values from child nodes, and so on.

Our implementation supports both *direct volume rendering*[21] and *direct surface rendering*[47]. The former was primarily designed for amorphous objects, but can also work with solid objects if an opacity field can be suitably defined. Backward ray casting, which samples in the direction away from the viewing position, is employed to take the advantage of possible early termination of the ray. Direct surface rendering allows the display of an iso-surface without explicit surface reconstruction. Being able to calculate a surface normal at each ray-surface intersection point, the method avoids the less accurate vertex normal calculations in both Gouraud and Phong shading, resulting in better rendering quality (Figure 8). Our implementation of the direct surface rendering algorithm facilitates the rendering of multiple iso-surfaces.

However, unlike implicit surfaces, a CVG scene is normally defined with multi-valued opacity fields; and unlike the conventional volume visualisation, a CVG scene may consist of many spatial objects with opacity fields defined in a variety of ways. In direct volume rendering, the main difficulty is to maintain a consistent opacity accumulation with different sampling intervals. The standard *volume rendering integral* for determining the visible colour at an arbitrary point t along a ray bounded by $[t_1, \infty)$ is:

$$C_{sum}(t) = \int_{u=t_1}^{u=t} C(u) \cdot \alpha(u) \cdot \beta(u) du$$
$$= \int_{u=t_1}^{u=t} C(u) \cdot \alpha(u) \cdot e^{-\int_{v=t_1}^{v=u} \beta(v) dv} du$$

where $C_{sum}(t)$ defines the accumulated colour at t, $C(u)$ is a colour function, $\alpha(u)$ is a density function and $\beta(u)$ is a transparency function. In practice, this is usually approximated by a discrete sampling process that accumulates $\boldsymbol{c}_{sum}$ and $\boldsymbol{o}_{sum}$ at each sample. The amount of colour $\boldsymbol{c}$ and opacity $\boldsymbol{o}$ to be added into $\boldsymbol{c}_{sum}$ and $\boldsymbol{o}_{sum}$ respectively at each sample thus depends on the sampling distance as well as $C(u)$ and $\alpha(u)$.

To solve this problem, we have introduced two sampling parameters: the standard and actual sampling distances. The *standard sampling distance* Δ specifies a distance such that if samples are taken at regular intervals $1\Delta, \ldots, k\Delta, \ldots$, along a ray, the opacity accumulated is considered to be "correct". In other words, the mapping from $\alpha(u)$ to an opacity has taken Δ into account. To achieve a high quality rendering, we normally use a much smaller interval $\delta < \Delta$. The opacity $\boldsymbol{o}_\delta$ oversampled with δ is corrected by:

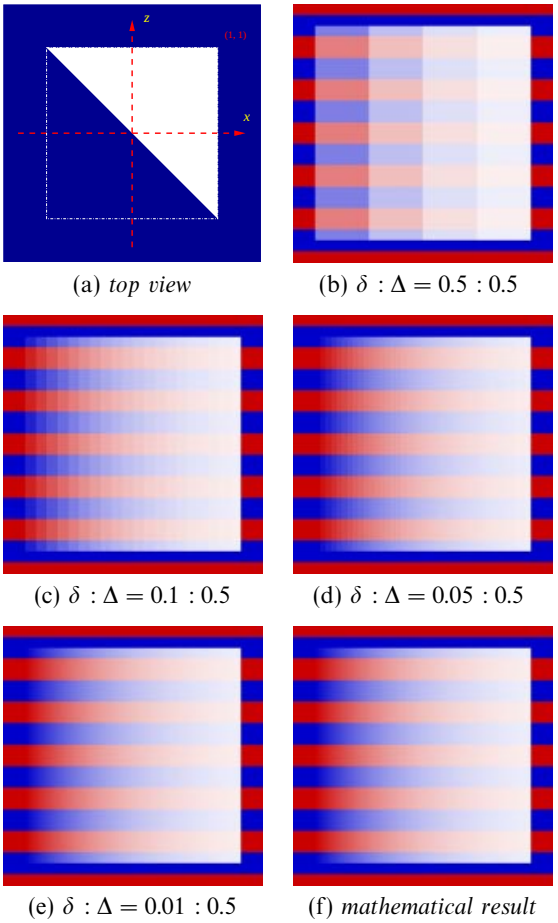
$$\boldsymbol{o}_\delta = 1.0 - (1.0 - \boldsymbol{o})^{(\delta/\Delta)}$$

resulting in $\boldsymbol{o}_\delta$ as the actual opacity to be accumulated. The proof of this correction can be derived using the standard discrete function for backward mapping (for ray casting from the viewer). Figure 9 shows a set of tests that demonstrate the sampling consistency.

Our testing has also shown that the high resolution sampling with a correction is generally more cost-effective than both image- and object-based supersampling. It can also be used for adaptive sampling and space leaping with a dynamic δ set according to the current density and complexity in the scene.

For each CVG tree, we also specify a sampling domain ϵ, which defines a small cubic domain $[x-\epsilon, x+\epsilon] \times [y-\epsilon, y+\epsilon] \times [z-\epsilon, z+\epsilon]$ at each sampling point, and which determines the positions of additional samples for normal computation should there be a "hit".

© The Eurographics Association and Blackwell Publishers Ltd 2000



(a) *top view* (b) $\delta : \Delta = 0.5 : 0.5$

(c) $\delta : \Delta = 0.1 : 0.5$ (d) $\delta : \Delta = 0.05 : 0.5$

(e) $\delta : \Delta = 0.01 : 0.5$ (f) *mathematical result*

Figure 9: *A prism with a uniform density is rendered against a strip background with different sampling distance δ. The opacity **o** obtained at each sampling point (assuming correct for Δ) is set to 0.5. It is not easy for naked eyes to distinguish (f) from (e) and (d).*

5.2. Disjoint Spatial Objects

Theoretically, any CVG scene can be modelled by a single CVG tree where disjoint spatial objects are banded together using $\boxed{\cup}$ operations. Nevertheless, this may lead to a great deal of inefficiency when a scene contains many independent objects.

In CSG, or equally in $\mathbf{O}(\Sigma_{\text{Boolean-opacity}})$, the following is considered to be trivial: given a point p and a CSG object $A \cup B$, "p is in A" implies "p is in $A \cup B$". In other words, if sampling A at p has a hit, there is no need to sample B.

However, this is not the case in CVG classes with real domain opacity fields. For instance, for $\mathbf{o}_1, \mathbf{o}_2 \in \mathbf{O}(\Sigma_{\text{4-colours}})$, given a sampling point p, the $\boxed{\cup}(\mathbf{o}_1, \mathbf{o}_2)$ operation needs to obtain the opacities of both $\mathbf{o}_1$ and $\mathbf{o}_2$ at p in order to determine the new opacity with the **max** operation on scalars. In other words, $\mathbf{o}_2$ has to be sampled no matter whether or not there is a hit at $\mathbf{o}_1$. Thus it is not desirable to apply $\boxed{\cup}$ operations to objects which do not intersect. We provide the user with the following two mechanisms to deal with this problem.

- One mechanism is a "nil" operator ⫽ for combining objects that do not intersect with each other. Given a volume object $⫽(\mathbf{o}_1, \mathbf{o}_2)$ and a sampling point p, we first recursively determine if p is inside the H-box of any terminal node of the subtree $\mathbf{o}_1$. If so, we call this a "hit", even when the resulting opacity at p is 0. The subtree $\mathbf{o}_2$ will be sampled only if there is not a hit at $\mathbf{o}_1$.
- The other mechanism is the specification of multiple CVG trees assuming that they do not intersect. From a modelling point of view, this is equivalent to combining all CVG trees using ⫽ operators, and seems to be a bit redundant. However, from a rendering point of view, each tree is sampled separately. This allows us to use different rendering methods and parameters (such as iso-values) for different CVG trees. For instance, we may render amorphous objects using direct volume rendering, while using direct surface rendering for those objects whose surface features are more important.

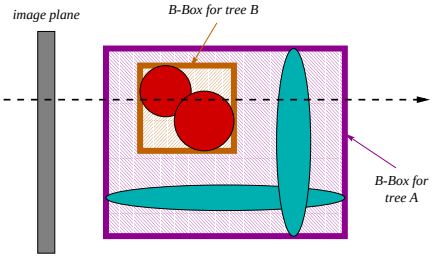


Figure 10: *Ambiguity with multiple CVG trees. If trees are sorted according to the points at which the ray enters the root-level B-boxes, tree A will be sampled first.*

Despite the advantage of using multiple CVG trees, there is a major drawback associated with this method. As illustrated in Figure 10, the determination of which tree to be sampled first may not be straightforward without employing a sophisticated depth-sorting algorithm. Similar to the depth-sorting algorithm for polygons[48], it is generally difficult to provide a satisfactory solution to all ambiguities without further partitioning. With our current implementation, multiple CVG trees can be used safely only if their B-boxes do not overlap. One possible solution is to store all sampled values (opacity and colour) without combining them during the rendering of those CVG trees with intersected B-boxes. The sampled values can then be sorted according to their positions along the ray before being combined together.

© The Eurographics Association and Blackwell Publishers Ltd 2000

6. Conclusions and Future Directions

We have described the core of a graphics modelling scheme, namely *Constructive Volume Geometry (CVG)*, which includes:

- a set of concepts which facilitate the constructive representation of spatial objects defined by mathematical scalar fields as well as discrete volume datasets;
- an algebraic framework that allows a variety of spatial object classes and operations to be formulated for different applications in a consistent manner;
- a hierarchical data representation scheme;
- a recursive rendering algorithm with both surface and volume rendering capabilities, together with a mechanism for consistent sampling of composite opacity fields.

In comparison with the existing constructive modelling methods such as CSG and implicit surfaces, CVG offers the following merits:

- CVG does not limit its operations to geometrical compositions only, and it can also be applied to combine physical properties that are associated with objects.
- With real domain opacity fields, CVG can be used to model and render both solid and amorphous objects.
- It accommodates objects which are defined mathematically by scalar fields as well as those built from sampled or simulated datasets such as medical images and other physical data.
- In classes such as $\mathbf{O}(\Sigma_{\text{4-colours}})$ and $\mathbf{O}(\Sigma_{\text{visualisation}})$, the spatial objects have true 3D "geometry" as well as heterogeneous colour properties. They operate on the interior as well as the exterior of objects, and therefore preserves the main geometrical properties in scalar fields such as volume density and multiple iso-surfaces.
- Images and solid textures can be naturally integrated into a scene as a volume object.

CVG is a general purpose modelling method, and it encompasses the main features of CSG and implicit surfaces while accommodating discrete volumetric datasets. It can be viewed as a major extension to the existing surface- and solid-based modelling schemes by introducing an extra dimension into the interior of objects and removing the necessity for boundary specification. It can also be considered as a high-level modelling scheme for volume visualisation, enabling operations for data manipulation (e.g. geometry clipping and colour mapping) to be consistently defined and implemented.

It is not intended for CVG to have a fixed set of classes or operations. The operations defined in this paper should be treated as templates rather than standards. However, this does not prevent one from defining a set of standard operations for a given class of graphical objects. It is indeed the authors' wish to encourage the standardisation of operations within a well-specified class though that is generally beyond the scope of this paper.

We intend to focus our future research effort on two aspects. In modelling, we would like to conduct a comprehensive study of the algebraic properties of CVG classes and operations, and to investigate the use of such algebraic properties in optimising CVG terms. In rendering, we would like to investigate a spatial partitioning scheme that would provide a satisfactory solution to the ambiguity problem associated in rendering multiple CVG trees.

References

1. A. A. G. Requicha, "Representations for rigid solids: theory, methods and systems", *Computing Surveys*, **12**(4), pp. 437–464 (1980).
2. A. A. G. Requicha and H. B. Voelcker, "Solid modeling: a historical summary and contemporary assessment", *IEEE Computer Graphics and Applications*, **2**(2), pp. 9–24, (1982).
3. A. A. G. Requicha and H. B. Voelcker, "Solid modeling: Current status and research directions", *IEEE Computer Graphics and Applications*, **3**(7), pp. 25–37, (1983).
4. A. A. G. Requicha, *Mathematical Models of Rigid Solids*, Technical Memo 28, Production Automation Project, University of Rochester, Rochester, NY, (1977).
5. J. F. Blinn, "A generalization of algebraic surface drawing", *ACM Transactions on Graphics*, **1**(3), pp. 235–256, (1982).
6. G. Wyvill and P. Sharp, "Volume and surface properties in CSG", in *Proceedings of Computer Graphics International '88: New Trends in Computer Graphics*, pp. 257–266, Springer-Verlag, (1988).
7. K. Perlin, "An image synthesizer", *ACM/SIG GRAPH Computer Graphics*, **19**(3), pp. 287–296, (1985).
8. G. Y. Gardner, "Simulation of natural scenes using textured quadric surfaces", *ACM/SIGGRAPH Computer Graphics*, **18**(3), pp. 11–20, (1984).
9. M. R. Stytz, G. Frieder and O. Frieder, "Three-dimensional medical imaging: algorithms and computer systems", *ACM Computing Surveys*, **23**(4), pp. 421–499, (1991).
10. S. E. Follin, "Scientific visualization", in A. Kent and J. G. Williams (eds), *Encyclopedia of Microcomputers*, **15**, pp. 147–178, Marcel Dekker, New York, (1998).

© The Eurographics Association and Blackwell Publishers Ltd 2000

11. J. C. Torres and B. Clares, "A formal approach to the specification of graphic object functions", *Computer Graphics Forum*, **13**(3) pp. C371–C380, (1994).

12. K. Meinke and J. V. Tucker, "Universal algebra", in S. Abramsky, D. Gabbay and T. S. E. Maibaum (eds), *Handbook of Logic in Computer Science*, Volume I, pp. 189–411, Oxford University Press, (1992).

13. J. Loeckx, H. D. Ehrich and M. Wolf, *Specification of Abstract Data Types*, Wiley/Teubner, (1996).

14. J. D. Foley, A. van Dam, S K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, (1990).

15. S. D. Roth, "Ray casting for modelling solids", *Computer Graphics and Image Processing*, **18**, pp. 109–144, (1982).

16. G. Wyvill, C. McPheeters and B. Wyvill, "Data structures for soft objects", *The Visual Computer*, **2**(4), pp. 227–234, (1986).

17. T. Duff, "Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry", *ACM/SIGGRAPH Computer Graphics*, **26**(2), pp. 131–138, (1992).

18. B. Wyvill, A. Guy and E. Galin, "Extending the CSG tree. warping, blending and Boolean operations in an implicit surface modeling system", *Computer Graphics Forum*, **18**(2), pp. 149–158, (1999).

19. A. Sherstyuk, "Fast ray tracing of implicit surfaces", *Computer Graphics Forum*, **18**(2), pp. 139–148, (1999).

20. W. Lorensen and H. Cline, "Marching cubes: a high resolution 3D surface construction algorithm", *ACM/SIGGRAPH Computer Graphics*, **21**(4), pp. 163–169, (1987).

21. M. Levoy, "Display of surfaces from volume data", *IEEE Computer Graphics and Applications*, **8**(5), pp. 29–37, (1988).

22. P. Sabella, "A rendering algorithm for visualizing 3D scalar fields", *ACM/SIGGRAPH Computer Graphics*, **22**(4), pp. 51–58, (1988).

23. L. Westover, "Footprint evaluation for volume rendering", *ACM/SIGGRAPH Computer Graphics*, **24**(4), pp. 59–64, (1988).

24. J. Wilhelms and A. Van Gelder, "A coherent projection approach for direct volume rendering", *ACM/SIGGRAPH Computer Graphics*, **25**(4), pp. 275–284, (1991).

25. G. Nielson, "Volume modelling", in M. Chen, A. Kaufman and R. Yagel (eds), *Volume Graphics*, pp. 29–48, Springer, London, (2000).

26. M. Levoy, "Efficient ray tracing of volume data", *ACM Transaction on Graphics*, **9**(3), pp. 245–261, (1990).

27. D. Laur and P. Hanrahan, "Hierarchical splatting: a progressive refinement algorithm for volume rendering", *ACM/SIGGRAPH Computer Graphics*, **25**(4), pp. 285–288, (1991).

28. A. Leu and M. Chen, "Modelling and rendering graphics scenes composed of multiple volumetric datasets", *Computer Graphics Forum*, **18**(2), pp. 159–171, (1999).

29. T. Malzbender, "Fourier volume rendering", *ACM Transaction on Graphics*, **12**(3), pp. 233–250, (1993).

30. S. Wang and A. Kaufman. "Volume sculpting", in *Proceedings of Symposium on Interactive 3D Graphics*, pp. 151-156, (April 1995).

31. T. A. Galyean and J. F. Hughes, "Sculpting: an interactive volumetric modeling technique", *ACM/SIGGRAPH Computer Graphics*, **25**(4), pp. 267–274, (1991).

32. R. S. Avia and L. M. Sobierajsk. "A haptic interaction method for volume visualization", in *Proceedings of IEEE Symposium on Volume Visualization*, pp. 197–204, San Francisco, CA, (October 1996).

33. S. M. Wang and A. Kaufman, "Volume sampled voxelization of geometric primitives", in *Proceedings of IEEE Symposium on Volume Visualization*, pp. 78–84, Los Alamos, CA, (October 1993).

34. S. Fang and R. Dai Srinivasan, "Volumetric CSG - a model-based volume visualisation approach", in *Proceedings of the 6th International Conference in Central Europe on Computer Graphics and Visualisation*, pp. 88–95, (1998).

35. D. E. Breen, S. Mauch and R. T. Whitaker, "3D conversion of CSG models models into distance volumes", in *Proceedings of IEEE/ACM Symposium on Volume Visualisation*, pp. 7–14, (1998).

36. M. Chen, A. E. Kaufman and R. Yagel (eds), *Volume Graphics*, Springer, London, (2000).

37. A. Kaufman, D. Cohen and R. Yagel, "Volume graphics", *IEEE Computer*, **26**(7), pp. 51–64, (1993).

38. M. Chen and J. V. Tucker, *Constructive Volume Geometry*, Technical Report, CS-TR-98-19, Department of Computer Science, University of Wales Swansea, (July 1998).

39. A. Ricci, "Constructive geometry for computer graphics", *Computer Journal*, **16**(2), pp. 157–160, (1973).

40. R. J. Oddy and P. J. Willis, "A physically based

© The Eurographics Association and Blackwell Publishers Ltd 2000

colour model", *Computer Graphics Forum*, **10**(2), pp. 121–127, (1991).

41. T. Porter and T. Duff, "Compositing digital images", *ACM/SIGGRAPH Computer Graphics*, **18**(3), pp. 253–259, (1984).
42. G. Nielson, "Tools for triangulations and tetrahedralizations and constructing functions defined over them", in G. Nielson, H. Hagen and H. Muller (eds), *Scientific Visualization: Overviews, Methodologies, Techniques*, pp. 429–525, IEEE Computer Society, (1997).
43. M. Chen, J. V. Tucker and A. Leu, "Constructive representations of volumetric environments", in M. Chen, A. Kaufman and R. Yagel (eds), *Volume Graphics*, pp. 97–117, Springer, London, (2000).
44. H. K. Tuy and L. T. Tuy, "Direct 2D display of 3D objects", *IEEE Computer Graphics and Applications*, **4**(10), pp. 29–33, (1984).
45. K. Perlin and E. M. Hoffert, "Hypertexture", *ACM/SIGGRAPH Computer Graphics*, **23**(3), pp. 253–262, (1989).
46. D. Kalra and A. Barr, "Guaranteed ray intersection with implicit surfaces", *ACM/SIGGRAPH Computer Graphics*, **23**(3), pp. 297–306, (1989).
47. M. W. Jones, *The Visualisation of Regular Three Dimensional Data*, PhD Thesis, University of Wales Swansea, (1995).
48. M. E. Newell, R. G. Newell and T. L. Sancha, "A solution to the hidden surface problem", in *Proceedings of ACM National Conference*, pp. 443–450, (1972).

Appendix A: Scalar Operations

We define the following operators on scalars $\in \mathbb{R}$ with an assumption that both positive and negative infinities are valid numbers:

$$\mathbf{max}(s_1, s_2) = \begin{cases} s_1 & s_1 \geq s_2 \\ s_2 & s_1 < s_2 \end{cases}$$

$$\mathbf{min}(s_1, s_2) = \begin{cases} s_1 & s_1 \leq s_2 \\ s_2 & s_1 > s_2 \end{cases}$$

$$\mathbf{add}(s_1, s_2) = s_1 + s_2$$

$$\mathbf{sub}(s_1, s_2) = s_1 - s_2$$

$$\mathbf{mult}(s_1, s_2) = s_1 \times s_2$$

$$\mathbf{div}(s_1, s_2) = s_1 \div s_2$$

$$\mathbf{select}(s_1, t_1, s_2, t_2) = \begin{cases} t_1 & s_1 \geq s_2 \\ t_2 & s_1 < s_2 \end{cases}$$

$$\mathbf{mix}(s_1, t_1, s_2, t_2) = \begin{cases} \frac{t_1|s_1| + t_2|s_2|}{|s_1| + |s_2|} & |s_1| + |s_2| \neq 0 \\ (t_1 + t_2)0.5 & |s_1| + |s_2| = 0 \end{cases}$$

$$\mathbf{cap}(s_1, s_2) = \begin{cases} s_1 & s_1 \leq s_2 \\ -\infty & s_1 > s_2 \end{cases}$$

$$\mathbf{trim}(s_1, s_2) = \begin{cases} s_1 & s_1 \geq s_2 \\ -\infty & s_1 < s_2 \end{cases}$$

We define the following operators on scalars $\in [0, 1]$:

$$\mathbf{add}_{01}(s_1, s_2) = \mathbf{min}(1, s_1 + s_2)$$

$$\mathbf{sub}_{01}(s_1, s_2) = \mathbf{max}(0, s_1 - s_2)$$

$$\mathbf{div}_{01}(s_1, s_2) = \mathbf{max}(0, \mathbf{min}(1, s_1 \div s_2))$$

$$\mathbf{mix}_{01}(s_1, t_1, s_2, t_2) = \begin{cases} \frac{t_1 s_1 + t_2 s_2}{s_1 + s_2} & s_1 + s_2 \neq 0 \\ (t_1 + t_2)0.5 & s_1 = s_2 = 0 \end{cases}$$

$$\mathbf{cap}_{01}(s_1, s_2) = \begin{cases} s_1 & s_1 \leq s_2 \\ 0 & s_1 > s_2 \end{cases}$$

$$\mathbf{trim}_{01}(s_1, s_2) = \begin{cases} s_1 & s_1 \geq s_2 \\ 0 & s_1 < s_2 \end{cases}$$

© The Eurographics Association and Blackwell Publishers Ltd 2000