CONSTRUCTIVE VOLUME GEOMETRY

Min Chen and John V. Tucker

Department of Computer Science
University of Wales Swansea
Singleton Park, Swansea SA2 8PP
United Kingdom
Email: {m.chen, j.v.tucker}@swansea.ac.uk

Abstract: Having evolved from volume visualisation, volume graphics is emerging as an important sub-field of computer graphics. This paper focuses on a fundamental aspect of volume graphics, the modelling of complex graphics objects and scenes using volume data types. We present an algebraic theory for volume data types, and describe a hierarchical modelling scheme, called *Constructive Volume Geometry (CVG)*, for building complex volume objects. We also outline a recursive algorithm for rendering scenes composed of CVG objects. The work has demonstrated the feasibility of using volume data types in general graphics applications, and the advantages of volume data types over surface/solid data types in some aspects.

Keywords: graphics modelling, volumetric data types, volume visualisation, volume graphics, algebraic methods, constructive volume geometry.

1. INTRODUCTION

The rapid development of computer hardware offers ample scope and encouragement to develop new methods and tools for computer graphics and visualisation. In particular, *volume graphics*, a combination of volume data types and traditional rendering techniques, is emerging as an important sub-field of computer graphics. Volume graphics has the potential to match and overtake surface graphics for the representation, manipulation, rendering and animation of 3D geometric scenes [KAUF93], just as 2D raster graphics has superseded vector graphics in many applications.

The majority of the existing schemes for modelling graphics objects deal with the geometric specification of solids and surfaces [REQU80, REQU82, REQU83]. Many of them, such as Constructive Solid Geometry [REQU77], 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.

Volume data types, where an object is no longer treated simply as a solid bounded by a surface but as a mass of points, can be employed to compensate the weaknesses of traditional solid modelling techniques. The method offers a simple means to describe the geometry and physical properties of a complex object uniformly using a set of scalar fields. Such data types are commonly used in medical imaging [STYT91] and scientific visualisation [UPSO88], and are obtainable from many digital scanning devices (such as CT, MRI and CCD TV) and computational processes (such as finite element analysis). However, programmers tend to regard volumetric representations as trivial 3D arrays, and manipulate them with "arbitrary" procedural operations. Apart from some study on applying octrees to volumetric data [AYAL85, LEVO90] and, more recently, on a scene modelling scheme [LEU98], there is not sufficient research into the methodologies for modelling complex graphics objects and scenes based on volume data types.

The work presented in this paper aims at providing a high-level mathematically well-founded modelling scheme for volume graphics. We formulate some general concepts of volume data types and spatial objects in three dimensional Euclidean space and show how to build complex scenes from similar component objects using algebraic operations. Thus, we propose an algebraic framework and methodology for the specification, representation, manipulation and rendering of volume data types which we call *Constructive Volume Geometry (CVG)*.

The CGV 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. Composing the operations of a CVG algebra and applying them to some spatial objects, creates more complex spatial objects. Notations for these operations are contained in the *signature* of the CVG algebra and compositions of operations are denoted by *CVG terms* over the signature. A range of algebraic techniques can be applied to these CVG terms to accomplish volume graphics tasks (e.g., CVG tree representations, specification of rendering using recursion on CVG terms). We will give a mathematically detailed account of the general framework, illustrating it with CVG algebras for some practical models.

A basic example is a CVG algebra for modelling spatial objects specified by their opacity. We give operations and laws for their application, and show that this generalises Constructive Solid Geometry (CSG). All our CVG algebras are extensions of the basic opacity-channel algebra. Another example is an CVG algebra for modelling spatial objects specified by four colour channels with operations $\boxed{\cup}$, $\boxed{\cap}$ and $\boxed{-}$, called volume union, intersection and difference, respectively. We consider rendering volume objects represented by CVG terms such as $\boxed{-}\,(\boxed{\cup}\,(\mathbf{o}_1, \mathbf{o}_2), \mathbf{o}_3)$.

Our CVG methodology has been guided by the algebraic theory of data types [WIRS91, WECH91 and MEIN92]. In this paper, we have employed and adapted only a few simple algebraic concepts (i.e. signature, algebra, term, tree, equation, homomorphism, isomorphism). The paper is self-contained and reader need not be familiar with the general algebraic theory of data.

In this paper, we will first, in Section Two, briefly examine the existing modelling schemes, and describe the problems which motivated this work. We will then lay down a algebraic foundation of scalar fields in Section Three, upon which the theory of CVG is developed. In Section Four, the core theory of CVG, in particular the algebraic specification of spatial objects and their composition will be described, and two example models will be given to illustrate the concepts and operations. The link between CVG and volume data types will be established in Section Five, leading to a practical solution for building CVG objects through finite data representations. We will address the implementation of CVG in Section Six with discussions on its data representations and rendering. A recursive rendering algorithm is given which is used to generate the figures in this paper. This will be followed by our concluding remarks in Section Seven.

2. BACKGROUND, PROBLEMS AND DESIDERATA

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.

2.1 Solid Modelling

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 solid [REQU82]. Those which "made waves" include:

- *Boundary Representations (b-reps)* [AGOS76, REQU77, REQU85] — A solid object is represented by segmenting its boundary into a finite number of faces. Planar faces are normally further segmented into bounding edges and vertices, while non-planar faces are specified with the support of additional representation schemes for curved surfaces, for example, parametric bicubic patches [FOLE90].
- *Constructive Solid Geometry (CSG)* [REQU77] — This scheme allow complicated objects to be represented as various ordered "union", "intersection" and "difference" of simpler objects, which may be bounded primitives or halfspaces. It is supported by the 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 variations, and are often used in conjunction with each other and other schemes, leading to many hybrid schemes.

- *Sweeping* [BINF71] — The volume of a solid is defined by sweeping a 2D or 3D object along a trajectory through space. The scheme provides users with an intuitive means for constructing a group of objects, and is particularly effective in describing objects that can be specified through translational or rotational sweeping.
- *Cell Decomposition* [AGOS76] — With this scheme, an object is decomposed into a set of solid primitives such as cubes, spheres and tetrahedra. Methodologically, it could be viewed as a restricted form of CSG, where objects are unions of disjoint cells. However, the disjointness may be exploited usefully in computing various integral properties of objects such as the volume of an object. The scheme, and the triangulated form in particular, has been extensively used finite element analysis [WAIT85].
- *Spatial Occupancy Enumeration* [MARC74] — An object under this scheme is essentially a list of spatial cells, which are normally cubes of a fixed size and lie in a fixed spatial grid. The scheme was designed as "coarse" approximations of solids, and was also combined with CSG to form a hybrid scheme [REDD78]. Although it possesses some theoretically desirable properties such as uniqueness and finiteness, the scheme has not been adopted extensively in solid modelling. Nevertheless, it introduced the term "volume element (voxel)" that played a much more important role in volumetric representations.
- *Octrees* [REDD78, MEAG82] — As a 3D analogue of the quadtree, octrees are a hierarchical variant of spatial occupancy enumeration, aiming mainly at storage reduction. With this scheme, each object is bounded by a cubical volume that is to be recursively subdivided into congruent disjoint octants where occupancy is indicated, until an appropriate balance between storage requirements and necessary complexity is reached.
- *Binary Space-Partitioning Trees* [THIB87] — The scheme was originally created for determining visible surfaces in the rendering process, and it builds a binary tree structure usually based on a boundary representation with planar faces.

There are also many hybrid schemes, and algorithms for conversion between different representations [REQU80].

2.2 Non-solid Object Modelling

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* — This scheme facilitates the representation of "blobby models" [BLIN82] and "soft objects" [WYVI86] through implicit surface functions, and the composition of more complicated objects based on the concept of "fields". Its elegance lies in the mapping from implicit functions in the real domain to surface-based objects primarily in the binary domain. In addition to its built-in composition capability, the concept of CSG was also applied to the scheme [DUFF92].
- *Volumetric Representations* — With this scheme, an object is represented by a "volume" that is commonly defined by a regular 3D grid, where each grid point is associated with a value. Although it is conceptually an extension of the spatial occupancy enumeration scheme, the scheme has been popularised through its extensive usage in 3D medial imaging [STYT91]. There are also representations allowing irregular grids or scatter point sets. The underlying concept of this scheme is "scalar fields" [FOLE90]. Two important factors that justify the popularity of the scheme are its simplicity in data structures and its power to represent an object's surface as well as its interior, and thereby to model solids as well as amorphous phenomena.
- *Particle Systems* [REEV83] — A particle system is defined by a collection of particles that evolves over time. The scheme has been used successfully to model objects such as fog, fire, smoke, fireworks and grass.
- *Fractals* [FOUR82, BARN93] — Fractal objects, which exhibit self-similarity, are generated by infinitely recursive processes, or approximated by finite ones. It was computer graphics that brought it into the attention of the world with spectacular images, while equipping itself with a sophisticated tool for modelling certain classes of objects including terrain and images.
- *Grammar-based Models* [SMIT84] — Under this scheme, objects are described by grammars and languages that are capable of representing symbolically parts of objects and their topological and geometrical dependency. The scheme has demonstrated its particular effectiveness through modelling plants.

There are also many special-purpose schemes, such as ocean-wave models [FOUR86], cloud models [GARD84] and cloth models [WEIL86], many of which are physically-based [BARR89].

2.3 Comparison and Discussions

There is a well-defined set of properties used to compare different schemes in solid modelling [REQU80]. Using Requicha's definition and his analysis of boundary representations and CSG, we summarise in Table 1 the properties of the four modelling schemes which are most commonly used and are relevant to this work. In the table, we consider only the basic representation method of each scheme. For example, the properties of CSG are all correct for

representations built from only bounded primitives, but not so if half-spaces are used as primitives. Neither did we include Boolean set operations on boundary representations [REQU85] as such as this would be regarded as a hybrid scheme. For volumetric representations, we adopt the definition where an object is referred to as a whole volume but not the segments within a volume.

Table 1: A summary of the properties of four modelling schemes.

	Boundary Reps	**CSG**	**Implicit Surfaces**	**Volumetric Reps**
Domains — descriptive power:				
object domain: suitable object types,	solids and surfaces	solids	blobby/soft objects	rigid and amor-phous objects
attribute domain: such as colour and opacity which are defined with,	objects or textures	primitives or objects	primitives or objects	voxels
mathematical domain: for defining topology.	bi- or trinary	binary	real	real
Validity — characteristics:				
rigidity: a shape is independent of location and orientation	Yes	Yes	Yes	Yes
orientability and closedness: each surface has distinguishable sides and no dangling portions	Yes	Yes	No	No
spatial finiteness: all objects occupy a finite space	Yes	Yes	No	Yes
finite describability: all objects are represenable in computer	Yes	Yes	Yes	Yes
operational closure: all composite objects posses the same characteristics as their components	N/A	Yes	Yes	N/A
Completeness/unambiguousness — a representation correponds to one single object.	Yes	Yes	Yes	Yes
Uniqueness — an object is uniquely described by its representation.	No	No	No	No
Conciseness — the "size" of representations.	Poor	Good	Good	Poor
Data creation — the ease with which objects may be created:				
through user interface,	Poor	Good	Fair	Poor
through digitisation.	Good	N/A	N/A	Good

Although it would be unnecessary and impractical, and perhaps even impossible, to seek unification of all these schemes into one super-scheme, it is desirable to have a general purpose scheme which exhibits a number of important features, such as *descriptive power*, *finite describability*, *constructive geometry*, *mathematical rigour*, and *rendering efficiency*.

The descriptive power of volume data types 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 simply data structures in the form of 3D arrays, which facilitate fast hardware and parallel implementations. Despite these features, it is still quite common for volumetric data to be converted to surfaces before being

manipulated or rendered [RANJ94]. Many suggested obstacles to using volume data types either result from misconceptions, or can be removed by recent technological developments.

Despite of various merits of using volume data types, it must be recognised that the conventional volume-based modelling method is not as sophisticated as those of other schemes. It lacks a constructive method that allows complex objects being built from simple ones; and a mature mathematical specification that supports combinational operations on volumetric datasets. In order for volume-based research to reach beyond the scope of visualisation applications and to play a more important role in graphics, it is essential to address such weaknesses of volume data types.

3. ALGEBRA FOR SCALAR FIELDS

The fundamental definitions of a *spatial* and *volume* object, and operations on objects, are based on scalar fields in three dimensional space $\mathbb{E}^3$ (i.e. $\mathbb{R}^3$ with the standard Euclidean metric). In this section we summarise properties of scalar fields, and operations on fields and their laws, which prepare the foundation for the general concepts of spatial objects and volume objects in Sections 4 and 5 respectively.

3.1 Scalar Fields.

Let $\mathbb{R}$ denote the set of all real numbers, and $\mathbb{E}^3$ denote 3D Euclidean space.

Definition: A *scalar field* is a function $F: \mathbb{E}^3 \to \mathbb{R}$. A scalar field F is *bounded* if there are $m, n \in \mathbb{R}$, $m \quad n$ such that for all $p \in \mathbb{E}^3$, $m \quad F(p) \quad n$; in this case clearly $F: \mathbb{E}^3 \to [m, n]$. The set of all scalar fields is $[\mathbb{E}^3 \to \mathbb{R}]$, and the set of scalar fields bounded by m, n is $[\mathbb{E}^3 \to [m, n]]$.

Note that we do not require a scalar field to be a continuous function.

In a typical computer graphics or visualisation application, we may use an *opacity field*

$$O: \mathbb{E}^3 \to \mathbb{R}$$

to specify the visibility O(p) of every point in $\mathbb{E}^3$, and other fields

$$A_1, A_2, \ldots A_k: \mathbb{E}^3 \to \mathbb{R}$$

for colours, reflection coefficients, etc., which are referred to as attributes generically.

In the following discussions, we assume scalar fields are bounded by 0 and 1, that is,

$$F: \mathbb{E}^3 \to [0, 1],$$

since for scalar fields this is the most commonly used real domain, and to which all other computer-representable domains can be easily mapped. When scalars are used to represent different attributes, we denote [0, 1] by S, T, etc. to reflect the name of the attributes. Thus the sets of scalar fields are $[\mathbb{E}^3 \to S]$, $[\mathbb{E}^3 \to T]$, etc. The names of scalar fields will be considered in a more formal manner in Section 4.2.

3.2 Operations

We are interested in a number of operations on scalar fields which we will derive from operations on scalars. Let S = [0, 1] be a set of scalars, and $\mathbb{R}^+$ denote the set of non-negative real numbers. The general form of an operation on scalars can be written as a function

$$g: S^u \times \mathbb{R}^{+v} \to S$$

that is, for $s_1, \ldots, s_u \in S$, $r_1, \ldots, r_v \in \mathbb{R}^+$, $u > 0$ and $v \quad 0$,

$$g(s_1, \ldots, s_u, r_1, \ldots, r_v) \in S.$$

We can extend the definition of such an operation to scalar fields by applying the operation to scalars at every point in $\mathbb{E}^3$. Let $[\mathbb{E}^3 \to S]$ be the set of all scalar fields on $\mathbb{E}^3$ with values in S. Given an operation g on scalars, the corresponding operation

$$G: [\mathbb{E}^3 \to S]^u \times \mathbb{R}^{+v} \to [\mathbb{E}^3 \to S]$$

on scalar fields is defined for all $p \in \mathbb{E}^3$,

$$G(F_1, \ldots, F_u, r_1, \ldots, r_v)(p) = g(F_1(p), \ldots, F_u(p), r_1, \ldots, r_v)$$

where $F_1, \ldots, F_u \in [\mathbb{E}^3 \to S]$, $r_1, \ldots, r_v \in \mathbb{R}^+$, $u > 0$ and $v \quad 0$. G is called the *pointwise extension* of g.

Note that the extension is uniquely determined: for operations g and h on scalars with their pointwise extensions G and H to operations on scalar fields,

$$g = h \Rightarrow G = H$$

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

Theorem: Let $g_1, \ldots g_k$ be operations on scalars. Let

$$t(g_1, \ldots g_k) = t'(g_1, \ldots g_k) \text{ in } S$$

be a valid equation where t, t´ are terms that formalise a sequence of compositions of $g_1, \ldots g_k$. Then if $G_1, \ldots G_k$ are the operations on scalar fields induced by $g_1, \ldots g_k$, we have

$$t(G_1, \ldots G_k) = t'(G_1, \ldots G_k) \text{ in } [\mathbb{E}^3 \rightarrow S].$$

This principle can be expressed more precisely and generally, in data type theory [MEIN92]. We will continue to use lower case names for operations on scalars and upper cases for scalar fields. To avoid repetition, we will give only the definitions and laws of scalar operations. A set of corresponding laws for scalar fields can be obtained easily.

3.3 Basic Operators for Scalars and Their Laws

Definitions: Let $s_1, s_2 \in S$, $r \in \mathbb{R}^+$, and we define the following operators on the interval S.

Algebra	scalars
Carriers	S=[0, 1], $\mathbb{R}^+$
Operations	**max**: S×S→S **add**: S×S→S **mult**: $\mathbb{R}^+$×S→S **min**: S×S→S **sub**: S×S→S
Definitions	**max**(s_1, s_2) = maximum(s_1, s_2) **sub**(s_1, s_2) = maximum$(0, s_1 - s_2)$ **min**(s_1, s_2) = minimum(s_1, s_2) **mult**(r, s) =minimum$(1, r \cdot s)$ **add**(s_1, s_2) = minimum$(1, s_1 + s_2)$

where '+', '–' and '·' are the ordinary arithmetic operators, and 0 and 1 are two constants in S.

From these definitions, we can obtain the following set of equational laws that hold:

1. Commutative Laws	$\mathbf{max}(s_1, s_2) = \mathbf{max}(s_2, s_1)$ $\mathbf{min}(s_1, s_2) = \mathbf{min}(s_2, s_1)$	$\mathbf{add}(s_1, s_2) = \mathbf{add}(s_2, s_1)$ $\mathbf{mult}(r, s) = \mathbf{mult}(s, r)$
2. Associative Laws	$\mathbf{max}(s_1, \mathbf{max}(s_2, s_3)) = \mathbf{max}(\mathbf{max}(s_1, s_2), s_3)$ $\mathbf{min}(s_1, \mathbf{min}(s_2, s_3)) = \mathbf{min}(\mathbf{min}(s_1, s_2), s_3)$ $\mathbf{add}(s_1, \mathbf{add}(s_2, s_3)) = \mathbf{add}(\mathbf{add}(s_1, s_2), s_3)$	
3. Distributive Laws	$\mathbf{add}(s_1, \mathbf{max}(s_2, s_3)) = \mathbf{max}(\mathbf{add}(s_1, s_2), \mathbf{add}(s_1, s_3))$ $\mathbf{add}(s_1, \mathbf{min}(s_2, s_3)) = \mathbf{min}(\mathbf{add}(s_1, s_2), \mathbf{add}(s_1, s_3))$ $\mathbf{mult}(r, \mathbf{max}(s_1, s_2)) = \mathbf{max}(\mathbf{mult}(r, s_1), \mathbf{mult}(r, s_2))$ $\mathbf{mult}(r, \mathbf{min}(s_1, s_2)) = \mathbf{min}(\mathbf{mult}(r, s_1), \mathbf{mult}(r, s_2))$ $\mathbf{mult}(r, \mathbf{sub}(s_1, s_2)) = \mathbf{sub}(\mathbf{mult}(r, s_1), \mathbf{mult}(r, s_2))$	
4. Indempotent Laws	$\mathbf{max}(s, s) = s$	$\mathbf{min}(s, s) = s$
5. Identity Laws	$\mathbf{max}(s, 0) = s$ $\mathbf{min}(s, 1) = s$ $\mathbf{add}(s, 0) = s$	$\mathbf{sub}(s, 0) = s$ $\mathbf{mult}(1, s) = s$ $(1 \in \mathbb{R}^+)$
6. Dominance Laws	$\mathbf{max}(s, 1) = 1$ $\mathbf{min}(s, 0) = 0$ $\mathbf{add}(s, 1) = 1$	$\mathbf{sub}(0, s) = 0$ $\mathbf{mult}(0, s) = 0$ $(0 \in \mathbb{R}^+)$
7. Absorption Laws	$\mathbf{max}(s_1, \mathbf{min}(s_1, s_2)) = s_1$ $\mathbf{max}(s_1, \mathbf{sub}(s_1, s_2)) = s_1$	$\mathbf{min}(s_1, \mathbf{max}(s_1, s_2)) = s_1$ $\mathbf{min}(s_1, \mathbf{add}(s_1, s_2)) = s_1$
8. Other Useful Laws	$\mathbf{sub}(s, 1) = 0$ $\mathbf{sub}(s_1, \mathbf{max}(s_2, s_3)) = \mathbf{min}(\mathbf{sub}(s_1, s_2), \mathbf{sub}(s_1, s_3))$ $\mathbf{sub}(s_1, \mathbf{min}(s_2, s_3)) = \mathbf{max}(\mathbf{sub}(s_1, s_2), \mathbf{sub}(s_1, s_3))$ $\mathbf{sub}(s_1, \mathbf{add}(s_2, s_3)) = \mathbf{sub}(\mathbf{sub}(s_1, s_2), s_3)$ $\mathbf{mult}((r_1+r_2), s) = \mathbf{add}(\mathbf{mult}(r_1, s), \mathbf{mult}(r_2, s))$	

We also note that

$\mathbf{mult}((r_1 \cdot r_2), s)$	$\mathbf{mult}(r_1, \mathbf{mult}(r_2, s))$
$\mathbf{mult}(r, \mathbf{add}(s_1, s_2))$	$\mathbf{add}(\mathbf{mult}(r, s_1), \mathbf{mult}(r, s_2))$
$\mathbf{sub}(s_1, \mathbf{sub}(s_2, s_3))$	$\mathbf{add}(\mathbf{sub}(s_1, s_2), s_s)$

We can also derive more complicated operations from these basic operations, such as the average function:

$$\mathbf{avg}(s_1, s_2) = \mathbf{add}(\mathbf{mult}(0.5, s_1), \mathbf{mult}(0.5, s_2)).$$

3.4 Additional Operators for Scalars

Let S and T be two sets of scalars, representing two different types of attributes, on which we have the following operations:

Algebra	two scalars
Carrier	S=[0, 1], T=[0, 1]
Operations	**combine**: S×T×S×T→T **select**: S×T×S×T→T **cap**: S×T×S×T→T
Definitions	$\mathbf{combine}\big((s_1,t_1),(s_2,t_2)\big)=\begin{cases}\dfrac{t_1\cdot s_1+t_2\cdot s_2}{s_1+s_2} & s_1\neq 0,\ \text{or } s_2\neq 0\\ \dfrac{t_1+t_2}{2} & s_1=s_2=0\end{cases}$ $\mathbf{select}\big((s_1,t_1),(s_2,t_2)\big)=\begin{cases}t_1 & s_1\geq s_2\\ t_2 & s_1<s_2\end{cases}$ $\mathbf{cap}\big((s_1,t_1),(s_2,t_2)\big)=\begin{cases}t_1 & s_1>s_2\\ 0 & s_1\leq s_2\end{cases}$

where '+' and '·' are the ordinary arithmetic operators.

4. SPATIAL OBJECTS AND THEIR COMPOSITION

4.1 Spatial Objects

In volume graphics, intuitively an object is considered to be a mass of points, with some properties associated to each point. The specification of these properties depends on the application concerned and unfortunately the rendering program used. We give a general definition of "spatial object" (or object for short), all of whose properties are called *attributes* and one of which is *opacity*.

Definition: A *spatial object* is a tuple

$$o=(O, A_1, \ldots, A_k)$$

of scalar fields defined in $\mathbb{E}^3$, including an opacity field O: $\mathbb{E}^3 \rightarrow \mathbb{R}$ specifying the visibility of every point in $\mathbb{E}^3$ and possibly other attribute fields $A_1, \ldots, A_k$: $\mathbb{E}^3 \rightarrow \mathbb{R}$, k 0.

The opacity field "implicitly" defines the "visible geometry" of the object. Given an opacity field $O: \mathbb{E}^3 \rightarrow [0, 1]$, 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. In order words, the "visible geometry" of the object can be defined in general as

$$\{p \in \mathbb{E}^3 \mid O(p) > 0\}.$$

A spatial object is *fully opaque* if for all $p \in \mathbb{E}^3$, $O(p) = 1$, and is *fully transparent* if for all $p \in \mathbb{E}^3$, $O(p) = 0$. We use the notations ■ and □ for fully opaque and fully transparent if objects respectively.

Note that we have deliberately avoided the introduction of the iso-surface concept [LORE87] which has been extensively used in volume visualisation. This is simply because we are interested in developing a modelling scheme that models objects in their three dimensional entirety, and that is independent from specific rendering algorithms or applications. One may argue the need for specifying points that belong to an object but are completely transparent. We feel to include a separate geometry field which specifies the surface or volume of an object but not its visibility would only introduce unnecessary complexity, because the need for specifying invisible points is very rare in graphics applications, though the above definition does not prevent such a field from being defined as an attribute.

4.2 Signature and Compatibility

Different applications define objects with different attributes. It is necessary for a modelling scheme to maintain the inter-operability of objects in a consistent manner. We therefore introduce syntax for naming and identifying scalar fields.

Definition: A *spatial object signature* is a collection of names for space, attributes and scalar fields; it has the following form:

Spatial object signature	
Space	euclid
Attributes	opacity, $attribute_1$, …, $attribute_k$
Fields	Opacity: euclid → opacity $Attribute_1$: euclid → attribute … $Attribute_k$: euclid → attributek

In this paper, we are assuming that space is interpreted by Euclidean space $\mathbb{E}^3$ and all attributes are interpreted by the interval [0, 1].

We have

$$\mathbf{O}(\ )$$

to denote the set of all spatial objects with signature . Clearly, using the above interpretation we have

$$\mathbf{O}(\ ) = \{(\mathbf{o}_0, \ldots, \mathbf{o}_k) \mid \mathbf{o}_i: \mathbb{E}^3 \rightarrow [0, 1]\ 0 \quad i \quad k\} = [\mathbb{E}^3 \rightarrow [0, 1]]^{k+1}$$

4.3 Operations on Objects

Operations on objects are built from operations on the scalar fields of the objects. Let $\mathbf{o}_i = (O_i, A_{i,1}, \ldots, A_{i,k})$ and $\mathbf{o}_j = (O_j, A_{j,1}, \ldots, A_{j,k})$ be two objects of the same signature . For example, we may define a general binary operation

$$\Phi: \mathbf{O}(\ ) \times \mathbf{O}(\ ) \rightarrow \mathbf{O}(\ )$$

as the composition of any set of 2(k+1)-ary operations $G_0, G_1, \ldots, G_k$ on scalar fields by:

$$\begin{aligned}\Phi(\mathbf{o}_1, \mathbf{o}_2) = (&G_0(O_i, A_{i,1}, \ldots, A_{i,k}; O_j, A_{j,1}, \ldots, A_{j,k}),\\ &G_1(O_i, A_{i,1}, \ldots, A_{i,k}; O_j, A_{j,1}, \ldots, A_{j,k}),\\ &\ldots,\\ &G_k(O_i, A_{i,1}, \ldots, A_{i,k}; O_j, A_{j,1}, \ldots, A_{j,k})).\end{aligned}$$

In most applications it is adequate to define Φ using only binary operations on the corresponding attribute fields as follows

$$\Phi(\mathbf{o}_i, \mathbf{o}_j) = (G_O(O_i, O_j), G_1(A_{i,1}, A_{j,1}), \ldots, G_k(A_{i,k}, A_{j,k}));$$

and sometimes allowing operations involving opacity fields in addition to the corresponding attribute fields as

$$\Phi(\mathbf{o}_i, \mathbf{o}_j) = (G_O(O_i, O_j), G_1(O_i, A_{i,1}, O_j, A_{j,1}), \ldots, G_k(O_i, A_{i,k}, O_j, A_{j,k})).$$

4.4 CVG Algebras

Our Constructive Volume Geometry is founded upon the following concept:

Definition: A *CVG algebra* consists of a set $\mathbf{S}(\,) \subseteq \mathbf{O}(\,)$ of spatial objects with spatial signature together with a collection $\Phi_1, \ldots, \Phi_m$ operations on these spatial objects. We write a CVG algebra in the following way:

CVG algebra	
Objects	$\mathbf{S}(\,)$
Operations	$\Phi_1: \mathbf{S}(\,)^{n_1} \rightarrow \mathbf{S}(\,)$ … $\Phi_m: \mathbf{S}(\,)^{n_m} \rightarrow \mathbf{S}(\,)$

We will shortly meet two spatial signatures and a selection of subsets $\mathbf{S}(\,)$ and operations Φ. Later (in Section 5) we will refine this general concept with computational properties.

4.5 CVG Terms

The purpose of a CVG algebra is to create spatial objects by building them from simpler objects using the operations. For instance, given a CVG algebra with say three binary operations Φ, Ψ and Δ, and spatial objects $\mathbf{o}_1$, $\mathbf{o}_2$, $\mathbf{o}_3$, $\mathbf{o}_4$, $\mathbf{o}_5$, algebraic expressions of the form:

$$\Phi(\Delta(\Phi(\mathbf{o}_1, \mathbf{o}_2), \Psi(\mathbf{o}_3, \mathbf{o}_4)), \mathbf{o}_5)$$

represent new spatial objects made by transforming the given $\mathbf{o}_1$, $\mathbf{o}_2$, $\mathbf{o}_3$, $\mathbf{o}_4$, $\mathbf{o}_5$. Such expressions are (the semantics of) high-level descriptions of spatial objects, and are at the heart of our CVG. We must make these high-level descriptions precise by defining the language of *CVG terms* whose semantics will be spatial objects. The definition of the syntax of the language proceeds in two steps.

Definition: A *CVG signature* Γ is a collection of names for a set of spatial objects and operations on that set. Recalling the general definition of a CVG algebra in 4.4, a CVG signature for that algebra will have the following form:

Spatial object signature	Γ
Spatial objects	so
Operations	ϕ_1: $so^{n_1} \rightarrow so$ … ϕ_m: $so^{n_m} \rightarrow so$

Now we can define the algebraic expressions formally:

Definition: Let $X = \{x_1, x_2, \ldots\}$ be a set of variables for spatial objects of spatial signature . A *CVG term* over the CVG signature Γ and variables X is an expression t that is recursively defined by

$$t ::= x_i \mid \phi_1(t_1, \ldots, t_{n_1}) \mid \ldots \mid \phi_m(t_1, \ldots, t_{n_m})$$

where $x_i \in X$ and t_j are CVG terms. The set of all CVG terms over Γ and X we denote CVG Term (Γ, X).

A CVG term is a syntactic definition of a spatial object. More precisely, there exists semantic mappings

$$[\![\ ]\!]: \text{CVG Term } (\Gamma, X) \rightarrow \mathbf{S}(\ )$$

such that for CVG term t

$$[\![\ t\ ]\!] = \text{spatial object of signature defined by t.}$$

It is easy to define these mappings by recursion on the signature of terms. Given a CVG algebra A with signature Γ (recall 4.4) and $\mathbf{o} = \mathbf{o}_1, \mathbf{o}_2, \ldots \in \mathbf{S}(\ )$

$$[\![\ ]\!]_{\mathbf{o}}: \text{CVG Term } (\Gamma, X) \rightarrow A$$

by

$$[\![x_i]\!]_{\mathbf{o}} = \mathbf{o}_i$$

$$[\![\phi_j(t_1, \ldots, t_{n_j})]\!]_{\mathbf{o}} = \Phi_j([\![t_1]\!]_{\mathbf{o}}, \ldots, [\![t_{n_j}]\!]_{\mathbf{o}})$$

with our standard interpretation of space and attributes.

4.6 Example I: Opacity-Channel Model

One of the simplest models in CVG may contain only an opacity channel that determines the visibility of $\mathbb{E}^3$. We define the signature of this model to be

Spatial Signature	$_{op}$ = opacity channel model
Space	euclid
Attributes	opacity
Fields	Opacity: euclid → opacity

and a set of operations below that allow us to compose objects.

CVG Algebra	opacity channel model
Spatial Objects	$\mathbf{O}(_{op}) = [\mathbb{E}^3 \rightarrow [0, 1]]$
Operations	**union** $\cup$**:** $\mathbf{O}(_{op})\times\mathbf{O}(_{op}) \rightarrow \mathbf{O}(_{op})$
	intersection $\cap$**:** $\mathbf{O}(_{op})\times\mathbf{O}(_{op}) \rightarrow \mathbf{O}(_{op})$
	difference $-$**:** $\mathbf{O}(_{op})\times\mathbf{O}(_{op}) \rightarrow \mathbf{O}(_{op})$
Definitions	$\cup(\mathbf{o}_1, \mathbf{o}_2) = \mathbf{MAX}(O_1, O_2)$
	$\cap(\mathbf{o}_1, \mathbf{o}_2) = \mathbf{MIN}(O_1, O_2)$
	$-(\mathbf{o}_1, \mathbf{o}_2) = \mathbf{SUB}(O_1, O_2)$

Here **MAX**, **MIN** and **SUB** are the pointwise extensions of **max**, **min** and **sub** in 3.3; and O_1, O_2 are opacity fields.

Figure 1 illustrates the effects in an x-y plane with z=0, which result from applying such operations to two spatial objects that are defined respectively by scalar fields

$$O_1(p) = \begin{cases} 1 - r_a & \text{if } r_a = \sqrt{(p_x + 0.5)^2 + p_y^2 + p_z^2} \le 1, \\ 0 & \text{otherwise;} \end{cases}$$

$$O_2(p) = \begin{cases} 1 - r_b & \text{if } r_b = \sqrt{(p_x - 0.5)^2 + p_y^2 + p_z^2} \le 1, \\ 0 & \text{otherwise.} \end{cases}$$

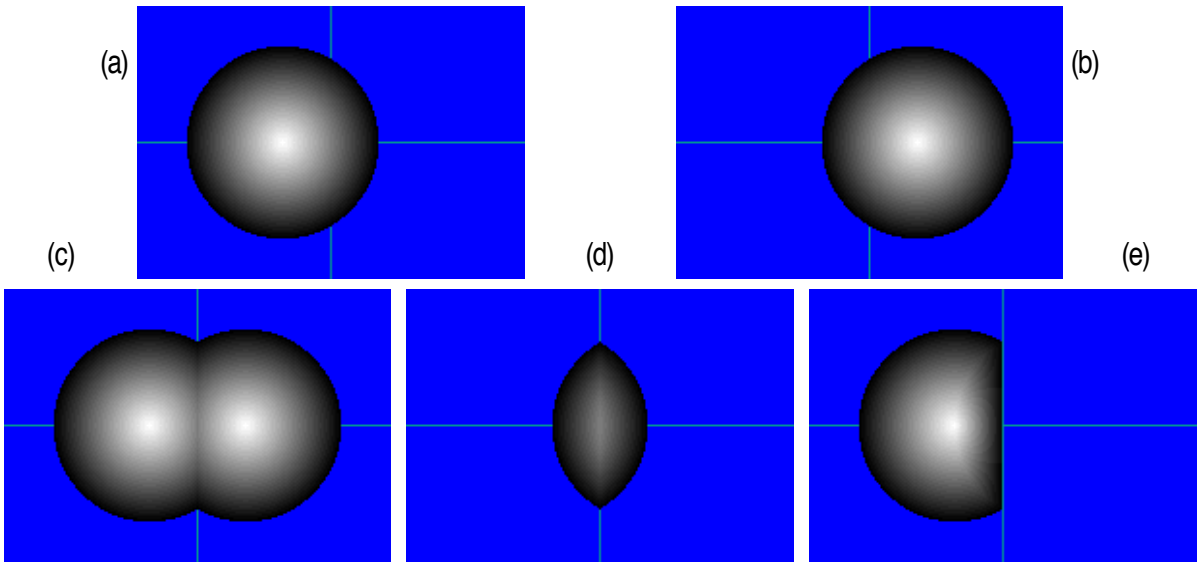


Figure 1. A 2D illustration of operations in the opacity channel model:
(a) $\mathbf{o}_1$, (b) $\mathbf{o}_2$, (c) $\cup$ ($\mathbf{o}_1$, $\mathbf{o}_2$), (d) $\cap$ ($\mathbf{o}_1$, $\mathbf{o}_2$), and (e) $-$ ($\mathbf{o}_1$, $\mathbf{o}_2$),
where the opacity is mapped onto greyscale except that the zero opacity is shown in blue.

As the "geometry" of a spatial object is defined through the visibility of every point in $\mathbb{E}^3$, the traditional concept of "boundary" is no longer as clear cut as in a surface based representation, but can be defined with an iso-surface, $O(p) = \tau$, in an opacity field. Inevitably, an operation on scalar fields, will affect different iso-surfaces differently. Figure 2 shows three iso-surfaces in $\mathbb{E}^3$ with **MAX**(O_1, O_2)=0.1, 0.4 and 0.7 respectively, and iso-surfaces become disjoint when the corresponding iso-values increase.

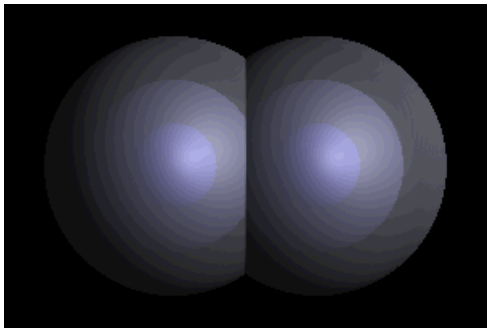


Figure 2. Three iso-surafces in scalar field **MAX**(O_1, O_2) with $O(p)$ = 0.1, 0.4 and 0.7.

Let ■ denote a fully opaque spatial object, and □ a fully transparent one as defined in 4.1. From the laws for the basic operations on scalars (3.3) and the definition of operations on objects (4.3), we can derive a set of laws for $\cup$, $\cap$, and $-$ as follows.

1. Commutative Laws	$\boxed{\cup}(\mathbf{o}_1, \mathbf{o}_2) = \boxed{\cup}(\mathbf{o}_2, \mathbf{o}_1)$ $\quad \boxed{\cap}(\mathbf{o}_1, \mathbf{o}_2) = \boxed{\cap}(\mathbf{o}_2, \mathbf{o}_1)$
2. Associative Laws	$\boxed{\cup}(\mathbf{o}_1, \boxed{\cup}(\mathbf{o}_2, \mathbf{o}_3)) = \boxed{\cup}(\boxed{\cup}(\mathbf{o}_1, \mathbf{o}_2), \mathbf{o}_3)$ $\boxed{\cap}(\mathbf{o}_1, \boxed{\cap}(\mathbf{o}_2, \mathbf{o}_3)) = \boxed{\cap}(\boxed{\cap}(\mathbf{o}_1, \mathbf{o}_2), \mathbf{o}_3)$
3. Indempotent Laws	$\boxed{\cup}(\mathbf{o}, \mathbf{o}) = \mathbf{o}$ $\quad \boxed{\cap}(\mathbf{o}, \mathbf{o}) = \mathbf{o}$
4. Identity Laws	$\boxed{\cup}(\mathbf{o}, \square) = \mathbf{o}$ $\quad \boxed{\cap}(\mathbf{o}, \blacksquare) = \mathbf{o}$ $\quad \boxed{-}(\mathbf{o}, \square) = \mathbf{o}$
5. Dominance Laws	$\boxed{\cup}(\mathbf{o}, \blacksquare) = \blacksquare$ $\quad \boxed{\cap}(\mathbf{o}, \square) = \square$ $\quad \boxed{-}(\square, \mathbf{o}) = \square$
6. Absorption Laws	$\boxed{\cup}(\mathbf{o}_1, \boxed{\cap}(\mathbf{o}_1, \mathbf{o}_2)) = \mathbf{o}_1$ $\quad \boxed{\cap}(\mathbf{o}_1, \boxed{\cup}(\mathbf{o}_1, \mathbf{o}_2)) = \mathbf{o}_1$ $\boxed{\cup}(\mathbf{o}_1, \boxed{-}(\mathbf{o}_1, \mathbf{o}_2)) = \mathbf{o}_1$
7. Other Useful Laws	$\boxed{-}(\mathbf{o}, \blacksquare) = \square$ $\boxed{-}(\mathbf{o}_1, \boxed{\cup}(\mathbf{o}_2, \mathbf{o}_3)) = \boxed{\cap}(\boxed{-}(\mathbf{o}_1, \mathbf{o}_2), \boxed{-}(\mathbf{o}_1, \mathbf{o}_3))$ $\boxed{-}(\mathbf{o}_1, \boxed{\cap}(\mathbf{o}_2, \mathbf{o}_3)) = \boxed{\cup}(\boxed{-}(\mathbf{o}_1, \mathbf{o}_2), \boxed{-}(\mathbf{o}_1, \mathbf{o}_3))$

As mentioned in Section 4.1, the opacity channel implicitly defines the "visible geometry" of an object. The geometrical features become more obvious if we substitute the interval [0, 1] by Boolean domain $\mathbb{B}=\{0, 1\}\subset[0, 1]$ in the opacity model.

Given a Boolean opacity field O: $[\mathbb{E}^3 \rightarrow \mathbb{B}]$, a point $p \in \mathbb{E}^3$ is said to be *inside a surface (or opaque)* if O(p) = 1, and *outside (or transparent)* if O(p) = 0. In this context, operations $\boxed{\cup}$, $\boxed{\cap}$ and $\boxed{-}$ are essentially operations on sets defined by Boolean scalar fields, and they are equivalent to those in CSG (Constructive Solid Geometry) [REQU77].

Theorem: The Constructive Solid Geometry (CSG) based on union $\cup$, intersection $\cap$ and difference – is embedded in the corresponding Boolean Opacity Only Model of CVG based on $\boxed{\cup}$, $\boxed{\cap}$, and $\boxed{-}$.

The precise formulation and proof of this requires an injective mapping ε from a CSG algebra to a CVG algebra such that:

$$\varepsilon(x_a \cup x_b) = \varepsilon(x_a) \boxed{\cup} \varepsilon(x_b),$$

$$\varepsilon(x_a \cap x_b) = \varepsilon(x_a) \boxed{\cap} \varepsilon(x_b),$$

$$\varepsilon(x_a - x_b) = \varepsilon(x_a) \boxed{-} \varepsilon(x_b),$$

where x_a and x_b are two CSG objects. Figure 3 illustrates such a correspondence. Thus, with our chosen operations, we can embed isomorphically a CSG algebra into a CVG algebra. The well known CSG laws for $\cup$, $\cap$ and $-$, can then be deduced from our CVG laws for $\boxed{\cup}$, $\boxed{\cap}$, and $\boxed{-}$. Of course, other operations of CSG (e.g., regularised set operations) would need corresponding additional operations and laws for the CVG algebra.

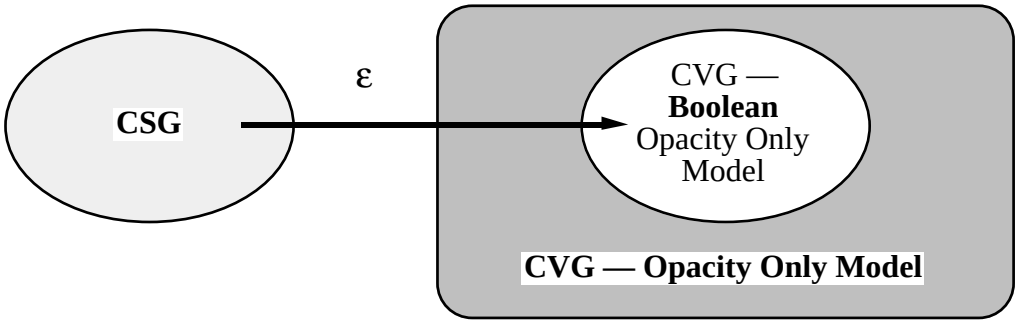


Figure 3. The algebraic mapping from CSG to the Boolean Opacity Only Model.

In practice, the CVG Boolean Opacity Only Model possesses the same modelling capability as the CSG Method. Figure 4 shows CVG objects $\boxed{\cup}$ ($\mathbf{o}_1$, $\mathbf{o}_2$) defined with Boolean scalar fields

$$O_1(p) = \begin{cases} 1 & \text{if } \sqrt{(p_x + 0.5)^2 + p_y^2 + p_z^2} \le 1, \\ 0 & \text{otherwise;} \end{cases}$$

$$O_2(p) = \begin{cases} 1 & \text{if } \sqrt{(p_x - 0.5)^2 + p_y^2 + p_z^2} \le 1, \\ 0 & \text{otherwise.} \end{cases}$$

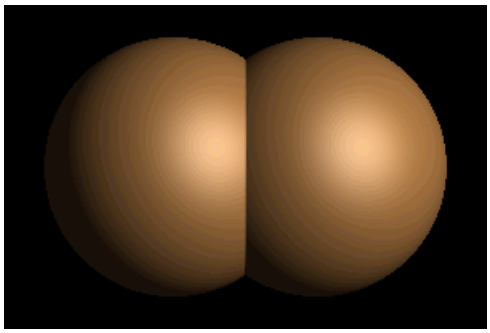


Figure 4. Spatial objects based on Boolean scalar fields.

From the perspective of surface-based modelling, operations defined in the real domain $\mathbb{R}$, or [0, 1] in our case, may seem to be an over-kill. Nevertheless, they are absolutely essential in volume graphics because of the presence of amorphous objects and the necessity for maintaining accuracy in finite volumetric representations. This will become obvious in the following discussions.

4.7 Example II: 4-Colour-Channel Model

Consider a more commonly used 4-colour-channel model proposed by Porter and Duff [PORT84]. It was originally designed to describe an image, and was later adapted for volumetric datasets in visualisation [LEVO88]. In this model, there are three colour channels, red, green and blue, and an alpha channel used to simulate opacity or assist anti-alisasing

techniques. We define the signature of this model as $_{4cc}$ =(opacity, red, green, blue) and a set of operations derived from Porter and Duff's work. These operations are intended for CSG in the domains of scalar fields, but not for colour mixing in terms of particle or fluid materials, which will be discussed briefly in Section 4.8.

Spatial Signature	$_{4cc}$ = 4 colour channel model
Space	euclid
Attributes	opacity, red, green, blue
Fields	Opacity: euclid → opacity Red: euclid → red Green: euclid → green Blue: euclid → blue

CVG Algebra	4 colour channel model
Spatial Objects	$\mathbf{O}(_{4cc}) = [\mathbb{E}^3 \rightarrow [0, 1]]^4$
Operations	**union** $\cup$**:** $\mathbf{O}(_{4cc})\times\mathbf{O}(_{4cc}) \rightarrow \mathbf{O}(_{4cc})$ **intersection** $\cap$**:** $\mathbf{O}(_{4cc})\times\mathbf{O}(_{4cc}) \rightarrow \mathbf{O}(_{4cc})$ **difference** $-$**:** $\mathbf{O}(_{4cc})\times\mathbf{O}(_{4cc}) \rightarrow \mathbf{O}(_{4cc})$
Definitions	$\cup(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{MAX}(O_1,O_2), \mathbf{SELECT}(O_1,R_1,O_2,R_2), \mathbf{SELECT}(O_1,G_1,O_2,G_2), \mathbf{SELECT}(O_1,B_1,O_2,B_2))$ $\cap(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{MIN}(O_1,O_2), , \mathbf{SELECT}(O_1,R_1,O_2,R_2), \mathbf{SELECT}(O_1,G_1,O_2,G_2), \mathbf{SELECT}(O_1,B_1,O_2,B_2))$ $-(\mathbf{o}_1, \mathbf{o}_2) = (\mathbf{SUB}(O_1,O_2), R_1, G_1, B_1)$

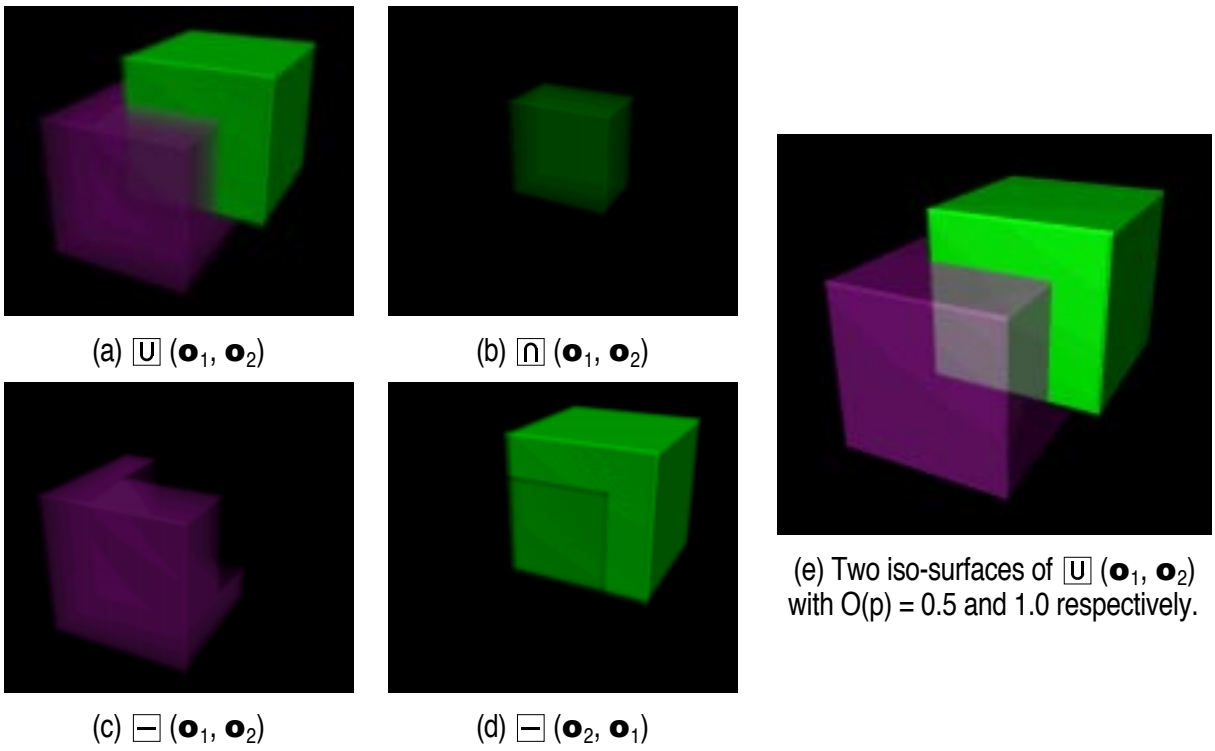
where **MAX**, **SELECT** and **SUB** are the pointwise extensions of **max**, **select** and **sub** in 3.3 and 3.4. Note that some of the laws given in 4.6 will not be applicable to $_{4cc}$, because of the **SELECT** operation used for colour fields. However, singularity conditions normally appears at only points where two spatial objects have the same opacity.

We may also define a scaling operation $\bullet$**:** $\mathbb{R}^{+4}\times V \rightarrow V$ as

$$\bullet((r_O,r_R,r_G,r_B), \mathbf{o}) = (\mathbf{MULT}(r_O,O),\mathbf{MULT}(r_R,R),\mathbf{MULT}(r_G,G),\mathbf{MULT}(r_B,B)).$$

Figure 5 shows the basic operations, $\cup$, $\cap$ and $-$, on two spatial objects. Spatial object $\mathbf{o}_1$ is defined with a cubic region within which {O, R, G, B} = {0.5, 1, 0, 1}, while $\mathbf{o}_2$ is defined with a different cubic region, within which {O, R, G, B} = {1, 0.5, 1, 0.5}. As $\mathbf{o}_2$ has a stronger opacity field than $\mathbf{o}_1$, one can clearly notice that the resulting objects are non-

symmetric. For example, in ⊔ ($\mathbf{o}_1$, $\mathbf{o}_2$), $\mathbf{o}_2$ appears to have penetrated into $\mathbf{o}_1$ but not vice versa. With difference operations, part of $\mathbf{o}_1$ has been segmented off by $\mathbf{o}_2$ through ⊟ ($\mathbf{o}_1$, $\mathbf{o}_2$), while $\mathbf{o}_1$ has only managed to reduce the opacity of part of $\mathbf{o}_2$ with ⊟ ($\mathbf{o}_2$, $\mathbf{o}_1$).



(a) ⊔ ($\mathbf{o}_1$, $\mathbf{o}_2$) (b) ⊓ ($\mathbf{o}_1$, $\mathbf{o}_2$)

(c) ⊟ ($\mathbf{o}_1$, $\mathbf{o}_2$) (d) ⊟ ($\mathbf{o}_2$, $\mathbf{o}_1$)

(e) Two iso-surfaces of ⊔ ($\mathbf{o}_1$, $\mathbf{o}_2$) with O(p) = 0.5 and 1.0 respectively.

Figure 5: Examples of basic CVG operations for $\sum_{4cc}$.

Similar to CSG, complex spatial objects can be constructed from elementary spatial objects such as cubes, spheres and so on. Figure 6 shows three simple spatial objects, a cube, a sphere and a cylinder, and two composite objects generated using CVG operations. The three simple spatial objects in Figure 6(a) are defined with the following field functions:

Green cube	$\{\mathrm{O,\ R,\ G,\ B}\}(\mathrm{p}) = \begin{cases} \{1, 0.4, 1, 0.4\} & \text{if } -1 \le p_x, p_y, p_z \le 1, \\ \{0, 0, 0, 0\} & \text{otherwise.} \end{cases}$
Peach sphere	$\{\mathrm{O,\ R,\ G,\ B}\}(\mathrm{p}) = \begin{cases} \{1 - r_s, 1, 0.7, 0.4\} & \text{if } r_s = \sqrt{p_x^2 + p_y^2 + p_z^2} \le 1, \\ \{0, 0, 0, 0\} & \text{otherwise.} \end{cases}$
Blue cylinder	$\{\mathrm{O,\ R,\ G,\ B}\}(\mathrm{p}) = \begin{cases} \{1 - r_c, 0.3, 0.3, 1\} & \text{if } r_c = \sqrt{p_x^2 + p_y^2} \le 1, -1 \le p_z \le 1 \\ \{0, 0, 0, 0\} & \text{otherwise.} \end{cases}$

To compose the spatial object shown in Figure 6(b), we first apply appropriate geometrical transformations, including scaling and translation in this case, to the simple objects, resulting in a flat rectangular slab **r**, a sphere **s** and two cylinders $\mathbf{c}_1$, $\mathbf{c}_2$. These spatial objects are then integrated together through a CVG term, $\boxminus$ ($\boxcup$ (**r**, **s**), $\boxcup$ ($\mathbf{c}_1$, $\mathbf{c}_2$)). Similarly, the composite object shown in Figure 6(c) is constructed using $\boxcup$ ($\boxminus$ ($\boxminus$ ($\mathbf{c}_1$, $\mathbf{c}_2$), **r**), **s**) with appropriate transformations.

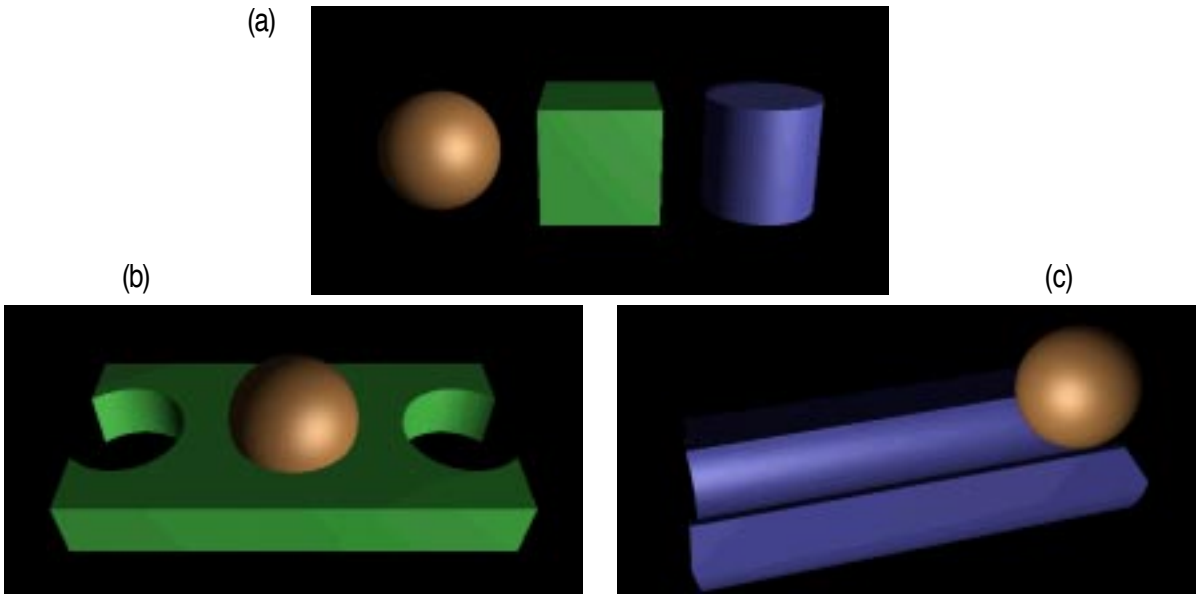


Figure 6. (a) Three simple spatial objects, **s**, **r**, **c**, and their compositions with (b) $\boxminus$ ($\boxcup$ (**r**, **s**), $\boxcup$ ($\mathbf{c}_1$, $\mathbf{c}_2$)), and (c) $\boxcup$ ($\boxminus$ ($\boxminus$ ($\mathbf{c}_1$, $\mathbf{c}_2$), **r**), **s**)

Unlike CSG objects that are normally modelled with boundary representations, spatial objects in $_{4CC}$ has true 3D "geometry" as well as 3D colour properties. The colour properties of a spatial object are manipulated in the same way as its opacity (i.e. geometry), though with a different field operation. By replacing the constant colour fields of simple objects with some field functions, CVG can be used to describe effectively the internal structures of objects, and subsequently to compose new objects in a more sophisticated manner. Figure 7 show such an example, where the colour fields of the sphere and cylinder are defined as:

Peach sphere	$\{R,\ G,\ B\}(p) = \begin{cases} \{0.3r_s + 0.7,\ 0.4r_s,\ 0.4r_s\} & \text{if } r_s = \sqrt{p_x^2 + p_y^2 + p_z^2} \le 1, \\ \{0,\ 0,\ 0\} & \text{otherwise.} \end{cases}$
Blue cylinder	$\{R,\ G,\ B\}(p) = \begin{cases} \{0.7r_c + 0.3,\ 0.7r_c + 0.3,\ r_c\} & \text{if } r_c = \sqrt{p_x^2 + p_y^2} \le 1, -1 \le p_z \le 1 \\ \{0,\ 0,\ 0\} & \text{otherwise.} \end{cases}$

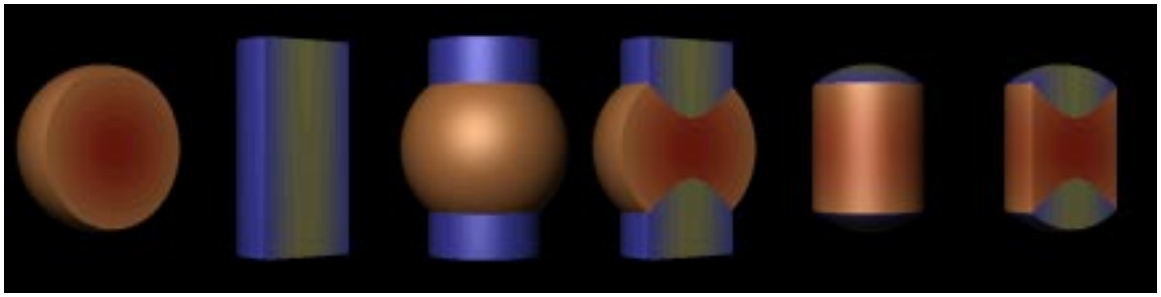


Figure 7: Isosurfaces of spatial objects with varying internal structures, from left to right:
$\boxminus(\mathbf{s}, \mathbf{r})$, $\boxminus(\mathbf{c}, \mathbf{r})$, $\boxcup(\mathbf{c}, \mathbf{s})$, $\boxminus(\boxcup(\mathbf{c}, \mathbf{s}), \mathbf{r})$, $\boxcap(\mathbf{c}, \mathbf{s})$, $\boxminus(\boxcap(\mathbf{c}, \mathbf{s}), \mathbf{r})$.

4.8 Other Possible Models

There are normally many other attributes associated with a graphics object, including coefficients for ambient, diffuse and specular reflections and specular reflection exponent. For those figures in 4.7, we used the **SELEC** operation, which is the pointwise extension of **select** in 3.4, for such attributes in a way similar to colour fields. There are also more complicated colour mixing operations suitable for particle systems and fluid materials. For example, Oddy and Willis introduced a 7-colour-channel model [ODDY91] based on an idealised physical analogy, where the colour of an object is separated into the part from pigmented *particles*, and that from a homogeneous *medium*. An additional channel defines the opacity of the material and the proportional effect of the two parts of colour.

With the scalar operations in 3.3 and 3.4 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. The 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.

5. VOLUMETRIC OBJECTS

We have so far considered the concepts of "scalar fields" and "spatial objects" in the abstract, without looking at the practical details of how they would be defined or computed. Of course, a scalar field can be defined by any function in $[\mathbb{E}^3 \to \mathbb{R}]$ as shown in 4.6 and 4.7. In the context of volume graphics, however, this is not the main objective of CVG proposed here. As mentioned in Section 2, this work concerns a modelling scheme that

(a) facilities finite representations of graphics objects,

(b) is suitable for geometrical manipulation and computation, and

(c) can be efficiently implemented and rendered.

Conventional volumetric representations satisfy (a) and (c) quite well, but can not be directly employed in constructing complex object. By transforming volume data types to spatial objects based on scalar fields, we shall be able to apply the algebraic operations defined above to volume data types indirectly. In this section, we will give a method which allow a spatial object to be derived from its original volumetric dataset.

First we consider a general boundedness property of spatial objects with aims (a) ~ (c) in mind. Then we define a practically useful class of spatial objects defined by interpolation methods on convex subsets of $\mathbb{E}^3$.

5.1 Volume Objects

We are interested in spatial objects whose "visible geometry" are contained within a finite region of $\mathbb{E}^3$.

A scalar field F: $\mathbb{E}^3 \to [0, 1]$ is *bounded* (respectively, *compact*) if there exists a bounded set $X \subseteq \mathbb{E}^3$ (respectively, compact) such that

$$x \in \mathbb{E}^3 - X \qquad \text{implies} \qquad F(x) = 0.$$

We say F is bounded by X.

Definition A spatial object $\mathbf{o} \in \mathbf{O}(\ )$ is a *volume object* if there is a bounded set X such that each scalar field of $\mathbf{o}$ is bounded by X. We denote $\mathbf{V}(\ )$ to the set of all volume objects of spatial signature .

Returning to our concept of a CVG algebra in 4.4, we note that the set $\mathbf{S}(\ )$ of spatial objects of interest are likely to be volume objects, i.e.

$$\mathbf{S}(\ ) \subseteq \mathbf{V}(\ ) \subseteq \mathbf{O}(\ )$$

and the operations $\Phi_1, \ldots, \Phi_m$ are likely to preserve boundedness. Clearly, the operations of $_{op}$ and $_{4cc}$ (in 4.6, 4.7) preserve boundedness.

5.2 Volumetric Scalar Fields

A practical method of creating a volume object is to derive a bounding set X from a finite set P of points and to derive scalar fields by interpolating values over X from 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 $\mathcal{Vol}(P)$ 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 $\mathcal{Vol}(P)$ can be uniquely determined by an interpolation function I defined upon the known scalar values, a *volumetric scalar field* F can be defined in $\mathbb{E}^3$ by

$$F(p) = \begin{cases} I\big(p,(p_1,v_1),\ldots,(p_n,v_n)\big) & p \in \mathcal{Vol}(P) \\ 0 & p \notin \mathcal{Vol}(P) \end{cases}$$

Such a field is defined by a tuple

$$(I, (p_1, v_1), \ldots (p_n, v_n)).$$

The most typical volumetric scalar field would be 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 $\mathcal{Vol}(P)$, and this, together with the grid of voxels, defines a volumetric scalar field. For a non-regular volume with scattered voxels, 3D Delaunay triangulation [HOPP92] may be applied to $\mathcal{Vol}(P)$, and unknown values in each tetrahedon are then determined by either tri-linear interpolation, or bary-centric interpolation. The latter method is commonly used in finite element anaysis, and is defined as follows.

Given an arbitrary tetrahedron with four vertices, p_1, p_2, p_3, p_4, whose values are known as v_1, v_2, v_3, v_4, the value v of any point p in the tetrahedron can be uniquely determined by:

$$v = \sum_{i=1}^{4} w_i v_i = \sum_{i=1}^{4} \frac{\nabla_i}{\nabla} v_i$$

where w_i is the i^{th} bary-centric coordinate of p, ∇ is the volume of the whole tetrahedron, and ∇_i is that of the sub-tetrahedron defined by p and the three tetrahedral vertices other than p_i.

5.3 Convex Volume Objects

Definition: A *convex volume object* based on an interpolation method I is an object that consists of a finite set of volumetric scalar fields all of which are defined upon the same $\mathcal{V}_{\mathcal{O}}(P)$ by the same interpolation method.

A convex volume object of signature based on method I, is finitely represented by a set of pairs of voxels and values of the form

$$\{ (p_i, v_{i,j}) \mid 1 \quad i \quad n, 0 \; j \; k \}$$

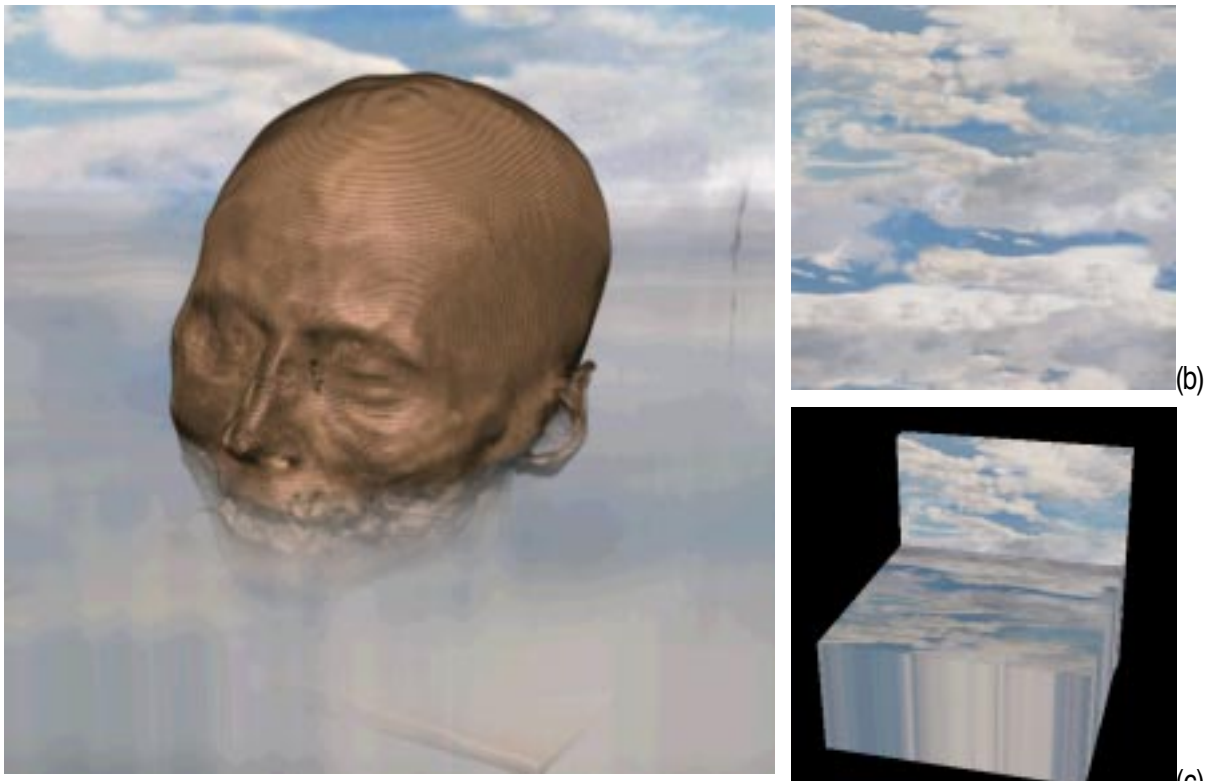
where $|P| = n$ and has k+1 scalar fields.

In volume visualisation [STYT91], for instance, a volumetric scalar field may be defined upon a CT dataset. By defining a few simple mapping functions, we obtain an opacity field and three colour fields, which form a spatial object $\mathbf{o} \in \mathbf{O}(_{4cc})$ as defined in 4.7. Since all scalar fields in $\mathbf{o}$ are derived from the same volumetric scalar field, they share a common $\mathcal{V}_{\mathcal{O}}(P)$. Hence $\mathbf{o}$ is a convex volume object.

We also notice that a 2D image is a convex volume object if one associates it with an opacity field. Figure 9 shows a scene constructed with three convex volume objects, namely **head**, **sky** and **clouds**. The opacity and colour fields of **head** are built from a CT dataset (from University of North Carolina) and colour fields are defined with appropriate mapping functions upon the opacity field. Both **sky** and **clouds** are constructed from an image (Figure 9(b)) with appropriate transformations (Figure 9(c)), and their colour fields are defined using the 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 brightness of the corresponding image pixel. This figure also demonstrates the capability of CVG in modelling both solid and amorphous objects. The CVG term for this scene is simply the union of three objects $\boxed{U}$ ($\boxed{U}$ (**sky**, **clouds**), **head**).

Let $\mathbf{C}(\,)$ be a class of all convex volume objects of signature based on method I. Given $\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n \in \mathbf{C}(\,)$, we can construct a composite object $\mathbf{o} \in \mathbf{V}(\,)$ by applying a finite number of CVG operations to $\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_n$. Clearly the operations in the models in 4.6 and 4.7 preserve boundedness, but not necessarily the convexity. Consider a CVG algebra with

signature Γ for **C**( ) as discussed in 4.5. The operations of Γ applied to **C**( ) generate a sub-algebra <**C**( )>$_\Gamma$ of **V**( ), and this is illustrated in Figure 10.



(a) (b) (c)

Figure 9. Man in the Sky.

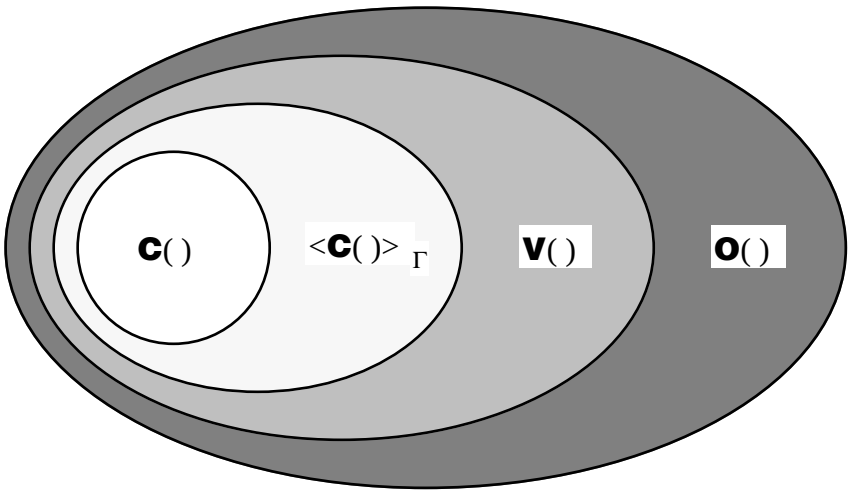


Figure 10. Classes of CVG objects.

The set of algebraic rules outlined above enable such a composition through a series of geometrical generalisations, that is, from a set of points, to convex volume object (defined

upon a convex hull $\mathcal{V}_{ol}$(P)), then a volume object (upon volumetric scalar field defined in $\mathbb{E}^3$), and finally a spatial object (upon general scalar field in $\mathbb{E}^3$); together with a series of operational decomposition from operations on spatial objects, to those on scalar fields, and then those on scalars. These algebraic rules not only provide a means for constructing complex CVG objects from simple convex volume objects, but also ensure that the modelling scheme governed by these rules is mathematically consistent. It is the general scalar fields at the high-end of the pipeline facilitates the inter-operability of spatial objects, while the finite representations of convex volume objects at the other end make data acquisition practical.

6 IMPLEMENTATION

The implementation of CVG techniques involves the design of CVG languages, data representation methods and rendering algorithms. Here we discuss briefly our implementation of CVG trees, and a recursive rendering algorithm.

6.1 Hierarchical Data Representations

The previous sections have presented the theoretical concepts for Constructive Volume Geometry that enable a volume object to be constructed from a set of convex volume objects can be expressed in a CVG term, such as $\cup$ ($-$ ($-$ ($\mathbf{c}_1$, $\mathbf{c}_2$), $\mathbf{r}$), $\mathbf{s}$) in 4.7. As with many other algebraic data types in computer science, a CVG term can be represented by a CVG tree, where terminal nodes represent convex volume objects and non-terminal nodes represent CVG operators.

Leu and Chen introduced recently a two-level representation scheme, called TROVE, for scenes composed of multiple volumetric datasets [LEU98]. The scheme separates objects from their underlying volumetric data and facilitates a high degree of data sharing and space reduction. We find that the design principles of TROVE can also be applied to CVG trees.

As shown in Figure 11, we organise the data of a CVG tree into two levels, namely the *voxel* level and *object* level. The voxel level contains all the raw volumetric datasets, each of which maintains its original point set and a local coordinate system. At the object level, a CVG tree is defined with a world coordinate system, and the position of each convex volume object (terminal node) is specified by a bounding box.

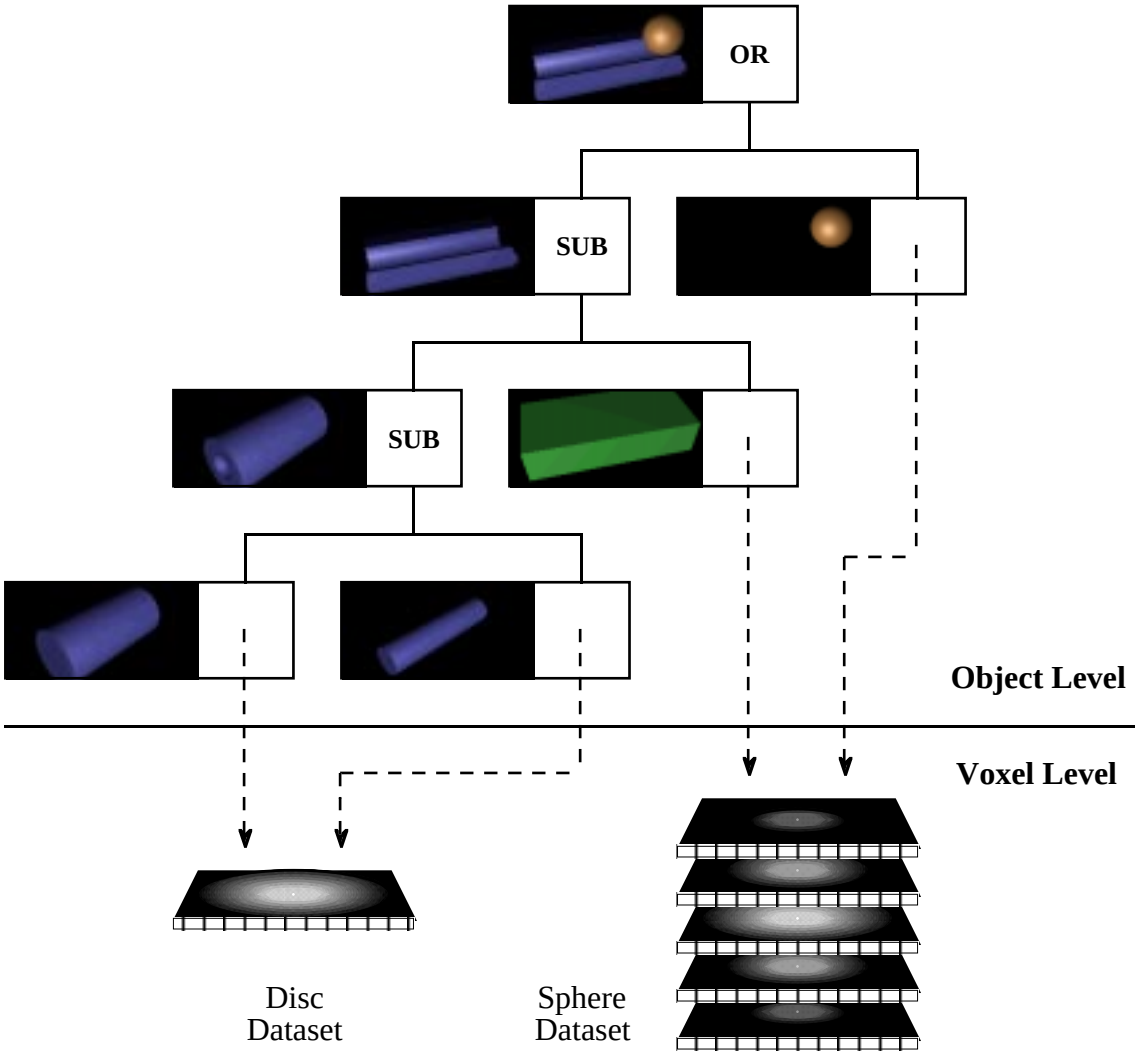


Figure 11. The data organisation of a CVG tree.

The bounding box of a convex volume object is a hexahedral box with quadrilateral faces. As in TROVE, it is not necessary for the adjacent faces or edges to be orthogonal, nor for the size of the object to correlate with the dimensions of the corresponding $\mathcal{vol}$(P). To maintain the generality of CVG trees where the representation of volumetric datasets is not restricted to regular 3D grids, we define the dimensions of a $\mathcal{vol}$(P) through a bounding box defined in the local coordinate system. Thus a mapping can easily be obtained from the two corresponding bounding boxes. In each terminal node, one may also specify opacity and colour mapping functions to define or redefine opacity and colour fields in a variety of ways, including constants, equations and look-up tables. For example, the green rectangular slab in Figure 11 is defined upon the sphere dataset with a constant opacity mapping and some geometrical transformations.

To assist an efficient rendering process, there is also x, y, z-extents stored in each node, which defines a rectangular region that bounds the corresponding sub-tree. In addition to CVG operators, we also allow geometric transformations to be specified through CVG trees, following the provision in many CSG implementations. The semantics of basic CVG trees can therefore be described as:

```
Object  Level:
           <CVG tree> ::=  <convex volume object>
                           | <CVG tree> <operator node> <CVG tree>
                           | <CVG tree> <transformation node>
<convex volume object> ::= <rectangular region in world coordinates>
                           <hexahedral box in world coordinates>
                           <volumetric data>
       <operator node> ::= <rectangular region in world coordinates>
                           <CVG operator>
 <transformation node> ::= <rectangular region in world coordinates>
                           <transformation parameters>

Voxel  Level:
     <volumetric data> ::= <dimensions in volume coordinates>
                           <channel dataset> [<channel dataset>]
     <channel dataset> ::= <volumetric scalar field>
```

This representation scheme facilitates high level data sharing and separates data to suit different computational processes. The scene in Figure 12 contains only two volumetric datasets, an MRI scan (from University of North Carolina), and an image of a disc. Both the circular base and the cylinder are defined from the disc dataset with appropriate transformation and colour mappings at the object level. Four convex volume objects are constructed from the same MRI scan with different colour mappings.

Figure 12. MRI sculpture.

6.2 A Direct Rendering Algorithm

Given a CVG object represented by a CVG tree, the goal of a rendering process is to generate a 2D image representing a view of the object. There are three classes of rendering methods in volume visualisation, namely surface reconstruction [LORE87], forward projection (splatting) [WEST90, WILH91] and ray casting [LEVO88, SABE88]. In principle, we could always transform the tree into a single volumetric object by combining scalar fields at each level of the tree, and then render the tree using one of the conventional methods. However, as discussed early, it would be more desirable to render the CVG tree directly by processing individual components as independently as possible. This is the manner that was adopted by most rendering algorithms designed for constructive solid geometry, for which past research has shown that ray casting provides the most effective and efficient mechanism. Moreover, in the context of volume rendering, surface reconstruction suffers from the inability to render amorphous structures, and forward projection has difficulties to produce high quality surface for solid models, while ray tracing is able to handle both types of objects.

Consider a ray R that is cast into a scene as illustrated in Figure 13, passing through a pixel in the image plane and intersecting with the bounding box of a CVG object at the root level of its CVG tree. Let the ray be represented by [`t·Direction+Origin`] where `Direction` is a normalised vector and `Origin` is a point in 3D space. The ray enters the box at a point where `t=TE`, and leaves at a point `t=TL`. Similar to volume ray casting, we sample the line segment between `TE` and `TL` at a regular interval.

The implementation supports both direct volume rendering [LEVO88] and direct surface rendering [JONE95]. The former, which was used to generate Figure 9 and most of the images in Figure 5, was primarily designed for amorphous objects, but can also work with solid objects if an opacity field can be suitably defined. With direct volume rendering, the ray accumulates some opacity and colour at each sampling point `S` between `TE` and `TL` to determine the colour of the corresponding pixel. The 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. Given an iso-value τ, the method follows each ray and examines pairs of successive samples, `S1` and `S2`. A ray-surface intersection is detected if O(`S1`) τ O(`S2`), or O(`S2`) τ O(`S1`).

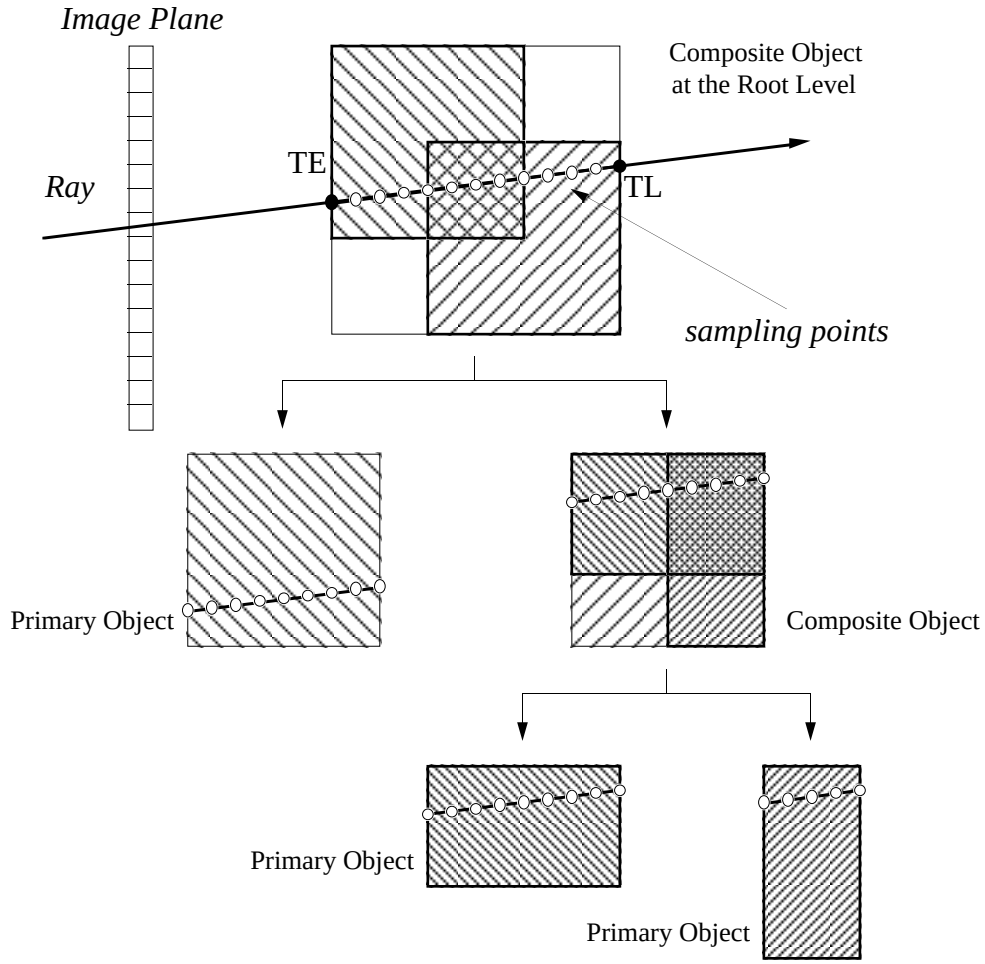


Figure 13. Direct volume rendering of a CVG tree.

Our implementation also supports the rendering of multiple iso-surfaces with varying display opacities (which is not necessarily the iso-values). Figure 14 shows a scene rendered using direct surface rendering, and it contains three volume datasets, a CT scan, an MRI scan and an image of rosewood texture. The semi-transparent mask is specified using the CT scan as ⊟ (**ct**, **box**) where **box** is used to remove the rear half of **ct**. The head is built from the MRI scan with a field scaling operation ⊡ ((k,1,1,1), **mri**) in order to eliminate the opacity overlap between the CT and MRI datasets. The complete CVG term for this scene is:

⊔ (⊔ (⊟ (**ct**, **box**), ⊡ ((k,1,1,1), **mri**)), **wood**).

Direct surface rendering was also employed to render many other CVG objects and scenes in this paper.

Figure 14. Man and mask.

7. CONCLUSIONS

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

- the concepts of spatial objects which are definable through scalar fields in Euclidean space $\mathbb{E}^3$;
- the concepts of operations on objects, CVG algebras and CVG terms;
- a set of basic operations on objects in the context of appropriate graphics models;
- some laws used to characterise operations;
- a hierarchical data representation scheme;
- a recursive rendering algorithm with both surface and volume rendering capabilities.

Unlike constructive solid geometry (CSG), CVG does not limit its operations to geometrical compositions only, and it can be applied to combine physical properties that are associated with objects, in addition to the geometrical property. It is intended for CVG *not* to fix a set of operations for all graphics models, 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 model. The operations defined in Sections 4.6 and 4.7 are treated as templates rather than standards. However, this does not prevent one from

defining a set of standard operations for a given graphics model. It is indeed the authors' wish to encourage the standardisation of operations within a well-specified model though it is generally beyond the scope of this paper.

REFERENCES

AGOS76 M. K. Agoston, *Algebraic Topology*, Marcel Dekker, New York, 1976.

AYAL85 D. Ayala, P. Brunet, R. Juan and I. Navazo, Object representation by means of non-minimal division quadtrees and octrees.

BARN93 M. F. Barnsley, *Fractals Everywhere*, Second Edition, Academic Press, Boston, 1993.

BARR89 A. H. Barr (ed.), *Topics in Physically Based Modeling*, Addison-Wesley, Reading, MA, 1989.

BINF71 T. Binford, Visual perception by computer, *Proc. IEEE Conference on Systems and Control*, Maimi, FL, December 1971.

BLIN82 J. F. Blinn, A generalization of algebraic surface drawing, *ACM Trans. Graphics*, 1(3):235-256, 1982.

DUFF92 T. Duff, Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry, *ACM/SIGGRAPH Computer Graphics*, 26(2): 131-138, 1992.

FOLE90 J. D. Foley, A. van Dam, S K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, 1990.

FOUR82 A. Fournier, D. Fussell and L. Carpenter, Computer rendering of stochastic models, *Communications of the ACM*, 25(6):371-384, June 1982.

FOUR86 A. Fournier and W. T. Reeves, A simple model of ocean waves, *ACM/SIGGRAPH Computer Graphics*, 20(4): 75-84, 1986.

JONE95 M. W. Jones, *The Visualisation of Regular Three Dimensional Data*, PhD Thesis, University of Wales Swansea, 1995.

HOPP92 H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, Surface reconstruction from unorganized points, *ACM/SIGGRAPH Computer Graphics*, 26(2): 71-78, 1992.

KAUF87 A. Kaufman, Efficient algorithms for 3D scan-conversion of parametric curves, surfaces and volumes, *ACM/SIGGRAPH Computer Graphics*, 21(4): 171-179, 1987.

LEU98 A. Leu and M. Chen, Direct Rendering Algorithms for Complex Volumetric Scenes, *Proc. 16th Eurographics UK Conference*, Leeds, March 1998, pp.1-15.

LEVO88 M. Levoy, Display of surfaces from volume data, *IEEE Computer Graphics and Applications*, 8(5):29-37, 1988.

LEVO90 M. Levoy, Efficient Ray Tracing of Volume Data, *ACM Trans. Graphics*, 9(3): 245-261, 1990.

LORE87 W. Lorensen and H. Cline, Marching cubes: a high resolution 3D surface construction algorithm, *ACM/SIGGRAPH Computer Graphics*, 21(4):163-169, July 1987.

MARC74 L. March and P. Steadman, *The geometry of Environment*, MIT Press, Cambridge, Mass., 1974.

MEAG82 D. Meagher, Geometric modeling using octree encoding, *Computer Graphics and Image Processing*, 19(2):129-147, June 1982.

MEIN92 K. Meinke and J. V. Tucker, Universal algebra, in S. Abransky, D. Gabbay and T. S. E. Maibaum (eds) Handbook of Logic in Computer Science, Volume I, Oxford University Press, 1992, pp.189-411.

MUEA91 S. Muraki, Volumetric shape description of range data using "Blobby Model", *ACM/SIGGRAPH Computer Graphics*, 25(4): 227-246, 1991.

ODDY91 R. J. Oddy and P. J. Willis, A physically based colour model, *Computer Graphics Forum*, 10(2):121-127, 1991.

PORT84 T. Porter and T. Duff, Compositing digital images, *ACM/SIGGRAP*H Computer Graphics, 18(3):253-259, 1984.

RANJ94 V. Ranjan and A. Fournier, Volume models for volumetric data, *IEEE Computer*, 27(7):28-36, 1994.

REEV83 W. T. Reeves, Particle systems — a technique for modelling a class of fussy objects, *ACM/SIGGRAPH Computer Graphics*, 17(3):359-376, 1983.

REDD78 D. Reddy and S. Rubin, *Representation of Three-Dimensional Objects*, CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1978.

REQU77 A.A.G. Requicha, *Mathematical Models of Rigid Solids*, Technical Memo 28, Production Automation Project, University of Rochester, Rochester, NY, 1977.

REQU80 A.A.G. Requicha, Representations for rigid solids: theory, methods and systems, *ACM Computing Surveys*, 12(4):437-464, 1980.

REQU82 A.A.G. Requicha and H. B. Voelcker, Solid modeling: a historical summary and contemporary assessment, *IEEE Computer Graphics and Applications*, 2(2):9-24, March, 1982.

REQU83 A.A.G. Requicha and H. B. Voelcker, Solid modeling: Current status and research directions, *IEEE Computer Graphics and Applications*, 3(7):25-37, October, 1983.

REQU85 A.A.G. Requicha and H. B. Voelcker, Boolean operations in solid modelling: boundary evaluation and merging algorithhms, *Proceedings of IEEE*, 73(1):30-44, January, 1985.

SABE88 P. Sabella, A rendering algorithm for visualizing 3D scalar fields, *ACM/SIGGRAPH Computer Graphics*, 22(4):51-58, 1988.

SMIT84 A. R. Smith, Plants, fractals and formal languages, *ACM/SIGGRAPH Computer Graphics*, 18(3):1-10, 1984.

STYT91 M. R. Stytz, G. Frieder and O. Frieder, Three-dimensional medical imaging: algorithms and computer systems, *ACM Computing Surveys*, 23(4):421-499, 1991.

THIB87 W. C. Thibault and B. F. Naylor, Set operations on polyhedra using binary space partitioning trees, *ACM/SIGGRAPH Computer Graphics*, 21(4):153-162, 1987.

UPSO88 C. Upson and M. Keeler, V-BUFFER: visiable volume rendering, *ACM/SIGGRAPH Computer Graphics*, 22(4): 59-64, 1988.

WAIT85 R. Wait and A. R. Mitchell, *Finite Element Analysis and Applications*, John Wiley & Son, Chichester, 1985.

WECH91 W. Wechler, Universal algebra for computer scientists, EATCS Monographs, Springer-Verlag, Berlin, 1991.

WEIL86 J. Weil, The synthesis of cloth objects, *ACM/SIGGRAPH Computer Graphics*, 20(4):49-54, 1986.

WEST90 L. Westover, Footprint evaluation for volume rendering, *ACM/SIGGRAPH Computer Graphics*, 24(4):59-64, 1988.

WILH91 J. Wilhelms and A. Van Gelder, A coherent projection approach for direct volume rendering, *ACM/SIGGRAPH Computer Graphics*, 25(4): 275-284, 1991.

WIRS91 M. Wirsing, Algebraic specification, in J van Lecuren (ed) Handbook of Theoretic Computer Science, Volume B: Formal Methods and Semantics, North-Holland, 1991, 675-678.

WYVI86 G. Wyvill, C. McPheeters and B. Wyvill, Data tructures for soft objects, *The Visual Computer*, 2(4):227-234, April 1986.