Volume Graphics (2005)
E. Gröller, I. Fujishiro (Editors)

Combining Point Clouds and Volume Objects in Volume Scene Graphs

Min Chen†

University of Wales Swansea, UK

Abstract

This paper describes an extension to the technical framework of Constructive Volume Geometry *(CVG) in order to accommodate point clouds in volume scene graphs. It introduces the notion of* point-based volume object *(PBVO) that is characterized by the opacity, rather than the geometry, of a point cloud. It examines and compares several radial basis functions (RBFs), including the one proposed in this paper, for constructing scalar fields from point clouds. It applies basic CVG operators to PBVOs and demonstrates the inter-operability of PBVOs with conventional volume objects including those procedurally defined and those constructed from volume datasets. It presents an octree-based algorithm for reducing the complexity in rendering a PBVO with a large number of points, and a set of testing results showing a significant speedup when an octree is deployed for rendering PBVOs.*

Keywords: Volume graphics, volume visualization, volume scene graph, constructive volume geometry, point-based modeling, point-based rendering, radial basis functions, ray casting.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Object Modeling – Object hierarchies; I.3.3 [Computer Graphics]: Image Generation – Display algorithms;

1. Introduction

Point-based modeling and rendering [PZvBG00, RL00] is one of the major advances in computer graphics recently. This collection of techniques enables direct and efficient process of complex geometric objects represented by large discretely sampled point clouds. As point-based rendering techniques are fundamentally associated with the splatting concept developed for direct volume rendering in the early 1990's [Wes90, CM93, MY96], it is relatively straightforward to realize a splatting-based rendering system that can handle both point clouds and volume datasets [ZPBG01]. However, this is less true for a ray-casting system that could potentially enjoy higher rendering quality, more rendering effects, more complex object composition, and higher level scene representations such as volume scene graphs.

In this paper, we address several technical issues in combining point clouds and conventional volume objects in *Constructive Volume Geometry* (CVG), which facilitates the modeling of complex objects using volume scene graphs and the direct rendering of volume scene graphs using ray-casting. Our contributions include:

- introducing the notion of *point-based volume object* (PBVO) that is characterized by the opacity, rather than the geometry (e.g., an isosurface), of a point cloud;
- examining and comparing several radial basis functions (RBFs) for constructing scalar fields from point clouds, and proposing a new RBF that offers more flexibility in achieving a desired blending effect;
- extending several basic CVG operators to PBVOs in volume scene graphs and demonstrating the inter-operability between PBVOs and conventional volume objects which are constructed from volume datasets or defined procedurally;
- developing an octree-based approach to the problem of determining a subset of points that are relevant to each sampling position in direct rendering a volume scene graph involving PBVOs, and conducting a series of tests showing a significant speedup when an octree is deployed for a large PBVO.

† e-mail: m.chen@swansea.ac.uk

The remainder of the paper is organized as follows. In Section 2, we give a brief overview of a collection of technical developments in volume graphics, implicit surfaces and point-based techniques, which provide the basis as well as motivation for this work. In Section 3, we introduce necessary definitions for PBVOs in the framework of CVG, and consider several RBFs, including the one proposed in this work, for constructing scalar fields of a PBVO. In Section 4, we discuss integration of PBVOs into volume scene graphs, and examine the inter-operation between PBVOs and conventional volume objects. We show the effectiveness of specifying and constructing complex objects using primitive objects defined on point clouds and volume datasets as well as those defined procedurally. We also investigate into the prospect of handling volume datasets in a point-based manner. In Section 5, we address several technical issues in rendering PBVOs and present an octree-based algorithm for sampling a volume scene graph in ray casting. We give a collection of testing results to aid our discussions on the performance of the algorithm. Finally, in Section 6, we offer our observations and briefly discuss possible future work.

2. Related Work

Many existing modeling schemes were designed to support *surface modeling* and *solid modeling* [RV82]. Those which are relevant to this work include boundary representations (b-reps), constructive solid geometry (CSG), spatial occupancy enumeration, implicit surfaces and octrees [Wat00]. There are also a number of modeling schemes capable of representing non-solid objects, or objects whose boundary cannot easily be determined. They include volume datasets, particle systems, point clouds and image-based modeling.

In particular, *implicit surfaces* facilitate the representation of 'blobby models' [Bli82], 'meta balls' [NHK*85] and 'soft objects' [WMW86] through scalar fields, and the composition of complicated objects from elemental field functions. Its elegance lies in the mapping from implicit functions in the real domain to surface-based objects primarily in the binary domain. The concept of CSG has been applied to implicit surfaces [Duf92, WGG99], and the octree method has been used for polygonizing implicit surfaces [Blo88] and computing ray-surface intersections [KB89]. The scheme has also been used to approximate volume datasets [HQ04].

The advances in *volume visualization* have produced a collection of methods for rendering *volume datasets*, including isosurfacing [LC87], ray casting [Lev88] and forward projection [Wes90]. Volume modeling possesses more descriptive power than surface and solid modeling. Though in most cases, voxels are organized into a grid or a mesh, in some cases, volume modeling with scattered data is necessary (e.g., [Nie93]). The concept of CSG has been applied to volume modeling through voxelization [WK93, FD98, BMW98]. The development of *Constructive Volume Geometry* (CVG) [CT00] has provided a theoretic framework for constructive volume modeling based primarily on real-domain operations on the opacity of volume objects, rather than Boolean operations on their geometry which is normally not explicitly or uniquely described in volume representations. CVG also facilitates volume scene graphs and direct rendering of such a scene graph [WC01]. The concept of CVG has also been extended to facilitate volumetric operations in implicit modeling in the form of FRep [PASS01].

In recent years, we have witnessed the rapid popularization of point-based modeling and rendering. The most significant examples of this development include Surfels [PZvBG00] and QSplat [RL00]. Other important developments include [GD98, ABC*01, SD01, ZPBG01]. In addition to the splatting approach commonly adopted in point-based rendering, ray tracing point clouds through intersection has been examined [SJ00]. Radial basis functions for building implicit surfaces upon point clouds and polygonizing such surfaces has been studied [CBC01], which represents a shift of focus from approximating point datasets economically to constructing point-based models directly and faithfully. The concept of CSG has also been applied to point-based modeling [AD04]. Recently, the use of the point-based approach for isosurfacing volume datasets has been investigated [ZPBG01, vHJ*04, LT04].

It is useful to highlight the close relationships among volume graphics, implicit surfaces and point-based techniques. *Points*, as modeling primitives, are extensively featured in all three classes of techniques. Moreover, the splatting approach, originally developed in volume visualization is playing perhaps a more significant role in point-based techniques. The essence of field-based modeling in implicit surfaces is further enriched in volume graphics for facilitating true 3D representations. This has motivated the author to examine the use of point-based objects in volume graphics.

3. Point-based Volume Object (PBVO)

3.1. Definitions

Let $\mathbb{R}$ denote the set of all real numbers, and $\mathbb{E}^3$ denote the 3D Euclidean space. A *scalar field* is a function $F : \mathbb{E}^3 \rightarrow \mathbb{R}$. Conceptually a scalar field $F(p)$ is a volumetric generalization of a surface function that models only those points on the surface. A volumetric representation of an object can thereby be specified as a set of scalar fields, $F_1(p), F_2(p), \ldots, F_k(p)$, that define the geometrical and physical properties of the object at every point p in $\mathbb{E}^3$. We call such an object a *spatial object*. In this work, we consider only spatial objects with four attribute fields, namely opacity (O), red (R), green (G) and blue (B), where $O, R, G, B : \mathbb{E}^3 \rightarrow [0,1]$. A spatial object is written as a tuple, $\mathbf{o} = (O, R, G, B)$.

Each scalar field can be defined procedurally, or on a discretely sampled dataset. In volume graphics, it is common to construct a scalar field from a volume dataset, where voxels

are organized in the form of a regular 3D grid and each voxel is associated with a value. Tri-linear interpolation is usually used to determine the unknown values in the bounding volume of the dataset.

In a similar manner, given a point cloud, $P = \{p_1, \ldots, p_n\}$, one can construct a scalar field by using 3D Delaunay tetrahedralization, and determine the unknown values in the bounding volume of P by using bary-centric interpolation, which is commonly used in finite element analysis. An alternative approach is to consider each point $p_i \in P$ as a *radial basis function* (RBF), $\omega(q, p_i, r_i)$, where q is an arbitrary point in $\mathbb{E}^3$ and $r_i \geq 0$ is the so-called *radius of influence* that defines a spherical bounding volume such that:

$$\omega(q, p_i, r_i) = 0, \forall q, \| q - p_i \| > r_i$$

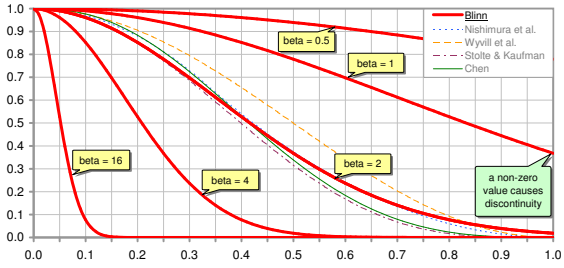
where $\| q - p_i \|$ denotes the Euclidean distance between q and p_i. For a collection of data values, $v_1, \ldots, v_n$, associated with $p_1, \ldots, p_n$ respectively, a scalar field V is therefore defined using a *blending function* as:

$$V(q) = \sum_{1 \leq i \leq n} \omega(q, p_i, r_i) v_i. \qquad (1)$$

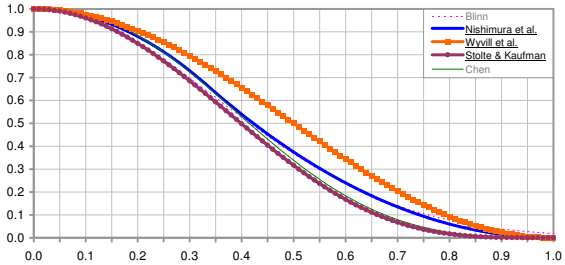
This approach forms the basis of several implicit modeling methods such as blobby models, meta balls and soft objects [Bli82, Mur91, WMW86]. However, in this work, instead of focusing on a particular implicit surface, $V(q) = 0$, or an implicit solid, either $V(q) \leq 0$ or $V(q) \geq 0$, we are interested the heterogeneous volumetric properties of $V(q)$ in modeling attribute fields of spatial objects.

For example, consider a discretely sampled point cloud where each p_i is associated with a confidence value and an intensity value. We can map the confidence value to r_i, and the intensity value to an opacity value, o_i. With an appropriate RBF, we can define an opacity field O as in Eq.(1). In volume modeling, O in effect defines the 'visible geometry' of the point cloud. Any point $q \in \mathbb{E}^3$ with $O(q) > 0$ is 'visible' or 'renderable' to a rendering algorithm, and may potentially contribute to a synthesized imagery representation of the point cloud.

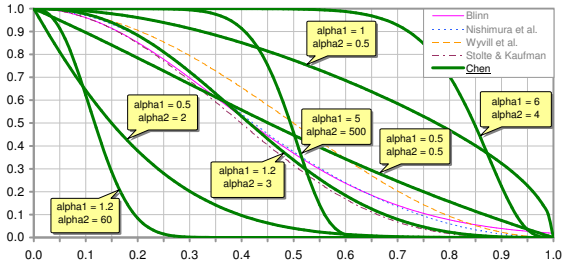
A spatial object **o** is a *volume object* if there is a bounded set $\mathrm{X} \subset \mathbb{E}^3$ such that the opacity field of **o** is bounded by X. In other words, it is certain that **o** is not visible in $(\mathbb{E}^3 - \mathrm{X})$, though it might also contain 'invisible geometry' in X. A volume object is called a *point-based volume object* (PBVO) if it has a least one attribute field that is built from a discrete point cloud. In the following discussions, we consider mainly PBVOs whose opacity fields are built from discrete point clouds. In Color Plate II, image (b) shows an example PBVO, built from the Stanford Bunny point set shown in image (a). Unlike traditional implicit modeling and point-based modeling that focuses on a specific isosurface, one can see a translucent shell of some thickness, which is determined mainly by the RBF used for its opacity field.



(a) Blinn's RBF ω^B



(b) Nishimura *et al.*'s RBF ω^N, Wyvill *et al.*'s RBF ω^W, and Stolte and Kaufman's RBF ω^W



(c) The new RBF proposed ω^C

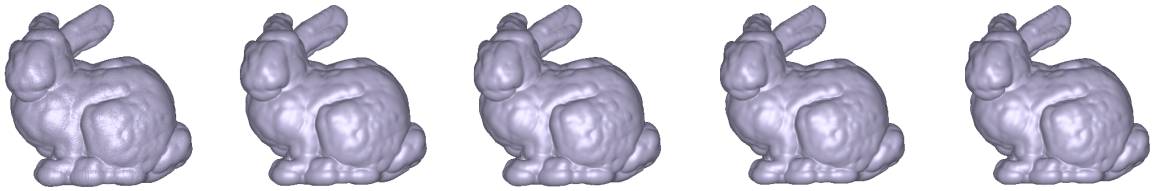
Figure 1: *Five radial basis functions (RBFs) considered.*

3.2. Radial Basis Functions (RBF)

As mentioned above, we are interested in blending functions for modeling attribute fields of spatial objects. Hence, such a function must satisfy the functional domain of these fields, that is, $O, R, G, B : \mathbb{E}^3 \rightarrow [0,1]$. It is therefore beneficial to take this constraint into account when considering RBFs. While there is no theoretic or practical difficulty in accommodating a negative RBF, it does not seem to offer any benefit to the modeling of PBVOs in this work. In general, one can always use the sign of v_i in Eq.(1) to control the additive or subtractive property of individual point elements.

There are several RBFs commonly used for implicit surface modeling. One of which is the Gaussian function used by Blinn [Bli82] for defining algebraic surfaces. To make the discussion more relevant to point-based modeling in volume graphics, we rewrite Blinn's RBF as follows:

$$\omega^B(q, p_i, r_i) = \begin{cases} e^{-\beta u_i^2} & \text{if } \| q - p_i \| < r_i \\ 0 & \text{if } \| q - p_i \| \geq r_i \end{cases}$$



Blinn's ω^B Nishimura *et al.*'s ω^N Wyvill *et al.*'s ω^W Stolte and Kaufman's ω^W Our proposed ω^C

Figure 2: *A comparison of five different RBFs for constructing a PBVO from a large point cloud (Stanford bunny) of 35947 points. Each PBVO is rendered using volume ray casting.*

where $u_i = \| q - p_i \| / r_i$ is the relative distance from q to p_i normalized by the *radius of influence*, r_i, and $\beta > 0$ is related to the standard deviation of the Gaussian function as shown in Figure 1(a). The main shortcoming of ω^B is that the basic Gaussian function is intended to be used 'globally', allowing every point element to have an influence on every point in space. When we restrict the radius of influence, discontinuity is often visually obvious in the scalar field resulting from the blending of different RBFs together.

A few researchers proposed several different polynomial functions in place of the exponential function in ω^B. They include ω^N, ω^W and ω^S which were proposed in [NHK*85, WMW86, SK97] as follows:

$$\omega^N(q,p_i,r_i) = \begin{cases} 1-3u_i^2 & \text{if } \| q-p_i \| < \frac{r_i}{3} \\ \frac{3(1-u_i)^2}{2} & \text{if } \frac{r_i}{3} \le \| q-p_i \| < r_i \\ 0 & \text{if } \| q-p_i \| \ge r_i \end{cases}$$

$$\omega^W(q,p_i,r_i) = \begin{cases} 1-\frac{4u_i^6-17u_i^4+22u_i^2}{9} & \text{if } \| q-p_i \| < r_i \\ 0 & \text{if } \| q-p_i \| \ge r_i \end{cases}$$

$$\omega^S(q,p_i,r_i) = \begin{cases} \frac{(\|q-p_i\|^2-r_i^2)^4}{r_i^8} & \text{if } \| q-p_i \| < r_i \\ 0 & \text{if } \| q-p_i \| \ge r_i \end{cases}$$

While these RBFs offer a good approximation of ω^B when β is around 2, they have lost some flexibility in controlling the shape of an RBF apart from changing the radius r_i (Figure 1(b)). To reintroduce some control flexibility as well as to involve the factor of distance attenuation, we experimented with the following RBF:

$$\omega^C(q,p_i,r_i) = \begin{cases} \left(1-\frac{2\|q-p_i\|^{2\alpha_1}}{1+\|q-p_i\|^2}\right)^{\alpha_2} & \text{if } \| q-p_i \| < r_i \\ 0 & \text{if } \| q-p_i \| \ge r_i \end{cases}$$

ω^C was partially inspired by Nielson's work on handling volumetric scattered data [Nie93], and it addresses a shortcoming of his RBF which would result in ∞ when q coincides with any p_i. The two parameters $\alpha_1 \ge 0.5$ and $\alpha_2 \ge 0$ enable considerable flexibility in controlling the shape of the RBF as shown in Figure 1(c). When $\alpha_1 = 1.2, \alpha_2 = 3$, for example, ω^C has a similar shape as ω^N, ω^W, ω^S and ω^B with $\beta = 2$. When $\alpha_1 = 1, \alpha_2 = 20$, ω^C has a similar shape as ω^B with $\beta = 6$. Unlike ω^B, ω^C maintains the basic G^0 continuity at the boundary of the RBF.

Color Plate III shows a comparison of five RBFs for constructing a PBVO from a point cloud. With the help of a PBVO built from a simple point set with five point elements, one can easily observe the discontinuity on the object in the top-left image which was rendered with ω^B. For a large sampled point cloud, as shown in Figure 2, when the radius of influence is carefully chosen, all five RBFs are shown to be reasonably effective.

Both ω^B and ω^C allow the control of the blending independently from the radius of influence, while ω^B offers a relatively limited degree of freedom (Figure 1). Figure 3 shows a few examples illustrating the effects of varying α_1 and α_2. A smaller value for α_1 results in a more gentle 'drop' on the ω^C curve, hence leading to a smoother blending, while a larger value enables ω^C to preserve more features defined by points. In conjunction with α_1, α_2 controls the 'position' of the drop, hence the 'strength' of ω^C.

All of the above RBFs operate within the $[0,1]$ domain. In other words, considering an individual point element p_i, it cannot propagate its data value v_i in a magnified manner. However, given a collection of data values, $v_1, v_2, \ldots, v_n$, it is still possible for the blending function in Eq.(1) to produce a value greater than the maximum data value, i.e., $V(q) > \max(v_1, v_2, \ldots, v_n)$, which is not consistent with our definition of a volume object. Let $w_{sum} = \sum_{1 \le i \le n} \omega(q,p_i,r_i)$. We therefore modify Eq.(1) as follows:

$$V(q) = \begin{cases} \sum \omega(q,p_i,r_i) v_i / w_{sum} & \text{if } w_{sum} > 1 \\ \sum \omega(q,p_i,r_i) v_i & \text{if } w_{sum} = 1 \\ \sum \omega(q,p_i,r_i) v_i / w_{sum}^{1-\eta} & \text{if } 0 < w_{sum} < 1 \\ 0 & \text{if } w_{sum} = 0 \end{cases} \quad (2)$$

where the summation range is $i \in [1,n]$. Eq.(2) operates in the range between $\min(0, v_1, \ldots, v_n)$ and $\max(v_1, \ldots, v_n)$, and it was used to synthesize all images in this paper. For

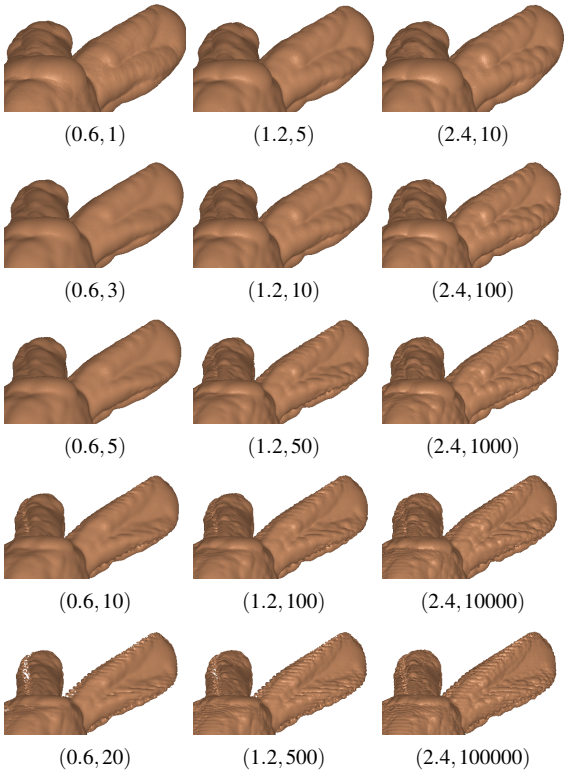


Figure 3: *Varying the two parameters of* ω^C, (α_1, α_2), *when rendering part of a PBVO (Stanford bunny). An isosurface,* $V(q) = 0.95$, *was directly rendered. While* α_2 *controls the strength of* ω^C, α_1 *affects the smoothness of blending.*

example, η is set to 1 for all images in Color Plate III, which preserves the principle features of $V(q)$ in Eq.(1).

4. PBVOs in Volume Scene Graphs

4.1. Volume scene graph

In the theoretic framework of CVG [CT00], a *volume scene graph* is an algebraic expression, called a *CVG term*, which involves a class of spatial objects and a family of constructive operations. In practice, a CVG term can be represented by a tree, where constructive operations are defined at non-terminal nodes, and elemental volume objects are defined at terminal nodes. Each subtree in effect defines a composite volume object, while the root represents the final composite volume object, or the *scene*. To facilitate the sharing of low level object data, we allow a CVG term to be realized using a directed acyclic graph with a single root, hence resulting in a volume scene graph. Geometrical transformations and transfer functions can be applied at each graph node.

To facilitate efficient processing of a CVG term, a bounding box is assigned to every node in a CVG tree. This in effect makes all spatial objects into volume objects. We utilize three coordinate systems for scene specification:

- *World Coordinate System* — a theoretically unbounded 3D domain where all objects in a scene are positioned.
- *Normalized Volume Coordinate System* — a unit cubic domain which is used to standardize the transformation between the bounding box of a volume object at the terminal node (defined in the world coordinate system) and a local *data coordinate system* that is data-dependent.
- *Data Coordinate System* — 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. For instance, given a volume dataset of $N_x \times N_y \times N_z$ voxels, our implementation determines the domain as $[0,N_x-1]\times[0,N_y-1]\times[0,N_z-1]$ when it loads the dataset. Given a point cloud, our implementation determines the domain automatically as:

$$x_1 = \min(p_{i,x} - r_i \,|\, i \in [1,n]), \quad y_1 = \ldots, \quad z_1 = \ldots;$$
$$x_2 = \max(p_{i,x} + r_i \,|\, i \in [1,n]), \quad y_2 = \ldots, \quad z_2 = \ldots.$$

In our implementation, we have included a set of built-in scalar fields, such as spherical, cylindrical, parabolic and hyperbolic fields, and random volumetric grids, all of which are defined in the normalized volume coordinate system.

4.2. CVG operators

In this work, we consider a class of spatial objects with four scalar fields in the form of $\mathbf{o} = (O,R,G,B)$. As an example, we give three basic CVG operations as follows:

- *union*: $\boxed{\cup}(\mathbf{o}_1,\mathbf{o}_2) = (\textbf{MAX}(O_1,O_2), \textbf{SELECT}(O_1,R_1,O_2,R_2), \textbf{SELECT}(O_1,G_1,O_2,G_2), \textbf{SELECT}(O_1,B_1,O_2,B_2))$;
- *intersection*: $\boxed{\cap}(\mathbf{o}_1,\mathbf{o}_2) = (\textbf{MIN}(O_1,O_2), \textbf{SELECT}(O_1,R_1,O_2,R_2), \textbf{SELECT}(O_1,G_1,O_2,G_2), \textbf{SELECT}(O_1,B_1,O_2,B_2))$;
- *difference*: $\boxed{-}(\mathbf{o}_1,\mathbf{o}_2) = (\textbf{SUB}_{01}(O_1,O_2),R_1,G_1,B_1)$;

where **MAX**, **MIN**, **SELECT** and $\textbf{SUB}_{01}$ are scalar field operations which can easily be derived from scalar operations by pointwise extension as given in [CT00], where one can also find some other CVG operators.

Color Plate IV shows the results of applying CVG operations to a PBVO **r** built from the Stanford bunny point set, and a procedurally defined cylindrical object **c**. Note that **c** comprises a translucent shell and a solid core. When it is subtracted from **r**, its translucent shell removes a portion of the opacity on the corresponding part of **r**, while its solid core removes all the opacity from the part of **r** where it intersects. This demonstrates that the bunny point set has been transformed to a true 3D volumetric representation, the 'geometry' of which is essentially defined by its opacity field.

Color Plate II(c) shows an image synthesized from a volume scene graph, $\boxed{\cup}(\boxed{-}(\mathbf{r},\mathbf{c}),\mathbf{h}))$, where **r** is a volume object built from a point cloud (Stanford bunny), **h** is another built from a volume dataset (UCSD rabbit heart), and **c** is

a procedurally-defined cylindrical object for making a hole on the exterior shell. The bunny is rendered as a translucent shell and the heart as a fully-opaque object.

Color Plate I shows a volume scene graph involving four PBVOs built from the same point cloud of 437645 points (Stanford dragon), all of which are immersed in artificial clouds represented by a volume dataset (Erlangen clouds). While the three coordinate systems facilitate the sharing of the common point data, CVG enables solid dragons to be combined with amorphous and translucent clouds.

4.3. Extracting Points from Volume Datasets

This also leads to an interesting perspective as to direct rendering a volume using only a subset of voxels near an isosurface, for which we have conducted an initial investigation and experimentation. In our experiments, we have noticed that, for a given isovalue τ, it is generally more effective to extract voxels of values in the range $[\tau, \tau+\varepsilon]$ if the voxel values inside the interested isosurface are mostly greater than those outside. Similarly, it is more effective to extract voxels of values in $[\tau, \tau-\varepsilon]$ if vice versa. Figure 4 shows a series of examples with different settings for ε, which is specified in terms of the normalized voxel value range between 0% and 100%. The greater ε is, the more voxels are extracted, and likely the better rendering result can be achieved. The radii of points are set according to the overall density of voxels, though anything less than 4 seems to have resulted in a high level of aliasing.

In comparison with Figure 4(a) which is rendered directly from the CT head dataset, none of the point-based volume objects has achieved a similar level of quality. However, images in (i)-(l) are of a reasonable quality, where major geometrical features such as the horizontal line in the original dataset are clearly visible. While further studies are necessary, the initial results are encouraging.

5. Ray Casting PBVOs

5.1. Rendering a Volume Scene Graph

Given a volume scene graph, the goal of a rendering process is to synthesize a 2D image representing a view of the scene. So far, ray casting is still the most appropriate means for directly rendering a volume scene graph, which features multiple volume objects, solid or translucent. Splatting has been effective in handling a single volume dataset as the order of voxels can easily be determined prior to the projection of splats. It has also been successfully deployed in point-based rendering for handling arbitrarily-ordered 'flat' splats by using image-space composition. The main obstacle for using splatting in directly rendering a volume scene graph is the necessity for combining volumetric properties (e.g., opacity and color) of intersected volume objects in the object space, and the difficulty in determining which point or voxel to be combined with which other points and voxels, unless they are all aligned on the same grid structure. Furthermore, establishing the projection order of 'volumetric' points and voxels among different objects in an arbitrary volume scene graph is not a trivial problem. Not mention that any procedurally defined volume object would require a process of discretization prior to the rendering. We therefore focus on the ray casting method in this work.

The basic ray casting mechanism is to sample at regular intervals along each ray cast from the view position. At each sampling position s_w in world coordinates, we recursively determine if s_w is inside the bounding box of the current CVG subtree, until we reach a terminal node. If s_w is inside the bounding box of the terminal node which contains an elemental volume object $\mathbf{o} = (O,R,G,B)$, we first transform s_w into s_n in the normalized coordinate system associated with $\mathbf{o}$, and then evaluate the four scalar fields, O,R,G,B.

When a scalar field is procedurally defined, for instance, by a function $F_{sphere}(x), x \in [0,1]^3$, we simply evaluate $F_{sphere}(s_n)$, since in this work all procedural scalar fields are defined in the normalized coordinate system. When a scalar field is defined on a volume dataset, we convert s_n to the local data coordinate system of the volume dataset, resulting in s_d. It is straightforward to identify the grid cell where s_d resides. A scalar value can be obtained for s_d by interpolating the values at the eight voxels which bound the cell.

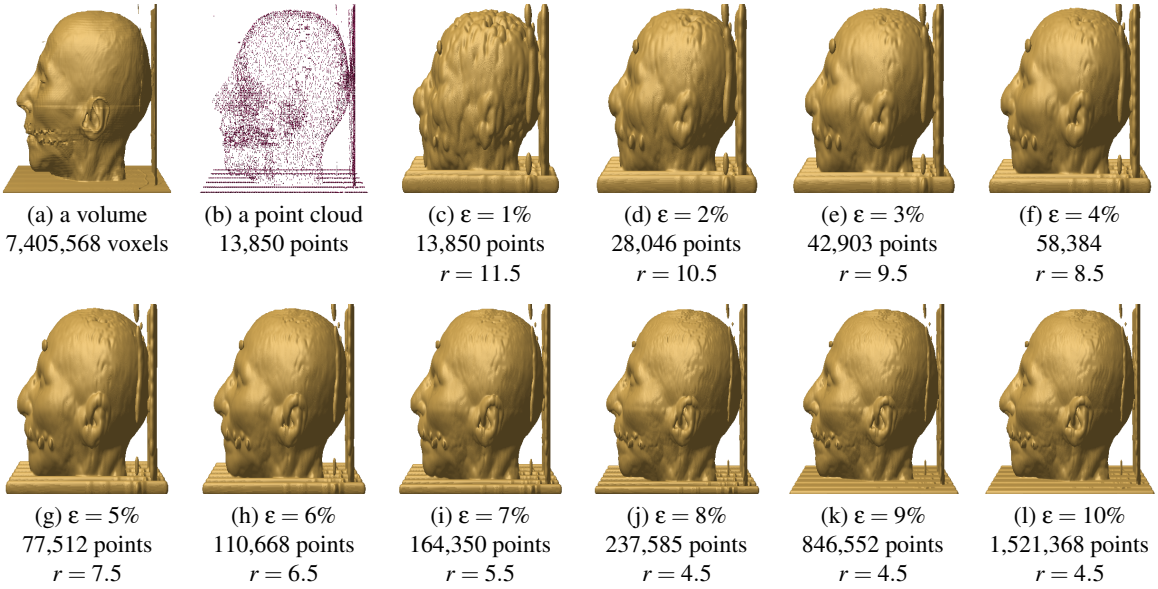
When a scalar field is defined on a point cloud $P = \{p_1, p_2, \ldots, p_n\}$, we first obtain s_d in data coordinates in a way similar to that with a volume dataset. We then identify a subset of points, $P' \subseteq P$, such that

$$P' = \{p'_i \,|\, p'_i \in P \text{ and } \| s_d - p'_i \| \leq r_i\}.$$

Given such a subset, we can sample the RBF of each $p'_i \in P'$, and obtain a scalar value by using Eq.(2). For a large point cloud, the most expensive cost in this process is the identification of P', as it involves a distance calculation against every point $p_i \in P$, rendering the whole process un-scalable when n increases.

Table 1 gives a set of timings for rendering two groups of randomly generated point clouds. The first group includes four point clouds with 10, 100, 1000, 10000 points respectively. All points are randomly placed on a spherical surface, representing a point-based surface. Point clouds in the second group are generated in a similar manner, but all points are randomly placed inside a spherical surface, representing a point-based volume.

The 'brute force' method results in timings that indicate an $O(n)$ complexity. Although this may seem to be acceptable in some other situations, it is not desirable in rendering a volume scene graph that may contain many PBVOs.



(a) a volume 7,405,568 voxels | (b) a point cloud 13,850 points | (c) $\varepsilon = 1\%$ 13,850 points $r = 11.5$ | (d) $\varepsilon = 2\%$ 28,046 points $r = 10.5$ | (e) $\varepsilon = 3\%$ 42,903 points $r = 9.5$ | (f) $\varepsilon = 4\%$ 58,384 $r = 8.5$

(g) $\varepsilon = 5\%$ 77,512 points $r = 7.5$ | (h) $\varepsilon = 6\%$ 110,668 points $r = 6.5$ | (i) $\varepsilon = 7\%$ 164,350 points $r = 5.5$ | (j) $\varepsilon = 8\%$ 237,585 points $r = 4.5$ | (k) $\varepsilon = 9\%$ 846,552 points $r = 4.5$ | (l) $\varepsilon = 10\%$ 1,521,368 points $r = 4.5$

Figure 4: *Comparing the visualization of the CT head dataset rendered as a conventional volume object and as a PBVO. Several point clouds were extracted from the original dataset with different numbers of points (voxels).*

n: number of points	10	100	1000	10000
Randomly generated points on a spherical surface				
a. brute force	16.62	52.12	413.61	2454.21
b. octree (preprocess)	0.00	0.00	0.04	2.38
c. octree (render)	14.68	15.36	17.92	15.43
speedup: a/(b+c)	1.13	3.39	23.03	137.8
Randomly generated points in a spherical volume				
d. brute force	16.68	52.34	419.38	3423.82
e. octree (preprocess)	0.00	0.00	0.02	1.08
f. octree (render)	17.34	16.20	20.22	27.66
speedup: d/(e+f)	0.96	3.23	20.72	119.13

Table 1: *Testing results for ray casting two groups of randomly generated point clouds. All the timings are in seconds, and were obtained on a 2GHz Pentium 4 with 1GB memory.*

5.2. An Octree-based Rendering Algorithm

We hence introduce an octree structure for partitioning a large point cloud in its local data coordinate system. In each level of recursion, a subtree contains only those points, which have some influence in the bounding box of the subtree. In other words, their RBFs have non-zero values in the bounding box. It is important to note that due to the non-zero radius of influence of each point, and the likely overlaps among the 'volumes of influence' of different points, a point element can belong to more than one terminal nodes.

In comparison with the brute force ray casting, Table 1 also gives the timings of ray casting with the support of an octree. One can easily observe a linear speedup in relation to the sizes of point clouds.

The construction of an octree is controlled by two parameters, namely the *maximum tree height/depth*, h_{max}, and the *'preferred' maximum number of points per terminal octant*, p_{max}. Whenever an octant reaches a tree depth of h_{max} or contains less points than p_{max}, the octant will be made into a terminal node. p_{max} is designed to maintain an optimal number of points in a terminal node, implying that it is unlikely that more than p_{max} points will have their 'volumes of influence' overlapped. However, given an arbitrary point cloud, it is always possible to have such a situation occurring. Thus the tree height control through h_{max} is particularly important for avoiding an infinite recursion. Table 2 gives a set of performance data when p_{max} changes from 1000 to 10. Three sets of tests were carried out, with 10000 random points on a spherical surface, 10000 random points in a spherical volume and 35947 points of the Stanford bunny, respectively. For all the tests, h_{max} is set to 7. Because of the point density of the bunny point set, the benefit of reducing p_{max} soon becomes less obvious, unless the reduction is coupled with an increase of h_{max}. In all cases, we can see the speedup achieved is proportional to the amount of memory space used by the octree.

For large point clouds, the amount of space consumed by an octree can be quite noticeable, especially when points are densely placed and the radius of influence is set to a rela-

tively large value. We therefore stores only indices, rather than the records of points, in the terminal nodes of an octree.

With ray casting, the coherence between consecutive sampling positions can also be exploited effectively. Each octree maintains a pointer to the octant of the last search, allowing a new search to start from the same octant. It is also possible to exploit the coherence between neighboring rays, though the benefits are less conclusive in our experimentation.

6. Conclusions

We have considered a challenging problem for integrating point clouds into volume graphics in a consistent and efficient manner. We have presented our solution under the framework of *Constructive Volume Geometry* (CVG) that enables us to accommodate point clouds in volume scene graphs. Through the introduction of the notion of *point-based volume object (PBVO)*, we can transform a point cloud, typically for defining a surface, to a volume object that is characterized by opacity, rather than geometry. We have examined and compared several *radial basis functions* (RBFs) for constructing scalar fields from point clouds. We have proposed a new RBF which is more controllable than the existing polynomial functions, and does not suffer from discontinuity as the Gaussian function. We have applied basic CVG operators to PBVOs in volume scene graphs and have demonstrated the inter-operability of PBVOs with conventional volume objects. We have presented an octree-based algorithm for reducing the complexity in rendering a PBVO with a large number of points. Our testing results have shown a significant speedup when an octree is deployed for rendering PBVOs. In addition, we have investigated into the suitability for directly rendering scattered voxels extracted from a volume dataset, and the initial results are interesting and encouraging.

This work has been implemented in a research prototype system. Our further work will be focus on embedding point-based modeling and rendering in *vlib*, an open source API for volume graphics [WC01].

Acknowledgments

The author gratefully acknowledges that this work has been supported by a UK EPSRC grant (GR/R25286/01). The author also wishes to acknowledge the sources of datasets used in the paper, which include the *Bunny* and the *Dragon* from Stanford University, the *Artificial Clouds* from the Universität Erlangen-Nürnberg, the *CT Head* from the University of North Carolina, Chapel Hill, and the *Rabbit Heart* from the University of California, San Diego.

References

[ABC*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., SILVA C.: Point set surfaces. In *Proc. IEEE Visualization* (2001), pp. 29–36.

[AD04] ADAMS B., DUTRÉ P.: Boolean operations on surfel-bounded solids using programmable graphics hardware. In *Proc. Eurographics Symposium on Point-based Graphics* (2004), pp. 19–24.

[Bli82] BLINN J.: A generalized algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (July 1982), 235–256.

[Blo88] BLOOMENTHAL J.: Polygonization of implicit surfaces. *Computer Aided Geometry Design 5*, 4 (1988), 341–355.

[BMW98] BREEN D. E., MAUCH S., WHITAKER R. T.: 3d conversion of CSG models models into distance volumes. In *Proc. IEEE/ACM Symposium on Volume Visualisation* (1998), pp. 7–14.

[CBC01] CARR J., BETSON R., CHERRIE J.: Reconstruction and representation of 3d objects with radial basis functions. In *Computer Graphics (Proc. SIGGRAPH 2001)* (2001), pp. 67–76.

[CM93] CRAWFIS R., MAX N.: Texture splats for 3d scalar and vector field visualization. In *Proc. IEEE Visualization* (1993), pp. 261–266.

[CT00] CHEN M., TUCKER J.: Constructive volume geometry. *Computer Graphics Forum 19*, 4 (2000), 281–293.

[Duf92] DUFF T.: Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics (Proc. SIGGRAPH 92) 26*, 2 (1992), 131–138.

[FD98] FANG S., DAI SRINIVASAN R.: Volumetric CSG – a model-based volume visualisation approach. In *Proc. 6th International Conference in Central Europe on Computer Graphics and Visualisation* (1998), pp. 88–95.

[GD98] GROSSMAN J. P., DALLY W. J.: Point sample rendering. In *Proc. Eurographics Workshop on Rendering* (1998), pp. 181–192.

[HQ04] HUA J., QIN H.: Haptics-based dynamic implicit solid modeling. *IEEE Transactions on Visualization and Computer Graphics 10*, 5 (2004), 574–586.

[KB89] KALRA D., BARR A.: Guaranteed ray intersections with implicit surfaces. *Computer Graphics (Proc. SIGGRAPH 89) 23*, 3 (1989), 297–306.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (Proc. SIGGRAPH 87) 21*, 4 (July 1987), 163–169.

[Lev88] LEVOY M.: Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 3 (May 1988), 29–37.

[LT04] LIVNAT Y., TRICOCHE X.: Interactive point-based isosurface extraction. In *Proc. IEEE Visualization* (2004), pp. 457–464.

[Mur91] MURAKI S.: Volumetric shape description of range data using 'blobby model'. *Computer Graphics (Proc. SIGGRAPH 91) 25*, 4 (1991), 227–235.

[MY96] MUELLER K., YAGEL R.: Fast perspective volume rendering with splatting by using a ray-driven approach. In *Proc. Visuzlization '96* (1996), pp. 65–72.

[NHK*85] NISHIMURA H., HIRAI A., KAWAI T., SHIRAKAWA I., OMURA K.: Object modeling by distribution function and a method of image generation. *Computer Graphics, Jounral of Papers given at the Electronics Communication Conference '85 (in Japanese) J68-D*, 4 (1985).

p_{max}: preferred maximal points per-terminal-node.	1000	500	100	50	40	30	20	10
	Randomly generated 10000 point elements on a spherical surface							
preprocessing time (sec.)	0.03	0.03	0.07	0.14	0.18	0.33	0.47	0.55
direct volume rendering time (sec.)	154.32	154.28	34.90	22.68	20.11	18.04	16.67	16.04
speedup: brute-force/(preprocess + render)	15.90	15.90	70.18	107.55	120.96	133.6	143.19	147.93
actual octree height	3	3	5	7	7	7	7	7
octree space (MB)	0.156	0.156	0.634	1.701	2.661	5.620	9.385	11.518
[min, max] numbers of points per leaf octant	[4,411]	[4,411]	[1,94]	[1,50]	[1,48]	[1,48]	[1,48]	[1,48]
	Randomly generated 10000 point elements in a spherical volume							
preprocessing time (sec.)	0.03	0.04	0.10	0.24	0.27	0.51	0.94	1.07
direct volume rendering time (sec.)	277.58	174.52	54.62	42.40	38.14	35.09	30.54	29.14
speedup: brute-force/(preprocess + render)	12.33	19.61	62.57	80.30	89.14	96.17	108.76	113.33
octree height	3	4	5	6	7	7	7	7
octree space (MB)	0.192	0.312	0.969	3.207	3.984	8.948	18.991	22.877
[min, max] numbers of points per leaf octant	[24,717]	[24,498]	[1,100]	[1,50]	[1,40]	[1,32]	[1,32]	[1,32]
	Stanford Bunny point set of 35947 point elements							
preprocessing time (sec.)	0.24	0.60	4.27	4.50	4.53	4.57	4.56	4.58
direct surface rendering time (sec.)	249.17	121.83	52.06	47.69	46.70	45.92	45.20	44.38
speedup: brute-force/(preprocess + render)	104.81	213.51	464.06	500.87	510.26	517.74	525.33	533.92
octree height	5	6	7	7	7	7	7	7
octree space (MB)	2.064	6.361	65.181	68.502	69.039	69.573	70.052	70.488
[min, max] numbers of points per leaf octant	[2,999]	[1,500]	[1,375]	[1,375]	[1,375]	[1.375]	[1,375]	[1,375]

Table 2: *Octree performance data for rendering two random point clouds and the Stanford bunny, obtained on the same computer as Table 1. The brute force rendering times for the three point sets are 2454.21, 3423.82 and 26140.58 sec. respectively.*

[Nie93] NIELSON G. M.: Scattered data modeling. *IEEE Computer Graphics and Applications 13*, 1 (1993), 60–70.

[PASS01] PASKO A., ADZHIEV V., SCHMITT B., SCHLICK C.: Constructive hypervolume modeling. *Graphical Models 63*, 6 (2001), 413–442.

[PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: surface elements as rendering primitives. In *Computer Graphics (Proc. SIGGRAPH 2000)* (2000), pp. 335–342.

[RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: a multiresolution point rendering system for large meshes. In *Computer Graphics (Proc. SIGGRAPH 2000)* (2000), pp. 343–252.

[RV82] REQUICHA A. A. G., VOELCKER H. B.: Solid modeling: a historical summary and contemporary assessment. *IEEE Computer Graphics and Applications 2*, 2 (1982), 9–24.

[SD01] STAMMINGER M., DRETTAKIS G.: Interactive sampling and rendering for complex and procedural geometry. In *Proc. Eurographics Workshop on Rendering Techniques* (2001), pp. 151–162.

[SJ00] SCHAUFLER G., JENSEN H. W.: Ray tracing point sampled geometry. In *Proc. Eurographics Workshop on Rendering Techniques* (2000), pp. 319–328.

[SK97] STOLTE N., KAUFMAN A.: Discrete implicit surface models using interval arithmetic. In *Proc. 2nd CGC Workshop on Computational Geometry* (Durham, 1997).

[vHJ*04] VON RYMON-LIPINSKI B., HANSSEN N., JANSEN T., RITTER L., KEEVE E.: Efficient point-based isosurface exploration using the span-triangle. In *Proc. IEEE Visualization* (2004), pp. 441–448.

[Wat00] WATT A.: *3D Computer Graphics*, 3rd ed. Addison-Wesley, 2000.

[WC01] WINTER A. S., CHEN M.: vlib: a volume graphics API. In *Proc. Volume Graphics 2001* (2001), pp. 133–147.

[Wes90] WESTOVER L.: Footprint evaluation for volume rendering. *Computer Graphics (Proc. SIGGRAPH 90) 24*, 4 (August 1990), 367–376.

[WGG99] WYVILL B., GUY A., GALIN E.: Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (1999), 149–158.

[WK93] WANG S. M., KAUFMAN A.: Volume sampled voxelization of geometric primitives. In *Proc. IEEE Symposium on Volume Visualization* (October 1993), pp. 78–84.

[WMW86] WYVILL G., MCPHEETERS C., WYVILL B.: Data structures for soft objects. *The Visual Computer 2*, 4 (1986), 227–234.

[ZPBG01] ZWICKER M., PFISTER H., BAAR J., GROSS M.: EWA volume splatting. In *Proc. IEEE Visualization* (2001), pp. 29–36.