VOLUMETRIC-CSG
— A MODEL-BASED VOLUME VISUALIZATION APPROACH

Shiaofen Fang[1]

Department of Computer & Information Science, Indiana University Purdue University Indianapolis

Rajagopalan Srinivasan[2]

Institute of Systems Science, National University of Singapore

ABSTRACT

This paper presents a *Volumetric-CSG* (VCSG) method for the representation of volumetric objects and their operation, such as transformations, cutting and Boolean operations. A new volume rendering algorithm is developed for visualizing the VCSG models. The algorithm first generates optimal target blocks for efficient model operations by adaptive subdivision of the target volume, and then volume renders the target blocks using a template-based octree projection process. Both the raycasting block projection and hardware assisted 3D texture mapping rendering methods are implemented.

Keywords – Volume rendering, Volumetric modeling, Raycasting, 3D texture mapping, CSG.

1 INTRODUCTION

1.1 Model-based Volume Visualization

Volume visualization[Kau91] is a technique to abstract, interpret and display large volume datasets that would otherwise be too complex to comprehend. Such volume datasets can come from a wide range of sources such as computed tomography (CT) and magnetic resonance imaging (MRI), or computer generated simulation data (e.g. in computational fluid dynamics). Current volume visualization algorithms are, however, mainly used fro viewing the structures of the entire volume dataset. Interactions with individual objects embedded within the volume dataset are very difficult due to the lack of a volumetric modeling scheme. On the other hand, volume visualization, together with object manipulations, is highly desirable in many practical applications. For example, in surgical planning and simulation[PVML96, MF89], users must be able to perform computer simulated operations, such as cutting, drilling, bending and repositioning, on individual bones, tissues or tumors, and visualize them at every step of the operation. The ability to extract, measure, move, track and alter featured objects is also essential to the interrogative visualization applications in many science and engineering fields.

Volume visualization techniques differ greatly from those used in conventional computer graphics dealing mostly with synthetic geometric models. The modeling of synthetic geometric objects is well developed (e.g. solid and surface modeling), and can support various object representation requirements and manipulation operations. Unfortunately, the same cannot be said about volumetric data. This is because current volume visualization algorithms are mostly dataset-based, i.e. the algorithms are designed for the rendering of complete volume datasets in regular format, with little or no structural information in the object level to support object operations. In other words, they do not have an associated volumetric modeling scheme for efficient object representation, construction and manipulation. Clearly, a systematic volumetric modeling scheme and associated rendering algorithms are required to explore the full potential of volume visualization.

1.2 Related work

Previous work on volumetric modeling is very limited. Most of the 3D object modeling efforts concentrate on the geometric shape modeling of synthetic objects, including curve/surface modeling and solid modeling[Far93, Hof89]. Their underlying representation methods are only capable of representing the surface boundaries, rather than the complete volumetric information. Geometric modeling and the corresponding surface graphics techniques are, in general, not sufficient for the modeling and rendering of volumetric objects because of their representational limitations, though some of these techniques, such as the *Constructive Solid Geometry* (CSG) method in solid modeling[Req80, RV85], can be extended to solve volumetric modeling problems as described in this paper.

Two common approaches in volume visualization are surface rendering and volume rendering. In surface rendering[LC87], iso-surfaces are extracted from the volume data based on constant field values, and then rendered and manipulated as polygons. Most existing surgical planning/simulation systems are based on surface rendering[PVML96, VMW83]. Since iso-surfaces

[1] 723 W. Michigan St., SL 280, Indianapolis, IN 46202-5132. e-mail: sfang@cs.iupui.edu

[2] Heng Mui Keng Terrace, Kent Ridge, Singapore 119597.

represent only a 2D subset of the object, this approach cannot effectively convey the complete volumetric information. Furthermore, an iso-surface normally contains a very large number of tiny polygons, and is, therefore, very inefficient and inaccurate for modeling purposes. Limited volume manipulations can, however, be realized by extending the thresholding approach to include semi-boundary and shell volumes[UO91, UO93], interval set[Guo95], or interval volumes[FMST96]. Volume rendering, on the other hand, attempts to directly display the entire volume dataset through semi-transparent images. This allows the viewers to peer inside the volume space for better comprehension and exploration of the dataset. Volume rendering algorithms can be roughly classified into two categories: the image space approach such as raycasting algorithms[Lev90, DH92, YK92], and the object space approach such as splatting and cell projection[UK88, Wes90, WG91]. Hybrid algorithms that combine the advantages of both approaches have also been developed[LL94, SFH97].

Volume graphics is a new 3D graphics paradigm proposed by Kaufman, Cohen and Yagel[KCY93]. It employs volume visualization and 3D raster techniques to display and manipulate 3D synthetic objects by converting the geometric models into a uniform volume buffer. The use of volume graphics method for CSG solid models has also been discussed in [SY95] and [WK94]. Apparently, volume graphics paradigm may also be applied to the rendering and manipulation of volumetric objects with non-binary field values. The advantages of this approach are algorithm uniformity and the view independence of the volume buffer. But since the volume buffer is a large regular discrete 3D array of voxels, a uniform model voxelization[KS86, Kau87a, Kau87b] for every step of the model operation is extremely inefficient for interactive applications. Furthermore, frequent model discretization and resampling in raster manipulations also generate unnecessary cumulative errors.

1.3 Overview

In this paper, we describe a *Volumetric-CSG* (VCSG) modeling scheme and its rendering algorithm for the interactive manipulation and visualization of volumetric models. A VCSG model, as an extension to the *Constructive Solid Geometry* (CSG) model in solid modeling, represents a volumetric object and its construction/manipulation process in a tree structure. The leaf nodes consist of the initial volume datasets. Intermediate nodes represent either unary operations, such as transformations, or binary operations, such as Boolean operations. A rendering algorithm is also developed for visualizing VCSG models. In this algorithm, instead of reconstructing the entire volume into a volume buffer as required in volume graphics, we generate optimal target blocks in the volume space directly from a VCSG model, by adaptive target volume subdivision, for efficient direct rendering. An octree data structure (called target octree) is used to keep track of this subdivision process. Template-based octree projection methods are employed to render all the target blocks using a raycasting formulation and hardware assisted 3D texture mapping, respectively. Section 2 presents the VCSG modeling scheme. The VCSG rendering algorithm is described in Section 3. Some implementation results are given in Section 4. Section 5 concludes the paper with additional remarks and future work.

2 VOLUMETRIC MODELING

A volumetric object is essentially a (irregular) subset of a regular volume field with scalar or vector field values. Since traditional geometric modeling is mainly concerned with the continuous surface boundaries of 3D objects, it is, in general, not suitable for the representation of volumetric objects that are discrete and volumetric in nature. But extensions of some solid representation methods may be made for volumetric objects. In this paper, we consider the volumetric extension of the *Constructive Solid Geometry* (CSG) representation in which solid primitives are combined into a more complex solid object by geometric transformations and Boolean operations. In volumetric environment, the primitives can be simply represented as regular volume datasets, which can come from either segmentation or separate data sources. For large sparse volumes, multiple volume primitives can be created from each dataset by subdividing the volume into subvolumes using data structures such as octree[SFH97].

2.1 Volumetric-CSG

Various operations, such as transformations and Boolean operations, can be applied to the initial primitive volumes to construct a more complex volumetric object. Model modification and manipulation may also be achieved by changing the operations or their parameters. Due to the regular sampling pattern in volume datasets, reformatting the volumetric object into a regular dataset after each operation is very time-consuming, particularly for interactive applications where the modification of the model can be fairly frequent. To avoid explicit volume reconstruction at every step of the operation, a *Volumetric-CSG* (VCSG) method is used to represent the volumetric construction process with a hierarchical sequence of operations. Conceptually, VCSG is a tree structure, as shown in Figure 1. The leaf nodes of the VCSG tree are the initial primitive volumes. Each intermediate node contains information on an operation to be performed on its child node(s), and therefore represents the resulting subvolume of this operation. With VCSG representation, object operations are simple nodal operations of the VCSG tree. The voxel level data access and operations are not performed in the VCSG modeling stage, but are, instead, integrated into the rendering process.

A volumetric operation is defined as a mapping from one or two volumes (called the source volume(s)) to a new volume (called the target volume). We call operations with only one source volume *unary* operations, and the ones with two source volumes *binary* operations. For the simplicity of discussion, we assume that all volumes are defined on a cubic 3-D domain $\mathcal{D}$ with scalar voxel values defined over $S \subseteq R$, where

$$\mathcal{D} = \{(x, y, z) \in R^3 \mid x, y, z \in [0, L]\}$$

2.2 Unary Operations

For an *unary* operation with a source volume:

$$V_1 = \{(P, I_1(P)) \mid P \in \mathcal{D}; I_1(P) \in S_1 \subseteq R\}$$

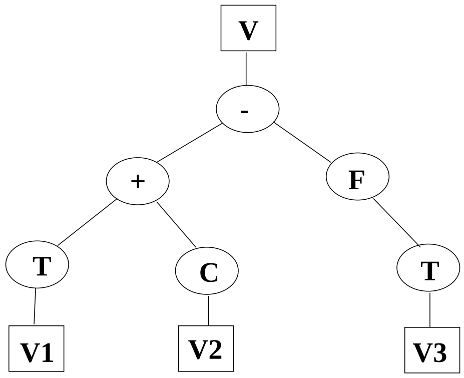

Figure 1: The VCSG representation for $V = (T(V_1) + C(V_2)) - F(T(V_3))$

the target volume is defined as:

$$V = \{(P, I(P)) \mid P \in \mathcal{D}; I(P) = f(I_1, P)\}$$

where $f() : \mathcal{D} \to R$ is a function associated with each *unary* operation. The following is a list of useful *unary* volumetric operations:

- **Affine transformation: "T"**. It can include translation, rotation, scaling, shearing and mirroring (reflection). Its VCSG node contains the 4×4 matrix, M, of the transformation and its inverse M^{-1}. Thus:

  $$V = \{(P, I(P)) \mid P \in \mathcal{D}; I(P) = I_1(M^{-1}P)\}$$

- **Transfering: "F"**. This operation does not explicitly change the geometry of the object, but, instead, alters the voxel values through a transfer function $g : R \to R$. It enables the use of different transfer functions for different subvolumes to create various rendering effects. The VCSG node contains either the parameters of the function g (e.g. the coefficients of a linear or quadratic function) or a pointer to a lookup table. The target volume is defined as:

  $$V = \{(P, I(P)) \mid P \in \mathcal{D}; I(P) = g(I_1(P))\}$$

- **Cutting: "C"**. It cuts away part of the object using a plane. The plane equation, $Ax + By + Cz + D = 0$, is stored in the VCSG node. The resulting target volume is defined as:

  $$V = \{(P, I(P)) \mid P \in \mathcal{D}; \quad I(P) = \begin{cases} I_1(P) & \text{if } Ax + By + Cz + D \geq 0 \\ 0 & \text{otherwise} \end{cases} \}$$

- **Deformation: "D"**. Volumetric objects are often deformable (e.g. the organs and tissues in medical applications), and deformation should, therefore, be an essential operation in volumetric modeling. In our approach, volume deformation is modeled as a 3D morphing function by landmark interpolation. Landmark movements are either defined interactively or derived from prescribed deformation forces or rules (e.g. physical modeling or biological growth pattern). A scattered data interpolation method will then generate a globally smooth morphing function on the volume space of the object to be deformed. Details about the landmark-based morphing approach can be found in [FRR96] and [FSHR96].

2.3 Binary Operations

Binary operations are basically Boolean operations, including Union ("+"), Intersection ("*") and Subtraction ("-"). For solid objects, they are defined as regularized set operations[Req80]. Boolean operations of synthetic objects in volume graphics are similarly defined as *voxblt* operations[Kau92]. But for volumes of scalar or vector fields, a *binary* mapping function is attached to each operation to resolve the field values in the two volumes involved. Let us define the two source volumes in a *binary* operation as:

$$V_1 = \{(P, I_1(P)) \mid P \in \mathcal{D}; I_1(P) \in S_1 \subseteq R\}$$

$$V_2 = \{(P, I_2(P)) \mid P \in \mathcal{D}; I_2(P) \in S_2 \subseteq R\}$$

and the target volume is:

$$V = \{(P, I(P)) \mid P \in \mathcal{D}; I(P) = I_1(P) <op> I_2(P)\}$$

where $< op >$: $R \times R \to R$ is the *binary* mapping function corresponding to a *binary* operation.

- **Union: "+"**.

  $$V = \{(P, I(P)) | \; P \in \mathcal{D}; I(P) = I_1(P) <+> I_2(P)\}$$

  where

  $$x <+> y = \begin{cases} x & \text{if } y \notin S_2 \\ y & \text{if } x \notin S_1 \\ u(x,y) & \text{otherwise} \end{cases}$$

  Union operations are normally used to add one object to another, to mix different materials together, or to combine multiple modality images (e.g. CT and MRI scans). Function $u(x, y)$ defines a composition of two 3D images. For 3D RGBA images, an analog can be easily made from the 2D image compositing techniques[PD84], using operators such as **over**, **plus** and **out**. For volumes of scalar or vector fields, this function is often more application dependent, and may be used to achieve many useful effects. For instance, $u(x, y) = \max(x, y)$ implies that higher density material has better visibility, while $u(x, y) = tx + (1 - t)y$ provides a more flexible blending of the materials. $u(x, y)$ may also be defined by a lookup table, i.e. $u(x, y) = \mathbf{lut}(x, y)$ for even greater flexibility. Yet another example is the fuzzy set operations used in modeling volume-sampled objects to resolve overlapping soft boundaries[WK94].

- **Intersection: "*"**.

  $$V = \{(P, I(P)) \mid P \in \mathcal{D}; I(P) = I_1(P) <*> I_2(P)\}$$

  where

  $$x <*> y = \begin{cases} 0 & \text{if } x \notin S_1 \text{ or } y \notin S_2 \\ u(x,y) & \text{otherwise} \end{cases}$$

  It generates the intersecting volume of the two source volumes. $u(x, y)$ can be defined exactly the same as in **Union**. **Intersection** operation is very useful for applications such as collision detection and fuzzy boundary analysis.

- **Subtraction: "-".**

  $V = \{(P, I(P)) | P \in \mathcal{D}; I(P) = I_1(P) <-> I_2(P)\}$

  where

  $$x <-> y = \begin{cases} x & \text{if } y \notin S_2 \\ 0 & \text{if } x \notin S_1 \\ u(x,y) & \text{otherwise} \end{cases}$$

  This operation subtracts the second object from the first one. $u(x, y)$ can also be defined the same way as in the **Union** operation. When $u(x, y) = 0$, it becomes the Boolean difference operation for solid objects. **Subtraction** is very useful in surgical simulation (e.g. drilling a hole from a bone).

2.4 Bounding Boxes

To facilitate efficient searching of regions of interest in a VCSG model, a hierarchy of bounding boxes is built within the data structure. To do so, we compute and store in each node of the VCSG tree a bounding box of the irregular subvolume the node represents. As to be seen in the next section, such bounding box hierarchy plays an important role in our rendering algorithm.

Since the bounding boxes of all initial primitive volumes are already defined by the sizes and positions of the volumes, the bounding boxes in the intermediate nodes of the VCSG tree can be directly computed from their child bounding boxes. This works as follows:

- **Transformation**: No change. But testing against the bounding box requires a prior inverse transformation of the object to be tested.
- **Transfering**: No change.
- **Cutting**: The child bounding box is cut and a new bounding box is computed.
- **Deformation**: The bounding box is estimated by morphing the eight vertices of the child bounding box and some additional sample points chosen on the six faces of the bounding box. This approach is not guaranteed to be accurate, but is simple and quite effective in general. A safer approach would be to compute the upper-bound of the Jacobian of the morphing function to estimate the maximum change of the bounding box. But we found that this often leads to overly large bounding bounding box and can consequently affect the algorithm's performance.
- **Union** or **Intersection**: compute the bounding box of the union or intersection of the two child bounding boxes.
- **Subtraction**: simply retain the bounding box of the first child node.

3 VOLUME RENDERING A VCSG MODEL

In order to use a regular volume rendering algorithm to visualize a VCSG model, a regular volume dataset, called the target volume, has to be reconstructed to encapsulate the VCSG model. In other words, every voxel of the target volume needs to be computed through the entire hierarchy of operations in the VCSG tree. This can be very time-consuming, especially for applications requiring interactive model manipulations, where changes of the VCSG tree is fairly frequent. In this section, we present a new model-based volume rendering algorithm that can be directly applied to a VCSG model, and efficiently carry out the modeling operations within the rendering process. The basic idea is to first adaptively subdivide the target volume space into various sized target blocks to obtain the optimal computational paths for the target blocks in the VCSG tree. These target blocks are then projected onto the projection plane to generate the final image, using either a raycasting formulation or the hardware assisted 3D texture mapping. The algorithm operates in three steps: target volume subdivision, target block sorting and target block projection.

3.1 Target Volume Subdivision

Since different regions of the target volume may involve only parts of the VCSG tree, it is important to identify such separate regions in the target volume and their respective paths in the VCSG tree to optimize the model computation. To do so, the algorithm employs an adaptive target volume subdivision process. The process starts from a uniform subdivision of the target volume space, represented as a complete target octree. An optimal test is applied to each leaf node of the target octree to determine whether its computational path in the VCSG tree is optimal. If not, the node is subdivided into eight smaller blocks, and the process continues recursively. Otherwise, the node is considered a satisfactory target block, i.e. no further subdivision of this block is necessary. The optimal test uses the bounding box hierarchy and traverses the VCSG tree in a depth-first order. For each VCSG node it encounters, the target block to be tested is compared with the bounding box of the node based on the following criteria:

1. If the block is entirely within the bounding box, no subdivision is necessary, and the traversal continues with its subtree. If the block is entirely outside the bounding box, the traversal skips the node's subtree, since it is unrelated to this block. The sequence of the in/out results from the above testing, along with the corresponding operations, is recorded in the block.
2. If the block is only partially inside the bounding box, it is subdivided into eight new blocks, and its previous testing results are passed on to the new blocks, i.e. the VCSG traversal for each new block starts from where the subdivision occurred.
3. For a **Deformation** node, the target block is subdivided until the morphing function can be approximated by trilinear interpolation within the target block. Details of this process has been described in [FSHR96].
4. When the size of the block reaches a pre-defined minimum (one voxel by default), no further subdivision is possible, and the block is simply assigned to be either inside or outside the bounding box, based on the position of its center point.

The above VCSG traversal process generates a sequence of VCSG nodes that the target block may intersect. This node sequence is called the optimal computational path of the target block. Following this optimal path, the algorithm reconstructs a subvolume for every target block. Basically, for each voxel in the target block, its values in all participating objects in the optimal path are fetched from their octree volumes, and computed using the corresponding operations recorded in the optimal path. The result of this computation is a regular subvolume for the target block. This target volume subdivision process only needs to be done once for every VCSG tree, and the result can be used for the rendering in all viewing angles. Furthermore, the bounding box hierarchy also allows for flexible local modifications. For instance, changes of some bounding boxes in the hierarchy (due to a small change of the VCSG tree) will cause the retesting of the target blocks against the changed bounding boxes only.

3.2 Target Block Sorting

The next step of the rendering algorithm is to sort the target blocks, created from the subdivision process, in either a front-to-back or a back-to-front order, depending on the rendering method (to be discussed in next subsection). Since the target blocks are already organized in the target octree, sorting for a given viewing direction is fairly simple. The signs of the three coordinates of the viewing vector determine the order in which the eight child octants of each node of the target octree are visited[FSHR96]. Based on this view-dependent child access order, a view-dependent depth-first traversal of the target octree will automatically pick the target blocks in the required order.

3.3 Target Block Projection

In this step, the target blocks are projected, in the required order, onto the projection plane to generate the final image. Since there can be thousands of target blocks, projecting all the blocks independently can be quite expensive. For parallel projection, however, a template-based block projection approach[SFH97] may be used to take advantage of the fact that all blocks have the same shape and orientation, and only a number of fixed sizes (in different levels of the target octree). Thus, the computation may be done for only one sample block with each size, and the result can be saved into a template, and later "pasted" to all other blocks of the same size. In the following, we will only discuss parallel projection rendering. We are currently still working on the extension of this approach for perspective projection. In our approach, two different image formation methods are used in block projection: the raycasting method and the 3D texture mapping method. Since these two projection methods have previously been reported in [SFH97], we will only present the basic ideas here. Details and the handling of complications can be found in [SFH97].

3.3.1 Raycasting

A front-to-back sorting order is used for raycasting projection, which is essentially a mini-raycasting process within each block. The block-raycasting results from all the blocks are blended and composited together to generate the rendered image Each block-raycasting result is defined by (see Figure 2):

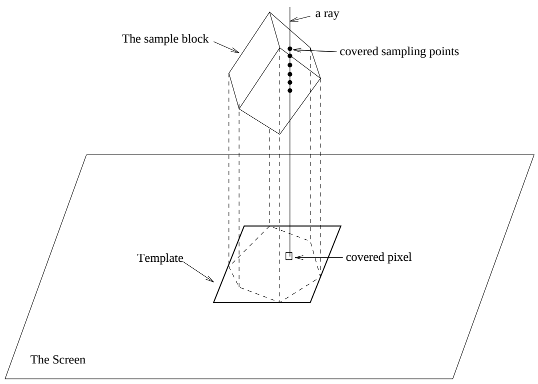

Figure 2: The template for block projection

1. *covered pixels*: the pixels that the block projects to;
2. *covered sampling points*: the raycasting sampling points along the rays shooting from the covered pixels that are within the block.

Using the template-based projection approach, the set of *covered pixels* and *covered sampling points* for blocks of the same size should be a simple translation of each other, i.e. for each size, the *covered pixels* and *covered sampling points* should be identified only once, saved in a template, and later "pasted" to all other blocks of the same size. A template is defined as a 2D array containing the image area of the *covered pixels*. Each entry of the 2D array contains simply two index numbers indicating the interval of the *covered sampling points*. To build the template, a sample block for each block size is used. The 2D image projection of the bounding box of the rotated sample block forms a rectangular image area containing all the *covered pixels* of the sample block. For every ray shooting from each *covered pixel*, the ray segment that is within the block is computed by normal ray/block intersection. The interval of the raycasting sampling points within this ray segment is stored in the template as the *covered sampling points*. For each block in the sorted block list, a displacement vector from the sample block is first computed. A template of appropriate size is translated by the displacement vector to obtain the *covered pixels* and *covered sampling points* of the block to perform raycasting within the subvolume of the block obtained earlier.

3.4 3D Texture Mapping

To use 3D texture mapping hardware, often available in high-end graphics workstations, a back-to-front projection order is normally required. For each target block, its subvolume is first defined as a 3D texture block and bound to the hardware as the current texture map. To render this block, polygons generated by intersecting a sequence of Z-planes and the block need to be sent, together with texture coordinates, to the texture mapping hardware in a back-to-front order.

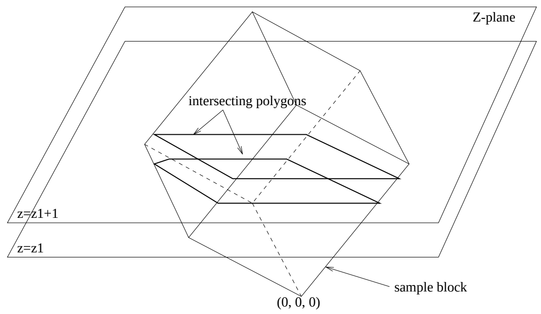

Figure 3: The template for Z-plane/block intersection

Using the template-based approach, similar to the raycasting method, the result of the Z-plane/block intersection for one sample block should be usable by all other blocks of the same size with a simple translation. As shown in Figure 3, the template is defined as a list of intersection polygons of consecutive Z-planes and the sample block. These polygons can later be translated to generate the intersection polygons for all other blocks of the same size. Texture coordinates at the polygon vertices are obtained by trilinear interpolation of the texture coordinates at the block vertices.

4 IMPLEMENTATIONS

Both the raycasting and 3D texture mapping versions of the rendering algorithm have been implemented in C++, and tested on an SGI R4400 workstation running at 150MHz with 64 Mbytes main memory. The texture memory on this machine is 4 Mbytes.

Some test results are shown in Figure 4, in which (c)(j)(k) are rendered with 3D texture mapping, and the rest are raycasting images. All original datasets have size 256^3. Figure 4(a) shows a craniofacial patient's CT data. In Figure 4(b), the tumor is subtracted, and a reflection of the left (normal) side of the face is then merged (using the left side's values) into the right side to generate a normal appearance. Multiple object rendering examples with different transfer functions are shown in Figure 4(c)(d). Figure 4(c) shows the eight octant subvolumes (slightly moved) of a human CT dataset, and Figure 4(d) is a CT dataset of two engine parts. The rest of the images show various binary operation results of a human head CT dataset and a synthetic knot dataset (Figure 4(e)). Figure 4(i) is the result of an intersection operation using the human head's values. Figure 4(f) shows the same intersection operation, but rendered with the knot's values. Their union operation is shown in Figure 4(h). Figure 4(g)(j)(k) show a similar set of the binary operations, except that the human head now is morphed by moving its landmarks to the corresponding biological landmarks in a Chimpanzee's head.

In table 1, the timings for some of the test examples are given. The time taken to build target blocks and the time for each subsequent rendering are given separately since the target blocks need to be built only once for each VCSG model. To save memory space, an octree data structure is used in storing the original volumes. Total memory usage for each example given in the table includes the octree volumes, target blocks and their subvolumes, and the templates. The rendering time is partitioned into three parts: building templates, block projection, and resampling/compositing/shading (for raycasting) or texture loading/binding time (for 3D texture mapping). For raycasting rendering, five light sources and depth cue are used for shading, and trilinear interpolation is used for resampling. For 3D texture mapping rendering, the block projection portion also includes the hardware texture mapping time.

Table 1: Rendering Performance Data

Dataset	rendering time (sec)				Target block building	memory (MB)
	build template	block projection	resam/blending/shading or: texture binding	total		
Fig.6 (b)	0.019	1.35	9.27	10.639	7.3	4.23
Fig.6 (c)	0.025	0.092	0.33	0.447	10.24	9.61
Fig.6 (d)	0.016	1.21	5.68	6.902	9.76	6.36
Fig.6 (f)	0.021	0.72	3.62	4.36	7.21	14.18
Fig.6 (h)	0.027	2.32	10.45	12.797	6.81	14.18
Fig.6 (j)	0.026	0.32	0.51	0.856	11.23	16.31
Fig.6 (k)	0.023	0.93	0.71	1.66	18.51	18.27

5 CONCLUSIONS

We presented a *Volumetric-CSG* modeling scheme and the associated rendering algorithm to support a model-based volume visualization paradigm. The VCSG scheme provides a compact and flexible representation method for volumetric objects and their construction and manipulation operations. The VCSG rendering algorithm can directly and efficiently visualize a VCSG model by optimizing the computational paths in the VCSG tree for different subvolumes in the target volume space. The template-based projection approach ensures that subdividing the target volume into many small blocks does not significantly affect the rendering efficiency.

In addition to providing a feasible modeling/rendering scheme for volume graphics, this approach may also have direct applications in many practical problems such as interrogative scientific visualization, and surgical planning and simulation. In the future, we plan to explore the full potential of the VCSG modeling and rendering in various applications, particularly in medical field. For example, using this technique in craniofacial surgical planning and simulation, most surgical operations can be modeled and rendered easily. In tumor sterotaxy[Kel91], as another example, this technique can be used to plan a tumor removal path to avoid structures, especially arteries, from being damaged.

6 ACKNOWLEDGMENTS

We would like to thank Prof. Joan Richtsmeier from the Johns Hopkins University, and Dr. Pheng Ann Heng from the Chinese University of Hong Kong for providing some of the excellent datasets. We especially want to thank Su Huang and Shalini Venkataraman for implementing many of the wonderful octree and rendering utilities. We would also like to thank Dr. Raghu Raghavan who has been extremely supportive throughout the course of this work, and Dr. Wieslaw Nowinski for some fruitful discussions.

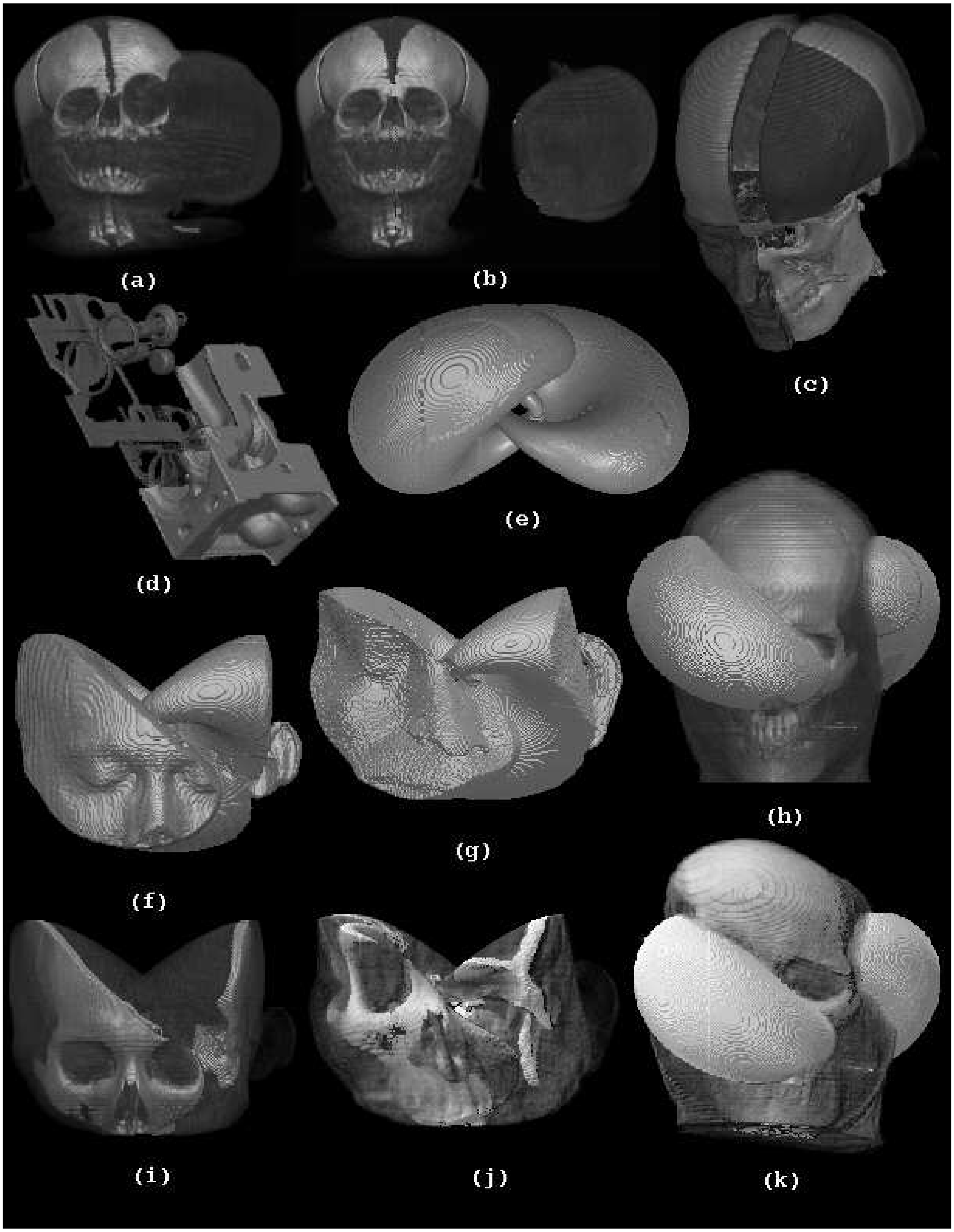

Figure 4: Rendering results of some VCSG models

REFERENCES

[DH92] J. Danskin and P. Hanrahan. Fast algorithms for volume ray tracing. In *Proc. 1992 Workshop on Volume Visualization*, pages 91–98, October 1992.

[Far93] G. Farin. *Curves and Surfaces for CAGD, A Practical Guide*. Academic Press, 1993.

[FMST96] I. Fujishiro, Y. Maeda, H. Sato, and Y. Takeshima. Volumetric data exploration using interval volume. *IEEE Trans. on Visualization and Computer Graphics*, 2(2):144–155, 1996.

[FRR96] Shiaofen Fang, Raghu Raghavan, and Joan T. Richtsmeier. Volume morphing methods for landmark-based 3D image deformation. In *: Proc. 1996 SPIE Medical Imaging, SPIE 2710, Newport Beach, CA*, pages 404–415, Febuary 1996.

[FSHR96] Shiaofen Fang, Rajagopalan Srinivasan, Su Huang, and Raghu Raghavan. Deformable volume rendering by 3D texture mapping and octree encoding. In *: Proc. of IEEE Visualization'96, San Francisco*, pages 73–80, October 1996.

[Guo95] Baining Guo. Interval set: A volume rendering technique generalizing isosurface extraction. In *Proc. of Visualization 95*, October 1995.

[Hof89] Christoph M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann Publishers, 1989.

[Kau87a] Arie Kaufman. An algorithm for 3D scan-conversion of polygons. In *Eurographics '87*, pages 197–208. North-Holland, August 1987.

[Kau87b] Arie Kaufman. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. In *SIGGRAPH '87*, volume 21, pages 171–179, July 1987.

[Kau91] Arie Kaufman. *Volume Visualization*. IEEE Computer Society Press, 1991.

[Kau92] Arie Kaufman. The *voxblt* engine: A voxel frame buffer processor. In A. A. M. Kuijk, editor, *Advances in Graphics Hardware III, Apringer-Verlag*, pages 85–102, 1992.

[KCY93] Arie Kaufman, D. Cohen, and Roni Yagel. Volume graphics. *IEEE Computer Graphics and Applications*, 13:51–64, 1993.

[Kel91] Patrick J. Kelly. *Tumor Stereotaxis*. Philadelphia, Saunders, 1991.

[KS86] Arie Kaufman and Eyal Shimony. 3D scan-conversion algorithms for voxel-based graphics. In *Proceedings of 1986 Workshop on Interactive 3D Graphics*, pages 45–75, October 1986.

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

[Lev90] Marc Levoy. Efficient ray tracing of volume data. *ACM Trans. on Graphics*, 9(3):245–261, July 1990.

[LL94] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. *SIGGRAPH'94*, pages 451–458, 1994.

[MF89] C. N. McEwan and K. Fukuta. Recent advances in medical imaging: Surgery planning and simulation. *World Journal of Surgery*, 13(4):343–348, 1989.

[PD84] Thomas Porter and Tom Duff. Compositing digital images. *Computer Graphics, SIGGRAPH'84*, 18(3):253–259, July 1984.

[PVML96] V. Patel, M. Vannier, J. Marsh, and L. Lo. Assessing craniofacial surgical simulation. *IEEE Computer Graphics and Applications*, 16(1):46–54, Januaryr 1996.

[Req80] A. A. G. Requicha. Representation for rigid solids: Theory, methods and systems. *Computing Surveys*, 12(4):437–464, December 1980.

[RV85] A. A. G. Requicha and H. B. Voelcker. Boolean opeartions in solid modeling: Boundary evaluation and merging algorithms. *Proc. IEEE*, pages 30–44, 1985.

[SFH97] Rajagopalan Srinivasan, Shiaofen Fang, and Su Huang. Volume rendering by template-based octree projection. In *8th Eurographics Workshop on Visualization in Scientific Computing*, April 1997.

[SY95] Naeem Shareef and Roni Yagel. Rapid previewing via volume-based solid modeling. In Chris Hoffman and Jarek Rossignac, editors, *Solid Modeling '95*, pages 281–292, May 1995.

[UK88] Craig Upson and Michael Keeler. V-buffer: Visible volume rendering. *Computer Graphics, SIGGRAPH'88*, 22(4):59–64, August 1988.

[UO91] J. K. Udupa and D. Odhner. Fast visualization, manipulation, and analysis of binary volumetric objects. *IEEE Computer Graphics and Applications*, 11(6):53–62, 1991.

[UO93] J. K. Udupa and D. Odhner. Shell rendering. *IEEE Computer Graphics and Applications*, 13(6):58–67, 1993.

[VMW83] M. W. Vannier, J. L. Marsh, and J. O. Warren. Three dimensional computer graphics for craniofacial surgical planning and evaluation. *Computer Graphics, SIGGRAPH '83*, 17(3):263–273, July 1983.

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

[WG91] Jane Wilhelms and Allen Van Gelder. A coherent projection approach for direct volume rendering. *Computer Graphics, SIGGRAPH'91*, 25(4):275–284, July 1991.

[WK94] Sidney Wang and Arie. Kaufman. Volume-sampled 3D modeling. *IEEE Computer Graphics and Application*, 14:26–32, September 1994.

[YK92] Roni Yagel and Arie Kaufman. Template-based volume viewing. In *Eurographics'92*, pages 153–167, 1992.