VOLUMETRIC CSG GRAPH AND ITS VOXELIZATION

Sema Koç[1], Ulus Çevik[2]

e-mail: skoc@gantep.edu.tr
Department of Electrical and Electronic Engineering, University of Gaziantep, 27310 Gaziantep, Turkey

Keywords: volume scene tree, blist, voxelization, slice-sweep

ABSTRACT

This paper introduces a new representation for the volumetric CSG scene graph. The Boolean list representation is used for the volume scene tree, which is differing from the traditional tree representation in the way that any volume tree expression can be evaluated without using recursion or stack. The scene evaluation is carried out by a slice-sweep voxelization algorithm.

I. INTRODUCTION

A volume scene can be constructed from various type of geometric or volumetric object such as curve, surface, solids and CT data set. A popular technique in scene composition is scene graph. It organizes geometric object and rendering parameters in tree-like hierarchical structure [1].

Evaluation of scene graph is carried out by voxelization algorithm. In recent years, a number of curve and surface voxelization algorithms have been proposed [2-6]. Solid voxelization on the other hand has not been sufficiently studied. Solid objects are normally represented by their boundary surfaces. Since the interior of solid object is not explicitly represented, solid voxelization is difficult and requires inside test for each voxel involved [7].

In [8], the evaluation of volume scene graph is done in a brute-force manner, i.e. for each point in volume space, recursively computing the value of scene graph starting from the root. This is very expensive procedure, and can only be used as preprocessing step, which is not practical for interactive applications [1].

In [7], a hardware voxelization algorithm was described. Although that algorithm is fast for small scale interactive applications, its performance have been limited by the need of generating intermediate object for each Boolean operation node and hardware restriction in blending function combination. This voxelization algorithm was improved by using point classification map for Boolean operations based on a frame buffer color encoding scheme [9]. But this algorithm only provides a binary volume, which may not provide the complete information in volumetric space.

In [1], another volume pipeline is used which each slice is applied to the entire CSG tree, rather than performing volume level voxelization for each CSG node named “slice sweeping”. Here, the basic idea is to generate a slice for each object in the scene first, and than apply the blending and filtering functions on the slice in postfix order of the volume scene tree. This algorithm needs a slice data structure to store intermediate result. 2D texture was used to represent slices in the slice stack. From the view of time spent, since the slice stack operation by 2D texture mapping will occupy a large proportion in practice, more slice stack operation leads to slower volume voxelization process.

Thus, in order to reach higher speed interior operation nodes must be reduced as much as possible during the design of volume scene tree.

We propose in this paper a new representation of volume scene tree, which is named blist [10]. In Blist formulation, Boolean expression is represented as a list of primitive instead of tree, and may be evaluated in pipeline fashion, combining at each step the result of classifying the cells against the current primitive with the result of the previous classification. The fundamental breakthrough provided here lies in the fact that the result of the previous classifications does not require the list of values of cell-primitive classification results, nor a stack of intermediate result of evaluating sub-expressions. Instead, Blist passes from one primitive to the next a simple label, which may be stored using at most log(H+1) bits, where H is the height of the CSG tree [10].

Using Blist representation volume scene tree expression can be evaluated without using recursion or stack.
The scene evaluation is carried out by a slice-sweep voxelization algorithm.

II. BLIST REPRESENTATION OF VOLUME SCENE TREE

In Blist formulation of volume scene tree each primitive represent input dataset or geometric model. The basic idea of our algorithm is generate a slice for each primitive in the list first, then evaluate blist by updating a label, when its value matches the primitive's name. At the end if the label on the voxel is 1(IN name) the voxel inside the volume scene otherwise, it is out. So, Boolean expression of volume scene tree is evaluated directly, combining steps used with traditional recursive evaluation is not necessary. The details of Blist formulation is as follows:

The Blist method transforms the CSG tree into decision graph. A primitive classifies a candidate voxel and depending on the result, forwards the voxel to one or another primitive. To convert volume CSG tree to blist representation firstly, convert tree into a positive form by applying de Morgan's law (i.e A-B→ A∩B', (A∩B)→A'UB', (A')'→ A) and propagating complements to the leaves, than rotate the tree by switching the left and right children at each node make the tree left heavy and than insert the resulting tree, T, as the left-most leaf of a two level tree: (T U OUT)∩IN. Than visit the leaves from left to right for each leaf, p, fill in the corresponding fields of BL[p]. Figure 1(a) shows volume scene tree and Figure 1 (c) shows Boolean list representation of this volume scene tree.

In Figure 1 (b) the leaves of tree are visited in the left-to-right order. Then number the leaves with increasing positive integers. For each leaf, numbered p, we compute its match, M. Note that each arrow first traces a path upwards, first reaching a node N1 coming from its left child, then reaching a node N2 with a different operator still coming from the left child. If N2 operator= U, the sign of p, Blist[p].sign is inverted. Then, the arrows follow the pointer to N2.rightChild and take the left-most child at each internal node, until they reach the matching leaf M. Then, if M does not yet a have name, it grab the lowest available strictly positive integer and use it as its name. We also store that name as the stamp, Blist[p], of p. on the Figure 1 (c), we show resulting names for the leaves (inside the circle) and their stamp, Blist[p].stamp, of p. Note that four leaves do not have a name and that only one name is needed for the entire tree.

Blist represent a CSG expression as a table called BL, of primitive entries. Here BL[p].primitive Reference is the reference to the primitive's description, which includes its type, parameters, color, BL[p].sign is the binary sign value, when set, that the result of classifying a cell against the primitive should be complemented. BL[p].name is the name associated with the primitive and BL[p].stamp contains the name of the next primitive in the list that should classify the voxel that are inside the current primitive if its sign is positive, or the voxels that are outside of the current primitive, if its sign is negative.

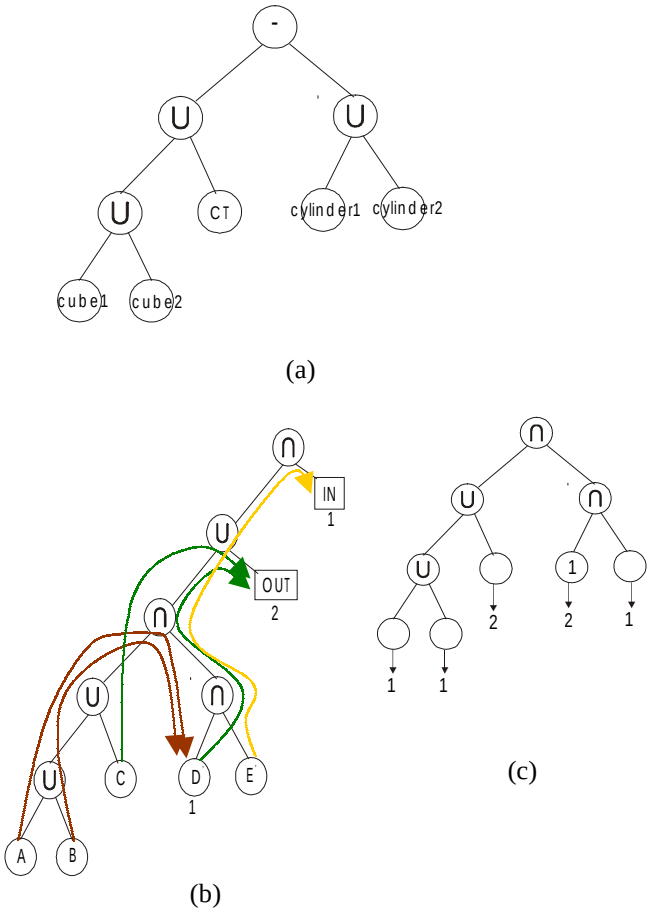

Figure 1 (a) Volume scene tree
(b) blist conversion process of volume scene tree
(c) converted tree

The Blist table resulting from conversion of volume scene tree in Figure 1 is as follows:

P	Bl[p].name	BL[p].sign	BL[p].Primitive Reference	Bl[p]. stamp
1	0	+	A	1
2	0	+	B	1
3	0	-	C	2
4	1	+	D	2
5	0	-	E	1

After filling Blist table, for the set membership classification a label is attached to each voxel and pass to the successive primitive in the Blist. When the label matches the primitive's name, the voxel is classified against the primitive. If the result of this classification matches the primitive's sign, the name on the primitive's stamp is put on the label, if not zero, name is put on the label of the voxel. At the end of this process, if the label on the voxel matches the IN's name, the voxel is inside the volume scene tree otherwise, it is out. In this representation, each voxel has a classification status with respect to all primitives and we can use the color of a primitive in its intersection with its active zone [11]. If the voxel lie in several primitives that overlap, in this case the result will be order dependent.

III. VOXELIZATION ALGORITHM

Voxelization is the process of generating volume dataset from a geometric model. Conceptually, voxelization is a set membership classification problem for all voxels in a volume against the given 3D model. In addition, from the view of the process, voxelization is a 3D scan conversion process.

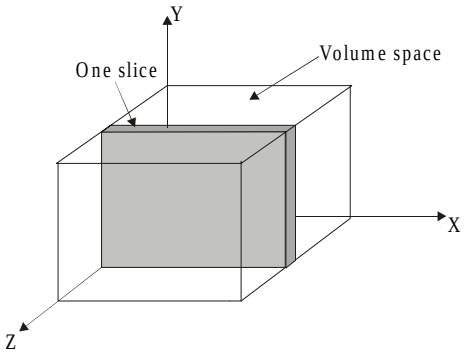

Figure 2. Volume space slicing

A cubic volume space is first defined over the CSG model. The algorithm proceeds slice by slice in a front-to-back order by moving the Z-plane, a plane parallel to the projection plane, along the viewing direction to generate slices for all primitives (Figure 2). For each slice, the algorithm defines the viewing volume of the system as a thin space between two adjacent Z-planes inside the volume boundary, and then renders each primitive within this viewing volume. When the algorithm moves from Z-plane to Z-plane, slices of the primitives are displayed and composited onto the frame buffer in a front-to-back order.
Using proper color-coding of the primitives, the algorithm can directly generate slices of the point classification indices in the frame buffer. These classification indices can then be mapped, by the PCM, to form a binary volume of the CSG model. The distance between adjacent Z-planes determines the resolution of the volume in the Z direction. The resolutions in the X and Y directions are determined by the size of the display window.

If the CSG tree has n primitives n distinct colors are assigned to the different primitives so that the color code of the jth primitive is a binary number, with the jth bit set to 1 and all other bits set to 0. For a spatial point P, the color of P with respect to the jth primitive, $C_j(P)$, is defined as the color of the jth primitive if P is inside the primitive, and o otherwise. Now, if we combine $C_1(p)$, $C_2(P)$,…$C_n(P)$ using a logical operation OR or XOR, the result, C(P), is exactly classification index of point P. Thus, for each Z-plane, algorithm generates a slice for each primitive, and then composites the slices from the primitives into one single slice of classification indices in the frame buffer using appropriate frame buffer pixel functions. This composition process is carried out as follows:

For a primitive defined by its surface boundaries, since only the boundary surfaces are drawn, we need to have a way to determine the interior points. The idea is similar to the voxelization algorithm described in [7], and is based on the principle that when a ray is shot from a pixel towards the jth primitive in the viewing direction, it has to enter the primitive object first (jth color bit becoming 1) and stay there (keeping the jth bit) until it exits the object (changing the jth bit to 0). This can be done by drawing the boundary surfaces of each primitive with a logical XOR operation (Figure 3). When a slice is complete, the frame buffer will not be cleared, i.e. the frame buffer content of the slice will be used for blending operations with subsequent slices. This way, the XOR operation will automatically set the jth color bit to 1 for all interior points, and 0 for all outside points. Since the pixel colors on the slice generated by the jth primitive has 0's at all bit positions except the jth, the XOR operation for the jth primitive will have no effect to the classifications of other primitives.

One the classification indices are generated in the frame buffer for each slice, the Point Classification Map can be applied to the frame buffer image to generate the final CSG classifications.

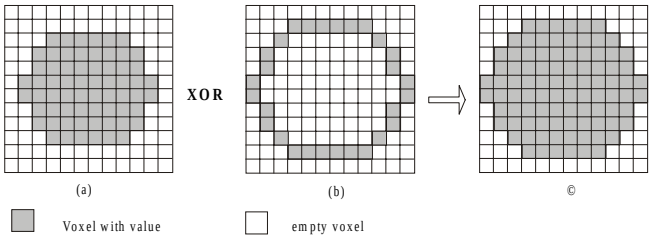

Figure 3. The process of solid voxelization for one slice by XOR
((a) the previous solid slice, (b) the boundary voxelization of current slice , (c) the final solid current slice)

IV. CONCLUSION

In this paper, we presented a new method for volume scene representation. With Blist representation results are directly obtained from volume scene expression without evaluating any Boolean operation

One approach for comparison of reported algorithm with the other algorithms in the literature is to compare the voxelization times. But comparing the execution times on different machines with different system configuration is not very meaningful. Nevertheless theoretically we can compare our algorithm with the existing methods. From the view of running time, the total running time T1 of this algorithm is log(H+1).N where H is the height of the tree and N is the number of slices. In [7], two objects in VST have to be separately converted in their own volumes before they are composted a new volume by blending operation and total running time T2 is $(M+M_i).N$ in theory, where M is the number of nodes, and Mi is the number of interior nodes. And in [1] total running time T3 is theoretically, MN. Obviously, T2>T3>T1.

From the view of occupied memory, the new algorithm reduced the storage requirement for each voxel to log(H+1) bits and there is no need to use slice stack, but in [7], the algorithm needs extra memories to store intermediate results of previous slices. That is one sliced-sized memory (2D texture memory) has to be spend for its next slice voxelization for each solid object in a VST. More specially, algorithm needs M slice buffer and in another algorithm [1], MN slice buffer is needed.

REFERENCES

1. Duoduo L., Fang S. Fast Volumetric CSG Modelling Using Standart Graphics System. 7th ACM symposium on Solid modeling and Applications, Saarbrucken, Germany, p.204-211, June 2002.
2. Cohen D, Kaufman. A Scan-conversion algorithms for linear an quadratic objects. In: Kaufman A, editor. Volume Visualization, Los Alamitos, CA: IEEE Computer Society Press, p.280-301, 1991..
3. Cohen D, Kaufman. A. 3D linear voxelization and connectivity control. IEEE Computer Graphics and Applications 17(6) 80-87,1997.
4. Huang J, Yagel R, Fillippov V. Kurzion Y. An accurate method for voxelization polygon meshes. Proceedings of the IEEE/ACM symposium on Volume Visualization p.199-226, 1998.
5. Kaufman A. Efficient algorithms for 3D scan-conversion of parametric curves, surfaces, and volumes. SIGGRAPH'87 p.171-179, 1987 .
6. Kaufman A. Efficient algorithms for 3D scan-conversion 3D polygons. Computer and Graphics; 12(2):213-219, 1988.
7. Fang S., Chen H. Hardware accelerated voxelization. Computer&Graphics 24(3):433-442. June 2000.
8. Nadeau D. Volume scene graphs. In Proc, IEEE/ACM Symposium on Volume Visualization. P. 49-56 Oct. 2000.
9. Fang S., Duoduo L. Fast CSG voxelization by Frame Buffer Pixel Mapping. In Proc. IEEE/ACM Symposium on Volume Visualization, p.43-48, Oct. 2000.
10. Rossignac J. Technical Report GIT-GVU-94-04. October 1998.
11. Rossignac J., Voelcker H. Active Zones in CSG for Accelerating Boundary Evaluation, Redundacy Elimination, Interference Detection and Shading Algorithms, ACM Transactions on Graphics, Vol.8, p.51-87,1989.
12. C.E. Prakash and S.Manohar. Volume rendering of unstructured grids-a Voxelization approach. Computer Graphics, 19(5):711-726 1995.
13. Karabassi E. A, Papaioannou, G., Theoharis T. A fast depth-buffer based voxelization algorithm, Journal of Graphics Tools, ACM, Vol.4, No.4, p.5-10,1999.
14. Çevik, U. Design of an FPGA Based Parallel Architecture Processor Displaying CSG Volumes and Surfaces. PhD. Thesis. University of Sussex, Brighton. June 1996.