Constructive Solid Geometry, Goldfeather algorithm and its followers

Computer Graphics, Josh Davis

Yanhan Lyu and Yijun Wang

README FIRST

In this project, we mainly focus on how to implement Goldfeather Algorithm to render Constructive Solid Geometry. Other improvements of Goldfeather and comparisons between Goldfeather and the improved algorithms are discussed in the paper. We have written 2 demo files. The 01CSGdemo.c uses stencil buffer to render a two-object CSG tree. It is based on the source code from Brad Grantham, and we rewrite it using GLFW and newer versions of OpenGL with shaders. The 02CSGdemo.c uses a more complex CSG rendering algorithm with more than two objects and it is written in GLUT. We did close reading of the code and tried to do the same thing using GLFW. The 03CSGdemo.c is our attempt of converting 02CSGdemo.c into GLFW, but we failed. The depth buffer and stencil buffer seem not to be working while the number of objects is correct. We write explicit comments in 01CSGdemo.c and 02CSGdemo.c to express our understanding of CSG rendering processes and the algorithm. The original authors of the source code are credited to in the files.

ABSTRACT

Constructive Solid Geometry(CSG) allows us to represent complex shapes and models with a set of boolean operations between primitives such as boxes, spheres and cylinders. It's a crucial component of many graphic softwares and have been implemented with many different

algorithms throughout the history. The Goldfeather algorithm is an improvement of previous boundary representation algorithms with binary-space partition tree. It increases the time efficiency of b-rep algorithm and allows interactive modification of the CSG objects. After the Goldfeather algorithm was proposed, two other algorithms based on it followed and further improved the algorithm and time efficiency.

AN INTRODUCTION TO CONSTRUCTIVE SOLID GEOMETRY

Constructive Solid Geometry(CSG) is an approach to render complex geometric shapes by applying boolean operations to primitives, such as boxes, spheres and cylinders. Relationships between two primitives are stored in a binary tree called a CSG tree (Figure 1&2). Its nodes can be either operations such as union, intersection and difference, or primitives.

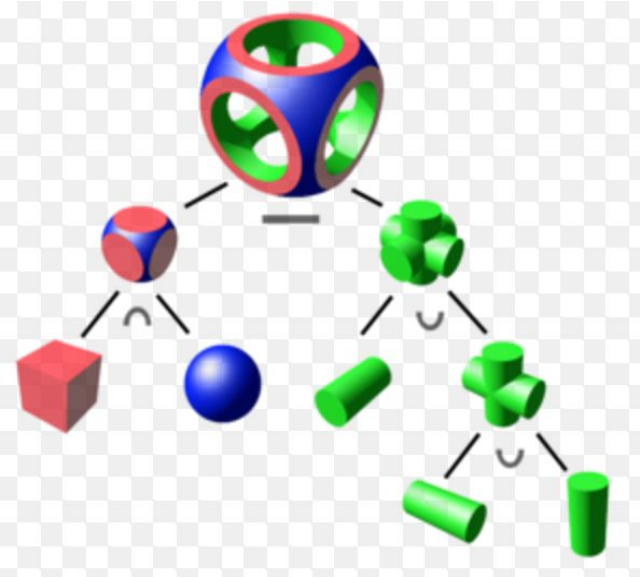


Figure. 1. An example of a CSG tree. Three kinds of primitives, box, sphere and cylinder are used here.

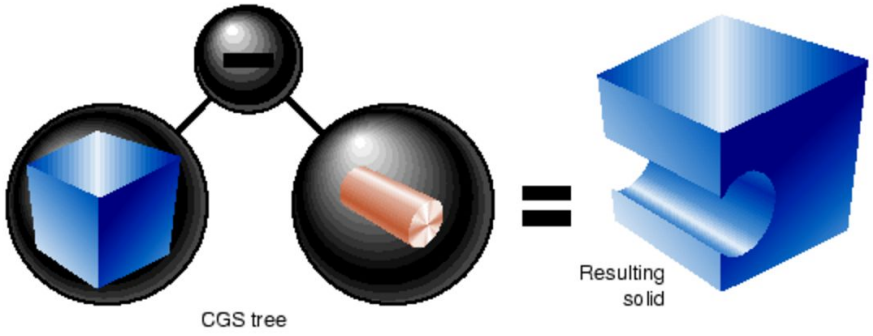


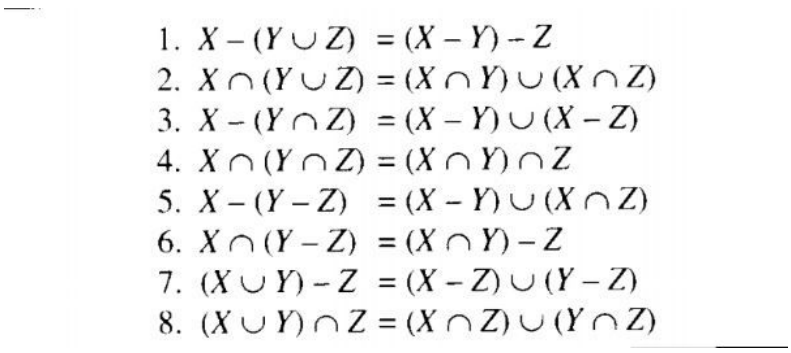
Figure 2. A simpler CSG with only two primitives and 1 operation. The root node is subtraction operation and its children are a box and a cylinder.

The rendering algorithms of CSG objects generally fall into two categories, image-space and object-space approach. The object-space approach can convert CSG directly into triangle, apply clipping to the primitives and then uses OpenGL to render the final shape, while image-space approach applies clipping of primitives on each pixel during the rendering process. Object-space approach is viewer-independent but requires more calculation and is less time efficient than one frame of calculation for image-space approach. In this paper, we are mainly discussing the algorithms using image-space approaches to clip surfaces depending on our view on a pixel basis. In our paper, we use clipping to refer to the process of determining which surface satisfies the operations in the CSG tree.

A SUMMARY OF GOLDFEATHER ALGORITHM

In Goldfeather's algorithm, a CSG model is represented by a Boolean expression, graphically shown as a CSG tree. Union can be think of as sum, and intersection and subtraction

can be think of as product. This CSG tree will need to be "normalized" into a sum-of-products form, to reorganize the operations and simplify the rendering process. A CSG tree is normalized if it's in the disjunctive normal form. Any CSG tree can be normalized using 8 simple logic transformations shown in figure 3. Figure 4 shows a CSG tree before and after normalization.



1. $X - (Y \cup Z) = (X - Y) - Z$
2. $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
3. $X - (Y \cap Z) = (X - Y) \cup (X - Z)$
4. $X \cap (Y \cap Z) = (X \cap Y) \cap Z$
5. $X - (Y - Z) = (X - Y) \cup (X \cap Z)$
6. $X \cap (Y - Z) = (X \cap Y) - Z$
7. $(X \cup Y) - Z = (X - Z) \cup (Y - Z)$
8. $(X \cup Y) \cap Z = (X \cap Z) \cup (Y \cap Z)$

Figure 3. Boolean equations used to normalize a CSG tree.

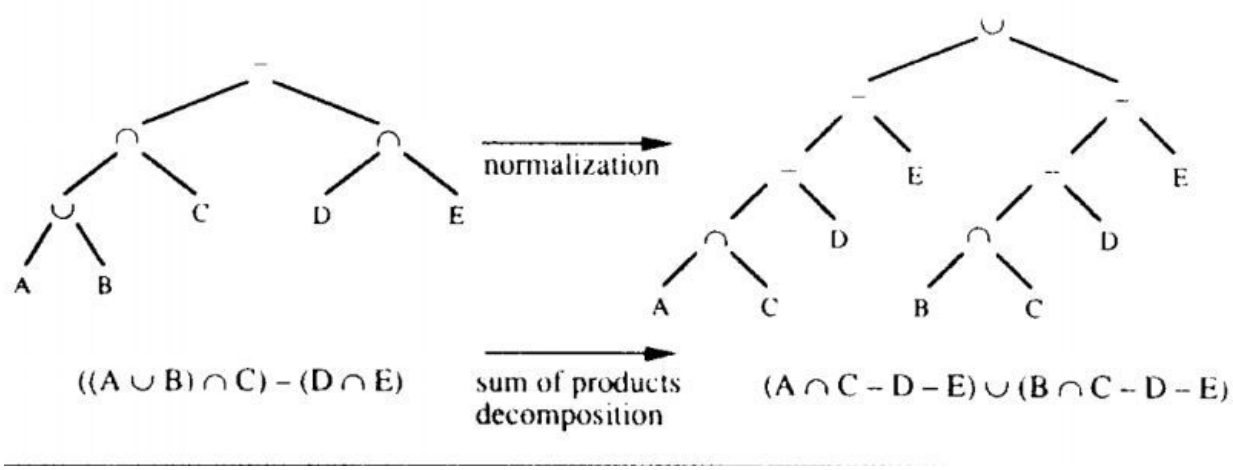


Figure 4. *Left:* A CSG tree before normalization. *Right:* A CSG tree after normalization. Notice that Union is at the top. The left child of Intersection or Subtraction is never Union and the right child of Intersection or Subtraction is always a primitive.

Since normalization might add too many nodes to the CSG tree, pruning of a normalized tree is needed to reduce time consumption of rendering unneeded nodes. A standard algorithm for pruning is the bounding-box algorithm. Only when the bounding box of a node intersects with the bounding box of the other node that it will operate with, this subtree will be kept in the

tree. Bounding box of primitives can be calculated easily with its maximum x,y and z values. Bounding box of boolean operation nodes is calculated as shown in Figure 5. There are more efficient ways to normalize and prune a CSG tree, but those algorithms are not discussed here because they are not related with OpenGL and computer graphics.

1. $\text{Bound}(A \cup B) = \text{Bound}(\text{Bound}(A) \cup \text{Bound}(B))$
2. $\text{Bound}(A \cap B) = \text{Bound}(A) \cap \text{Bound}(B)$
3. $\text{Bound}(A - B) = \text{Bound}(A)$

Figure 5. Calculation of bounding box of boolean operations.

Rendering each primitives requires rendering the primitives itself and trim it with every other primitives in the tree. As a result, this algorithm has a O(n^2) time complexity. Depending on the geometry of the CSG tree, the time complexity of rendering falls into O(n) and O(n^2).

IMPLEMENTATION OF THE GOLDFEATHER ALGORITHM

In Goldfeather algorithm, we need at least 1 z-buffer (the other z-buffer can be created by copying the z-buffer into memory storage), a color buffer and a stencil buffer. A stencil buffer is an auxiliary buffer which stores one integer value per pixel. It's used in Goldfeather's algorithm to determine which part of an image to be drawn.

The pseudocode is shown below:

```
Initialize the z-buffer to z-far
For each product A in CSG tree{
    For each primitive P in this product A{
        If P should be subtracted{
            Draw the back of P into the temporary z-buffer
```

```
        } else if P should be intersected {
            Draw the front of P into the temporary z-buffer
        }
        For other primitives B which are not P{
            clip the temporary z-buffer against B
        }
        copy and merge the temporary z-buffer into the z-buffer
    }
}
```

We do clipping of a z-buffer by counting how many surfaces are in front of this pixel. This can be done through stencil buffer. The stencil buffer is used to increase or decrease the counter for each fragment drawn into a pixel and we only update z-buffer when the counter equals the desired number representing where the surface is.

IMPROVEMENT OF GOLDFEATHER ALGORITHM

	Goldfeather	Layered Goldfeather	SCS
Z-buffer	2	2	2
Stencil buffer	1	1	1
Color buffer	1	1	1
Time Efficiency	O(n^2)	O(kn)	O(kn)

Improved z-buffer based CSG render algorithm（Layered Goldfeather）

This algorithm is an improvement based on Goldfeather's algorithm. It has a O(kn) time complexity, where k is the depth complexity of the CSG tree representing the operations from a specific viewing direction. K can be determined by counting the number of primitives in front of each pixel and find the maximum number of all the pixels in an image. Each layer of depth complexity is rendered at a time, instead of rendering each primitive at a time. A set of primitives in each layer is extracted using the stencil buffer. Clipping of each layer requires a small adjustment of the clipping algorithm in Goldfeather algorithm. The resultant time complexity of rendering a CSG model is O(kn).

Sequenced Convex Subtraction (SCS)

The sequenced convex subtraction is based on subtractions of convex object in a specific sequence. In calculating SCS product, we only need one z-buffer and do not perform z-buffer copying. Rendering more SCS products needs some z-buffer management. The algorithm of SCS can be optimized using depth complexity. The biggest advantage of SCS algorithm is that the sequences that SCS uses to performs CSG rendering is formulated as permutation embedding sequences, which can be formed by deleting other entries. For example, if we want to perform concatenation of n copies of s1, in this case, n=3, s1= abc, the length of the sequence is n^2, since the length is s1s1s1 = abcabcabc. However, if we reverse s1 and denote it as s2, s1s2s1 = abc cba abc = abcba, which has length 6.

We could optimize it using maximum depth complexity of the subtracted objects. If the specific number is known, shorter subtraction sequences can be used. Since if we know that k, which is the depth complexity, and k <n, then if we have n=3, k=2, s1=abc, s2 = cba, s1s2 = abcba.

TEST OF THE ALGORITHMS

Dataset	Algorithm	Frame Rate (frame/sec)		
		O2	Indigo2	TNT2
Widget	Goldfeather	5.20	15.18	1.41
	Layered Gold.	3.51	10.85	1.01
	SCS Product	14.93	26.18	15.27
Grid	Goldfeather	0.31	0.73	0.15
	Layered Gold.	1.10	2.54	0.71
	SCS Product	1.64	3.66	2.67
Swiss Cheese	Goldfeather	0.09	0.34	0.06
	Layered Gold.	0.36	0.74	0.31
	SCS Product	0.46	0.98	0.74
Machine Tool	Goldfeather	5.41	18.98	1.46
	Layered Gold.	4.15	14.81	1.23
	SCS Product	4.93	11.12	6.95
Hollow Pipe	Goldfeather	1.48	4.74	0.41
	Layered Gold.	0.72	2.43	0.22
	SCS Tree	3.24	8.46	1.57

In this test, we see that rendering performances are actually hardware-dependent. They may have relatively different cost of copying z-buffer and triangle rasterization. For example, although SCS may have less time doing copy z-buffer because of its improvement, the hardware may have slow triangle rasterization compared with z-buffer copying, leading to a lower time efficiency compared with Goldfeather.

References:

Stewart, Nigel, Geoff Leach, and Sabu John. "An improved z-buffer CSG rendering algorithm." *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware - HWWS '98* (1998): n. pag. Web.

Stewart, Nigel, Geoff Leach, and Sabu John. "A Z-Buffer CSG rendering algorithm for convex objects." (2000).

Goldfeather, J., S. Monar, G. Turk, and H. Fuchs. "Near real-time CSG rendering using tree normalization and geometric pruning." *IEEE Computer Graphics and Applications 9.3* (1989): 20-28. Web.