Interactive CSG Trees Inside Complex Scenes

Jan Ohlenburg[*]
Fraunhofer FIT
Sankt Augustin, Germany

Jan Müller[†]
Fraunhofer FIT
Sankt Augustin, Germany

1 Introduction

Constructive Solid Geometry (CSG) is widely used in modeling tools such as CAD applications. A number of algorithms exists for different scenarios, e.g. for interactive use of CSG modeling the Z-buffer algorithm is a simple but very efficient approach [1], known as the Goldfeather algorithm. For larger CSG trees and non-interactive modeling B-rep algorithms calculate the resulting polygons for a view-independent representation, but are too slow for real-time calculation. A good overview of the different approaches to CSG is given in [2], which also was the motivation for our work.

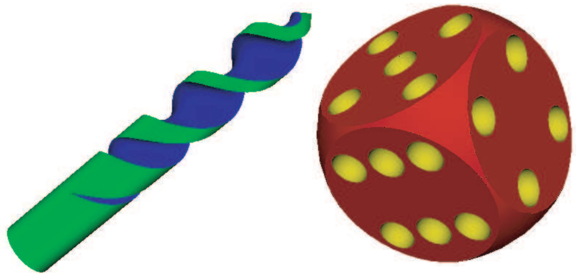
Figure 1: Two sample objects generated by CSG operations.

Image-based methods like the Goldfeather algorithm have the disadvantage that the resulting CSG objects cannot be used inside a complex virtual environment (VE), without adding the whole environment to the CSG tree. The algorithm computes a normalized CSG tree with the boolean operations union, subtract and intersect using hardware-accelerated Z-buffer tests. In order to calculate the operations the depth and the stencil buffers are used, corruption the current as well as the resulting depth buffer. This makes it impossible to simply render the CSG tree as part of a VE. On the other hand, if the whole VE has to be added to the CSG tree, the advantage of interactivity is lost.

We present an interactive approach, which overcomes this limitation. It uses similar methods as the Goldfeather algorithm to calculate the CSG tree, but it is able to insert its result into a complex VE.

2 Implementation Outline

We calculate the different CSG operations by the same techniques described by the Goldfeather algorithm, but opposing to the common algorithm we operate on eight virtual pixel buffers. Our algorithm accepts operands of the boolean operations to be a previous CSG result, a geometric object or a group of objects. The color and depth buffers of the front and the back faces have to be generated, i.e. 4 pixel buffers for each operand. The add and the subtract operations operate directly on the pixel buffers of the first operand (A), therefore only changes have to be written.

The addition of two operands ($A+B$) is very simple. The resulting color and depth buffer for the front face are generated by comparing the front faces of A and B, areas where B's depth values are smaller than A's the content of B's pixel buffers are written into A's. The back face is generated accordingly, only here A is changed if B's back is behind A's back.

In case of a subtraction ($A-B$), the overlapping areas of both primitives are determined using their four depth values (front and back faces). In regions where the front of B is closer than the front of A and B's back is intersecting A, the back face of B will be visible. Therefore the back face of B is written into the resulting pixel buffer at these regions. Otherwise, the front face of A will be visible in the resulting image or A is inside B and thus completely deleted. Again, the back face is calculate accordingly.

The intersection result is generated slightly different and starts with empty pixel buffers. The front face of A is written into the front face pixel buffers only where it intersects B, and B's front is merge with these buffers only at areas where it intersects A. The back face pixel buffers A's and B's back faces are used where they intersect the other object.

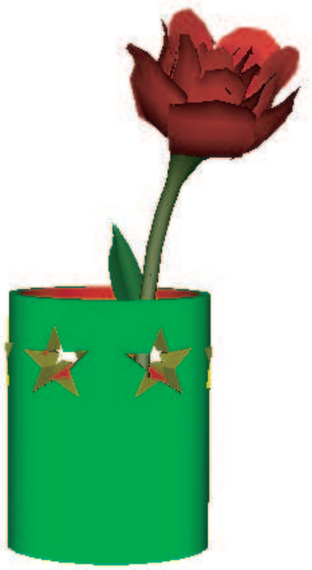
Figure 2: The vase is created using CSG operations, the rose is just part of the scene.

Since the CSG operations return their pixel buffers, the algorithm can reuse these buffers to calculate the whole tree, because each operation performs on pixel buffers representing arbitrary objects.

See Figure 1 for two examples, which are generated by CSG trees using add, subtract and intersect. The screw on the left is generated

[*]e-mail: jan.ohlenburg@fit.fraunhofer.de
[†]e-mail: jan.mueller@fit.fraunhofer.de

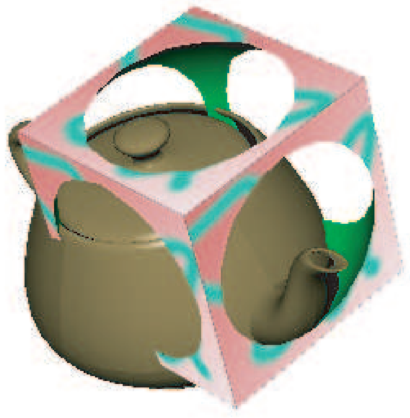

Figure 3: The CSG result can be intersecting the teapot, which is not part of the CSG tree.

by subtracting the sum of two spirals from a cylinder. The dice on the right is generated by an intersection of a sphere and a cube and finally 21 yellow spheres are subtracted from the surface.

In our render engine, CSG nodes are part of the normal scene graph hierarchy and can have arbitrary children, namely geometric objects, groups of objects and other CSG nodes. The scene graph is rendered using two passes, during the first pass all geometric objects and CSG nodes are collected and put into different lists. In the second pass all opaque nodes are rendered first, than the result of each CSG tree is merged with the current frame buffer and at last all transparent objects are rendered. Since the pixel buffers of the CSG trees are calculated during the first render pass, they can be used to insert the result into the current frame buffer. See Figures 2 and 3 for examples, where a CSG trees are rendered together with the scene objects, namely the rose and the teapot, which are not part of the CSG trees.

3 Conclusion and Future Work

We have presented an approach on interactive CSG modeling inside complex VE that handles CSG trees of reasonable size in real-time. It transfers the results of single CSG operations to the main memory to increase the CSG's flexibility. Our solution can be integrated in any renderer, since it does not rely on any assumptions about the structure of the scene graph. Also it does not rely on special functionality like stencil buffer comparison, second depth buffers or GPU shaders. An interactive performance is solely achieved by using quick pixel buffer traversal and bounding pixel area determination: CSG operations can only alter those regions that are covered by both CSG primitives. Also early pixel termination is crucial for interactive frame rates.

As part of our future work we would like to extend our approach by B-rep algorithms to use the interactive approach during modeling and to generate the resulting object afterwards.

References

[1] J. Goldfeather, 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):20–28, 1989.

[2] N. Stewart, G. Leach, and S. John. An Improved Z-Buffer CSG Rendering Algorithm. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 25–30, 1998.