2007

Surface Reconstruction from Constructive Solid Geometry for Interactive Visualization

Doug Baldwin
SUNY Geneseo, baldwin@geneseo.edu

Surface Reconstruction from Constructive Solid Geometry for Interactive Visualization

Doug Baldwin

Department of Computer Science
SUNY Geneseo

Abstract. A method is presented for constructing a set of triangles that closely approximates the surface of a constructive solid geometry model. The method subdivides an initial triangulation of the model's primitives into triangles that can be classified accurately as either on or off of the surface of the whole model, and then recombines these small triangles into larger ones that are still either entirely on or entirely off the surface. Subdivision and recombination can be done in a preprocessing step, allowing later rendering of the triangles on the surface (*i.e.*, the triangles visible from outside the model) to proceed at interactive rates. Performance measurements confirm that this method achieves interactive rendering speeds. This approach has been used with good results in an interactive scientific visualization program.

1 Introduction

Constructive solid geometry (CSG) is a technique for modeling three-dimensional solids as set-theoretic combinations of simple primitive shapes [1]. Common primitives are such shapes as cylinders, cones, spheres, polyhedra, *etc.* Combining operations typically include union, intersection, and difference or complement. CSG was first used to represent solid models for computer aided design and manufacturing, and has since found applications in computer graphics and other areas.

The work reported in this paper is motivated by a need to render CSG-defined geometries in certain particle physics visualizations. Specifically, a visualization tool named IViPP [2] is being developed to support visual analysis of results from the MCNPX [3] simulator. MCNPX simulates reactions between subatomic particles, using a form of CSG to describe the physical environment within which the reactions occur. IViPP needs to display both particle data and the geometry of the surrounding environment. It must update its displays at interactive rates, fast enough for users to smoothly rotate, zoom, and similarly manipulate their view.

This paper's main contribution is a method for constructing a small set of triangles that closely approximates the surface of a CSG model. The set of triangles can be constructed in a preprocessing step, and subsequently rendered at interactive rates. Section 2 compares this approach to previous ways of rendering

CSG, while Section 3 describes the method itself. Section 4 presents data regarding the performance and effectiveness of this approach. Section 5 summarizes the work's status and suggests directions for further research.

2 Background and Previous Work

The method described in this paper builds a mesh of triangles representing a CSG model's surface; similar approaches have also been pursued by other researchers. For example, the "constructive cubes" algorithm [4] adapts the marching cubes isosurface construction algorithm [5] to approximate the surface of a CSG model. The ACSGM approach [6] is also based on marching cubes, but is considerably more sophisticated than constructive cubes in how it approximates the surface of the CSG model. Chung [7] uses a three-stage method, consisting of spatial subdivision followed by triangulation proper followed by triangle refinement to preserve sharp edges and corners. More recently, Čermák and Skala describe a method for triangulating implicit surfaces [8] that could be adapted to CSG. The method proposed in the present paper is conceptually simpler than ACSGM or Chung's approach, and unlike any of the previous efforts, detects unnecessarily small triangles and combines them into larger ones to reduce the total number of triangles. Like constructive cubes and ACSGM, this paper's method can trade image quality for computing resources. However, the primary resource consumed by this paper's method is time, whereas constructive cubes consumes significant amounts of both time and memory (the resource requirements of ACSGM are not discussed in [6]).

Triangulating the surface of a CSG model can be an unacceptable bottleneck for CAD applications in which users want to edit models and see the results in real time [9]. However, in visualization, geometry is often static, and users need only simple real-time interactions (*e.g.,* changes of viewpoint). Triangulated surfaces are attractive in such settings, because the triangulation itself can be rendered very quickly, and it only needs to be constructed once—after construction, different renderings of the same set of triangles display the surface from whatever viewpoints are needed.

One of the oldest alternatives to surface triangulation for CSG rendering is ray tracing [10]. However, classic ray tracing cannot be done at interactive speeds, due to the need to compute multiple ray-primitive intersections for every pixel in the display. Speeds can be improved by dividing the CSG model into spatial subregions in such a way that only a few primitives lie in each region [11]; recent research has divided the model in such a way that the primitives in each subregion can be ray traced in the GPU [12]. GPU-assisted ray tracing achieves interactive speeds on small to medium-size models, but does require custom GPU programming. In contrast, the approach introduced in this paper reduces CSG rendering to drawing triangles, something that is well-supported by modern graphics hardware without custom programming.

Goldfeather [13] proposed a method that uses hardware depth and color buffers to evaluate and render CSG models. Wiegand [14] adapted this idea to

more widely available graphics hardware, using a depth buffer (albeit capable of being saved to and restored from main memory) and a stencil buffer, and showed how to access that hardware via standard APIs such as OpenGL. Subsequent work [15, 16] improved the asymptotic execution time of depth-and-stencil-buffer rendering algorithms, and showed how to perform the necessary buffer comparisons in the GPU [9, 17]. These improvements have yielded interactive rendering speeds for some models, but the algorithms still require considerable computation for each frame, and are sensitive to hardware characteristics (e.g., depths of buffers, memory-to-frame-buffer bandwidths, GPU capabilities).

Liao and Fang describe a volumetric approach to CSG rendering [18] that builds a three-dimensional texture map from a CSG model and then renders the model by drawing slices through that texture map. Because this approach can build the texture map prior to rendering, it can run in constant time per frame, and can easily achieve interactive speeds. However, it requires large amounts of memory for the texture map, and care must be taken to avoid aliasing in the rendered images.

3 Surface Reconstruction by Triangle Subdivision

A small set of triangles that approximates the surface of a CSG model can be constructed by the process illustrated in Figure 1. The process begins with triangulations of the model's primitives, created without concern for how primitives interact in the overall model. Each triangle from a primitive is then subdivided into smaller triangles. Vertices of these subtriangles coincide as much as possible with intersections between the edges of the triangle and the surface of the model. Subdivision continues until the triangles are small enough to be classified as either inside the modeled object, on its surface, or outside the object without producing visual anomalies. Only those triangles on the model's surface need be drawn in order to render the model. The total number of such triangles is reduced by recombining subtriangles into their parent triangle whenever the parent's subtriangles are either all on the surface or all off (inside or outside) it.

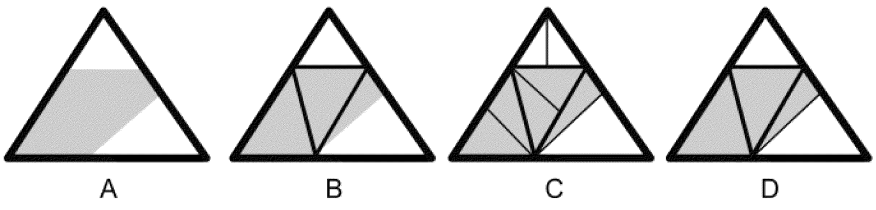

Fig. 1. Triangle subdivision. Initial triangle, with portion on model's surface in gray (A); subtriangles after one (B) and two (C) levels of subdivision; subtriangles are recombined into their parent if all all are on or all are off the surface (D)

3.1 Triangle Subdivision

The visual quality of the images produced by triangle subdivision, and the speed with which they can be rendered, depend on how triangles are divided in order to approximate the CSG model's surface. Specifically, the division scheme should yield a small number of triangles, while faithfully representing the surfaces, edges, and corners of the model. Triangle subdivision is therefore driven by changes in the classification of triangle edges relative to the CSG model (*i.e.,* whether an edge is inside the modeled object, outside it, or on its surface). Changes in classification can only happen at intersections between the edge and the surface of the CSG model, which are found in the standard manner [10]: solve the equations for the points at which edges intersect primitives, split edges into segments at these points, and then combine segments according to the Boolean operations used to combine primitives. To avoid the need for extreme numeric accuracy to determine that a segment lies on a surface of a primitive, triangle edges are considered to lie on a primitive if either that primitive is the "source" for the edge's triangle (primitive p is the source for triangle t if t is one of the triangles produced by triangulating p, or if t is a subtriangle of a triangle whose source is p), or if the entire edge lies within a small tolerance of the primitive's surface. Once the points at which classifications change are known, a triangle is divided into subtriangles according to rules 1 through 4 below. Figure 2 summarizes these rules, using dots to indicate points at which an edge's classification changes.

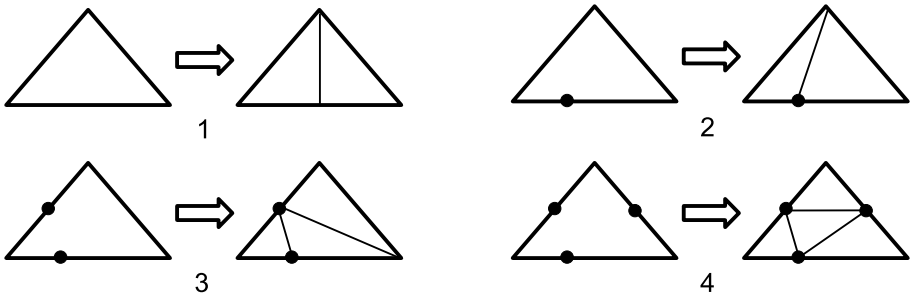

Fig. 2. Triangle subdivision rules

1. If no edge of the triangle changes classification, split the triangle into two subtriangles along the line from the center of the longest edge to the opposite vertex. Note that even if no edge changes classification, the triangle's interior might; this rule ensures that subdivision continues until any changes in classification anywhere in a triangle must intersect an edge.
2. If exactly one edge changes classification, split the triangle into two subtriangles along the line from one of the points at which that edge's classification changes to the opposite vertex.

3. If two edges change classification, split the triangle into three subtriangles, along a line connecting one classification-change point from each edge, and along the line from one of these points to the opposite vertex. If either edge changes classification multiple times, use the classification-change points closest to the edges' common vertex.
4. If all three edges change classification, split the triangle into four subtriangles along lines between one change point from each edge. For two of the edges, use the change points closest to the common vertex, as in Rule 3; for the third edge, choose a change point arbitrarily.

These rules keep the overall number of triangles small by maximizing the likelihood that different subtriangles will lie on different sides of an edge of the CSG model (rules 2, 3, and 4), or by making progress towards reducing the size of triangles to the point where further splitting is unnecessary (rule 1).

3.2 Subdivision Order and Stopping Criteria

Both the order in which triangles are considered for subdivision and the criteria for stopping subdivision can be tuned to reduce the number of triangles produced and the time required to do so. Tuning is supported by storing triangles awaiting subdivision in a priority queue. Triangles are subdivided when they are removed from this queue, with subtriangles placed back in the queue. The goal of the priority function is to divide visually important triangles before unimportant ones, but different functions can reflect different measures of importance. For example, triangles may be prioritized by size (divide larger, and thus visually more prominent, triangles before smaller ones), age (divide older triangles, likely to be cruder approximations to the model's surface, before newer triangles), *etc.* In all cases, a priority of 0 or less indicates that a triangle should be classified without further subdivision. Priority functions can take advantage of this fact to limit the time or other resources used by triangle subdivision: setting priorities to 0 when the allocated resources have been exhausted stops further subdivision and forces the triangles currently in the priority queue to receive "best guess" classifications.

When a triangle no longer needs subdividing, it is classified as either on the model's surface or off the surface, based on the heuristic that triangles are on the surface if and only if more than half of the total length of their edges is.

It is sometimes possible to stop subdividing a triangle sooner than the priority function would. In particular, if all of a triangle's edges are on the surface, or all are off, and no edge changes its classification, then subdivision can stop as soon as the shortest edge is shorter than the model's minimum feature size. This early end to subdivision is permissible because when a triangle's shortest edge is shorter than the model's minimum feature, any feature that affects the classification of the triangle's interior must also intersect an edge. If no such intersections occur, as indicated by no edge changing its classification, then the entire triangle has the same classification as its edges. Minimum feature size must be estimated, but simple estimates work well. For example, the implementations

discussed below estimate minimum feature size as half the size of the smallest feature in any primitive.

Finally, triangles that are "small" by various measures are discarded instead of being further divided. In particular...

- Subtriangles that cover only a small fraction of their parent's area leave large siblings to be processed by future subdivisions and are visually insignificant. To avoid creating such subtriangles, changes in edge classification in either the first or last 0.05% of an edge are ignored when dividing triangles.
- Triangles with nearly collinear vertices are numerically unstable. Therefore, triangles are discarded if the sum of the lengths of the two short edges is nearly equal to the length of the longest edge.
- Triangles with nearly equal vertices are also numerically unstable. Therefore, triangles are discarded if, for any edge, the ratio l/d is close to the machine epsilon for floating point numbers, where l is the length of the edge and d is the distance from the origin to one of the edge's endpoints (l/d close to epsilon indicates that changes in vertex coordinates across the edge will be difficult to recognize numerically).

3.3 The Complete Algorithm

Figure 3 summarizes the complete algorithm for triangulating and rendering a CSG model. This figure presents two central functions, "triangulate" and "draw." "Triangulate" generates a list of triangles that approximates the surface of a CSG model, while "draw" renders these triangles to some display device. Calls on "triangulate" and "draw" may be separated. In particular, the time-consuming triangulation can be done once when the model is created (or when it changes), with only the faster drawing done for each frame.

4 Results

A prototype program for CSG rendering by triangle subdivision has been coded using C++ and OpenGL. Figure 4 shows two CSG models rendered by this program. For both models, triangles were prioritized for subdivision by the length of their longest side, and subdivision stopped when that length was less than the equivalent of five pixels on the display device.

The left-hand image in Figure 4 is a perforated shell, constructed by subtracting a number of cylinders (the perforations) from a difference of two spheres (the shell). Table 1 presents the performance of triangle-subdivision rendering on several variations on this model. The variations differ in the number of cylinders removed from the shell, and thus in the total number of primitives in the model. The "Triangles Created" and "Triangles Drawn" columns are the total number of triangles created during subdivision, and the number actually used for drawing after recombination, respectively. Recombination is able to remove about 90% of the triangles generated. "Frames per Second" is the rate at which

```
triangulate( CSG model m )
    list of triangles l = triangulations of primitives in m
    priority queue q = priority queue containing triangles in l
    while q is not empty
        dequeue triangle t from q
        if t.priority <= 0
            v = sum of lengths of parts of edges of t on m's surface
            d = sum of lengths of edges of t
            if v > d / 2
                t.classification = ON
            else
                t.classification = OFF
        else if t's shortest edge is shorter than m's minimum feature
                    and no edge of t changes its classification
            if all edges of t have classification ON
                t.classification = ON
            else
                t.classification = OFF
        else if t is not "small"
            t.subtriangles = divide t per rules in Figure 2
            for each triangle s in t.subtriangles
                enqueue s in q
            t.classification = UNKNOWN
    end while
    for each triangle t in l
        recombine( t )
    return l

draw( triangle t )
    if t.classification == UNKNOWN
        for each triangle s in t.subtriangles
            draw( s )
    else if t.classification == ON
        render t to display

recombine( triangle t )
     if t.classification == UNKNOWN
         for each triangle s in t.subtriangles
             recombine( s )
         if all members of t.subtriangles have same classification, c
             t.classification = c
```

Fig. 3. Triangulating and rendering a CSG model

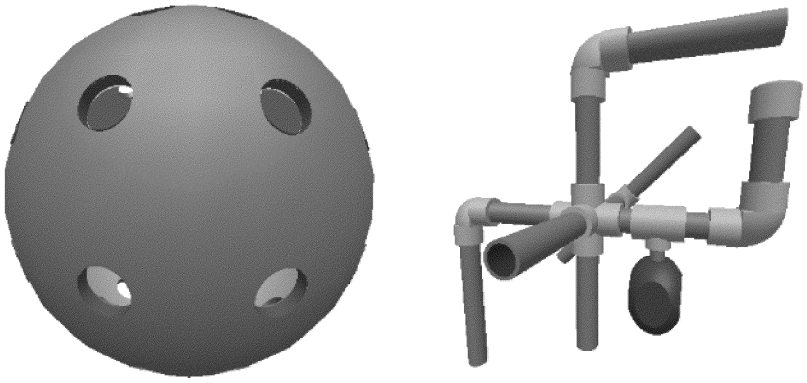

Fig. 4. Some CSG models rendered by triangle subdivision: perforated shell (left) and pipe network (right)

the triangulated model is rendered. Speeds of over 10 frames per second were achieved in all cases. All data was collected on an Apple MacBook Pro with 2.16 GHz Intel Core 2 Duo processor and ATI Radeon X1600 video chipset, running under Mac OS X 10.4.10 and OpenGL 2.0. Frame rates were measured using Apple's OpenGL profiler.

Table 1. Triangulation and rendering data for perforated shells

Primitives	Triangulation Time (Sec)	Triangles Created	Triangles Drawn	Frames per Second
8	2.58	30260	3990	89.7
16	9.20	55166	6692	54.2
28	27.12	93336	10597	34.2
44	65.57	142558	15973	22.5
64	141.44	209060	22596	16.1
88	270.11	285209	30594	12.0
116	420.13	332448	35426	10.2

CSG rendering by triangle subdivision has also been incorporated into IViPP, which is beginning to see production use as a visualization tool for physics research. Typical geometries in this setting consist of several tens of "cells," each described by a CSG expression containing on the order of 1 to 3 primitives. IViPP treats each cell as a separate CSG model. Triangle subdivision is used to triangulate each cell, and cells are rendered by rendering their surface triangles. A standard depth buffer then handles occlusions of one cell by another, occlusions of particle tracks by cells and cells by tracks, *etc.* IViPP prioritizes triangles for subdivision by size and the time that has elapsed since beginning triangulation: during the first second, a triangle's priority is the length of its

shortest edge; after one second priorities are always 0, preventing further subdivision. IViPP is successfully visualizing models as large as a nuclear reactor consisting of 71 cells collectively containing 138 primitives. IViPP takes about 1 minute to triangulate this model, and renders it at about 12 frames per second (using the same computer configuration that produced the performance data for the perforated shell).

5 Conclusions

A new way of rendering CSG models at interactive speeds has been described. Distinguishing features of this method include using triangle subdivision followed by recombination to generate a small set of triangles that closely follows the surface of the CSG model, a priority mechanism that allows image quality to be balanced against resource usage, and the ability to separate time-consuming analysis of the CSG model from rendering. Rendering speeds of over 10 frames per second have been achieved on models containing over 100 primitives, an entirely adequate speed for interactive applications. The method is in successful production use in the IViPP visualization program.

The factor that ultimately limits the utility of this method is triangulation time, not rendering time. However, time can be a factor in prioritizing triangles for subdivision, allowing triangulation time to be controlled. Experience with IViPP suggests that such time limits do not seriously compromise image quality.

CSG rendering by triangle subdivision can be extended in a number of ways. Perhaps most significantly, it assumes that triangulations of primitives are easy to generate. While this is true for many primitives, some CSG systems support primitives whose triangulations are not obvious (*e.g.,* general quadric surfaces). The method also needs to be tested on very large CSG models, *i.e.,* ones containing thousands of primitives. Since triangles divide independently of each other, the concurrency provided by multicore processors seems a promising way to speed triangle subdivision, but has not yet been investigated. Finally, additional combinations of priority schemes and resource limits for subdivision may yield further improvements in image quality and triangulation time.

6 Acknowledgments

This work was supported in part by a grant from the U. S. Department of Energy through the University of Rochester Laboratory for Laser Energetics. Ryan Kinal, Heather Warren, and Justin Hagstrom are largely responsible for incorporating the rendering method described here into IViPP. Dr. Sharon Stephenson and her students at Gettysburg College are early adopters of IViPP, and provided the reactor model mentioned in Section 4. Thanks to Dr. Stephen Padalino of SUNY Geneseo for support of the entire IViPP project.

References

1. Requicha, A.A.G.: Representations for rigid solids: Theory, methods, and systems. ACM Computing Surveys **12** (1980) 437–464
2. Baldwin, D.: Architecture of the IViPP particle visualization program. Available at `http://cs.geneseo.edu/~baldwin/ivipp/ivipparch.html` (2004)
3. Los Alamos National Laboratory: MCNPX™ Users Manual (LA-UR-02-2607). (2002) Available at `http://mcnpx.lanl.gov/opendocs/versions/v230/MCNPX_2.3.0_Manual.pdf`.
4. Breen, D.E.: Constructive cubes: CSG evaluation for display using discrete 3-D scalar data sets. In: Proceedings of Eurographics '91, Elsevier Science Publishers (1991) 127–142
5. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3D surface construction algorithm. In: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press (1987) 163–169
6. Purgathofer, W., Tobler, R.F., Galla, T.M.: ACSGM: An adaptative CSG meshing algorithm. In: Proceedings of CSG '96, Information Geometers Ltd. (1996)
7. Chung, C.W., Chuang, J.H., Chou, P.H.: Efficient polygonization of CSG solids using boundary tracking. Computers and Graphics **21** (1997) 737–748
8. Čermák, M., Skala, V.: Adaptive edge spinning algorithm for polygonalization of implicit surfaces. In: Proceedings of Computer Graphics International 2004, IEEE (2004) 36–43
9. Hable, J., Rossignac, J.: CST: Constructive solid trimming for rendering BReps and CSG. IEEE Transactions on Visualization and Computer Graphics **13** (2007)
10. Roth, S.D.: Ray casting for modeling solids. Computer Graphics and Image Processing **18** (1982) 109–144
11. Chuang, J.H., Hwang, W.J.: A new space subdivision for ray tracing CSG solids. IEEE Computer Graphics and Applications **15** (1995) 56–62
12. Romeiro, F., Velho, L., de Figueiredo, L.H.: Hardware-assisted rendering of CSG models. In: Proceedings of the XIX Brazilian Symposium on Computer Graphics and Image Processing, IEEE Computer Society (2006) 139–146
13. Goldfeather, J., Molnar, S., Turk, G., Fuchs, H.: Near real-time CSG rendering using tree normalization and geometric pruning. IEEE Computer Graphics and Applications **9** (1989) 20–28
14. Wiegand, T.: Interactive rendering of CSG models. Computer Graphics Forum **15** (1996) 249–261
15. Stewart, N., Leach, G., John, S.: Linear-time CSG rendering of intersected convex objects. In: Proceedings of the 10th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision. (2002) 437–444
16. Stewart, N., Leach, G., John, S.: Improved CSG rendering using overlap graph subtraction sequences. In: Proceedings of the International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia – GRAPHITE 2003, ACM Press (2003) 47–53
17. Guha, S., Krishnan, S., Munagala, K., Venkatasubramanian, S.: Application of the two-sided depth test to CSG rendering. In: Proceedings of the 2003 Symposium on Interactive 3D Graphics, ACM Press (2003) 177–180
18. Liao, D., Fang, S.: Fast volumetric CSG modeling using standard graphics system. In: Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications, ACM Press (2002) 204–211