LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-676678

Surface Triangulation for CSG in Mercury

D. Engel, M. J. O'Brien

August 26, 2015

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Surface Triangulation for CSG in Mercury

Daniel Engel[1,2], *Matthew O'Brien*[1]
[1]*Lawrence Livermore National Laboratory*, [2]*Georgia Institute of Technology*

ABSTRACT

Visualization routines for rendering complicated geometries are very useful for engineers and scientists who are trying to build 3D prototypes of their designs. A common way to rapidly add interesting features to a 3D model is through the use of a concept called Constructive Solid Geometry. CSG uses compositions of the boolean set operations to manipulate basic geometric primitives to form more complicated objects. The most common boolean operations employed are union, intersection, and subtraction. Most computer-aided design software packages contain some sort of ability visualize CSG. The typical workflow for the user is as follows: The user specifies the individual primitive components, the user arbitrarily combines each of these primitives with boolean operations, the software generates a CSG tree structure which normally stores these solids implicitly with their defining equation, the tree is traversed and a general algorithm is applied to render the appropriate geometry onto the screen. Algorithms for visualizing CSG have been extensively developed for over a decade. Points sampled from the implicit solids are typically used as input by variations of algorithms like marching cubes and point-cloud surface reconstruction. Here, we explain a surface triangulation method from the graphics community that is being used for surface visualization in the framework of a Monte-Carlo neutron transport code called Mercury.

INTRODUCTION

The CSG object representation is particularly important for users of the Monte-Carlo neutron transport code called Mercury. It allows them to construct complicated geometries and generate a visual representation of the objects to guarantee their validity. In Mercury, users write an input file that includes a CSG description of how they want to create a geometry for their simulation. Mercury's parser passes the main code the necessary information for it to construct a CSG tree data structure. This tree serves multiple purposes. Primarily, it is actually used in the physics simulation for detecting where in the 3D environment particles exist at a particular point in time and tracking them to surfaces. However, for visualization purposes, the tree is used to gather information about the final set of objects that should be rendered.

It is worth noting that for this application, the primary concern is performing CSG operations on concave and convex closed surfaces. Traditionally we use surface rather than solid to refer to the boundary of these geometric bodies. Additionally, mesh is used to refer to an approximate polyhedral representation of a given geometric surface. Mercury has two general approaches for solving the CSG problem. The ray tracer is the most natural in a Monte-Carlo framework and performs excellently at displaying the appropriate geometry. However, at this point, ray tracing suffers from being computationally expensive and not realistic for real-time visualization on limited hardware.

Another more interesting approach is mesh generation based on the CSG specification. Many of Mercury's current mesh generation methods for solving the CSG problem involve making successive approximations to an implicitly defined series of surfaces. These methods are accurate, but some leave visual artifacts. These arise when the surfaces are “not smooth” or have

local points where they are non-differentiable.

PREVIOUS WORK

Ray Tracer

For each pixel, a ray is fired into the scene from the viewer's perspective. These rays essentially sample objects in the scene to determine which object is visible to the viewer. The material properties of the chosen sampled object are used to determine the final color for the pixel and the scene is constructed appropriately(Figure 1).

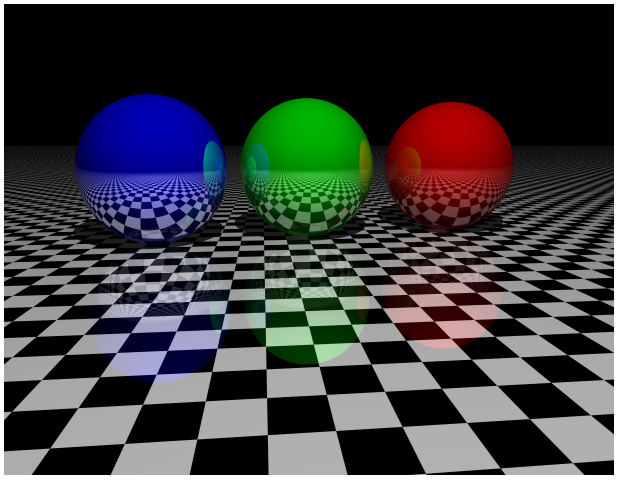


Figure 1: *A sophisticated implementation of a ray tracer with many components of the shading model used.*

The obvious downside to this approach is that upon every frame update, the pixel values have to be recomputed. Acceleration structures such as octrees(Figure 2) can be used to enhance this per frame calculation by spatially decomposing the scene into geometric regions.

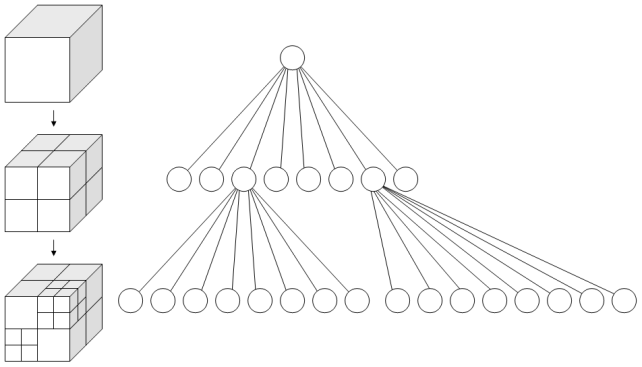


Figure 2: A visual representation of the octree data structure.

This helps to reduce the number ray-intersection tests during the ray trace calculation.

Mixed-Cell

This method begins with specified spatial grid to begin defining the resulting mesh. It uses an idea similar to adaptive mesh refinement to recursively subdivide grid cells to compute a total volume fraction. The terminating condition is reached when all nodes of a sub-cell lie in the same material or the user-specified subdivision threshold is reached. The volume fraction of these individual cells are passed to Visit which performs material interface reconstruction from the cell volumes. These are used to generate a final mesh(Figure 3).

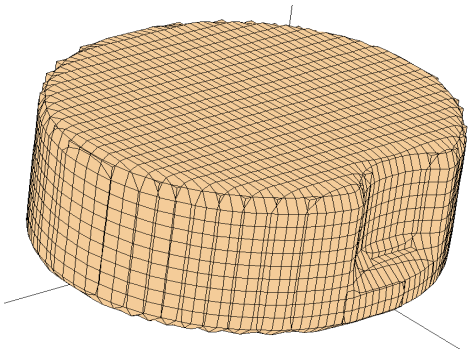


Figure 3: *A CSG subtraction generated using the conformal method with a 100x100x100 grid size.*

Numerical Integration

An alternative method for computing volume fractions of cells is employed in this method. It's a 2D numerical quadrature rule for computing the volume fraction of CSG cells in graphics mesh cells. The user has the option to select various quadrature rules to adjust the accuracy. This volumetric information is passed to Visit which performs the surface reconstruction and creates a final mesh.

Conformal Mesh

Space is partitioned into uniform spatial cells. For each of the nodal points on the grid, the point is relocated to the nearest surface to "conform" to the CSG cell. This is done similarly to how rays are sent into the scene using the ray tracing method(Figure 4). When multiple CSG surfaces occupy a cell, this method reverts to the Mixed-Cell method.

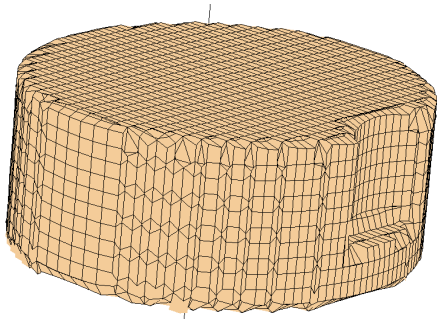


Figure 4: *A CSG subtraction generated using the conformal method with a grid size of 100x100x100.*

CONTRIBUTIONS

The more recent additions take a dramatically different approach to solving the CSG problem. Rather than focusing on defining a graphics mesh globally, these methods deal with each CSG primitive individually to preserve the local behavior of the mesh.

Adaptive Local Convex Subdivision ALCS("Local Catmull-Clark")

The motivation behind this scheme was to develop an algorithm that could arbitrarily triangulate a given convex body. Traditional Catmull-Clark subdivision is a method for successively refining a coarse object to obtain a much smoother surface(Figure 5).

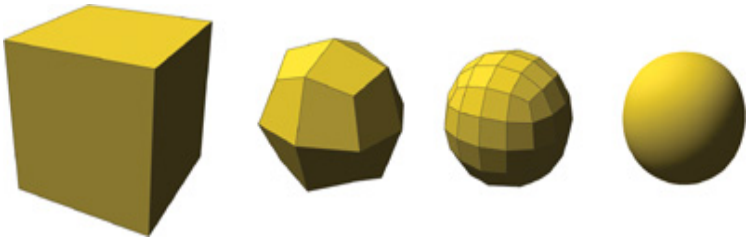


Figure 5: *A depiction of traditional Catmull-Clark ran three times to smooth a cube into a sphere.*

This image illustrates how Catmull-Clark subdivision is a global operation and refines all faces equally. For the CSG problem, we needed a notion of local refinement and wanted the ability to subdivide a series of faces independent of the adjacent neighbors. This led to a fundamentally different set of data structures for mesh representation that made the mesh easy to locally traverse and update.

Collectively, these data structures have been referred to as the spoke representation for a mesh.

Spoke
Vertex Index
Next Spoke Index
Twin Spoke Index
Face Index
Spoke Depth

Figure 6: *The spoke data structure.*

A spoke can visually be considered as a directed edge of a graph or a half-edge for a mesh representation(Figure 7).

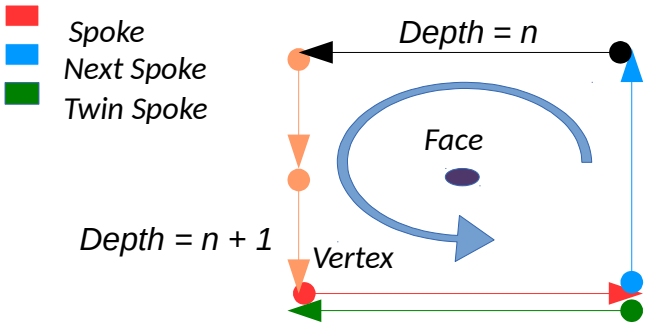


Figure 7: *A visual representation of how the spoke data structure is being used to communicate topology information.*

The separation of the mesh data into smaller data structures is done in an attempt to contrast the geometric from the topological information of a mesh. The connectivity and adjacency information is contained within the face list and spoke table, while the vertex table contains the geometric spatial coordinates of all of the vertices.

Because the goal of this method is to obtain the closest approximation to a surface, it does not use most of the averaging steps found in traditional Catmull-Clark subdivision. Instead there is a projection step, where upon subdivision of a face, the centroid is placed on the surface of the actual surface using a ray trace, and newly added vertices along the edge are also projected to the surface in a similar way. Each face is stored in a priority queue using the index into the face list and the error metric associated with that face. Each of the faces is locally subdivided until an error threshold is met(Figure 8).

$$\vec{c} = \frac{1}{n}\sum_{i=0}^{n} \vec{v}_i, where \vec{v}_i \in V$$

$$\vec{n_{avg}} = \frac{1}{n}\sum_{i=0}^{n} \vec{n}_i, where\ \vec{n}_i = N(\vec{v}_i)$$

$$\vec{r_{max}} = max_{0 \leq i \leq n} \|\vec{v}_i - \vec{c}\|$$

$$p(t) = \vec{c} + t\,\widehat{n_{avg}}$$

Solving for t given the surface's implicit description yields:

$$f(p(t_{surf})) = 0$$

$$Error\ Metric = \frac{\|\vec{c} + t_{surf}\widehat{n_{avg}}\|}{\|\vec{r_{max}}\|}$$

Figure 8: *Description of how the error metric is computed for each face.*

In order to preserve the correct topological information of the mesh, careful steps have to be placed into the algorithm to handle the multiple cases that arise upon subdivision of an individual face. While subdividing a face, each spoke is traversed and an spoke-subdivision routine is executed. The spoke-subdivision algorithm has three different possible cases. We call a spoke coarse when both its corresponding face and its twin spoke's face have the same level of refinement(Figure 9). Similarly, a fine spoke is one where
the neighboring face has undergone one level of refinement compared to the current face(Figure 10). A super-fine spoke is one where the the refinement difference between the two adjacent faces is greater than one(Figure 11). The general algorithm for face subdivision
determines one of these three possible cases and updates the mesh data structures accordingly.

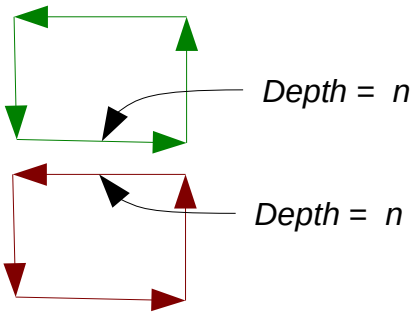


Figure 9: *An image of the coarse neighbor case.*

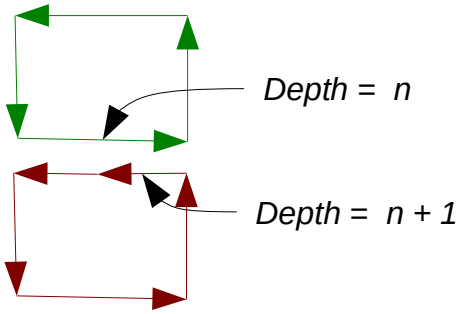


Figure 10: *An image of the fine neighbor case.*

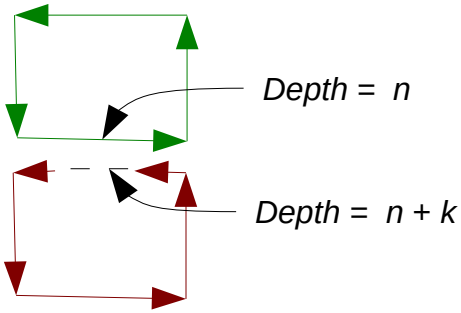
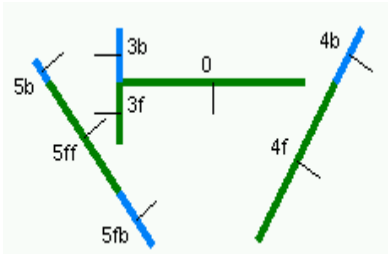


Figure 11: *An image of the super-fine neighbor case.*

BSP tree probing

The intention of this algorithm is to combine arbitrary closed triangulated meshes given a boolean operation. BSP(Binary Space Partitioning) trees offer a very natural way of dividing space into smaller domains. Instead of being limited by the constraints of a cartesian grid, BSP trees allow space to be chopped into areas that characterize the local orientation of faces on a mesh(Figure 12).



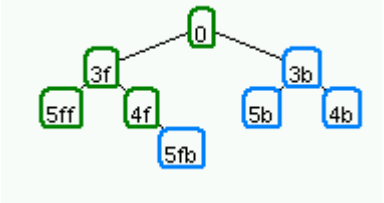
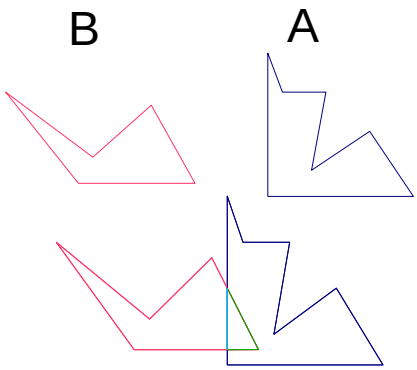


Figure 12: *A depiction of how the BSP tree partitions space, and how the binary tree structure is organized.*

Given an arbitrary non-convex triangulated mesh, upon inserting the triangles into a BSP tree, particular subtrees correspond to convex meshes that are composed to make the original non-convex mesh. This allows us to simplify our problem of handling the triangulation of non-convex bodies to one of handling a collection of convex bodies(Figure 13).



$$I(X) = Interior(X)$$

$$A_{outside} = \{x : x \in \partial A \wedge x \notin I(B)\}$$

$$B_{outside} = \{x : x \in \partial B \wedge x \notin I(A)\}$$

$$A_{inside} = \{x : x \in \partial A \wedge x \in I(B)\}$$

$$B_{inside} = \{x : x \in \partial B \wedge x \in I(A)\}$$

AND $A \cap B = A_{inside} \cup B_{inside}$

OR $A \cup B = A_{outside} \cup B_{outside}$

SUBTRACT $A - B = A_{outside} \cup B_{inside}$

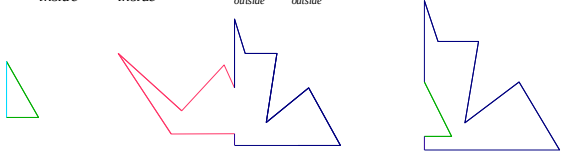
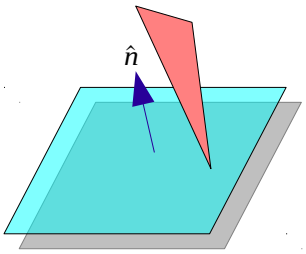


Figure 13: *A 2D demonstration of how we construct the boolean operations using subsets of the original primitives.*

The BSP algorithm begins by traversing the CSG tree. Whenever it encounters a convex primitive, it executes a triangulation routine on the primitive and builds a BSP tree from those triangles. The ALCS scheme could be used to generate triangulated quads or specific triangulation routines for each primitive could be implemented. When constructing the BSP tree, three separate cases emerge for inserting a triangle(Figure 14).

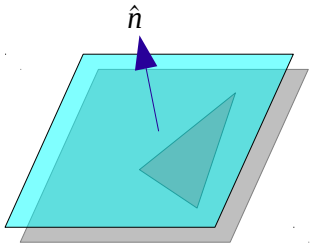
$f(v_i) > 0 \; \forall \; v_i \in T$



a)

b)

$f(v_i) < 0 \; \forall \; v_i \in T$



$\exists v_i, v_j \in T : f(v_i) f(v_j) < 0$

c)

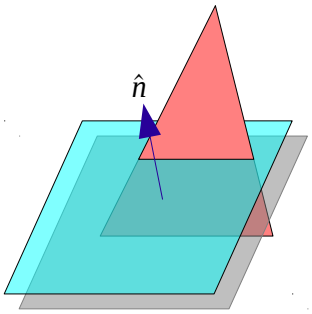


Figure 14: *Enumeration of the cases for tree insertion.*

For each node of the tree, a triangle is tested to see if it is in front or behind of the current triangle corresponding to the node(Figure 14). This process is recursively performed until a triangle can be inserted in front or behind another node in the tree. An interesting third case emerges when the inserted triangle crosses the current node's triangle(Figure 14c). If all of the vertices of the triangle do not lie on this node's triangle, then we slice the inserted triangle into three smaller triangles and recursively insert them into the BSP tree. If exactly one of the inserted triangle's vertices lies on the node's triangle, then we instead slice the inserted triangle into two triangles and recursively add them to the tree.

Once each tree is constructed, two trees are taken at a time if a boolean operation is to be applied on the both of them. Each tree uses a collection of triangles from the other tree to perform a probe operation similar to the insertion. For all of the triangles, the probe operation queries the tree to assemble a set of triangles that are inside and outside of the mesh.

The result is four new sets. From these partitions of the mesh into components, we can use set unions to construct the desired mesh to represent the desired boolean operation.

RESULTS & FUTURE WORK

After applying the BSP probing algorithm to two triangulated cylinders, the following results were produced.

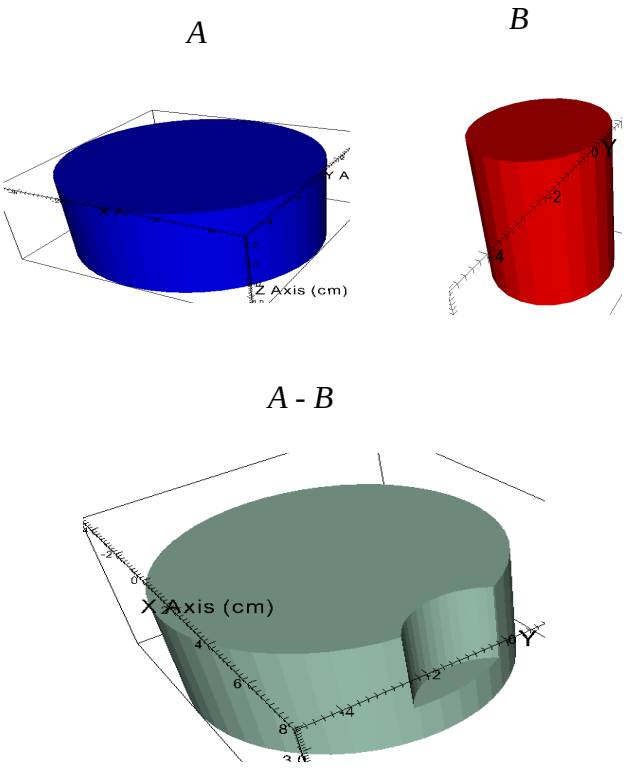


Figure 15: CSG operations using BSP method.

With the BSP probing method, one feature worth noticing is the sharp, crisp edges that are preserved from the original geometries. This is a desired property of the final CSG. Unlike previous methods that Mercury has used, the BSP method preserves these points where locally the derivative on the surface is discontinuous.

The BSP probing method has two noteworthy drawbacks currently, however. It only works on two triangulated primitives currently. However, this can easily be extended by determining an efficient way to decompose chains of boolean expressions into simpler expressions involving the inside and outside sets. Additionally, after the CSG has been triangulated, there tend to be several unnecessary triangles as a result of the BSP building step in the algorithm. On some occasions, the merging of neighboring triangles might result in the same looking shaded surface. It is beneficial to consider these cases in the future and use mesh simplification routines to reduce the overall polygon count.

ACKNOWLEDGEMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.