Converting CSG models into Meshed B-Rep Models Using Euler Operators and Propagation Based Marching Cubes

Marcos de Sales G. Tsuzuki
Senior Member, ABCM
mtsuzuki@usp.br

Fabio K. Takase
Member, ABCM
fktakase@usp.br

Murilo Antônio S. Garcia
murilogarcia@yahoo.com

Thiago de Castro Martins
thiago@usp.br
Escola Politécnica, University of Sao Paulo
Dep. Mechatronics and Mech. Syst. Eng.
05508-900 São Paulo, SP. Brazil

The purpose of this work is to define a new algorithm for converting a CSG representation into a B-Rep representation. Usually this conversion is done determining the union, intersection or difference from two B-Rep represented solids. Due to the lack of explicit representation of surface boundaries, CSG models must be converted into B-Rep solid models when a description based on polygonal mesh is required. A potential solution is to convert a CSG model into a voxel based volume representation and then construct a B-Rep solid model. This method is called CSG voxelization, conceptually it is a set membership classification problem with respect to the CSG object for all sampling points in a volume space. Marching cubes algorithms create a simple mesh that is enough for visualization purposes. However, when engineering processes are involved, a solid model is necessary. A solid ensures that all triangles in the mesh are consistently oriented and define a closed surface. It is proposed in this work an algorithm for converting CSG models into triangulated solid models through propagation based marching cubes algorithm. Three main new concepts are used in the algorithm: open boundary, B-Rep/CSG Voxelization mapping and constructive triangulation of active cells. The triangles supplied by the marching cubes algorithm need not be coherently oriented; the algorithm itself finds the correct orientation for the supplied triangles. The proposed algorithm restricts the exploration to the space occupied by the solid's boundary. Differently from normal marching cubes algorithms that explore the complete sampled space.
Keywords*: solid model, marching cubes algorithm, triangular meshes*

Introduction

The Constructive Solid Geometry (CSG) representation allows users to define complex 3D solid objects by hierarchically combining simple geometric primitives using Boolean operations and affine transformations (Hoffmann, 1989). It is a very popular and powerful solid modeling scheme, and it is particularly suitable for interactive object manipulations and design. Traditionally, CSG primitives are defined by simple analytic objects, such as cubes, cylinders and spheres. Some recent CSG algorithms can also support primitives that are general solid models defined by their boundary surfaces. An explicit representation of the boundary is not available in a CSG model.

CSG is a widely used modeling paradigm, but several algorithms in different fields, require a description based on polygonal mesh. Due to the lack of explicit representation of surface boundaries, CSG visualization is not directly supported by standard graphics systems. Boender et. al. (1994) showed that in order to derive an accurate finite element mesh from a CSG model, a B-Rep model has in fact to be derived. They proposed an algorithm based in two steps, where the first step is the boundary evaluation of the CSG model, which converts the CSG model into a B-Rep representation by computation of its boundary. Tobler et al. (1996) proposed an approach to convert CSG solid models into B-Rep solid models based on the marching cubes algorithm. They focused on the marching cubes algorithm and did not explain anything about how the B-Rep solid model is created. Kamel and Chen (1991) defined an algorithm where the primitives are meshed, and remeshing is performed on the interference region. This algorithm is very hard to implement and has several special cases.

A solution for the rendering of volumetric CSG models is to convert a CSG model into a voxel based volume representation and then construct a B-Rep solid model. This method is called CSG voxelization, conceptually it is a set membership classification problem with respect to the CSG object for all sampling points in a volume space. The marching cubes algorithm (Lorensen and Cline, 1987) was originally proposed as a tool for CSG voxelization. Recently, several authors (Viceconti et al., 1999) have used the marching cubes as a tool for visualization of CSG models and/or medical images. Lee et al. (2005) used the marching cubes algorithm and improved the geometrical quality of the generated triangles, that is measured either by its angles and its edge´s length.

Related Works

It is estimated that in a 3D domain containing the CSG model, a well designed algorithm that visits only the cells intersected by the CSG model's boundary, from now refered as active cells, will have a computational cost of $O(n^{2/3})$ (Itoh and Koyamada, 1995), where n is the number of cells. Consequently, algorithms which perform an exhaustive covering of cells are found to be inefficient (computational cost of $O(n)$, spending a large portion of time visiting cells which do not contribute to the contour, also named as empty cells. The CSG boundary is also named as isosurface.

The majority of the techniques for accelerating the extraction of isocontours do so by limiting the number of cells that are visited, thereby reducing the overhead associated with the inevitable search for active cells. Newman and Yi (2006) classified the methods that minimize unnecessary operations on empty cells according to their main processing characteristics: hierarchical geometric approaches (Galin and Akkouche, 2000; Whilhems and Gelder, 1992), interval-based and propagation-based (Bajaj et al., 1996; Itoh et al., 2001; Itoh and Koyamada, 1995; Shekkar et al., 1996). Typically, these methods minimize the operations via representations that can efficiently encode regions of non-activity.

Wilhems and Van Gelder (1992) were the first to use octrees to avoid examining empty cells. The algorithm starts with a setup phase that creates the octree. Each node of the octree contains the maximum and minimum scalar values among the cells in the sub-volume. When the user specifies an isovalue, the algorithm starts the isosurface-finding phase, which examines the volume by traversing

Paper accepted August, 2007. Technical Editor: Glauco A. de P. Caurin.

from the root of the octree. All the sub-volumes with minimum values higher than the isovalue or maximum values lower than the isovalue are then excluded. Galin and Akkouche (2000) proposed a recursive octree subdivision algorithm until a given level of precision is reached, converging to the implicit surface. The algorithm propagates through an octree inflating and deflating strategy.

Interval-based representations are another class of data structures that are useful in avoiding traversal of empty cells. One advantage of many interval-based approaches is their operational flexibility; since these approaches operate in an interval space rather than in the geometric space of the mesh. Shen and Johnson (1995) proposed an interval-based isosurface extraction algorithm. The algorithm is divided into two parts: setup and isosurface extraction. The algorithm uses the concept of cell's extreme value that is defined as the maximum and minimum scalar values at the corners of the cell. Straightforwardly, only those cells that have lower minimum value and higher maximum value than a given isovalue are intersected by an isosurface. To efficiently and accurately locate candidate cells without searching the entire set of data, they sort the cells by their extreme values. The isosurface extraction algorithm locates all active cells and polygonizes them.

Several propagation-based approaches generate the isosurface by recursively visiting adjacent cells. The adjacent cells are inserted into a queue. The cells inserted in the queue are marked, so that they are not inserted twice. The cells are removed from the queue and they are triangulated by the Marching Cubes method or some other polygonization method. The cells adjacent to the just processed cell that are not marked, are inserted in the queue. By repeating the above, an isosurface is generated when the queue becomes empty. The cell propagation heuristic can be roughly classified as: (1) propagate through isocurves with fixed coordinate (breadth-first search) (Bajaj et al., 1996; Itoh and Koyamada, 1995), (2) vertex/edge adjacency (Itoh et al., 2001), (3) no heuristic at all (Shekkar et al., 1996).

Propagation-based approaches to isosurface construction avoid traversal of empty cells since the propagation process visits only active cells. Propagation-based approaches do not use forward cube by cube marching but rather propagate outward from some active seed cell. Most propagation-based approaches require manual selection of seed cells since automatic selection can be challenging. For example, the method of Shekkar et al. (1996) recursively propagates from a user specified cell using isosurface connectivity. However, a few automatic seed cell selection schemes exist (Bajaj et al., 1996; Itoh and Koyamada, 1995; Kreveld et al., 1997). Another advantage to using a propagation approach over other techniques is that surfaces are easily transformed into a triangle strip representation for more efficient rendering. Also of importance is the fact that shared vertices between cells are more efficiently located, as we are considering only a single closed contour at any given time. One drawback of the propagation-based algorithms proposed in the literature is that adjacent cells or triangles are stored in a stack or queue.

A simple mesh can be enough for visualizations purposes. However, when engineering processes are involved, a solid model is necessary. A solid model ensures that all triangles in the mesh are consistently oriented, i.e. have their vertices listed clockwise counterclockwise. A finite element mesh is valid only if the mesh formed by the external faces of its elements is closed and orientable (Mäntylä, 1988). An algorithm for calculating volumetric properties generates meaningful results only if the mesh models a closed and oriented surface. The solid model created through the algorithm proposed in this work explicitly guarantees all those conditions. Using a B-Rep data structure, the adjacency information required by the algorithm presented in this paper is explicitly available.

It is proposed in this work an algorithm for converting a CSG model into a triangulated B-Rep solid model through a propagation based marching cubes algorithm. The triangles supplied to the algorithm need not be coherently oriented; the algorithm itself finds the correct orientation for the supplied triangles. The technique presented here is new as the propagation traverses the CSG's boundary that is represented in the B-Rep data structure. It is not necessary to mark visited cells and the algorithm uses no queue or stack for adjacency cell storage. Further, the algorithm has only one phase, eliminating the setup necessary in other approaches. In the CSG voxelization, the setup phase corresponds to evaluate the CSG model over the entirely 3D domain.

The rest of the paper is organized as follows. In section 2, the concepts behind the CSG representation are briefly explained and the recursive algorithm to query CSG models is showed. In section 3, the concepts of the B-Rep and the Euler operators are briefly explained. The Euler operators are used in the definition of the proposed algorithm. In section 4, the marching cubes algorithm is briefly explained. In section 5, the proposed algorithm is presented. Some results are presented in section 6. Discussion and future works are presented in section 7. Conclusions are presented in section 8.

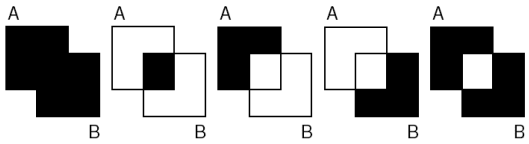

Figure 1. The five non-trivial Boolean combinations of two sets (from left to right): A+B= {a : a ∈ A or a ∈ B}, A∩B= {a: a ∈ A and a ∈ B }, A-B= {a : a ∈ A and a ∉ B}, B-A= { a: a ∉ A and a ∈ B}, and the symmetric difference (A-B)+(B-A).

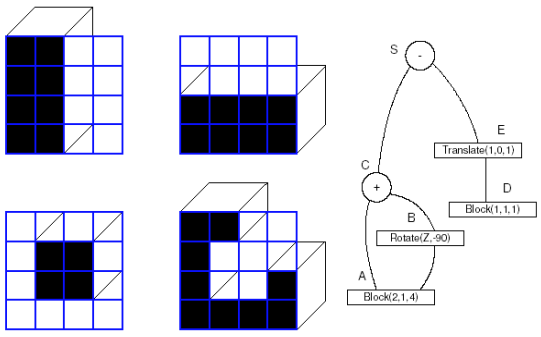

Figure 2. The instances A, B, and E are shown (left) superimposed on the same reference grid. The solid S was specified by the following sequence of commands: A=Block(2,1,4); B=Rotated(A,Z-axis,-90); C=A+B; D=Block(1,1,1); E=Translated(D,1,0,1); S=C-E; The corresponding CSG graph (right) has 2 leaf primitives, 2 transformation nodes, and 2 regularized Boolean operation nodes.

CSG Representation

Constructive representations capture a construction process, which defines the solid by a sequence of operations, that instantiate or combine modeling primitives or the results of previous constructions. They often capture the user's design intent in a high level representation that may be easily edited and parameterized.

CSG is the most popular constructive representation. Its primitives are parameterized solids, which may be simple shapes (such as cylinders, cones, blocks) or more complex features suitable for a particular application domain (such as slots or counter-bored holes). The primitives may be instantiated multiple times (possibly with different parameter values, positions, and orientations) and

grouped hierarchically. Primitive instances and groups may be transformed through rigid body motions (which combine rotations and translations) or scaling.

The transformed instances may be combined through regularized Boolean operations: union, intersection, and difference. These regularized operations perform the corresponding set theoretic Boolean operations, and then transform the result into an r-set by applying the topological interior operation followed by the topological closure. They always return valid (although possibly empty) solids. Although other Boolean operations may be offered, these three are convenient and sufficient, because amongst the 16 different Boolean combinations of two sets, *A* and *B*, 8 are unbounded, 3 are trivial, and only 5 are useful for solid modeling: the union $A+B$, the intersection $A \cap B$, the differences $A-B$ and $B-A$, and the symmetric difference, $(A-B)+(B-A)$, as shown in Fig. 1.

Figure 2 illustrates how a simple syntax may be used to specify a solid in CSG. Parsing such syntax yields a rooted graph, whose leaves represent primitive instances and whose internal nodes represent transformations or Boolean operations that produce solids. The root represents the solid corresponding to the CSG graph. CSG representations are concise, always valid in the r-set modeling domain, and easily parameterized and edited. Many solid modeling algorithms work directly on CSG representations through a divide-and-conquer strategy, where results computed on the leaves are transformed and combined up the tree according to the operations associated with the intermediate nodes. However, CSG representations do not explicitly carry any information on the connectivity or even the existence of the corresponding solid. These topological questions are best addressed through some form of boundary evaluation, where a whole or partial B-Rep is derived algorithmically from the CSG model.

Point Classification for CSG Solids

The CSG tree can be viewed as an implicit description of the modeled solid's geometry that must be evaluated in order to create graphical output or perform calculations. A CSG representation is a tree. This immediately suggests a divide and conquer, or recursive descent algorithm for computing the point classification. The algorithm is designed as follow (Requicha, 1980):

```
/* Evaluate property P of a CSG tree */
P *Tree_P (CSG_Tree *S, args)
{
    if (S->op = = <primitive>)
        return primitive_P (S, args);
    else
        return combine_P (Tree_P (S->left, args),
                          Tree_P (S->right, args), S->op);
}

/* Combine two evaluation of P with set operation 0p */
P *Combine_P (CSG_Tree *left_P, CSG_Tree *right_P, in 0p)
{ … }

/* Evaluate P for a primitive */
P *Primitive_P (CSG_Tree  *S, args)  { … }
```

Boundary Representation (B-Rep)

B-Rep solid models emerged from the polyhedral models used in computer graphics for representing objects and scenes for hidden line and surface removal. They can be viewed as enhanced graphical models that attempt to overcome some problems by including a complete description of the bounding surface of the object. There are three primitive entities face, edge and vertex. The geometric information attached to them form the basic constituents of B-Rep models. In addition to geometric information such as face equation and vertex coordinates, a B-Rep model must also represent how the faces, edges and vertices are related to each other. According to Mäntylä (1988), it is customary to bundle all information of the geometry of the entities under the term geometry of a boundary model and, similarly, information of their interconnections under the term topology. It is possible to say that the topology is a glue that ties the geometry. With the objective of algorithm simplification, mainly in the determination of the circuit of edges surround a face, the halfedge entity was created. It was observed that the edge in the original winged-edge data structure (Baumgart, 1975) had two main functions: represent the circuit of edges surround the face and to represent the real edge. The algorithm to determine the circuit of edges surrounds a face was very complex with several rules. Some researchers observed that separating these two functions, the algorithm becomes much simpler (Toriya and Chiyokura, 1991). This way, modern solid modelers have one entity to represent the edge itself and the circuit of edges surrounds the face is represented by a circuit of halfedges. An edge participates in two circuits, each one with opposite directions. The edge is represented by two halfedges, each halfedge is used in one circuit. The correct orientation of the circuit of halfedges is fundamental to guarantee the integrity of a solid model. Fig. 3 shows circuits of halfedges surround faces. A face can have holes inside to represent protrusions or depressions. In this case a face has one outer loop and zero or more inner loops. The loop represents one circuit of halfedges.

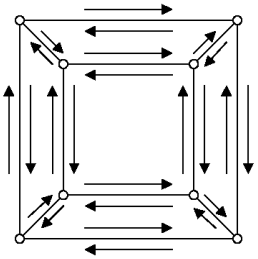

Figure 3. Planar diagram of a B-Rep solid model with its oriented circuit of halfedges.

Euler Operators

Euler operators were originally introduced by Baumgart (1975) in the context of the winged-edge data structure. In order to manipulate the topological entities and at the same time ensure validity of the model, the Euler operators are used satisfying Euler's law. The Euler-Poincaré law relates faces, edges, vertices and inner loops in a quantitative manner for solid models:

$$v - e + f = 2 - 2h + r \tag{1}$$

where v is the number of vertices, e is the number of edges, f is the number of faces, h is the number of through holes and r is the number of internal loops. It has been proved by Mäntylä (1988), that Euler operators form a complete set of modeling primitives for manifold solids. More precisely, every topologically valid polyhedron can be constructed from an initial polyhedron by a finite sequence of Euler operators. Euler operations represent a conceptually clean way to modify a mesh. Insertion and deletion of edges, vertices, faces, rings and genus are executed, while a valid orientated 2-manifold connectivity is maintained, and they are invertible. The five topological operators and their inverse operators are illustrated in Fig. 4, basically following a proposal from Mäntylä (1988).

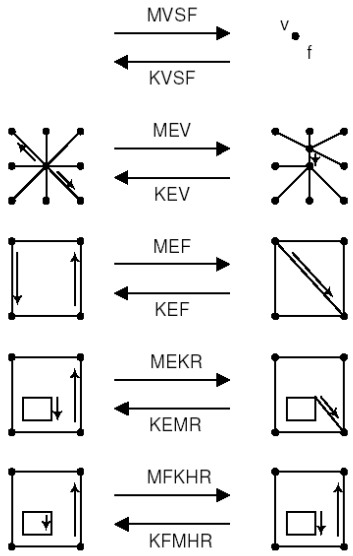

Figure 4. Planar diagrams of Euler operators.

The first operator is **MVSF** (Make Vertex Solid Face), it creates a new solid with just one vertex and one face. It's corresponding inverse operator **KVSF** (Kill Vertex Solid Face) is applicable to a solid consisting of only one vertex and one face. It removes the vertex and the face. The operator **MEV** (Make Edge Vertex) subdivides the circuit of edges surrounds the given vertex, creating one edge and one vertex. There is a special version of the operator, used in this work, to create a dangling edge. The inverse operator **KEV** (Kill Edge Vertex) removes an edge and a vertex. The operator **MEF** (Make Edge Face) divides a face in two by adding a new edge between two vertices. Its inverse operator **KEF** (Kill Edge Face) must be applied to an edge that is adjacent to two distinct faces.

The operator **MEKR** (Make Edge Kill Ring) joins two distinct circuits of halfedges. The new edge is interpreted as belonging twice to the new circuit of halfedges, once in its both orientations. The inverse operator **KEMR** (Kill Edge Make Ring), is applicable to an edge belonging twice to a circuit of edges. The operator **KFMRH** (Kill Face Make Ring Hole) creates a hole through the solid model. Its inverse operator **MFKRH** (Make Face Kill Ring Hole) removes one hole through.

Marching Cubes Algorithm

Marching Cubes is an algorithm for computing triangular meshes from discrete sampled volume data over voxel-based volumes (Lorensen and Clide, 1987). The isosurface is located in a cube of eight voxels. The marching cubes algorithm determines how the isosurface intersects this cube. Each vertex of the cube is classified into positive and negative vertices, depending whether the sampled value associated to vertex is greater or not than an isovalue. Thus, there are 2^8=256 possible configurations of a cube. The usual implementation stores those 256 configurations in a lookup table that encodes the tiling of the cube in each case. A boundary face is one of the six sides of a cube. A border is one of the four rims of a face. An isovertex is the intersection of an isosurface with a border. A boundary edge is the connection between two isovertices within a face and an internal edge is the connection between two isovertices within a cube.

The same configuration can be tiled in various ways and the 256 entries lookup table does not distinguish between those. Those ambiguities can appear on the boundary face or inside the cube. Nielson (2003) introduced the concept of deciding on ambiguous faces by using bilinear interpolation on faces. The internal ambiguity arises when two diagonally opposite vertices of a cube can be connected through the interior of the cube, creating a kind of tunnel. Lewiner et al. (2003) solved the internal ambiguities by using trilinear interpolation.

CSG to B-Rep Conversion Algorithm

The algorithm traverses the CSG boundary represented by the B-Rep data structure using three main concepts: open boundary, B-Rep/CSG voxelization mapping and constructive triangulation of active cells.

This algorithm classifies each vertex as internal or external according to the point classification algorithm for CSG solids. In this context, the bilinear and trilinear interpolation are not useful. Then, the ambiguity appearing in the boundary face is solved by applying the point classification algorithm at the center of the ambiguous boundary face (see Fig. 5). The ambiguity appearing in the interior of the cube is solved by applying the point classification algorithm at the center of the cube (see Fig. 6).

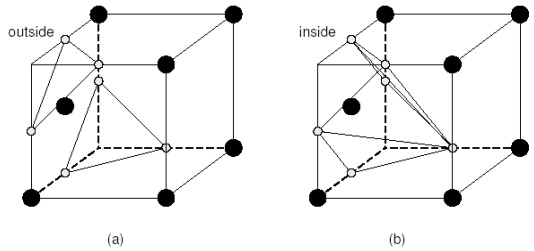

Figure 5. Ambiguity appearing on the boundary face. The point classification algorithm is applied to the center of the ambiguous boundary face and resulted as external (case (a)) or internal (case (b)). The configuration in (a) has no internal edges.

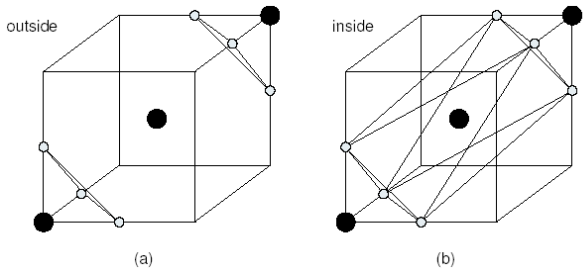

Figure 6. Ambiguity appearing in the center of the cube. The point classification algorithm is applied to the center of the cube and resulted as external (case (a)) or internal (case (b)).

The open boundary represents the external contour of the triangles already added to the B-Rep solid model and simultaneously defines the position where new triangles can be added. It is represented as a loop in the B-Rep data structure. Triangles are incrementally added to the open boundary. When a new triangle is added, the open boundary is recalculated. The open boundary defines the traverse direction.

An active cell is triangulated, and its triangles have boundary edges and in some configurations, internal edges are present (see Figs. 5(b) and 6(b)). This set of triangles is added to the open boundary. After including a triangle with internal edges, the open boundary will have internal edges in its composition. After the set of triangles associated to this active cell is completely included, the open boundary is defined uniquely by boundary edges.

The B-Rep/CSG voxelization mapping uses the coordinates present in the B-Rep solid model to determine the associated active cells. The algorithm searches for new active cells to continue the traverse when the open boundary is uniquely composed by boundary edges. The boundary edge belongs to one created triangle and using its three coordinates the proposed algorithm determines the active cell that originated it. One boundary edge has two vertices, and based on their coordinates it is possible to obtain the coordinates of two adjacent active cells. Using these two results, the proposed algorithm determines the next active cell to be used.

The triangulation of the next active cell happens in a constructive manner. At least the boundary edge that was used to find the next active cell is known. The proposed algorithm checks if adjacent boundary edges are present in any open boundary. This is done by using the coordinates of the vertices limiting each boundary edge and checking if it is adjacent to the current active cell. The active cell configuration is determined using the adjacent boundary edges present in the open boundaries and their associated triangle's normals. The normal in a B-Rep solid model points outwards. Using this set of information some vertex classification is determinated without evaluating the point classification algorithm, as shown in Fig. 7. The remaining vertex classification is determined using the point classification algorithm.

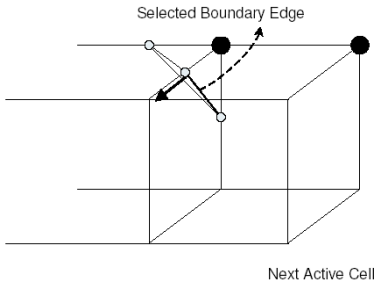

Figure 7. The cube configuration is determined by processing its adjacent boundary edges present in the open boundaries, the rest of the vertices are classified by applying the point classification algorithm.

Once the set of triangles is determined, they are added to the open boundary using Euler Operators (see Fig. 8). As a consequence, the open boundary is automatically updated after the end of the operation. There are only six different possibilities of cases to implement triangle attachment:

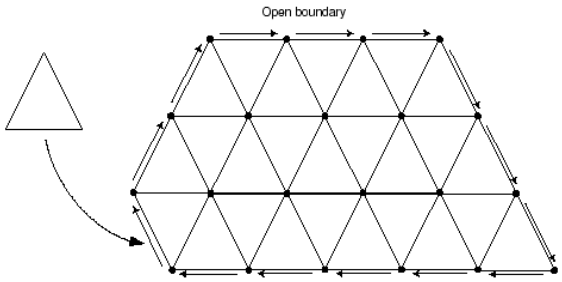

Figure 8. The open boundary is defined as a circuit of halfedges, a triangle is attached to the open boundary.

- The first triangle;
- Triangle with one edge created;
- Triangle with two adjacent edges created;
- Triangle with one edge created and a vertex created belonging to the same open boundary. The vertex is not adjacent to the edge;
- Triangle with one edge created and a vertex created belonging to a different open boundary;
- The last triangle with three adjacent edges created.

At the end, the final situation of the open boundary matches exactly the last triangle to be attached. During the execution of the algorithm, there will be at least one open boundary. There should be multiple open boundaries in the case of objects topologically equivalent to a torus or multiple torus. The open boundaries are stored in a list.

The Algorithm

The algorithm is presented below. Initially, the algorithm searches for the first isocube and then processes the first case. The while loop is done while there are open boundaries to be processed. The next isocube is obtained by processing the first halfedge from the open boundary. The list of connected triangles adjacent to the given halfedge is determined. The triangles are added to the solid model according to the case classification and the appropriate processing is done.

1: $c \leftarrow$ retrieve First Cube ()
2: solid $\leftarrow$ process Case 1 (c),
3: **while** < exist open boundary to be processed > **do**,
4: $he \leftarrow$ < get boundary edge from open boundary >,
5: $c \leftarrow$ < get next active cell > (he),
6: triangle Stack $\leftarrow$ < triangle active cell > (c),
7: **while** < exist triangle in stack > **do**,
8: triangle $\leftarrow$ <get triangle from stack >,
9: < insert triangle using the appropriate routine > (triangle),
10: **end while**,
11: **end while**.

The First Triangle

Figure 9 shows the steps to create the first triangle. It is necessary to firstly apply a **MVSF**, two **MEV**s and one **MEF**. Two circuits of halfedges are created. The algorithm calculates the normals of both faces and the face with a normal pointing to the external side is an internal face. The other face contains the open boundary and it is pushed in the stack..

1: processCase1 (c)
2: triangle $\leftarrow$ processCube (c),
3: $v_1 \leftarrow$ triangle.get_first_vertex (),
4: $v_2 \leftarrow$ triangle.get_second_vertex (),
5: $v_3 \leftarrow$ triangle.get_third_vertex (),
6: solid $\leftarrow$ **MVSF** (v_1),
7: **MEV** (solid, v_1, v_2); **MEV** (solid, v_2, v_3); **MEF** (solid v_1, v_3),
8: openBoundaryList.Add (get_open_boundary (solid, c)),
9: return solid.

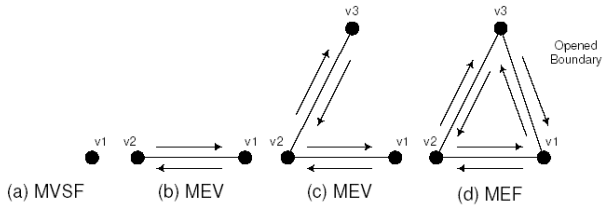

Figure 9. The first triangle is created.

Triangle with One Edge Created

The third vertex does not belong to any open boundary. It is necessary to apply one **MEV** and one **MEF**. The open boundary is automatically updated. Fig. 10 shows the steps to create this triangle.

1: processCase2 (solid, *he*, v_3)
2: **MEV** (solid, *he*.get_start_vertex (), v_3),
3: **MEF** (solid, v_3, *he*.get_end_vertex ()).

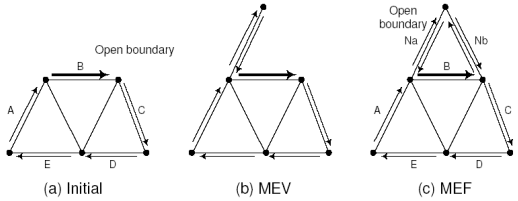

Figure 10. Triangle with one edge created. The final open boundary is represented by the following circuit of halfedges: A-Na-Nb-C-D-E.

Triangle with two Adjacent Edges Created

It is necessary to apply one **MEF**. There are two possible situations, the third vertex is after or before the edge in the open boundary. Fig. 11 shows the steps to create this triangle when the third vertex is after the edge in the circuit.

1: processCase3 (solid, *he*, v_3)
2: if *he*.next ().get_end_vertex () == v_3 **then**,
3: **MEF** (solid, *he*.get_start_vertex (), v_3),
4: **else**,
5: **MEF** (solid, v_3, *he*.get_end_vertex ()),
6: **end if**.

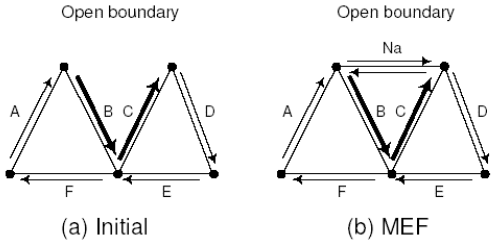

Figure 11. Triangle with two adjacent edges created. The final opened boundary is represented by the following circuit of halfedges: A-Na-D-E-F.

Triangle with one Edge and the Third Vertex Created on the same Open Boundary

The edge and the third vertex are on the same circuit of halfedges. Then, it is necessary to create two new edges connecting the third vertex with the edge. This is done by applying **MEF** twice (see Fig. 12). A new open boundary is created and added to the list. The sequence that the vertices are supplied to the **MEF** is important, the halfedge associated with the first vertex has the new circuit of halfedges.

1: processCase4 (solid, *he*, v_3)
2: new OpenBoundary ← **MEV**(solid, *he*.get_start_vertex(), v_3),
3: **MEF** (solid, *he*.get_end_vertex (), v_3),
4: openBoundaryList.Add (new OpenBoundary).

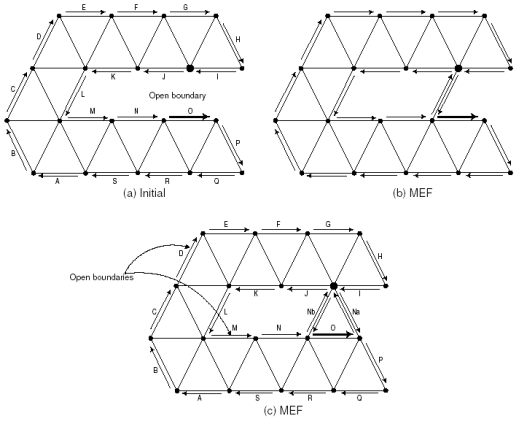

Figure 12. Triangle with one edge and the third vertex created on the same open boundary. Initially, the solid model has one open boundary: A-B-…-R-S. After the triangle attachment, the solid model has two open boundaries: A-B-…-H-I-Na-P-Q-R-S

Triangle with one Edge and the Third Vertex Created on Different Open Boundaries

The edge and the third vertex are on different open boundaries. This situation occurs in solids topologically equivalent to a torus. This way, it is necessary to create a through hole. This case is implemented by applying one **KFMRH**, one **MEKR** and one **MEF** (see Fig. 13). The open boundary is deleted and removed from the stack.

1: processCase5 (solid, *he*, v_3, open, aux_open)
2: **KFMRH** (solid, open, aux_open),
3: **MEKR** (solid, *he*.get_start_vertex(), v_3),
4: **MEF** (solid, v_3, *he*.get_end_vertex ()),
5: openBoundaryList.Remove (aux_open).

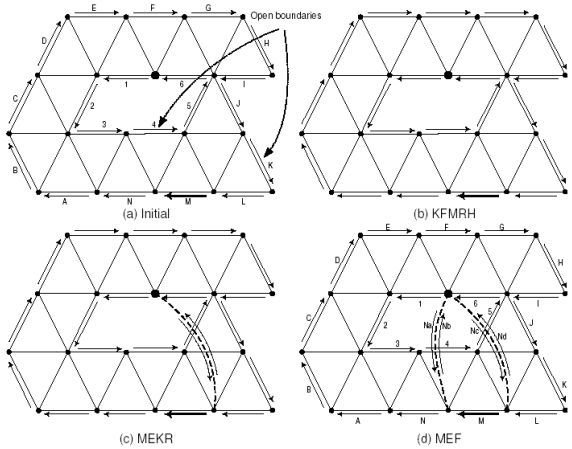

Figure 13. Triangle with one edge and the third vertex created on different open boundaries. Initially, the solid model has two open boundaries: A-B-…-M-N and 1-2-3-4-5-6. After the triangle attachment, the solid model has only one open boundary: A-B-…-L-Nd-1-2-3-4-5-6-Na-N.

Triangle with Three Adjacent Edges Created

In this case the triangle is already created, then nothing is done. The associated open boundary is deleted and removed from the list.

1: processCase6 (open_Boundary),
2: openBoundaryList.Remove (open_Boundary).

Results

In Fig. 14 the B-Rep model construction is illustrated. In a first step the vertex of each cube that lies on the CSG model boundary are classified as being internal or external to the solid (Fig. 14.a). With this classification, the isopoints are easily identified (Fig. 14.b). The isopoints are connected (Fig. 14.c) through the development of one (Fig. 15.a) or more open boundaries (Fig. 15.b) using Euler Operators. This approach avoids ambiguities and generates a solid representation with a valid topology (Fig. 14.d). More complex solids were used to test the algorithm and can be seen in Fig. 16, Fig. 17 and Fig. 18.

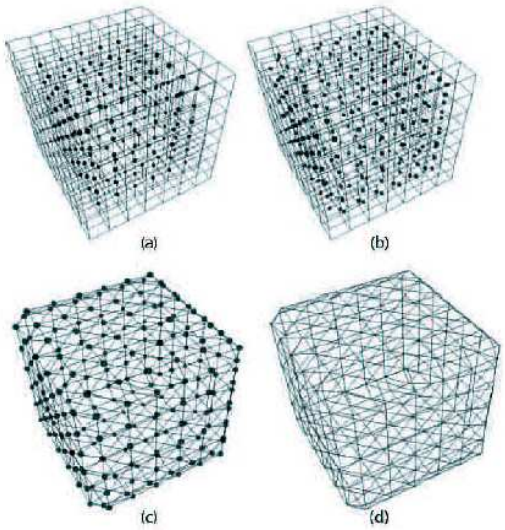

Figure 14. Solid construction steps: (a) cubes with internal vertices highlighted, (b) isopoints, (c) isopoints connected, (d) final solid.

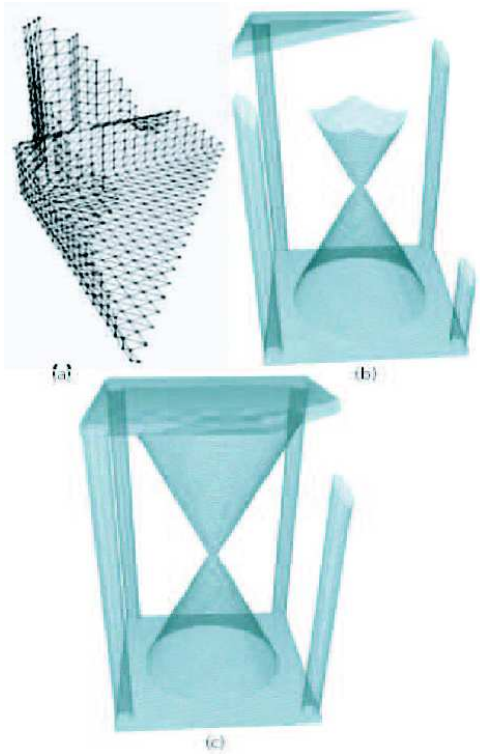

Figure 15. Samples of opened boundaries: (a) beginning of the solid construction with only one opened boundary, (b) intermediate step with 5 opened boundaries, (c) advanced step of the solid construction.

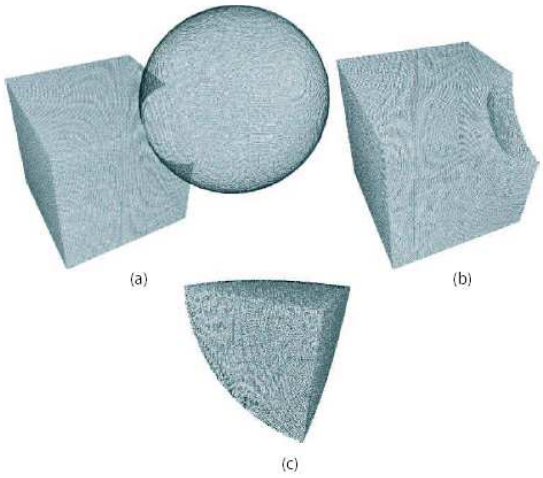

Figure 16. Simple Boolean operations: (a) union with 143.698 triangles, (b) subtraction with 76.796 triangles, (c) intersection with 61.184 triangles.

Figure 17. Die constructed with one box and several spheres (246.272 triangles).

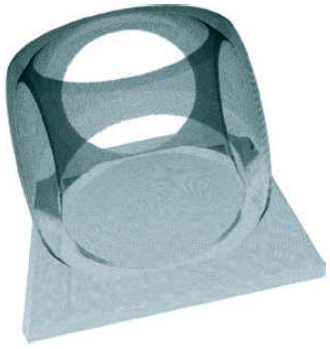

Figure 18. Complex solid with 553.632 triangles).

Conclusions and Future Works

The CSG representation defines an implicit representation where the 3D domain *D* must be discretized and evaluated. Then, the best performance is provided by algorithms that even do not execute the point classification algorithm at empty cells. This fact makes the hierarchical geometric, interval-based and conventional approaches very costly.

When an active cell is visited, it is necessary to know if a new open boundary must be created or not. Conventional propagation based approaches cannot know when such happens and always pushes adjacent active cells to a stack or queue. This fact increases too much the memory usage. Another drawback is the verification if

the adjacent active cell was already visited or not before pushing them.

Marching cubes algorithm implementation often use lookup table of predefined cube configurations. For each configuration encountered, a match has to be found in the predefined lookup table with cube configurations, so that a set of triangles approximating the isosurface can be determined. From the programming point of view, this method of predefining cube configurations and their connected triangles in the correct orientation is tedious and error prone.

The proposed algorithm has a better computational performance as only the vertices of the active cells are investigated using the point classification algorithm for CSG models. It is not necessary to verify if an adjacent cell was already visited or not, as the proposed algorithm has control over visited active cells. The active cell triangulation is easier implemented, as it is not necessary to produce a table with predefined cube configurations, as it is done in a constructive approach. Further, the triangles must not be coherently oriented, as the algorithm will insert the triangles in the correct orientation.

The algorithm proposed here is based on three main concepts: open boundary, B-Rep/CSG Voxelization mapping and constructive triangulation of active cells. The algorithm makes use of several types of adjacency information available in the B-Rep data structure. The proposed algorithm with very few modifications can be used to generate B-Rep solid models from 3D medical images. Another possible application is the generation of correct B-Rep solid models from unstructured triangular meshes. B-Rep solid modelers export triangular meshes for rapid prototyping and in some special situations the exported model does not represent a closed boundary. The algorithm proposed here can be used to find regions in the unstructured triangular that remain open, and an algorithm to correct close them can be used.

Acknowledgements

The first author was partially supported by CNPq.

References

Bajaj, C., Pascucci, V., Schikore, D., 1996, “Fast isocontouring for improved interactivity”, Proceedings of 1996 IEEE Symposium on Volume Visualization, San Francisco, pp. 39-46.

Baumgart, B. G., 1975, “Polyhedral representation for computer vision”, AFIPS Conference, Vol. 44, pp. 589-596.

Boender, E., Bronsvoort, W. F., Post, F. H., 1994, “Finite-element mesh generation from constructive-solid-geometry models”, *Computer-Aided Design*, Vol. 26, No. 5, pp. 379-392.

Galin, E., Akkouche, S., 2000, “Incremental polygonization of implicit surfaces”, *Graphical Models*, Vol. 62, No. 1, pp. 19-39.

Hoffmann, C. M., 1989, “Geometric and Solid Modeling: An Introduction”, Morgan Kaufmann Publishers.

Itoh, T., Koyamada, K., 1995, “Automatic isosurface propagation using an extrema graph and sorted boundary cell lists”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 4, pp. 319-327.

Itoh, T., Yamaguchi, Y., Koyamada, K., 2001, “Fast isosurface generation using the volume thinning algorithm”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 7, No. 1, pp. 32-46.

Kamel, H. A., Chen, L., 1991, “Integration of solid modeling and finite element generation”, *Computer Methods in Applied Mechanics and Engineering*, Vol. 89, pp. 485-496.

Kreveld, M. van, Oostrum, R. van, Bajaj, C., 1997, “Contour trees and small seed sets for isosurface traversal”, Proceedings of the 13th ACM Symposium on Computational Geometry, Nice, pp. 212-219.

Lee, S.-W., Senot, A., Jung, H.-Y., Prost, R.; 2005, “Regularized marching cubes mesh”, Proceedings of ICIP 2005 - IEEE International Conference on Image Processing, Vol. 3, pp. 788-791.

Lorensen, W. E. and Cline, H. E., 1987, “Marching cubes: a high resolution 3D surface construction algorithm”, *Computer Graphics*, Vol.21, No. 4, pp. 163-169.

Lewiner, T., Lopes, H., Vieira, A. W., Tavares, G., 2003, “Efficient implementation of marching cubes' cases with topological guarantees”, *Journal of Graphics Tools*, Vol. 8, No. 2, pp. 1-15.

Mäntylä, M., 1988, An Introduction to Solid Modelling, Computer Science Press.

Newman, T. S., Yi, H., 2006, “A survey of the marching cubes algorithm”, *Computers & Graphics*, Vol. 30, No. 5, pp. 854-879.

Nielson, G. M., 2003, “On marching cubes”, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 9, No. 3, pp. 283-297.

Requicha, A. A. G., 1980, “Representation for rigid solids: theory, methods and systems”, *Computing Surveys*, Vol. 12, No. 4, pp.437-464.

Shekhar, R., Fayyad, E., Yagel, R., Cornhill, J., 1996, “Octree-based decimation of marching cubes surfaces”, Proceedings of the IEEE Visualization Conference, San Francisco, pp. 335-342.

Shen, H. W., Johnson, C. R., 1995, “Sweeping simplices: a fast iso-surface extraction algorithm for unstructured grids”, Proceedings of the IEEE Visualization Conference, pp. 143-150.

Tobler, R. F., Galla, T. M. and Purgathofer, W., 1996, “ACSGM-an adaptive CSG meshing algorithm”, Proceedings of CSG 96-Set Theoretic Solid Modelling Techniques and Applications, pp. 17-31.

Toriya, H. and Chiyokura, H., 1991, “3D CAD Principles and Applications”, Berlin, Springer-Verlag.

Viceconti, M., Zannoni, C., Testi, D., Cappello, A., 1999, “CT data sets surface extraction for biomechanical modeling of long bones”, *Computer Methods and Programs in Biomedicine*, Vol. 59, pp. 159-166.

Wilhelms, J., Gelder, A. van, 1992, “Octrees for faster isosurface generation”, *ACM Transactions on Graphics*, Vol. 11, No. 3, pp. 201-227.