New Algorithm for Converting a CSG Representation Into a B-Rep Representation

Garcia, M. A. S.
Department of Mechatronics and Mechanical Systems Engineering, EPUSP
murilo.garcia@poli.usp.br

Vogel, N.
Department of Mechatronics and Mechanical Systems Engineering, EPUSP
nelson.vogel@poli.usp.br

Tsuzuki, M. S. G.
Department of Mechatronics and Mechanical Systems Engineering, EPUSP
mtsuzuki@usp.br

Abstract. *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. We will define another approach where the space used by the solids is determined as a volumetric representation. Then, using the volumetric representation a B-Rep Solid Model is created. Such an algorithm can be used in the future for creating B-Rep Solid Models from Three Dimensional Medical Images. Generating a full Solid Model instead of the traditional approach of only Surface Representations for 3D medical images, has the advantage that mass properties are easily extracted from a Solid Model, such as: volume, moment of inertia and mass centre. The surface representation does not guarantee that a closed volume is created. It is enough for visualization. However, for engineering purposes as analysis or stereolitography manufacturing, a closed volume is necessary. A solid model is a complete representation for engineering purposes. The construction of a Solid Model is different from the traditional construction of a surface model; thus we modified the marching cubes algorithm to reach this objective.*

Keywords: *CSG, B-Rep, Solid Model, Marching Cubes*

1. 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; Requicha, 1980). 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. Using voxel-based volume representations, a further extension can include objects extracted from volume data sets using intensity thresholding. These volume data sets may come from various types scanning of real objects, such as CT, MRI, and microscopy images, or from the sampling of implicit or procedural functions. Such extended CSG models are sometimes called Volumetric CSG models and are very useful in applications such as medical imaging, surgical applications, and amorphous phenomenon modeling (Fang and Liao, 2000).

Due to the lack of explicit representation of surface boundaries, CSG display is not directly supported by standard graphics systems. Although several interactive CSG rendering algorithms have previously been developed (Thibault and Naylor, 1987; Westermann and Thomas, 1998), they cannot be directly applied when volume data sets are involved. A potential solution for the rendering of volumetric CSG models is to convert a CSG model into a voxel based volume representation and then construct a R-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. This is the approach used in this work.

2. 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. Constructive Solid Geometry (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 counterbored 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 Figure 1.

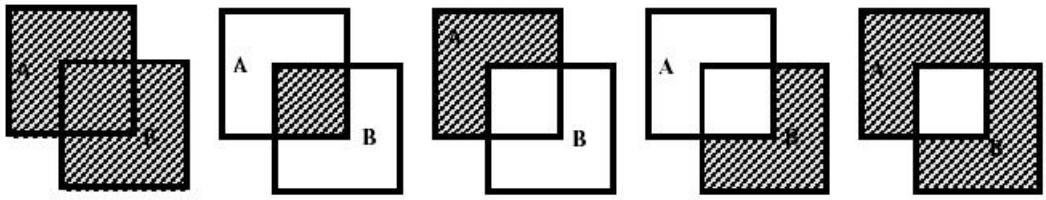

Figure 1. The five non-trivial Boolean combinations of two sets (from left to right): $A + B = \{a : a \in A \text{ or } a \in B\}$, $A \cap B = \{a : a \in A \text{ and } a \in B\}$, $A - B = \{a : a \in A \text{ and } a \notin B\}$, $B - A = \{a : a \in A \text{ and } a \notin B\}$, and the symmetric difference $(A - B) + (B - A)$.

Figure 2 illustrates how a simple syntax may be used to specify a solid in CSG. Parsing such a 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.

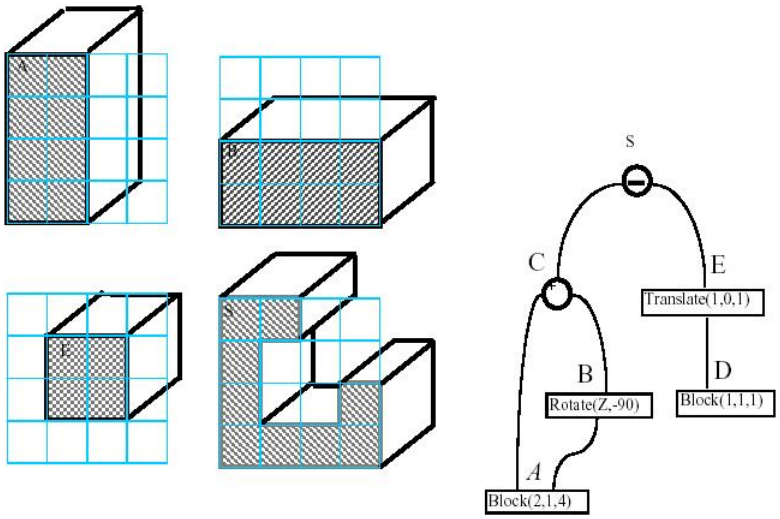

Figure 2. The instances A, B, and E are shown (left) superimposed on the same reference grid. The solid S was specified by the following sequence of commands: $A = Block(2, 1, 4)$; $B = Rotated(A, Z - axis, -90)$; $C = A + B$; $D = Block(1, 1, 1)$; $E = Translated(D, 1, 0, 1)$; $S = C - E$; The corresponding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

2.1 Point Classification for CSG Solids

The CSG tree can be viewed as an implicit description of the geometry of the solid modeled 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 evaluations of P with set operation Op */
P *Combina_P(CSG_Tree *left_P, CSG_Tree *right_P, int Op)
{ ... }

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

3. 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, and 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 tie together the geometry. With the objective of algorithm simplification, mainly in the determination of the circuit of edges surround a face the half-edge 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 can become much simpler (Toriya and Chiyokura, 1991). This way, the modern solid modelers have one entity to represent the edge itself and another entity to represent the circuit of edges surrounds the face. As the B-Rep model can become more complex, it is necessary to add a new primitive: ring (or internal loop). 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.

3.1 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, faces and inner loops in a quantitave manner for solid models:

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

where v is the number of vertices, e is the number os edges, f is the number of faces and r is the number os internal loops. It has been proved by Mäntylä in 1984, 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. Therefore, Euler operators are powerful operations. There are basic Euler operators which are pairwise inverse:

1. Solid creation: a vertex, face and solid are created (<**MVSF**> - Make Vertex Solid Face);
2. Solid removal: a vertex, face and solid are removed (<**KVSF**> - Kill Vertex Solid Face);
3. Vertex splitting: a vertex is split into two vertices connected by a new edge (<**MEV**> - Make Edge Vertex);

4. Vertex joining: two neighbouring vertices are merged into one and the edge between them is deleted. This operator is the inverse of vertex splitting (<**KEV**> - Kill Edge Vertex);
5. Face splitting: a face is split into two faces by inserting a new edge between two vertices of its outer loop (<**MEF**> - Make Edge Face);
6. Face joining: two neighbouring faces are merged by deleting an edge between them. This operator is the inverse of face splitting (<**KEF**> - Kill Edge Face);
7. Loop splitting: a loop is split into two loops, one of them being a new inner boundary of the loop's face, by deleting an edge. This operator will thus create a new inner loop (<**KEMR**> - Kill Edge Make Ring);
8. Loop joining: two loops, at least one of them being an inner loop, are joined to form one loop. Thus, an inner loop is deleted. This operator is the inverse of loop splitting (<**MEKR**> - Make Edge Kill Ring);
9. Create hole: joins two faces, and creates a through-hole. This a necessary operator when a torus is created (<**KFMRH**> - Kill Face Make Ring Hole);
10. Remove hole: creates a new face and remove a through hole (<**MFKRH**> - Make Face Kill Ring Hole).

4. Marching Cubes Algorithm

Marching Cubes is an algorithm for rendering isosurfaces in volumetric data (Loresen and Cline, 1987). The basic notion is that we can define a voxel(cube) by the pixel values at the eight corners of the cube. If one or more pixels of a cube have values less than the user-specified isovalue, and one or more have values greater than this value, we know the voxel must contribute some component of the isosurface. By determining which edges of the cube are intersected by the isosurface, we can create triangular patches which divide the cube between regions within the isosurface and regions outside. By connecting the patches from all cubes on the isosurface boundary, we get a surface representation. This algorithm is often used to extract the surface of medical organs. It provides a fast and easy way to get from serial sections to a complete 3D object

There are a number of 256 possible cube configurations in each of which, the isosurface is triangulated. The weaving wall algorithm is proposed, in which all 256 cube configurations are explicitly defined (Baker,1989). This method, however, is tedious and error prone. In the standard marching cubes implementation, the use of symmetry reduces the number of cases to 15. The complementary symmetry is defined as the equivalence between complementary configurations (Loresen and Cline, 1987). Two configurations are defined as complementary, if the action of the logical NOT operator on one of them generates the other. Besides complementary simmetry, it is possible to rotate one case by any degree over any of the three primary axis and/or mirror the shape across any of the three primary axis. The use of symmetry that reduces the number of cube configurations can produce topologically incoherent surfaces, or 'holes' in certain cases of two adjacent cubes. This is indicated as Type A 'hole problem' (see Figure 3).

A lookup table of predefined cube configurations is the common element of every variation of the standard marching cubes algorithm. However, the algorithm proposed by Delibasis et al. (2001) generates the resulting triangles in every possible cube configuration, without resorting to any predefined cases. This approach orders the isosurface points directly in polygons rather than triangles.

5. CSG to B-Rep Convertion Algorithm

The algorithm proposed by Delibais et al. (2001) constructs the 2D configurations shown in Figure 4.(a), (b) and (c). However, the ambiguity present in cases shown in Figures 4.(d) and (e) (this ambiguity originated the type A hole) is not solved. The algorithm proposed in this work solves the ambiguity, distinguishing when it is necessary to create one case or the other. It is necessary to use the information from the neighboring isocubes to solve such ambiguity.

5.1 Defining the Existence of an Edge

Edges are generated connecting two isopoints placed on the same cube's face and based on the following rules:

- **Condition 0.** The two isopoint must be placed on the same cube's face;
- **Condition 1.** The two isopoint must share one adjacent vertex classified as internal;
- **Condition 2.** The two isopoint must have distinct adjacent vertices (one internal and the other one external) in a way that one adjacent vertex classified as internal must be adjacent to one classified as external;
- **Condition 3.** The two isopoint must share one adjacent vertex external and the other three face's vertices must be classified as internal.

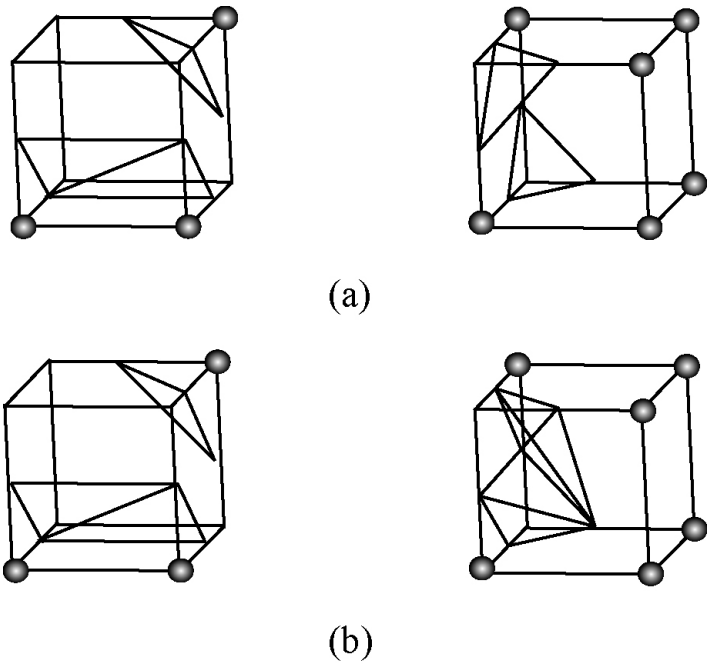

Figure 3. (a) Type A hole example produced by the standard marching cubes algorithm. (b) Solution for the problem.

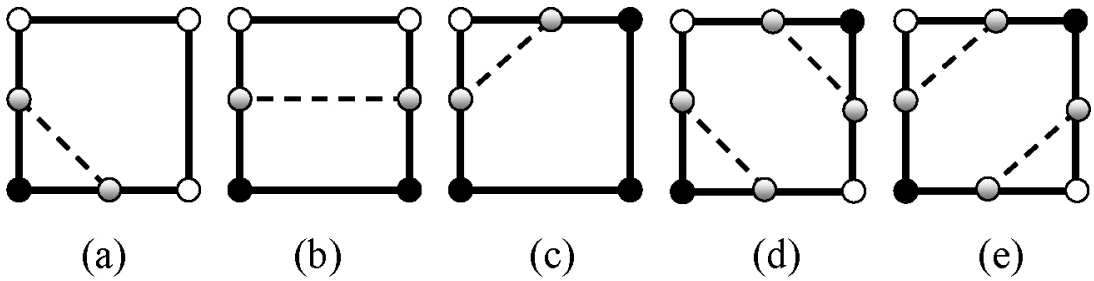

Figure 4. Isopoint located in the middle of the cube's edge and the condition for isopoint existence (vertices in opposite sides of the border).

Finally, if condition 0 associated to condition 1, 2 or 3 is true, then an edge is created.

5.2 The Algorithm

Initially, it is necessary to find an isocube that is on the boundary of the solid. This first isocube is pushed on a stack. While the stack is not empty, the following procedure is done. An isocube is popped from the stack and it is processed. Firstly, the neighboring isocubes that were already processed, are found. The edge configuration of the lateral faces is defined by two distinct methods: the edge configuration is retrieved from the lateral faces of the already processed isocubes; or calculated as shown in the previous section. If it is necessary to calculate the edge configuration and the ambiguity situation is present, then this isocube is not processed and pushed back to the stack.

Considering the case where the ambiguity situation does not exist. In this case, an associated set of triangles is added to the solid and the neighboring isocubes that were not yet processed are pushed back to the stack. There are four different situations for triangle creation: the solid is empty, the triangle has one vertex already created, the triangle has two vertices already created and the triangle has three vertices already created.

5.3 B-Rep solid generation

It is necessary to apply the Euler operators in the proper order to add triangles to the solid. There are four different situations of triangle creation. Each situation is detailed bellow:

5.3.1 No triangle has been generated yet

This case happens only once, it is the first triangle of the solid. It is necessary to firstly apply a **<MVSF>**, then 2 **<MEV>**s and, finally, one **<MEF>**.

5.3.2 Triangle with one Vertex Already Created

Given a vertex a new triangle is created. There are two different situations, as shown in Figure 5. However, the case shown in Figure 5.(c) is not considered in this proposal because an ambiguity is presented. There are two possibilities to create a new triangle in situation shown in Figure 5.(c). The cases shown in Figure 5.(a) and (b) can be implemented by applying two **<MEV>**s and a **<MEF>**.

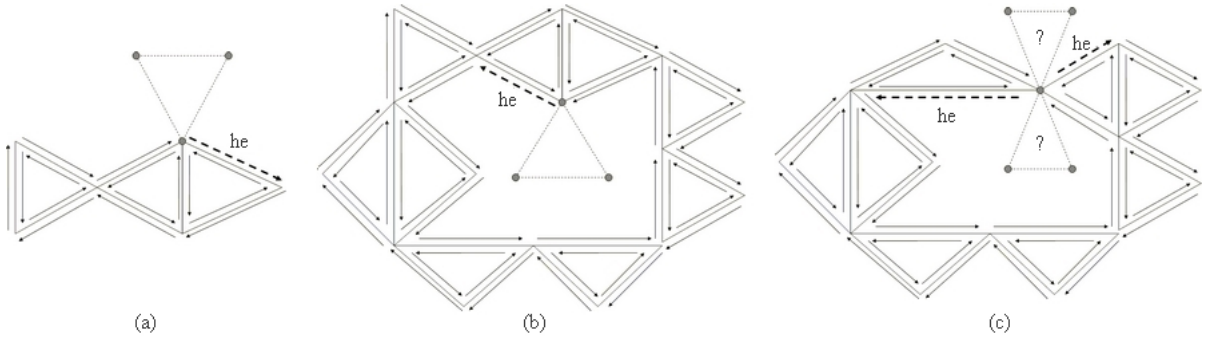

Figure 5. Possible situations where a triangle is created with one vertex already created.

5.3.3 Triangle with two Vertices Already Created

When two vertices are already created, it is necessary to check if exists an edge connecting both vertices. Figures 6. (a), (b) and (c) show all possible situations. These cases can be implemented by applying one **<MEV>** and one **<MEF>**.

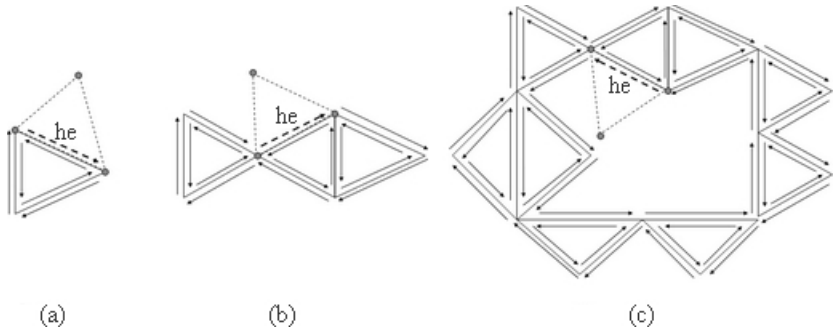

Figure 6. Possible situations where a triangle is created with two vertices already created.

5.3.4 Triangle with three Vertices Already Created

Figure 7 show all possible situations to create a triangle when three vertices are already created. There are four different situations, they described bellow:

- **Situation 1** In this case the triangle is already created, then nothing is done. Usually, this is the last triangle to be added to the solid (see Fig. 7.(a));
- **Situation 2** In this case the edges connecting the three vertices already exist (see Fig. 7.(b)) and it is necessary that they are on the same loop. It is necessary to close the face by connecting the first vertex to the third applying a **<MEF>** Euler operator;
- **Situation 3** In this case, the three vertices are on the same loop. However, there is only one edge connecting two vertices. Then, it is necessary to create two new edges connecting the third vertex to the other two. This is done by applying **<MEF>** twice (see Fig. 7.(c));

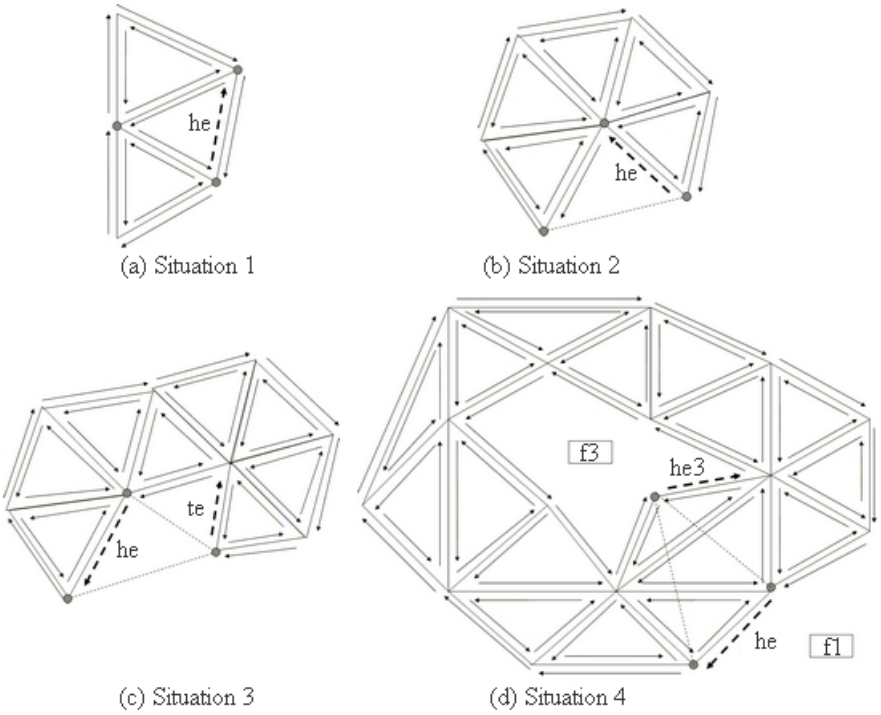

Figure 7. (a), (b), (c) and (d) the possible situations in which the triangle has three vertices already created.

- **Situation 4** In this case, two vertices are on the same loop and they are connected by an edge. The third vertex is in a different loop. This situation occurs in solids topologically similar to a torus. This way, it is necessary to create a through hole. This case is implemented by applying a **<KFMRH>**, a **<MEKR>** and a **<MEF>** (see Fig. 7.(d)).

After finishing all computation, the result is a B-Rep solid with only triangular faces.

6. Results

The algorithm was tested using an ellipsoid with the following equation

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1. \tag{2}$$

The size of the isocube is variable, controlling the level of detail of the resulted solid. Figure 8.(a) shows a sphere with low level of detail. Figures 8.(b), (c) and (d) show the same ellipsoid with different levels of detail.

6.1 Conclusions and Future Works

A new algorithm for CSG to B-Rep convertion was proposed and implemented. The new algorithm is a combination of CSG voxelization and marching cubes. As future work, the new algorithm will be extended to represent 3D medical images in a B-Rep Solid Modeler.

7. Acknowledgements

This research was partially supported by CNPq.

8. References

Baker, H.H.,1989, “Building surfaces of evolution: the Weaving Wall”, *International Journal of Computer Vision*, Vol.3, pp. 51-71.

Boissonat, J. D., 1988, “Shape reconstruction from planar cross sections”., *Computer Vision, Graphics and Image Processing*, Vol.44.

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

Delibasis, K. S., Matsopoulos, G. K., Mouravliansky, N. A. and Nikita, K. S., 2001, “A novel and eficient implementation of the marching cubes algorithm”, *Computerized Medical Imaging and Graphics*, Vol.25, No. 4, pp. 343-352.

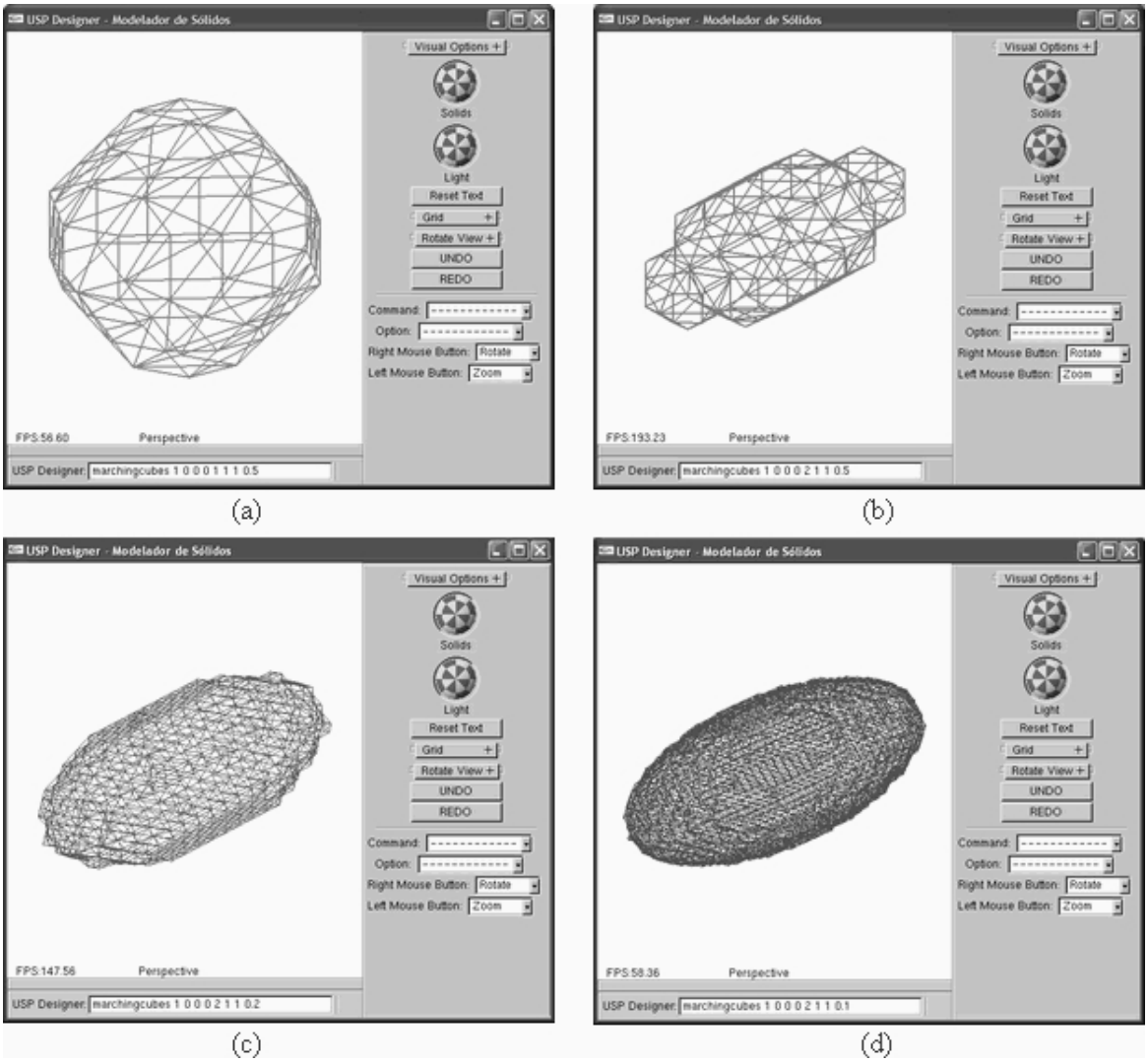

Figure 8. (a) sphere with low level of detail. (b), (c) and (d) an ellipsoid with different levels of detail.

Fang, S. and Liao, D., 2000, "Fast CSG voxelization by frame buffer pixel mapping". *Proceedings of the 2000 IEEE symposium on Volume visualization*, Salt Lake City, Utah, United States, pp. 43-48.

Fuchs H., Kedem Z. M., Uselton S. P., 1977, "Optimal surface reconstruction from planar contours". *Communication of the ACM*, Vol.20.

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.

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

Requicha, A. A. G., 1980, "Representation for rigid solids: Theory, methods and systems". *Computing Surveys*, 12(4):437-464.

Thibault, W. C. and Naylor. B. F., 1987. "Set operations on polyhedra using binary space partitioning trees". *SIGGRAPH'87*, pp. 153-162.

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

Westermann, R. and Thomas. E., 1998. "Efficiently using graphics hardware in volume rendering applications". *SIGGRAPH'98*, pp. 169-177.

Zhou, C., Shu, R. and Kankanhalli, M. S., 1995, "Selectively Meshed Surface Representation", *Computer & Graphics*, Vol.16, No. 6, pp. 793-804.

9. Responsibility notice

The author(s) is (are) the only responsible for the printed material included in this paper