

# Boolean Operations in Open-Source Blender Project

Marc Freixas, Sergi Grau and David Silva

Grup d'Informàtica a l'Enginyeria (GIE), UPC, Barcelona, Spain

---

## Abstract

*This paper describes the work of a new implementation of the Boolean operations of Blender. Blender is a modelling and animation 3D software with GNU General Public License (GPL). Boolean operations are a useful tool for modelling. Previous implementations of Blender Boolean operations have some drawbacks and the Blender users are not totally satisfied with them. The proposed implementation avoids the existing drawbacks of previous implementations and helps users in their modelling stage.*

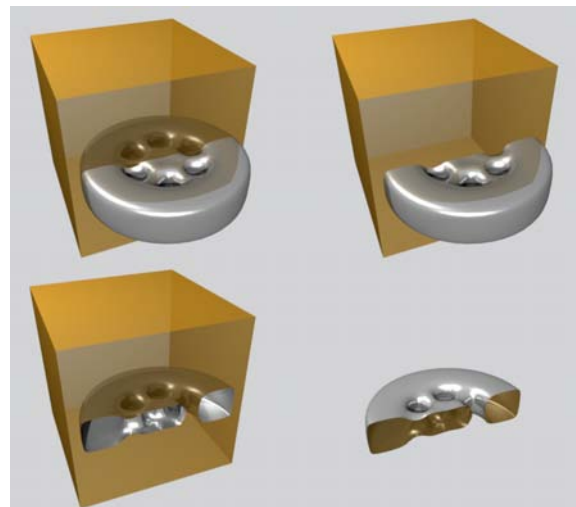
Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Graphics Utilities I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling I.3.6 [Computer Graphics]: Methodology and Techniques

---

## 1. Introduction

In this paper we present the work consisting of a new implementation of Blender Boolean operations. Blender is a modelling and animation 3D software with GNU General Public License (GPL) and it is maintained by a really active community. Blender features are closed to other commercial programs but some aspects still need to be improved.

Many important editing operations can be expressed in terms of Boolean operations on closed objects. Shapes are created using union, intersection or difference. For example, a union operation merges shapes; the difference operation scoops out a hole (see Figure 1). However some users do not use Boolean operations because the original implementation in Blender provides unsatisfactory results. It generates meshes that have duplicated vertices, and they are not sewed (see Figure 6 b). Some developers have offered alternative solutions to this problem. For example, there is one script called MegaBool, that implements Boolean operations in Python. The results of MegaBool are usually satisfactory, but it is a fact that interpreted languages do not have an acceptable computational cost, and it is not easy to integrate this script inside the Blender's source. Another problem of MegaBool is that it does not take into account the properties of the original objects in the final object (see Figure 7 c). The properties of an object do not matter in the modelling stage, but they are needed in the rendering stage.



**Figure 1:** *Upper-left: original objects. Upper-right: union. Down-left: subtraction. Down-right: intersection.*

## 2. Previous work

Boolean operations can be performed in several solid modelling system. Constructive Solid Geometry (CSG) is commonly used for specifying solid models as Boolean combi-

nations of primitives [Req80,HHK89]. CSG expressions can be converted to a polygonal mesh through boundary evaluation, but conversion algorithms are too slow for real-time graphics [RV85, Taw91, KCF\*02, BGM93]. Requicha and Voelcker [RV85] introduce first the boundary evaluation and merging process. Their approach consists basically of three stages. First, it splits the objects by the intersection region (*subdivision stage*). Second, it classifies faces with regard to the objects (*classification stage*). At last, it reconstructs the resulting object using those faces that were correctly classified according to the type of Boolean operation (See union, intersection or difference in Table 1) (*reconstruction stage*).

Boolean Operation	A	B
$\cap$	in(B)	in(A)
$\cup$	out(B)	out(A)
$-$	out(B)	in(A)

**Table 1:** Object classification according to the type of Boolean operation.

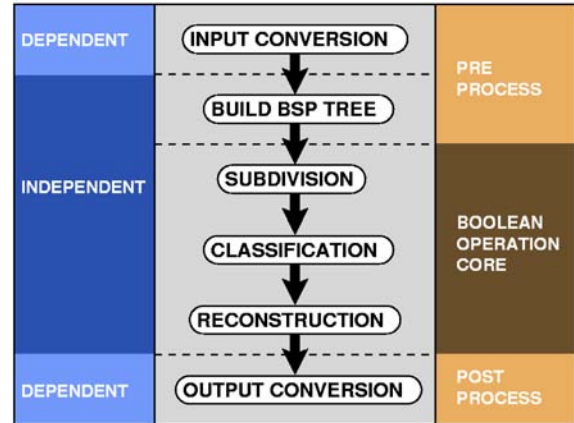
There are several authors that propose different techniques for the subdivision stage. Requicha and Voelcker [RV85] and Miller [Mil88] use an edge-by-edge approach. Mäntylä [Män86] suggests an algorithm called *vertex neighbourhood classifier* based on an edge-by-face strategy. Later, other authors like Hoffmann, Hopcroft and Karasick [HHK89] and Chiyoruka [Chi88] also suggest boundary evaluation algorithms based on a face-by-face approach. Kunwoo Lee [Lee99] describes an algorithm that computes the intersected segments between two faces (called xegment). Then it takes into account the xegments to subdivide each face. Rivero and Feito [RF04] present a technique to detect intersection between triangular faces, and their consistent subdivision.

Binary Space Partitioning Trees (BSP trees [FAG83]) have been used for classification in several strategies ([TN87,NAT90,PY89]). The BSP tree of a solid object can be used to classify any point with regard to the object. The point can be inside (IN), outside (OUT) or on the boundary (ON) of the solid. Using this basic test, the classification of one face according to a solid object is also simple. In the classification stage, there is one BSP tree for each object, and the faces are classified using the BSP of the other object.

### 3. Boolean operations algorithm

The implementation presented in this paper had to fit inside an existing platform (Blender), where the data structure used to represent an object is a mesh of triangles or quads. This fact simplifies the representation of the general B-Rep structure.

For easiness and robustness, Boolean union ( $\cup$ ) and difference ( $-$ ) are implemented in terms of intersection ( $\cap$ ) and



**Figure 2:** Boolean Operation Algorithm's Pipeline can be subdivided into dependent or independent on the type of the operation (left). This pipeline can also be subdivided into preprocess, core or postprocess stages (right).

complement ( $\neg$ ) (See Table 2). Complement operation consists in flipping normal vectors and inverting vertex order of all the mesh faces.

Boolean Operation	Equivalence
$A \cup B$	$\neg((\neg A) \cap (\neg B))$
$A - B$	$A \cap (\neg B)$

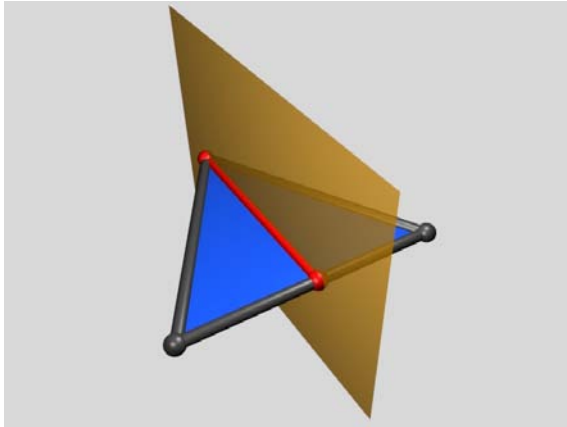
**Table 2:**  $\cup$  and  $-$  expressed in terms of  $\cap$  and  $\neg$ .

This simplification subdivides the pipeline of the Boolean operations algorithm in two kind of stages: dependent or independent on the type of the operation ( $\cap, \cup, -$ ) (See Figure 2).

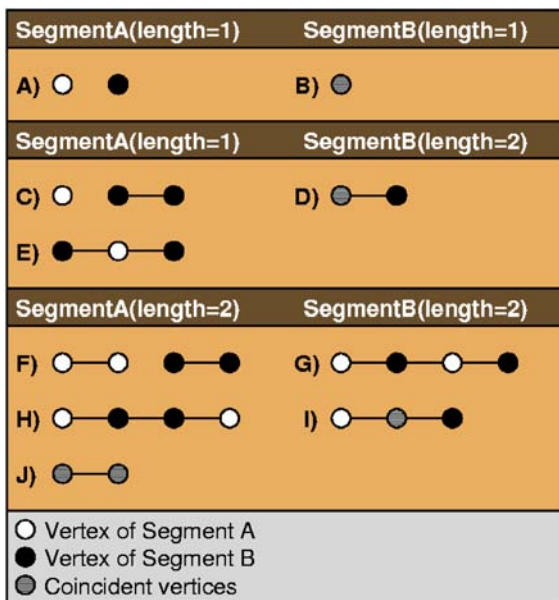
The core of the strategy follows the three stages explained before: subdivision, classification and reconstruction stages. Before these stages, there is a preprocess that prepares all the structures needed for the next stages. Firstly, the input meshes are negated if it is required. The other tasks of the preprocess are explained on each stage that it needs. After all the stages, there is a postprocess that negates the output mesh in case it is needed (in  $\cup$  operation).

#### 3.1. Subdivision stage

The subdivision stage is based on Kunwoo Lee [Lee99], but it is simplified because the input faces are only triangles (quads are split in the preprocess). In this face-by-face approach we intersect each face of one object against the faces of the other object. For each couple of faces, it computes their xegment and splits both faces.



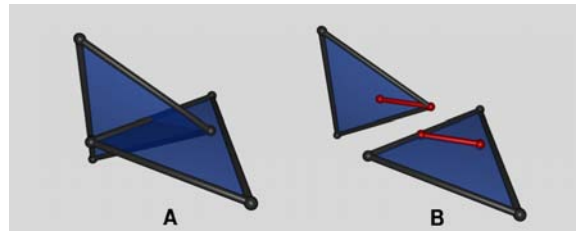
**Figure 3:** Intersection between a triangle and plane is shown with a red segment. Each end of this segment can lie on existing vertex or edge of the triangle.



**Figure 4:** In case that exists intersection between both triangles, the intersection of each one against the plane of the other face generates two segments (with one or two points). Segment A represents their points in white, and segment B in black. Both segments are collinear and can overlap. The xegment is the corresponding overlapping region.

**3.1.1. Xegment creation process**

This process computes the possible intersection between two faces, intersecting each face against the plane of the other face. Intersection between the face and the plane generates a segment with one or two points that can lie on vertices or edges of the face (see red segments in Figure 3). When



**Figure 5:** A shows the intersection between two triangles. Red segments represents in B the computed xegment on each triangle.

there is intersection between both faces, the segments are overlapped. Figure 4 shows all the configurations of two segments. Two faces do not intersect when there are not two segments or when they have configuration A or B. The region shared by both segments represents the intersection segment between both faces (xegment).

When the xegment does not begin or end on a face vertex, a new vertex is created and added to the mesh vertices set. When it begins or ends on a vertex of only one face, this one is used. But when the xegment begins or ends on vertices of both faces, one of them replaces the other. The vertex that persists is used in the split process. This criterion of vertex replacement avoids a posterior sewing process of the final mesh in the reconstruction stage.

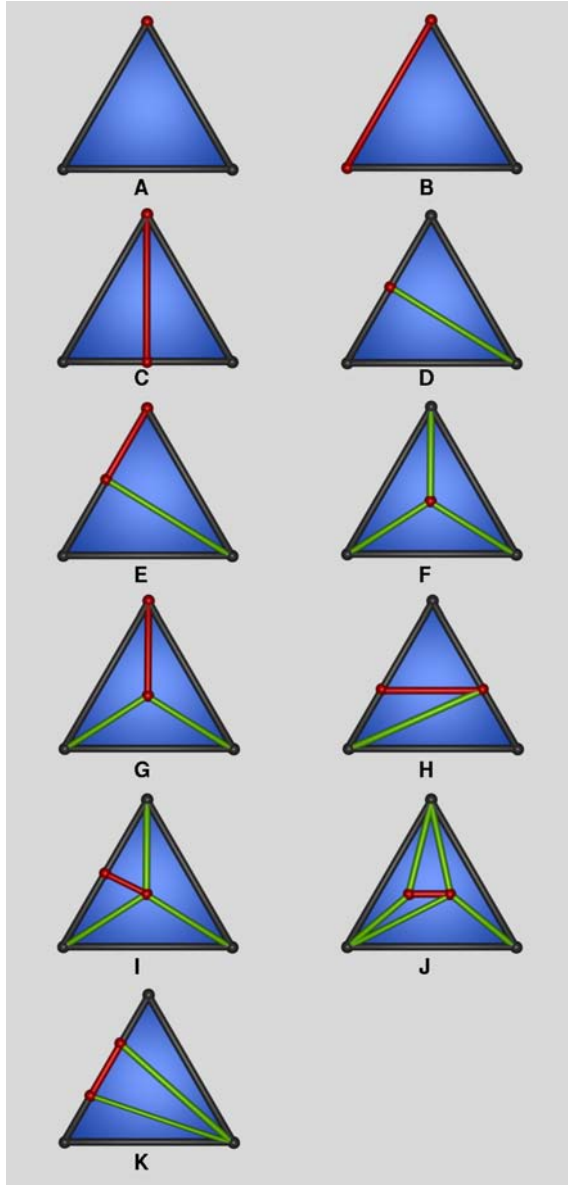
**3.1.2. Split process**

This process subdivides the faces taking into account the xegment computed in the previous process (see Figure 5). Figure 6 shows the possible locations of a xegment (red) with regard to a face and the new edges (green) resulting from the splits. The old faces are discarded and the new generated faces are added to the face-by-face process.

**3.2. Classification stage**

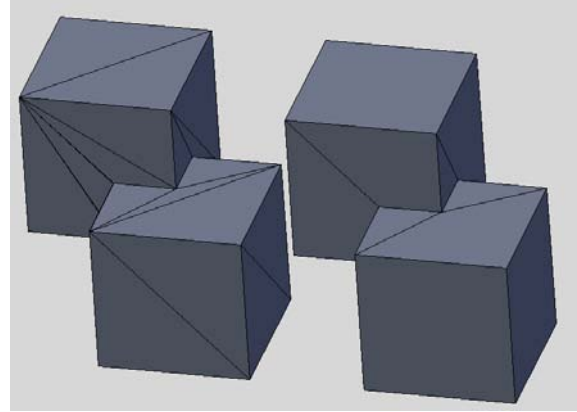
This stage receives a set of faces for each object (that are original faces or come from original faces). For both objects, this stage classifies its faces using the BSP tree of the other object. The BSP trees had been computed in the preprocess (see Figure 2).

The faces resulting from the subdivision stage can only have vertices classified as IN and ON, or OUT and ON. This happens due to the BSP tree being constructed using the object faces and these same faces subdivide the face of the other object, if it is required. This property allows to simplify the classification as IN, OUT or ON and accelerates this stage. In case that at least, one of the vertices is IN, the face is also IN. If one of the vertices is OUT, the face is OUT. Otherwise the face is ON. When a face is classified ON we compare its normal vector with the plane normal vector of



**Figure 6:** All configuration of the xegment inside a triangle. Red segments in B are the resulting xegment on each triangle, and green segments are the new edges created by the splitting process. Triangles A and B do not need to split. Configurations {C,D,E} and {F,G} use the same split strategy. Cases H, I, J and K do a particular split

the BSP node, if both are the same we will classify IN, and OUT otherwise.



**Figure 7:** Left object represents the result of a Boolean operation without reconstruction stage. Right object represents the result of a Boolean operation with reconstruction stage.

### 3.3. Reconstruction stage

The reconstruction stage has been simplified because there are not duplicated vertices but only those faces that represent the boundary of the final object, and all the faces are connected. These faces could be a result of the Boolean operation. The drawback of this output is that it could be very fragmented (see Figure 7). The objective of this implementation is to help Blender users to design their objects. So if they receive a mesh with unnecessary split faces, they could be disappointed with the result. For this reason, we use the reconstruction stage to reduce the number of faces, merging those faces that come from the same original face. This stage generates a mesh of triangles and quads, although it receives only triangles. Only convex quads are allowed so before creating a quad we check its convexity and we discard it if it is not a convex quad.

The reconstruction process is subdivided in two stages:

- Merge of faces removing vertices.
- Merge of faces removing edges.

Both are executed successively, until one of them ends without producing any merge.

#### 3.3.1. Merge via removing vertices

We say that a vertex is candidate to be removed or *removable* if it didn't exist on the initial vertices set. For this reason, we only can remove those vertices generated during one of the intermediate Boolean operation stages. The rest of vertices and faces without *removable* vertices will be ignored during this stage.

For each *removable* vertex  $v_i$  of a mesh  $M$ , the first step consists in finding the set of faces  $L_{i1} \dots L_{iN}$  where

- $L_{ij}$  is the faces set of  $M$  that contains  $v_i$  and come from the original face  $C_j$ .

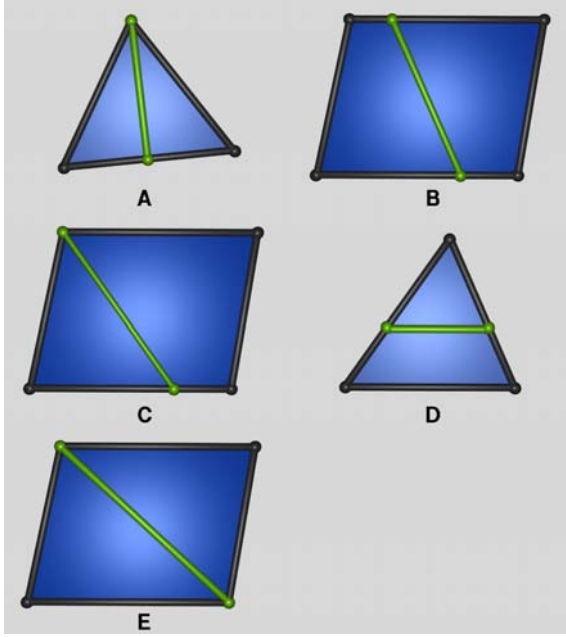


Figure 8: Possible configurations of merge faces.

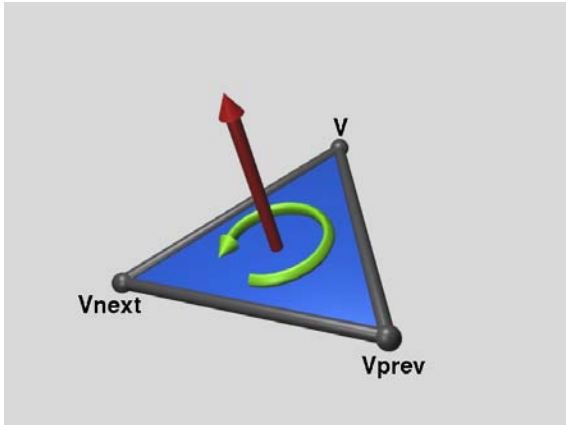


Figure 9: The vertices  $v_{prev}$  and  $v_{next}$  of  $v$  inside a triangle are computed according to the vertex order.

- $L = L_{i1} \cup \dots \cup L_{iN}$  is the faces set of  $M$  that contains  $v_i$ .

The next step consists in the application of merge patterns on those faces in the same set  $L_{ij}$  (see Figure 8 A-D). Each of these patterns receives two faces with  $v_i$ , and in case of success it produces a new face without it as a result of the merge. The new faces are added in a new set  $L'$  while the merged faces are removed from their corresponding set  $L_{ij}$ . The process ends when neither of the patterns can't be applied or all set  $L_{ij}$  is void. In the first case, we have two or more faces that could not be merged, so  $v_i$  can't be removed

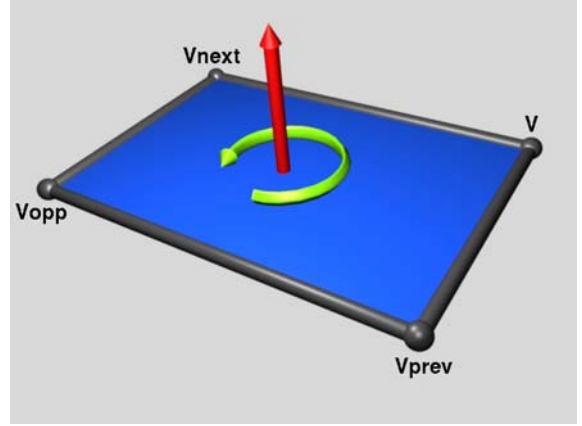


Figure 10: The vertices  $v_{prev}$ ,  $v_{next}$  and  $v_{opp}$  of  $v$  inside a quad are computed according to the vertex order.

and the mesh keeps intact. In the second case,  $v_i$  and the faces of  $L$  are removed from the mesh, and the faces of  $L'$  are added.

The patterns B-D in Figure 8 present as a special feature, the fact that they require to remove a second vertex in order to merge the input faces. This second vertex is added to a list of pending vertices that must be removed to eliminate  $v_i$ , generating a chain of remove-dependencies between the vertices. If the process fails for one of them, neither can be removed. Otherwise, the vertices and their faces are removed, and the new merged faces are added.

Figures 9 and 10 show definitions of  $v_{prev}$ ,  $v_{next}$  and  $v_{opp}$  and here we define some functions used later:

$$\begin{aligned} \text{Collinear}(a,b,c) &\Leftrightarrow \text{Segment}(a,b) \parallel \text{Segment}(b,c) \\ \text{Between}(a,b,c) &\Leftrightarrow a \neq b \wedge a \neq c \wedge a \in \text{Segment}(b,c) \end{aligned}$$

The pattern A can be formalised as:

**Require:**  $\text{Triangle}(v_0, v_1, v_2) \wedge \text{Triangle}(w_0, w_1, w_2) \wedge \exists i, j : 0..2 : v_i = w_j$   
**if**  $v_{i_{prev}} = w_{j_{next}} \wedge \text{Between}(v_i, v_{i_{next}}, w_{j_{prev}})$  **then**  
     **return**  $\text{Triangle}(v_{i_{prev}}, w_{j_{prev}}, v_{i_{next}})$   
**else if**  $v_{i_{next}} = w_{j_{prev}} \wedge \text{Between}(v_i, v_{i_{prev}}, w_{j_{next}})$  **then**  
     **return**  $\text{Triangle}(v_{i_{prev}}, w_{j_{next}}, v_{i_{next}})$   
**end if**

The pattern B can be formalised as:

**Require:**  $\text{Quad}(v_0, v_1, v_2, v_3) \wedge \text{Quad}(w_0, w_1, w_2, w_3) \wedge \exists i, j : 0..3 : v_i = w_j$   
**if**  $v_{i_{prev}} = w_{j_{next}} \wedge \text{Between}(v_i, w_{j_{prev}}, v_{i_{next}}) \wedge \text{Between}(w_{j_{next}}, w_{j_{opp}}, v_{i_{opp}})$  **then**  
     **return**  $\text{Quad}(w_{j_{opp}}, w_{j_{prev}}, v_{i_{next}}, v_{i_{opp}})$   
**else if**  $v_{i_{next}} = w_{j_{prev}} \wedge \text{Between}(v_i, v_{i_{prev}}, w_{j_{next}}) \wedge \text{Between}(v_{i_{next}}, v_{i_{opp}}, w_{j_{opp}})$  **then**  
     **return**  $\text{Quad}(v_{i_{opp}}, v_{i_{prev}}, w_{j_{next}}, w_{j_{opp}})$

**end if**

The pattern C can be formalised as:

**Require:**  $Quad(v_0, v_1, v_2, v_3) \wedge Triangle(w_0, w_1, w_2) \wedge \exists i : 0..3, j : 0..2 : v_i = w_j$   
**if**  $v_{i_{prev}} = w_{j_{next}} \wedge Between(v_i, v_{i_{next}}, w_{j_{prev}}) \wedge \neg Collinear(w_{j_{prev}}, v_{i_{prev}}, v_{i_{opp}})$  **then**  
  **return**  $Quad(v_{i_{opp}}, v_{i_{prev}}, w_{j_{prev}}, v_{i_{next}})$   
**else if**  $v_{i_{next}} = w_{j_{prev}} \wedge Between(v_i, v_{i_{prev}}, w_{j_{next}}) \wedge \neg Collinear(w_{j_{next}}, v_{i_{next}}, v_{i_{opp}})$  **then**  
  **return**  $Quad(v_{i_{opp}}, v_{i_{prev}}, w_{j_{next}}, v_{i_{next}})$   
**end if**

The pattern D can be formalised as:

**Require:**  $Quad(v_0, v_1, v_2, v_3) \wedge Triangle(w_0, w_1, w_2) \wedge \exists i : 0..3, j : 0..2 : v_i = w_j$   
**if**  $v_{i_{prev}} = w_{j_{next}} \wedge Between(v_i, v_{i_{next}}, w_{j_{prev}}) \wedge Between(v_{i_{prev}}, w_{j_{prev}}, v_{i_{opp}})$  **then**  
  **return**  $Triangle(v_{i_{next}}, v_{i_{opp}}, w_{j_{prev}})$   
**else if**  $v_{i_{next}} = w_{j_{prev}} \wedge Between(v_i, v_{i_{prev}}, w_{j_{next}}) \wedge Between(v_{i_{next}}, v_{i_{opp}}, w_{j_{next}})$  **then**  
  **return**  $Triangle(v_{i_{prev}}, w_{j_{next}}, v_{i_{opp}})$   
**end if**

### 3.3.2. Merge via removing edges

This stage is much simpler than the previous one. It generates quads merging two triangles, using the pattern of the Figure 8 E, when these triangles share an edge and come from the same parent face. If this pattern is successful, the input triangles are removed from the mesh and the resulting quad is added to its faces set. This pattern can be formalised as follow:

**Require:**  $Triangle(v_0, v_1, v_2) \wedge Triangle(w_0, w_1, w_2) \wedge \exists i, j : 0..2 : v_i = w_j$   
**if**  $v_{i_{prev}} = w_{j_{next}} \wedge \neg Collinear(v_{i_{next}}, v_i, w_{j_{prev}}) \wedge \neg Collinear(v_{i_{next}}, v_{i_{prev}}, w_{j_{prev}})$  **then**  
  **return**  $Quad(v_i, v_{i_{next}}, v_{i_{prev}}, w_{j_{prev}})$   
**else if**  $v_{i_{next}} = w_{j_{prev}} \wedge \neg Collinear(v_{i_{prev}}, v_i, w_{j_{next}}) \wedge \neg Collinear(v_{i_{prev}}, v_{i_{next}}, w_{j_{next}})$  **then**  
  **return**  $Quad(v_i, w_{j_{next}}, v_{i_{next}}, v_{i_{prev}})$   
**end if**

## 4. Results

Table 7 shows some figures that compare the original Blender Booleans with our implementation and the Mega-Bool script. We can see that the original strategy produces too many triangles and the meshes are not solids (a,g,j). The other two strategies preserve original quads (b,c). MegaBool constructs clean meshes with spectacular nice triangles (l), but sometimes generates unexpected holes (i) and does not assign the correct property values for each faces, like its material (c). In the second row (d,e,f), two solids that are coincident by one face and one vertex are joined. Megabool (f)

computes a wrong union operation, and the original strategy (d) does not detect the coincidence.

The performance tests have been done in a Pentium IV 3Ghz with 512Mb of memory. We can consult the used samples sizes in the Table 3. In Table 4 we can observe the results of computing the different Boolean operations with the sample models. In this Table we can see that the old version is faster than the new one, but we have seen this one does not compute the solid object well. In the case of the Mega-bool Python script, it is considerably slower. For the dense models it has memory problems and it does not compute the result. This low performance is due to the interpreted language implementation.

In Table 5 we show the performance of each pipeline stage. The most expensive stage in the Boolean process is the subdivision stage, followed by the reconstruction stage. As a future work we aim to improve the subdivision stage so that it realizes less and better divisions, and as a consequence we will be able to reduce the work for the reconstruction stage.

Sample	size of mesh A	size of mesh B
sphere drill	1984	12
two spheres	1984	1984
wheel low	4704	12
wheel high	18816	12

**Table 3:** Samples used in the performance tests.

Old version	$\cap$	$\cup$	$-$
sphere drill	0.22''	0.23''	0.25''
two spheres	0.47''	0.49''	0.49''
wheel-low	0.19''	0.20''	0.20''
wheel-high	0.73''	0.80''	0.82''
MegaBool	$\cap$	$\cup$	$-$
sphere drill	2.78''	2.90''	3.03''
two spheres	10.82''	10.82''	11.10''
wheel-low	out of mem	out of mem	out of mem
wheel-high	out of mem	out of mem	out of mem
New version	$\cap$	$\cup$	$-$
sphere drill	0.50''	0.49''	0.51''
two spheres	1.59''	2.04''	1.63''
wheel-low	1.05''	0.97''	0.98''
wheel-high	22.97''	27.07''	26.72''

**Table 4:** Execution times for all Boolean implementations.

## 5. Conclusions and future work

The present work has allowed to introduce us to the world of free software and to be able to collaborate with the Blender community. The fact that everybody has access to the Blender source code turns it an ideal platform for the development of projects related to computational geometry.

<b>sphere drill</b>	$\cap$	$\cup$	$-$
Preprocess	0.21''	0.20''	0.22''
Subdivision	0.17''	0.17''	0.17''
Classification	0.06''	0.06''	0.06''
Reconstruction	0.06''	0.06''	0.06''
Total	0.50''	0.49''	0.51''
<b>two spheres</b>	$\cap$	$\cup$	$-$
Preprocess	0.42''	0.38''	0.40''
Subdivision	0.81''	0.78''	0.78''
Classification	0.30''	0.28''	0.29''
Reconstruction	0.06''	0.60''	0.16''
Total	1.59''	2.04''	1.63''
<b>wheel-low</b>	$\cap$	$\cup$	$-$
Preprocess	0.22''	0.22''	0.21''
Subdivision	0.61''	0.53''	0.56''
Classification	0''	0''	0.01''
Reconstruction	0.22''	0.22''	0.20''
Total	1.05''	0.97''	0.98''
<b>wheel-high</b>	$\cap$	$\cup$	$-$
Preprocess	1.77''	1.70''	1.69''
Subdivision	12.14''	13.62''	13.56''
Classification	0.02''	0.02''	0.02''
Reconstruction	8.86''	11.55''	11.27''
Total	22.97''	27.07''	26.72''

**Table 5:** Execution times for each Boolean stage.

We recoded Blender's Boolean Operations, and they return intuitive results that conserve properties of the input objects. This new implementation is available since Blender 2.40.

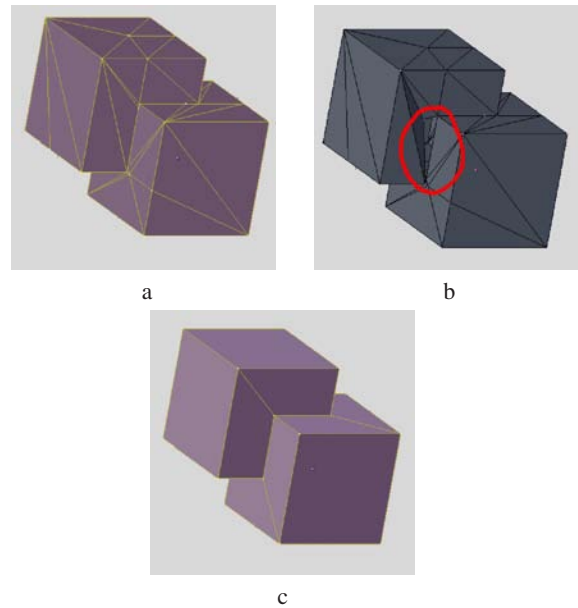
We will try to speed up the algorithm by simplifying the subdivision stage. This stage will be divided in two sub-stages. First we will compute all segments, using a face-by-face test, without splitting the triangles. Next we will tessellate all faces avoiding unnecessary splits.

## 6. Acknowledgements

This work has been made thanks to the program of scholarships "Summer of Code 2005" of Google for the development of free software projects. We thank Alex Ewering and the Blender community for their aid and technical support. This work has been partially supported by a CICYT grant MAT2005-07244-C03-03 and TIN2004-06326-C03-01.

## References

[BGM93] BANERJEE R. P. K., GOEL V., MUKHERJEE A.: Efficient parallel evaluation of csg tree using fixed number of processors. In *SMA '93: Proceedings on the second ACM symposium on Solid modeling and applica-*



**Table 6:** Union of two cubes. (a&b) Some vertices lay on edges. The result is not a solid mesh and it has a lot of unnecessary vertices. (c) The result is a well sewed mesh with the minimum number of necessary vertices.

tions (New York, NY, USA, 1993), ACM Press, pp. 137–146. 2

[Chi88] CHIYOKURA H.: *Solid Modeling with Database: Theory and Implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988. 2

[FAG83] FUCHS H., ABRAM G. D., GRANT E. D.: Near real-time shaded display of rigid objects. In *SIGGRAPH '83: Proceedings of the 10th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1983), ACM Press, pp. 65–72. 2

[HHK89] HOFFMANN C. M., HOPCROFT J. E., KARASICK M. E.: Robust set operations on polyhedral solids. *IEEE Comput. Graph. Appl.* 9, 6 (1989), 50–59. 2

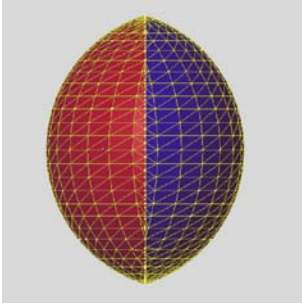
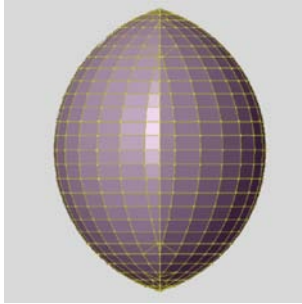
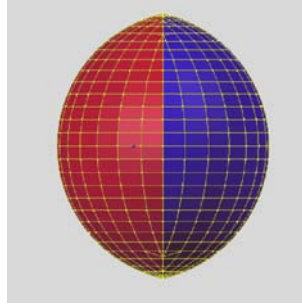
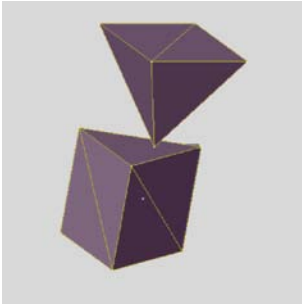
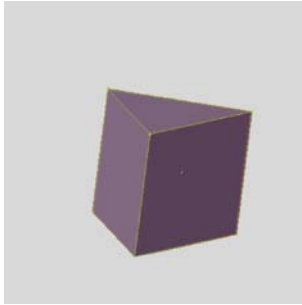
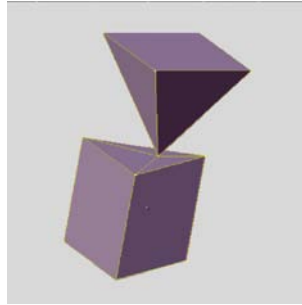
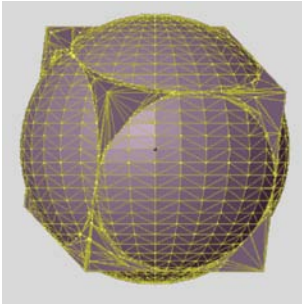
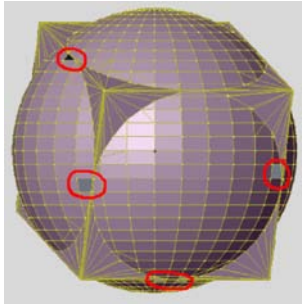
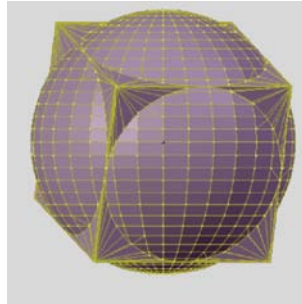
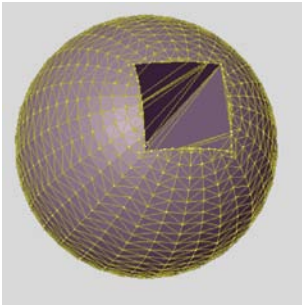
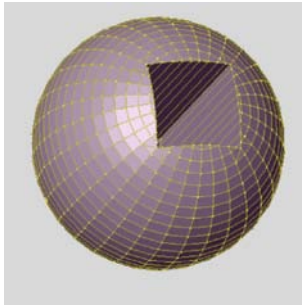
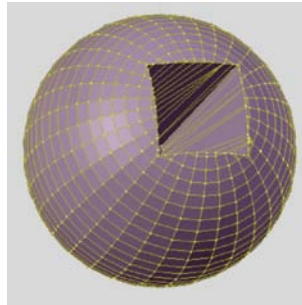
[KCF\*02] KEYSER J., CULVER T., FOSKEY M., KRISHNAN S., MANOCHA D.: Esolid—a system for exact boundary evaluation. In *SMA '02: Proceedings of the seventh ACM symposium on Solid modeling and applications* (New York, NY, USA, 2002), ACM Press, pp. 23–34. 2

[Lee99] LEE K.: *Principles of CAD/CAM/CAE Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 2

[Män86] MÄNTYLÄ M.: Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Trans. Graph.* 5, 1 (1986), 1–29. 2

- [Mil88] MILLER J. R.: Analysis of quadric-surface-based solid models. *IEEE Comput. Graph. Appl.* 8, 1 (1988), 28–42. [2](#)
- [NAT90] NAYLOR B., AMANATIDES J., THIBAUT W.: Merging bsp trees yields polyhedral set operations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), ACM Press, pp. 115–124. [2](#)
- [PY89] PATERSON M. S., YAO F. F.: Binary partitions with applications to hidden surface removal and solid modelling. In *SCG '89: Proceedings of the fifth annual symposium on Computational geometry* (New York, NY, USA, 1989), ACM Press, pp. 23–32. [2](#)
- [Req80] REQUICHA A. A. G.: Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv.* 12, 4 (December 1980), 437–464. [2](#)
- [RF04] RIVERO M. L., FEITO F.: Refinamiento de mallas triangulares. Aplicación para el cálculo de operaciones Booleanas en 3D. *XIV Congreso Español de Informática Gráfica* (2004), 77–90. [2](#)
- [RV85] REQUICHA A., VOELCKER H.: Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *P-IEEE* 73 (1985), 30–44. [2](#)
- [Taw91] TAWFIK M. S.: An efficient algorithm for csg to b-rep conversion. In *SMA '91: Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications* (New York, NY, USA, 1991), ACM Press, pp. 99–108. [2](#)
- [TN87] THIBAUT W. C., NAYLOR B. F.: Set operations on polyhedra using binary space partitioning trees. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1987), ACM Press, pp. 153–162. [2](#)



Old Version	Megabool script	New Version
 <p data-bbox="454 593 470 616">a</p>	 <p data-bbox="790 593 805 616">b</p>	 <p data-bbox="1125 593 1141 616">c</p>
 <p data-bbox="454 963 470 985">d</p>	 <p data-bbox="790 963 805 985">e</p>	 <p data-bbox="1125 963 1141 985">f</p>
 <p data-bbox="454 1332 470 1355">g</p>	 <p data-bbox="790 1332 805 1355">h</p>	 <p data-bbox="1125 1332 1141 1355">i</p>
 <p data-bbox="454 1702 470 1724">j</p>	 <p data-bbox="790 1702 805 1724">k</p>	 <p data-bbox="1125 1702 1141 1724">l</p>

**Table 7:** Comparison between the different strategies.