Visualization algorithm for CSG polyhedral solids
Anne Verroust

To cite this version:
Visualization algorithm for CSG polyhedral solids

Anne Verroust
INRIA, Domaine de Voluceau,
78150 LE CHESNAY CEDEX, FRANCE
LIENS, Ecole Normale Supérieure,
45 rue d’Ulm, 75230 PARIS CEDEX 05, FRANCE

Abstract

An algorithm is presented here to visualize CSG solids in wire frame with hidden faces eliminated. The approach taken is to construct the image of the CSG solid directly from the CSG tree. This algorithm takes into account face coherence property and the depth of the faces to minimize the number of rays fired during the process. It mixes a two dimensional polygonal clipping and a ray casting algorithm.

Key words : solid modelling, constructive solid geometry, hidden surface algorithms, ray-casting.

Introduction

Constructive Solid Geometry (CSG) is one of the classical representations of three-dimensional solids. It is particularly well adapted to ray-tracing algorithms [Kaj] which produce realistic images with reflections, refractions and transparencies. The aim, here, is to make a CAD part of a ray-tracing program. Thus it is necessary to generate images rapidly from CSG models. There are two ways to solve this problem:

• construct a boundary surface model from the CSG tree and use a hidden surface algorithm on it (see [HLT] and [RVo] for more details).

• compute directly the image of the CSG tree without computing the boundaries of the CSG solid.

We are interested more particularly in the second type of methods. Most of them use as basis the ray-casting method, first introduced by Roth [Rot]. As this method is time consuming, many authors have proposed improvements to this algorithm.

Roth himself introduces box enclosure to eliminate some parts of the screen and, when the silhouette of the solid is wanted, sample the screen with rays and then locate the first visible face via a binary search. This can induce imprecisions in the final image if the sample size is too large.

Bronsvort, Jansen and Van Wijk [BJV] propose also to use sampling and scan line enclosure instead of box enclosure and simplify the CSG tree in an “active CSG tree” to reduce boolean calculation in each ray. However, they recognize that the computation
time is too big to make the visualization being interactive.

Atherton [Ath] uses several coherence properties (span, visible-segment, scan-line and object coherences) to fire rays in crucial points of the screen. His solution is adapted to scan-line visualization.

Some authors improve the computation of the first visible surface in ray-tracing algorithm ([GHW] and [Jan]). Nevertheless they are concerned by ray-tracing more than by ray-casting and their improvements are interesting when a great amount of rays are fired.

More recently several authors have taken much attention to depth coherence in their algorithms:

Crocker [Cro] introduces the notion of “invisibility coherence” to decrease the visualization time for scan-line surface algorithms. The intuitive idea is to eliminate the surfaces that are likely to be invisible, knowing their minimal z-depth and the different z-depths of the previous scan line. He also adapts his method to Constructive Solid Geometry by improving Atherton’s algorithm. His measurements show the advantages of the method.

Okino, Kakazu and Morimito [OHM] and Requicha and Rossignac [RRo] both use an extended depth buffer in their visualization algorithms.

The approach presented here is largely inspired by Atherton’s solution [Ath]. Atherton’s visualization process is adapted to scan-line algorithms, thus he uses span and scan-line coherence. We want to display a CSG solid composed of polyhedral objects in wire frame. The final image is viewed as a partition of the screen in polygonal zones, each polygonal zone representing a part of the projection of a face of the CSG solid in the screen (any polygonal zone of the screen will be called window). Window coherence is used in the algorithm. The main idea is the following:

The result of the ray-cast on objects transformed by a perspective orthographic transform depends only on the respective order of the z-depths of the different objects encountered. Thus, if we make a subdivision of the screen in windows where the z-depth order of the faces covering each window is ensured to be constant inside it, at least until the first visible face, it is enough to fire a ray from any point inside the window to know the visible face for all the points inside the window.

In a primary version [Ver], we have followed this idea and made a subdivision of the screen in windows such that:

- if a face of an object overlaps a window, this face entirely contains the window.
- if two faces intersect in the space, no window may contain any portion of the intersecting line except on its boundary.

In the current version, rectangle enclosure and a kind of “active CSG tree” [BJV] is used to eliminate some parts of the screen. This algorithm will be described in the next section.

In most cases, this subdivision is too thin. The problem is then to minimize the cardinality of the subdivision without altering the final image. To this purpose, using the
z-depth of the faces,
- during the two dimensional clipping process, some *certainly* invisible faces are eliminated.
- we compute only spatial intersections which may change the result of the ray-cast. They are called *consequent* intersections.

These eliminations are explained and a new algorithm of visualization is described in the third section of the paper.

Some measurements of the new algorithm are also given in the final section.

**Description of the visualization process**

The CSG solid is a CSG tree of polyhedral primitives. There are no condition on the type of allowed polygons, it is assumed that different faces of a same object do not intersect in the space. During the whole process, we work with the projections of the objects on the screen except when we compute the spatial intersections of objects and when we fire a ray. Thus the primitives are identified with collections of faces of the screen during most of the explanation of the algorithm.

The algorithm is decomposed in three phases:

- “preparation phase”: a first rough partition of the screen into rectangles.
- “clipping phase”: a subdivision of the screen in windows where the result of the ray-cast is ensured to be constant for all the points inside each window.
- “merging phase”: a final subdivision of the screen where the windows associated to the same visible face are joined.

Before going into the description of these three phases, some functions used during the whole process are introduced:

FUNCTIONS

RAY-CAST(window,cover_list,visible) : fires a ray from a point P inside the window, evaluate the boolean expression of the z-depths of the faces of COVER_LIST on the “active CSG tree” corresponding and put in VISIBLE the result obtained.

INSERT(list,face) : inserts face in the list and returns it. This insertion preserves the existing order in the list.

DEL(list,face) : deletes face from the list and returns the result.

EXTRACT(list,face) : returns the first face of the list.

SPATIAL(window,list1,list2) : returns a collection of windows, result of the partitionment of the window by the projections of the spatial intersections between the faces of list1 and the faces of list2.
INTER(window, face) : computes the windows resulting from the intersection of the window and the face. These windows have DEL(candidate list, face) as CANDIDATE_LIST and INSERT(cover list, face) as COVER_LIST and no visible face associated.

DIFF(window, face) : computes the windows resulting from the difference of the window and the face. These windows have the same COVER_LIST than the initial window and DEL(candidate list, face) as CANDIDATE_LIST and no visible face associated.

MERGE(window1, window2) : computes the windows resulting from the union of window1 and window2.

The faces of CANDIDATE_LIST and COVER_LIST are lexicographically sorted by:
- the order of the objects in a postorder traversal of the CSG tree,
- the decreasing order of the minimum z-depth of the faces, \( z_{min}(face) \),
- the decreasing order of the maximum z-depth of the faces, \( z_{max}(face) \).

**Preparation phase**

To begin, a perspective transform is made on all the primitives. During this transformation, the faces without thickness are eliminated and the others are oriented in clockwise order. The faces of each object are sorted in decreasing order versus their minimal and their maximal z-depths. A bounding rectangle is associated to each primitive. We make then a rough partition \( P \) of the screen using the CSG tree with the bounding rectangles as leaves. We subdivide the overlapping rectangles and eliminate the rectangles whose associated “active CSG tree” is the empty tree. This process is described in details in [Ver]. At the end of the phase, for each rectangle, CANDIDATE_LIST contains exactly the faces of the active CSG tree that could overlap the rectangle and COVER_LIST and VISIBLE are empty.

**Clipping phase**

During this phase, we mix three different processes:
- the clipping of a window and a face on the screen,
- the ray-cast process and
- the computation of the spatial intersections of faces.

We recursively subdivide the windows \( W \) of the partition \( P \) maintaining, at each step, the variables CANDIDATE_LIST, COVER_LIST and VISIBLE associated to each window as follows:

For each window \( W \) of \( P \) having an empty VISIBLE face associated,
- If CANDIDATE_LIST is not empty
F = EXTRACT(CANDIDATE_LIST),
replace W in P by DIFF(W,F) ∪ INTER(W,F),
else
   if COVER_LIST is not empty
      For each W’ in SPATIAL(W,COVER_LIST,COVER_LIST)
      VISIBLE = RAY-CAST(W’,COVER_LIST,VISIBLE),
      if VISIBLE is empty, delete W’ from P
   else
      delete W from P.

Thus at the end of this phase, P contains windows of the screen having a non empty VISIBLE face associated.

Merging phase
In this phase we try successively to merge the windows of P when they have the same VISIBLE face associated:

   For any window W in P,
      NOCHANGE = TRUE,
      For any window W’ in P having the same VISIBLE than W,
      if MERGE(W,W’) is equal to a unique window
         NOCHANGE = FALSE,
         replace W and W’ by MERGE(W,W’) in P
      if NOCHANGE is TRUE, W cannot be merged,
         insert W in the final partition.

The four functions DIFF(), INTER(), MERGE() and SPATIAL() use a two dimensional polygon clipper, introduced by [AWc] and described in details in [Ver]. This algorithm allows to compute intersections, union and difference of two polygons that may have “holes” in their contour. Thus, it allows as primitive of the CSG solid any type of polyhedral object.

Computation reduction
In this section we explicit and justify the two notions of “certainly” invisible face and “consequent” spatial intersections for a window. We present then a modified version of the visualization algorithm to include these reductions.

Certainly invisible face
The objects of the CSG tree are classified in three classes:

- union objects.

Objects of the CSG tree which are involved only in union operations or, which is equivalent, the objects whose ancestor in the CSG tree are only union operations.
• *left objects.*

Objects of the CSG tree which appear at the left of their first difference or intersection ancestor.

• *right objects.*

Objects which are neither union nor left objects.

By extension, the faces will be differentiated in union, left or right faces.

Example 1. The CSG tree of Figure 1 has one left object A, three right objects B, C and D and two union objects E and F.

![CSG tree composed of 6 primitives](image)

The philosophy here is to eliminate a face from a CANDIDATE_LIST of a window using the least amount of computation time and without omitting possible spatial intersections. Thus only the maximal and the minimal z-depths of the faces are considered.

Given a window W and a face F of its CANDIDATE_LIST, the elimination of F from the CANDIDATE_LIST differs, according to the type of the face.

- If F is an union face, it will be eliminated when F is entirely behind an union face of the COVER_LIST. In this case, F may belong to the boundary of the CSG solid, but it is not visible in this window.

- If F is a right face, F is involved in at least one intersection or difference operation. F is eliminated
  - either when F is hidden by a union face,
  - or when F cannot take part of the boundary of the CSG solid, at least in that zone of the screen. It is the case when the face is entirely behind all the left faces belonging to COVER_LIST. This condition is due to the existing ordering of CANDIDATE_LIST.
Nevertheless they have a common characteristic: they are behind the first visible face inside the window W.

More precisely, given a window W, the first face F of its CANDIDATE_LIST is said certain invisibility if and only if:

- if F is an union face,
  \[ z_{\text{min}}(F) \geq \min \{ z_{\text{max}}(F') \mid F' \text{ union face belonging to COVER_LIST} \} \]
- if F is a right face,
  \[ z_{\text{min}}(F) > \min (\max \{ z_{\text{max}}(F') \mid F' \text{ left face belonging to COVER_LIST} \}, \min \{ z_{\text{max}}(F') \mid F' \text{ union face belonging to COVER_LIST} \}) \]

Because of the ordering of CANDIDATE_LIST, when F is certainly invisible, all the faces of CANDIDATE_LIST belonging to the same object are also invisible.

A new function is added:

\[ \text{ELIM}(\text{window, candidate_list}) : \text{successively eliminates the first faces of candidate_list which are certainly invisible. The function stops when candidate_list is empty or when the first face is not certainly invisible.} \]

Example 2:
Let us consider the scene of Figure 2. For the clarity of the example, we suppose that the minimal and the maximal z-depths of the faces cutting the plane \( y = y_0 \) appear in

![Diagram of the scene](image)

Figure 2: The scene is composed of two objects A and B.
the plane. The faces of A and B are ordered in \(a_1, \ldots, a_4\) and in \(b_1, \ldots, b_4\) with respect to the order induced by their minimal and maximal z-depths (the faces without thickness in the screen have been eliminated at the entrance).

As the tree is traversed in postorder, we have:

If the CSG tree is

\[
\left(\bigcup_{a_2, a_3, a_4 \text{ and all the faces of B are certainly invisible}} \bigg/ \bigg\{A, B\}\bigg)\right.
\]

in the window corresponding to \(a_1\).

\[
\left(\bigcap_{b_2, b_3 \text{ and } b_4 \text{ are certainly invisible in all the}} \bigg/ \bigg\{A, B\}\bigg)\right.
\]

or \(\bigg/ \bigg\{A, B\}\bigg)\) to the segments \(x_1x_2, x_2x_3\) and \(x_3x_4\).

\[
\left(\bigcup_{b_2, b_3, b_4 \text{ and } a_2, a_3, a_4 \text{ are certainly invisible}} \bigg/ \bigg\{B, A\}\bigg)\right.
\]

in the window corresponding to \(a_1\).

\[
\left(\bigcap_{\text{no face is certainly invisible.}} \bigg/ \bigg\{B, A\}\bigg)\right.
\]

or \(\bigg/ \bigg\{B, A\}\bigg)\)

One can see in this example, that the order of the primitives in the tree has an effect on the number of certainly invisible faces for a window:

the tree has to be constructed from the front to the bottom of the scene.

Thus, to take advantage of the notion of “certainly” invisible face, the initial CSG tree \(T\) is transformed into an equivalent one \(T'\) as follows:

all the union objects are extracted from \(T\). They are inserted in a CSG tree \(T_i\) in such a way that \(T_i\) is a binary search tree w. r. t. the \((z_{\text{min}}, z_{\text{max}})\) lexicographic order.

the CSG tree \(T - \{\text{union objects}\}\) is reorganized by:

- Eliminating the empty leaves: if \(\emptyset\) represents the empty tree,

\[
\left(\bigcup_{A, \emptyset, \emptyset, A} = A \text{ we built the “active CSG tree” corresponding to } T - \{\text{union objects}\}\right)
\]
Using recursively the following rules, while making a postorder traversal of $T - \{\text{union objects}\}$:

If $A$ and $B$ are CSG trees such that $(z_{\text{min}}(A), z_{\text{max}}(A)) > (z_{\text{min}}(B), z_{\text{max}}(B))$, then:

- $\cap$ is replaced by $\cap$
  \[
  \begin{array}{c}
  \_ \\
  / \ \\
  A \ B \\
  \_ \ \\
  \end{array}
  \begin{array}{c}
  \_ \\
  / \ \\
  B \ A \\
  \_ \ \\
  \end{array}
  
- $\cup$ is replaced by $\cup$
  \[
  \begin{array}{c}
  \_ \\
  / \ \\
  A \ B \\
  \_ \ \\
  \end{array}
  \begin{array}{c}
  \_ \\
  / \ \\
  B \ A \\
  \_ \ \\
  \end{array}
  
when $z_{\text{min}}(B) > z_{\text{max}}(A)$, $A$ and $B$ represent disjoint CSG solids. Then:

- $- = A$ and $\cap = \emptyset$
  \[
  \begin{array}{c}
  \_ \\
  / \ \\
  A \ B \\
  \_ \ \\
  \end{array}
  \begin{array}{c}
  \_ \\
  / \ \\
  A \ B \\
  \_ \ \\
  \end{array}
  
and the minimum and the maximum z-depths of the subtrees are deduced from their subtrees using the rules:

- $z_{\text{min}}(A \cap B) = \sup (z_{\text{min}}(A), z_{\text{min}}(B))$,
- $z_{\text{max}}(A \cap B) = \inf (z_{\text{max}}(A), z_{\text{max}}(B))$,
- $z_{\text{min}}(A \cup B) = \inf (z_{\text{min}}(A), z_{\text{min}}(B))$,
- $z_{\text{max}}(A \cup B) = \sup (z_{\text{max}}(A), z_{\text{max}}(B))$

and $z_{\text{min}}(A - B) = z_{\text{min}}(A)$, $z_{\text{max}}(A - B) = z_{\text{max}}(A)$.

Let $T_r$ be the resulting CSG tree.

the CSG tree $T'$ is equal to

- $T' = \cap$ if $T_r$ and $T_l$ are both non empty
  \[
  T_l \quad \cap \quad T_r
  
- else $T' = T_r$ if $T_l$ is the empty tree
  \[
  T' = T_l
  
As the presence of union objects can have an effect in the elimination of right faces, $T_l$ has to be examined before $T_r$, thus it is the left son of $T$.

Let us call this process $T' = \text{REORDER}(T)$. The tree $T'$ is built
from the front to the bottom of the scene for the union objects,
and from the front to the bottom of the scene when it is possible for the others.
REORDER($T$) is called before entering the preparation step.

**Consequent spatial intersections**

In the clipping phase, the spatial intersections occurring in a window are computed only when all the faces of CANDIDATE\_LIST have been examined. At this point of the visualization algorithm, COVER\_LIST contains a sufficient set of faces to compute the ray-cast inside all the window. To justify the name of consequent spatial intersection let us explain our intuitive reasoning:

the result of the ray-cast may differ in two points $P$ and $P'$ of a window if some faces of COVER\_LIST intersect in front of the face visible in $P$ or in $P'$. An intersection of this type changes the z-depth order and thus may modify the result of the ray-cast.

Thus, given a window $W$ and a point $P$ inside $W$, the *consequent* spatial intersections are the intersections of faces belonging to COVER\_LIST and faces appearing in front of the visible face on $P$.

The function RAYCAST is modified as follows:

```plaintext
RAY-CAST(window, cover\_list, consequent\_list, visible) :
put in VISIBLE the result of the ray-cast on $P$ and put in
CONSEQUENT\_LIST the faces whose z-depth on $P$ is smaller or equal than the
z-depth of VISIBLE. If the result of the ray-cast is empty, CONSEQUENT\_LIST
is exactly COVER\_LIST.
```

Some unnecessary rays may be fired, but, if we take into account “Atherton’s face coherence” [Ath] (the occurrence of surface intersecting is relatively rare), in most of the cases the SPATIAL does not subdivide the window.

Example 3:
In Figure 3 the faces have the same property than in Example 2: their minimal and maximal z-depths appear in the projection plane.

On $W$ : COVER\_LIST=$\{a_1,a_2,b_1,b_2,c_1,c_2\}$
On $P$, RAY-CAST returns VISIBLE $=a_1$ and z-depth sort gives $a_1,\ldots$

SPATIAL computes intersections of the faces $a_1,b_1$ and $a_1,c_1$. $W$ is split in three windows $W_1$, $W_2$ and $W_3$.

In $W_1$ and $W_2$, the ray-cast is constant and equal to $a_1$.
In $Q$ inside $W_3$, the ray-cast gives VISIBLE $=b_1$ and the z-depth sort gives $c_1,b_1,\ldots$

SPATIAL computes intersections of the faces $b_1$ and $c_1$.
In this example the spatial intersections computed are all of importance for the computation of the first visible face.
Conclusion

Consider Table 1 in conjunction with Figures 4-10. It can be remarked that the computing time depends mainly on the size of the first subdivision. Thus all the causes of increase or reduction of the subdivision affect the computing time in the same way; particularly:

- the concentration and the imbrication of the objects more than the number of faces (compare the results of figure 5 and figure 8)
- the point of view chosen (c.f. the difference between figure 8 and figure 10)
- the classes of the objects present in the CSG tree

This algorithm is an improvement of the ray-casting algorithm. It seems a little slower but still comparable with Atherton’s scan line visualization [Ath]. The modified version of Atherton’s algorithm introduced very recently by Bronsvoort [Bro] seems nevertheless very efficient and faster than our algorithm.

Anyhow our images are different than Atherton’s or Bronsvoort’s ones, as we obtain a description of the screen in polygons and not in lines.

The use of octree structure jointly with the construction of the screen subdivision may induce a better elimination of useless subdivisions than our reasoning on the minimum and maximum z-depths of the faces. This is an orientation to improve the algorithm.
<table>
<thead>
<tr>
<th>references of figures</th>
<th>number of primitives in the tree</th>
<th>number of facets</th>
<th>number of union objects</th>
<th>number of left objects</th>
<th>number of right objects</th>
<th>size of the first subdivision</th>
<th>time taken (*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>13</td>
<td>162</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>665</td>
<td>282</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>548</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>647</td>
<td>239</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>187</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>524</td>
<td>153</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>63</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>555</td>
<td>192</td>
</tr>
<tr>
<td>9</td>
<td>49</td>
<td>294</td>
<td>49</td>
<td>0</td>
<td>0</td>
<td>855</td>
<td>305</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>63</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>259</td>
<td>120</td>
</tr>
</tbody>
</table>

(*) The measurements correspond to the CPU time in seconds of all the visualization process on a VAX/785 without floating point accelerator.

Table 1: Description of the figures and timing results

Acknowledgments

This work has been done in the first phase at the “Institut National de l’Audiovisuel” in the team of D. Borenstein, J.C. Hourcade and A. Nicolas. It has been continued at the “Institut National de Recherche en Informatique et Automatique” and terminated in the Computer Science Laboratory of the “Ecole Normale Supérieure”. It is written in C language on a VAX 785 working on Unix system.

References


Figure 4:
the CSG tree is composed of 13 primitives. The subdivision of the screen contains 665 windows after the clipping phase. Time taken: 282 s. Most operations involved in the CSG tree are difference or intersection operations. Thus the facets of these objects are less eliminated than the union facets. That explains the size of the subdivision.
Figure 5:
The CSG tree is composed of 17 polyhedral primitives. It involves only union operations. Thus we obtain only 647 windows in the subdivision of the screen after the clipping phase. The visualisation process takes 239 s.
Figure 6:
The subdivision of the screen after the clipping phase. It contains 524 windows. One can see that some facets of front objects have been eliminated: they correspond to some ‘‘surely invisible facets’’. 
Figure 7:
Final image corresponding to the subdivision of Figure 6. The CSG tree is composed of 14 polyhedral primitives. Time taken: 153 s.
Figure 8:
The CSG tree is the union of 4 polyhedral primitives. Time taken: 192 s.
Figure 9:
Each chair is the union of 6 primitives and the table is the union of 7 primitives. The CSG tree is composed of 49 polyhedral primitives.
Time taken: 305 s.
Figure 10:
The same CSG solid than in Figure 8 but the point of view is different. Time taken: 120 s.