Visualization algorithm for CSG polyhedral solids

A Verroust

An algorithm is presented here to visualize CSG solids in wireframe 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 the 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.

solid modelling, constructive solid geometry, hidden surface algorithm, ray-casting

Constructive solid geometry (CSG) is one of the classical representations of three-dimensional solids. It is particularly well adapted to ray-tracing algorithms[1] 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[2,3]
- compute directly the image of the CSG tree without computing the boundaries of the CSG solid

The second type of method is of particular interest here. Most of the approaches using this method use as a basis, the ray-casting method, first introduced by Roth[4]. 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.

Bronsvoort, Jansen and Van Wijk[5] also proposed 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 too much computation time is needed to make the visualization interactive.

Atherton[6] used 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 have improved the computation of the first visible surface in the ray-tracing algorithm[7,8]. Nevertheless they are concerned with ray-tracing more than with ray-casting and their improvements are interesting when a great number of rays are fired.

Liens, Ecole Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05, and INRIA, 78150 Le Chesnay, France

More recently, several authors have paid much attention to depth coherence in their algorithms. Crocker[9] introduced 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 adapted his method to the use of CSG by improving Atherton's algorithm. His measurements showed the advantages of the method.

Okino, Kakazu and Morimito[10] and Requicha and Rossignac[11] both used an extended depth buffer in their visualization algorithms.

The approach described here is largely inspired by Atherton's solution. Atherton's visualization process is adapted to scan-line algorithms, thus he uses span and scan-line coherence. A CSG solid composed of polyhedral objects is to be displayed in wireframe. 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 described in 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 the screen is subdivided into windows where the z-depth order of the faces in each window is ensured to be constant within it, at least until the first visible face, it is sufficient 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[12], the author has used this idea and subdivided the screen into windows such that:

- if a face of an object overlaps a window, this face entirely contains the window
- if two faces intersect in 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'[5] have been 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 2D clipping process, the faces that are 'certainly' invisible are eliminated
- only spatial intersections that might change the result of the ray-cast are computed, these are called consequent intersections

These eliminations are explained and a new visualization algorithm is described in the third section of the paper. Some measurements of the new algorithm are also given in the final section.

VISUALIZATION PROCESS DESCRIPTION

The CSG solid is a CSG tree of polyhedral primitives. There are no conditions set for the type of polygons allowed, it is assumed that different faces of the same object do not intersect in space. During the whole process, we work with the projections of the objects on the screen except when the spatial intersections of objects are computed and when a ray is fired. Thus the primitives are identified with collections of faces on 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 into windows where the result of the ray-cast is sure to be constant for all the points inside each window
- merging phase
  - a final subdivision of the screen where windows associated to the same visible face are joined

Before describing these three phases, some functions used during the process are introduced.

Functions

RAY-CAST(window,cover_list,visible) : fires a ray from a point P inside the window, evaluates the Boolean expression of the z-depths of the faces of COVER_LIST on the corresponding active CSG tree and puts the result obtained into VISIBLE.

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 that result from the partitioning 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 between the window and the face. These windows have the same COVER_LIST as 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:

- order of the objects in a postorder traversal of the CSG tree
- decreasing order of the minimum z-depth of the faces, z_{min}(face)
- decreasing order of the maximum z-depth of the faces, z_{max}(face)

Preparation phase

Initially, all the primitives undergo perspective transformation. 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 against their minimal and maximal z-depths. A bounding rectangle is associated with each primitive. A rough partition, P, is made of the screen using the CSG tree with the bounding rectangles as leaves. The overlapping rectangles are subdivided and those rectangles whose associated active CSG tree is the empty tree are eliminated. This process is described in detail by Verroust[12]. At the end of this phase, for each rectangle, CANDIDATE_LIST contains exactly those faces of the active CSG tree that could overlap the rectangle, and COVER_LIST and VISIBLE are empty.

Clipping phase

During this phase, three different processes are mixed:

- clipping of a window and a face on the screen
- ray-cast process
- computation of the spatial intersections of faces

The windows, W, of the partition, P, are subdivided recursively maintaining, at each step, the variables CANDIDATE_LIST, COVER_LIST and VISIBLE associated with 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 those windows of the screen that have a nonempty VISIBLE face associated with them.

Merging phase

In this phase, the merging of the windows of P when they have the same VISIBLE face associated is tried successively:

```
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 2D polygon clipper, introduced by

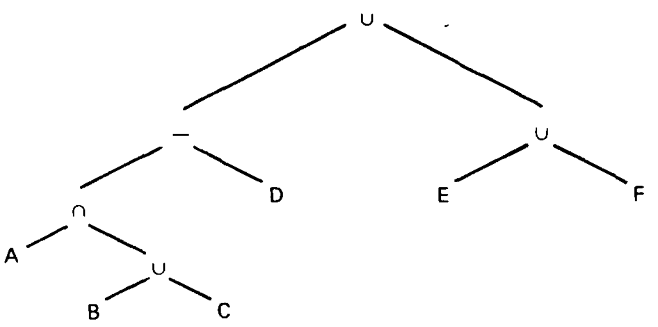


Figure 1. CSG tree composed of six primitives

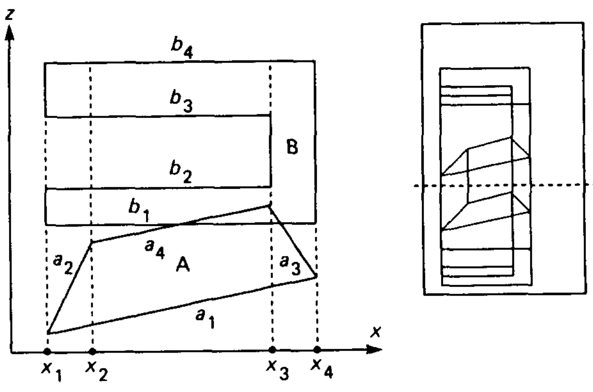


Figure 2. Scene composed of two objects, A and B

Atherton and Weiler[13] and described in detail in Verroust[12]. This algorithm allows the computation of intersections, union and difference of two polygons that may have 'holes' in their contour. Thus, it allows any type of polyhedral object as a primitive of the CSG solid.

COMPUTATION REDUCTION

In this section, the two notions of 'certainly' invisible faces and 'consequent' spatial intersections for a window are explained and justified. A modified version of the visualization algorithm that includes these reductions is presented.

'Certainly' invisible faces

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 ancestors 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 as 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.

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 a union face, it will be eliminated when F is entirely behind the 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 of difference operation. F is eliminated:
  - either when F is hidden by a union face
  - or when F cannot take part in 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 results from 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 to be 'certainly' invisible if and only if:

- F is a union face
  $z_{\min}(F) \geq \min \{z_{\max}(F') \mid F' \text{ union face belonging to COVER\_LIST}\}$
- F is a right face
  $z_{\min}(F) > \min (\max \{z_{\max}(F') \mid F' \text{ left face belonging to COVER\_LIST}\}, \min \{z_{\max}(F') \mid F' \text{ union face belonging to COVER\_LIST}\})$

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:

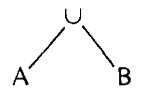
ELIM(window,candidate_list) : successively eliminates the first faces of candidate_list which are certainly invisible. The function stops when candidate_list is empty or when the first face is not certainly invisible.

Example 2

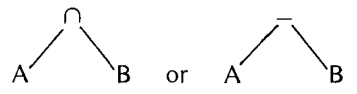
Consider the scene of Figure 2. For the sake of clarity in the example, suppose that the minimal and the maximal z-depths of the faces cutting the plane $y = y_0$ appear in the plane. The faces of A and B are ordered in $a_1, \ldots, a_4$ and in $b_1, \ldots, b_4$ with respect to the order induced by their minimal and maximal z-depths (the faces without thickness in the screen have been eliminated at the beginning).

As the tree is traversed in postorder, the following results are obtained.

If the CSG tree is:



a_2, a_3, a_4 and all the faces of B are certainly invisible in the window corresponding to a_1.



b_2, b_3 and b_4 are certainly invisible in all the windows covering a_1 (i.e. corresponding to the segments x_1x_2, x_2x_3 and x_3x_4).

$$\begin{tikzcd}[column sep=small] & \cup \arrow[dl, dash] \arrow[dr, dash] & \\ B & & A \end{tikzcd}$$

b_2, b_3, b_4 and a_2, a_3, a_4 are certainly invisible in the window corresponding to a_1.

$$\begin{tikzcd}[column sep=small] & \cap \arrow[dl, dash] \arrow[dr, dash] & \\ B & & A \end{tikzcd} \quad \text{or} \quad \begin{tikzcd}[column sep=small] & - \arrow[dl, dash] \arrow[dr, dash] & \\ B & & A \end{tikzcd}$$

no face is certainly invisible.

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 top to the bottom of the scene.

Thus, to take advantage of the notion of the '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_l in such a way that T_l is a binary search tree w. r. t. the $(z_{\min}, z_{\max})$ lexicographic order. The CSG tree T – {union objects} is reorganized by two processes.

First, eliminating the empty leaves : if 0 represents the empty tree,

$$\begin{tikzcd}[column sep=small] & \cup \arrow[dl, dash] \arrow[dr, dash] & \\ A & & 0 \end{tikzcd} = \begin{tikzcd}[column sep=small] & \cup \arrow[dl, dash] \arrow[dr, dash] & \\ 0 & & A \end{tikzcd} = A$$

the 'active CSG tree' is built corresponding to T – {union objects}.

Second, using the following rules recursively, while making a postorder traversal of T – {union objects}. If A and B are CSG trees such that $(z_{\min}(A), z_{\max}(A)) > (z_{\min}(B), z_{\max}(B))$, then

$$\begin{tikzcd}[column sep=small] & \cap \arrow[dl, dash] \arrow[dr, dash] & \\ A & & B \end{tikzcd} \quad \text{is replaced by} \quad \begin{tikzcd}[column sep=small] & \cap \arrow[dl, dash] \arrow[dr, dash] & \\ B & & A \end{tikzcd}$$

$$\begin{tikzcd}[column sep=small] & \cup \arrow[dl, dash] \arrow[dr, dash] & \\ A & & B \end{tikzcd} \quad \text{is replaced by} \quad \begin{tikzcd}[column sep=small] & \cup \arrow[dl, dash] \arrow[dr, dash] & \\ B & & A \end{tikzcd}$$

when $z_{\min}(B) > z_{\max}(A)$, A and B represent disjoint CSG solids. Then

$$\begin{tikzcd}[column sep=small] & - \arrow[dl, dash] \arrow[dr, dash] & \\ A & & B \end{tikzcd} = A \quad \text{and} \quad \begin{tikzcd}[column sep=small] & \cap \arrow[dl, dash] \arrow[dr, dash] & \\ A & & B \end{tikzcd} = 0$$

and the minimum and the maximum z-depths of the subtrees are deduced from their subtrees using these rules:

$$\begin{aligned} &z_{\min}(A \cap B) = \sup\,(z_{\min}(A), z_{\min}(B)),\\ &z_{\max}(A \cap B) = \inf\,(z_{\max}(A), z_{\max}(B)),\\ &z_{\min}(A \cup B) = \inf\,(z_{\min}(A), z_{\min}(B)),\\ &z_{\max}(A \cup B) = \sup\,(z_{\max}(A), z_{\max}(B)),\\ &z_{\min}(A - B) = z_{\min}(A), z_{\max}(A - B) = z_{\max}(A) \end{aligned}$$

Let T_r be the resulting CSG tree. The CSG tree T′ is equal to

$$T' = \begin{tikzcd}[column sep=small] & \cap \arrow[dl, dash] \arrow[dr, dash] & \\ T_l & & T_r \end{tikzcd} \qquad \text{if } T_r \text{ and } T_l \text{ are both non empty}$$

else

$$\begin{aligned} T' &= T_r && \text{if } T_l \text{ is the empty tree}\\ T' &= T_l && \text{in the other cases} \end{aligned}$$

As the presence of union objects can have an effect on the elimination of right faces, T_l has to be examined before T_r, thus it is the left son of T.

Call this process T′ = REORDER(T). The tree T′ is built in two ways: from the top to the bottom of the scene for the union objects; and from the top to the bottom of the scene when it is possible for the others. REORDER(T) is called before entering the preparation phase.

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 the whole window. To justify the name of consequent spatial intersection, the author's intuitive reasoning is explained. 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

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 than or equal to 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 'Atherton's face coherence' (surface intersection occurrence is relatively

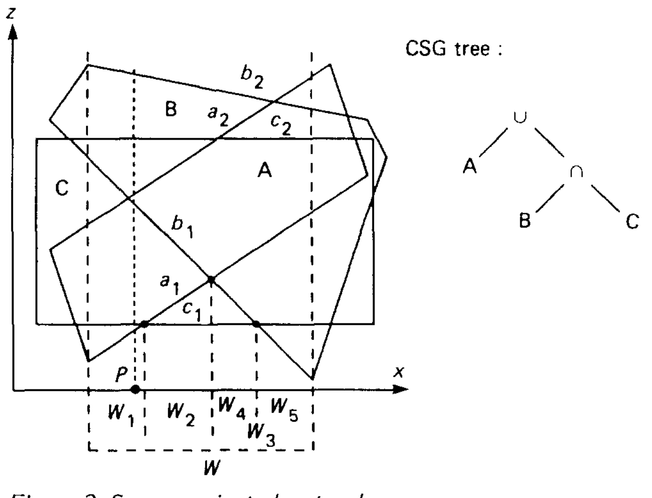


Figure 3. Scene projected onto plane y = y₀

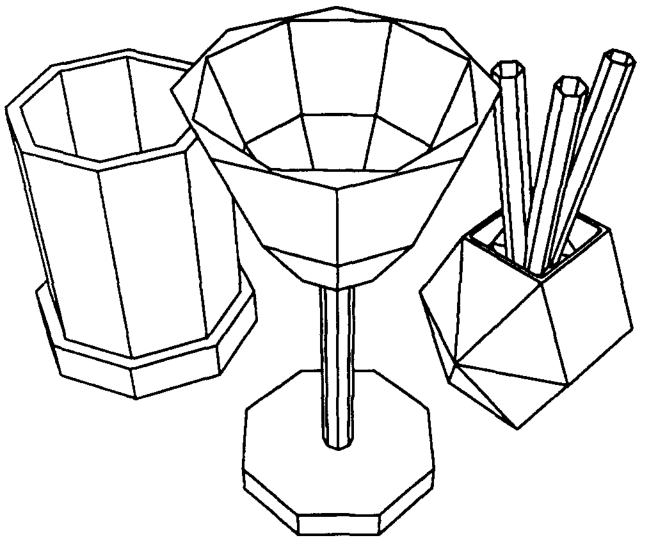


Figure 4. CSG tree is composed of 13 primitives. The subdivision of the screen contains 665 windows after the clipping phase. Time taken: 282 s. Most operations involved in the CSG tree are difference or intersection operations. Thus the facets of these objects are eliminated less than the union facets. That explains the size of the subdivision

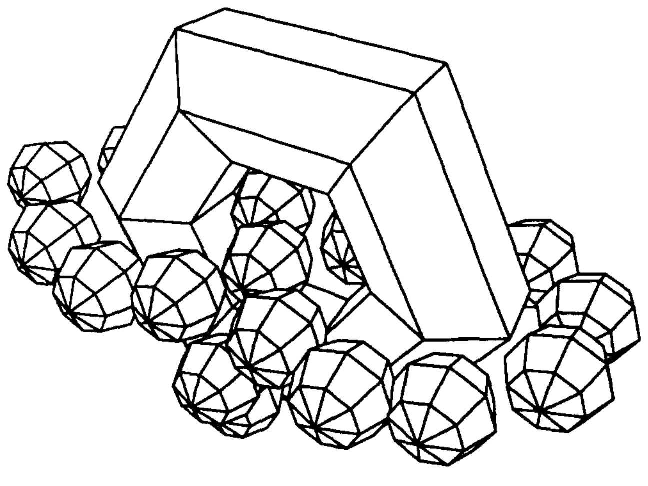


Figure 5. CSG tree is composed of 17 polyhedral primitives. It involves only union operations. Thus only 647 windows are obtained in the subdivision of the screen after the clipping phase. The visualization process takes 239 s

rare) is taken into account; in most cases SPATIAL does not subdivide the window.

Example 3

Consider the scene of Figure 3. The faces have the same property as 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 into 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 important to the computation of the first visible face.

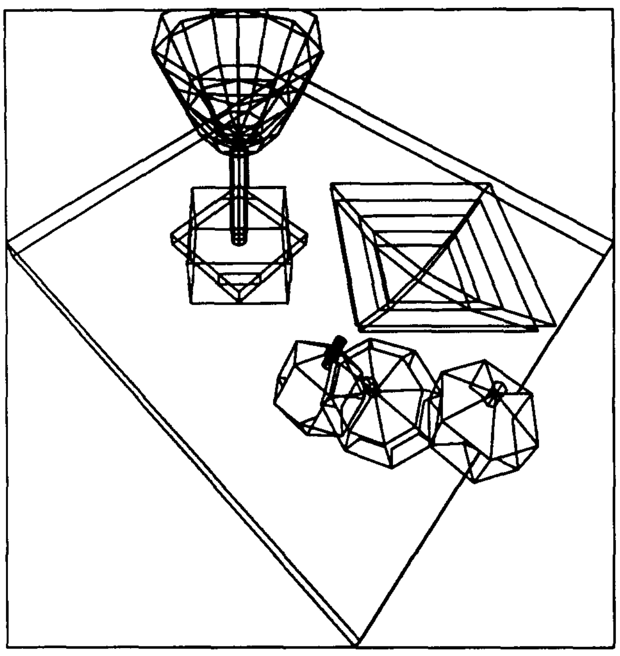


Figure 6. Subdivision of screen after clipping phase. It contains 524 windows. One can see that some facets of front objects have been eliminated : they correspond to some 'certainly invisible facets'

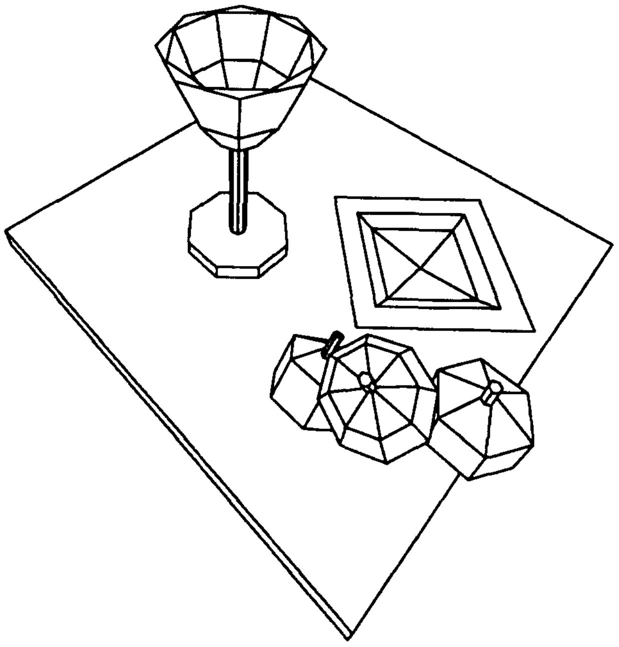


Figure 7. Final image corresponding to the subdivision of Figure 6. The CSG tree is composed of 14 polyhedral primitives. Time taken: 153 s

CONCLUSIONS

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 overlapping 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

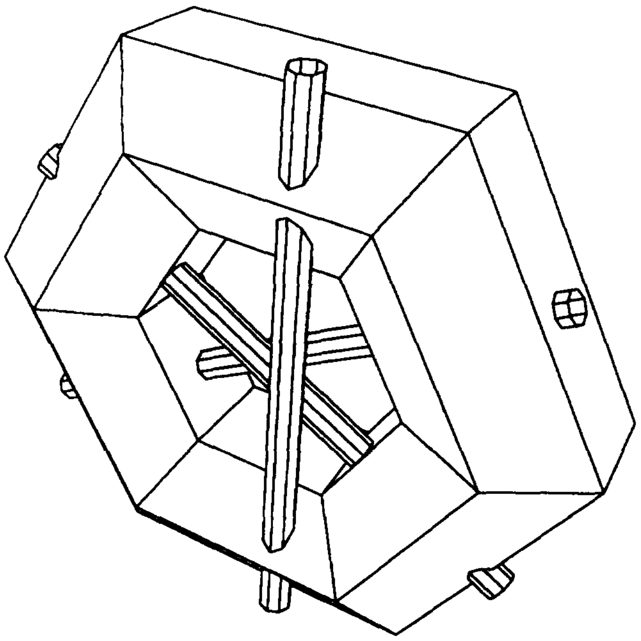


Figure 8. CSG tree is the union of 4 polyhedral primitives. Time taken: 192 s

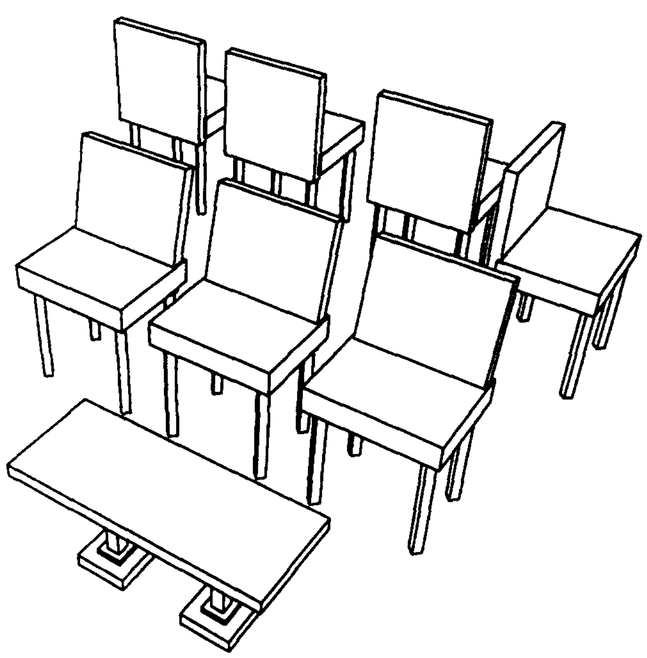


Figure 9. Each chair is the union of 6 primitives and the table is the union of 7 primitives. The CSG tree is composed of 49 polyhedral primitives. Time taken: 305 s

It can also be noted that this algorithm is an improvement of the ray-casting algorithm. It seems a little slower but, still comparable with Atherton's scan line visualization[6]. Nevertheless, the modified version of Atherton's algorithm introduced very recently by Bronsvoort[14] seems very efficient and faster than the algorithm presented here. However, the image is different from Atherton's or Bronsvoort's ones. In this case, the screen is described in polygons and not in lines.

The use of the octree structure along with the construction of the screen subdivision may eliminate useless subdivisions in a better way than the reasoning on the minimum and maximum z-depths of the faces. This is a possible way to improve the algorithm.

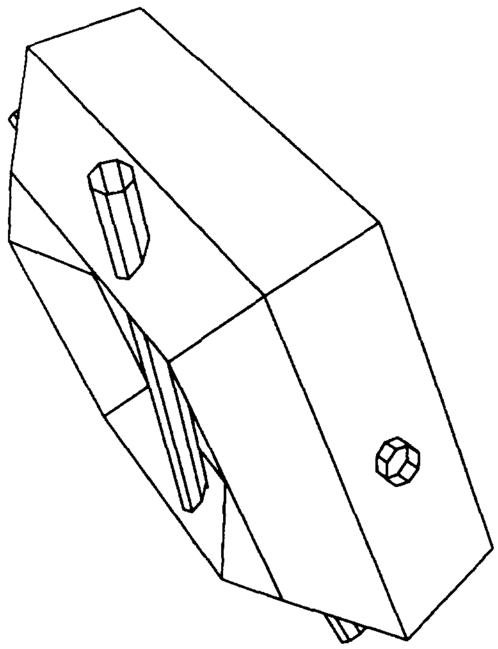


Figure 10. Same CSG solid as in Figure 8 but the viewpoint is different. Time taken: 120 s

Table 1. Description of the figures and timing results

References to Figures	Number of primitives in the tree	Number of facets	Number of union objects	Number of left objects	Number of right objects	Size of first subdivision	Time taken*
4	13	162	5	4	4	665	282
5	17	548	17	0	0	647	239
6	14	187	8	6	6	524	153
8	4	63	4	0	0	555	192
9	49	294	49	0	0	855	305
10	4	63	4	0	0	259	120

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

ACKNOWLEDGEMENTS

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 completed in the Computer Science Laboratory of the Ecole Normale Supérieure. It is written in C on a VAX/785 running under Unix.

REFERENCES

1 **Kajiya, J T** 'Tutorial on ray tracing' *Course Notes of Siggraph '84* (1984)

2 **Hughes, J F, Laidlaw, D H and Trumbore, W B** 'Constructive solid geometry for polyhedral objects' *Comput. Graph.* Vol 20 No 4 (August 1986) pp 161–169

3 **Requicha, A A G and Voelcker, H B** 'Boolean operations in solid modeling: boundary evaluation and merging algorithms' *IEEE Comput. Graph. & Appl.* Vol 5 No 1 (January 1985) pp 30–34

4 **Roth, S D** 'Ray casting for modelling solids' *Comput. Graph. Image Process.* No 18 (February 1982) pp 109–144

5 **Bronsvoort, W F, Jansen, F W and Van Wijk, J J** 'Two methods for improving the efficiency of ray casting in solid modelling' *Comput.-Aided Des.* Vol 16 No 1 (January 1984) pp 51–55

6 **Atherton, P R** 'A scan line hidden surface removal procedure for constructive solid geometry' *Comput. Graph.* Vol 17 No 3 (July 1983) pp 73–82

7 **Greenberg, D P, Hooper, G and Weghorst, H** 'Improved computational methods for ray tracing' *ACM Trans. Graph.* Vol 3 No 1 (January 1984) pp 52–69

8 **Jansen, F W** 'A CSG list priority hidden surface algorithm' *Eurographics '85 Proc.* (1985) pp 51–62

9 **Crocker, G A** 'Invisibility coherence for faster scan-line hidden surface algorithms' *Comput. Graph.* Vol 18 No 3 (July 1984) pp 95–102

10 **Okino, N, Nakazu, Y and Morimoto, M** 'Extended depth-buffer algorithms for hidden-surface visualization' *IEEE Comput. Graph. & Appl.* Vol 4 No 5 (May 1984) pp 79–88

11 **Requicha, A G and Rossignac, J** 'Depth-buffering display techniques for constructive solid geometry' *IEEE Comput. Graph. & Appl.* Vol 6 No 8 (September 1986) pp 29–39

12 **Verroust, A** 'A CSG visualization algorithm using polygon clipping' *Rapport INRIA No. 461,* INRIA, Domaine de Voluceau, 78150 Le Chesnay, Cedex, France (December 1985)

13 **Atherton, P R and Weiler, K** 'Hidden surface removal using polygon area sorting' *Comput. Graph.* Vol 11 No 2 (Summer 1977) pp 214–222

14 **Bronsvoort, W F** 'Techniques for reducing Boolean evaluation time in CSG scan-line algorithms' *Comput.-Aided Des.* Vol 18 No 10 (December 1986) pp 533–538