Interactive Inspection of Solids: Cross-sections and Interferences

Jarek Rossignac, Abe Megahed, Bengt-Olaf Schneider
Interactive Geometric Modeling, IBM T.J. Watson Research Center

Abstract

To reduce the cost of correcting design errors, assemblies of mechanical parts are modeled using CAD systems and verified electronically before the designs are sent to manufacturing. Shaded images are insufficient for examining the internal structures of assemblies and for detecting interferences. Thus, designers must rely on expensive numerical techniques that compute geometric representations of cross-sections and of intersections of solids. The solid-clipping approach presented here bypasses these geometric calculations and offers realtime rendering of cross-sections and interferences for solids represented by their facetted boundaries. In its simplest form, the technique is supported by contemporary high-end graphics workstations. Its variations, independently developed elsewhere, have already been demonstrated. Our implementation is based on the concept of a cut-volume interactively manipulated to remove obstructing portions of the assembly and reveal its internal structure. For clarity, faces of the cut-volume which intersect a single solid are hatched and shaded with the color of that solid. Interference areas between two or more solids are highlighted. Furthermore, to help users find the first occurrence of an interference along a search direction, we have developed an adaptive subdivision search based on a projective approach which guarantees a sufficient condition for object disjointness. The additional performance cost for solid-clipping and interference highlighting is comparable to the standard rendering cost. An efficient implementation of the disjointness test requires a minor extension of the graphics functions currently supported on commercial hardware.

CR Categories and Subject Descriptions:
I.3.3 [Computer Graphics]: Picture/Image Generation – *Display Algorithms;* **I.3.5** [Computational Geometry and Object Modeling]: Solid Representation; **I.3.7** [Three-Dimensional Graphics and Realism]: Visible Surface Algorithms; **J.6** [Computer Aided Engineering]: Computer Aided Design.

Keywords: Cross-section, Clipping, Interferences.

Authors' address:
IBM Research, P.O. Box 704, Yorktown Heights, NY 10598.
Rossignac: jarek@watson.ibm.com, 914-784-7630.
Schneider: bosch@watson.ibm.com, 914-784-6002.

1. Introduction

Design errors discovered at the manufacturing or assembly stages result in expensive engineering changes and production delays. Manufacturers of mechanical goods have invested in advanced graphics hardware and in the solid modeling technology hoping that they will eliminate the need for clay models and drastically reduce the number of design errors prior to fabrication. Although designers can interactively visualize subsets of an assembly using shaded or wireframe pictures, they need cross-sections and interference highlights to understand how components fit together in tight assemblies. For example, the shaded image of the small assembly in Figure 1 may be produced in realtime on most high-end graphics workstations, but reveals neither the internal structures of the assembly nor the interferences between its components.

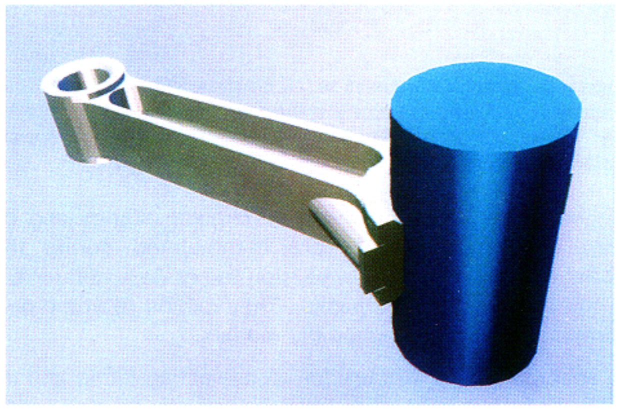

Figure 1. ***A small mechanical assembly****: The interference between the cylinder and the connecting rod is not apparent.*

The availability of an informationally complete solid modeling representation of each assembly component permits the automatic calculation of cross-sections and the calculation of interferences. Unfortunately, classical implementations of these functions rely on expensive geometric operations, which, when applied to models of industrial complexity, increase the system's response time far beyond tolerable limits for interactive sessions.

This paper describes new techniques for automatically: (1) **filling and shading multi-facetted cross-sections through solids, (2) identifying and highlighting areas of interference in a cross-section, and (3) positioning cross-sectioning planes at the beginning of interference or contact regions.**

The solid-clipping techniques presented here exploit existing graphics architectures in novel ways to create, in realtime, shaded images showing cross-sections, cut-outs, and interference regions. Cut-outs are discussed in Section 2. Interferences are addressed in Section 3. The result of a combination of these techniques is illustrated in Figure 2, where a portion of the assembly was cut away using a user-specified cut-volume and where the interference region between the two components was highlighted in red. Furthermore, the cross-section areas are hatched for clarity and the cut-away portions are indicated using transparent faces and silhouette lines.

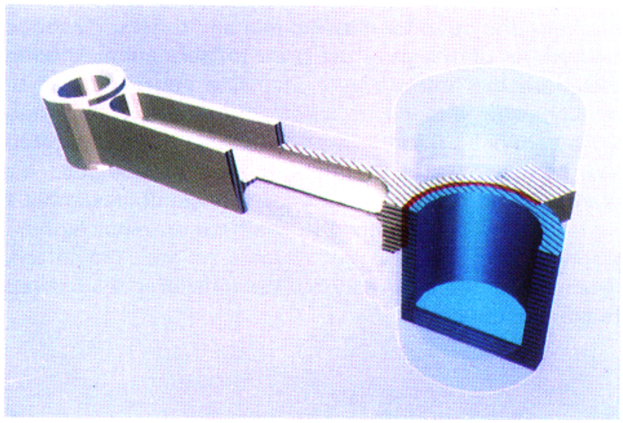

Figure 2. ***Graphics inspection techniques***: *A multi-facet cut-volume is removed to show the internal structure of the assembly. The resulting cross-sections are displayed in the appropriate color and hatched. Red areas indicate interferences.*

These techniques are based on clipping planes and on auxiliary bit-planes that are manipulated during the standard surface scan-conversion to create and later exploit appropriate pixel-masks. They exhibit realtime performance for simple assembly models.

A solution similar to ours for cross-section filling and interference highlights has been independently developed at Silicon Graphics Inc. by Kurt Akeley in 1991 [1]. It is discussed in Section 3.1. Hewlett Packard's graphics library also offers filling and interference highlights, but no description of the underlying techniques is available. Since the manual mentions "the collection of capping edge data" and "cap polygons" [6], we conjecture that an approach different from ours is used.

The automatic detection of interferences is described in Section 3.2. Its requires feedback from the graphics hardware to the application. An efficient implementation of this feedback loop is not supported on commercially available workstations; thus we simulate it by a software inspection of the frame buffer.

2. Solid-clipping

This section describes a new technique for computing in realtime images of solids, or of assemblies of solids, from which user-controlled linear half-spaces or polyhedral cut-volumes have been subtracted. The technique leverages on the recent support of auxiliary clipping planes and pixel-masks in the rendering pipeline of high-end graphics workstations [13].

Clipping planes are commonly used for surface-clipping, i.e. to trim the objects' faces prior to display. The difference between the solid-clipping technique presented in this section and the previously available surface-clipping is illustrated in Figures 3 and 4 using a single-face cut-volume, i.e. a volume bounded by a single clipping plane. Figure 3 shows the effect of the standard surface-clipping, which treats each solid as a hollow shell, because clipping planes do not fill pixels, but merely limit the extent of faces. The image is confusing, since the viewer must mentally reconstruct the areas where the clipping plane intersects the solids. Figure 4 shows the result of the new solid-clipping technique, which, in addition to clipping the solids' faces, also fills the regions of intersection between each solid and the clipping plane. These cross-section regions are hatched to visually differentiate them from other surfaces in the assembly.

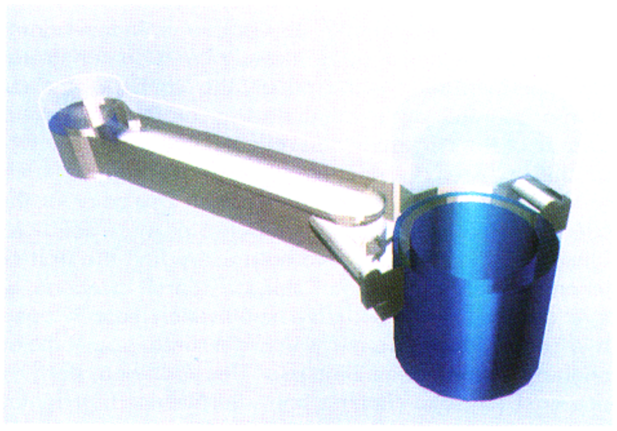

Figure 3. ***Surface-clipping***: *The standard surface-clipping technique correctly removes portions of the solids' faces, but does not fill in the cross-section areas.*

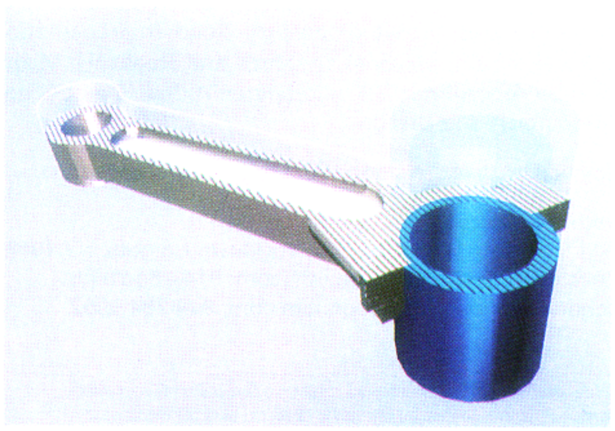

Figure 4. ***Solid-clipping***: *In addition to the surface clipping of Figure 3, the cross-section of each solid by the clipping plane is hatched and shaded using the color of the solid.*

A standard way to produce the image of Figure 4 is to perform the Boolean difference operation between the solids and the cut-volume and to display the result. A slightly better approach combines surface clipping with the display of a cross-sections computed as the geometric intersection of the solid with a plane [7]. A CSG formulation of the result may also be used with special-purpose direct CSG rendering hardware [3-5, 11].

The technique described in this section provides an alternative which neither requires the hardware used for efficiently rendering CSG models nor any complex geometric intersection calculations. It works with any boundary representation for solids, provided that the scan-conversion method used by the graphics hardware satisfies the following parity condition.

When the entire solid fits between the front and the back clipping planes, each pixel is visited an even number of times during the scan-conversion of the solid's faces.

Early scan-conversion techniques did not guarantee the parity condition at pixels traversed by the projection of a common edge between two faces. A reliable implementation of the methods presented here requires a "true point-sampling" scan-conversion [9].

To establish which points of a clipping plane lie inside any given solid, we use the following property [14].

A point Q lies inside a bounded solid S if and only if Q is not on the boundary of S and if a semi-infinite line (ray) starting at Q intersects the boundary of S at an odd number of isolated transversal intersection points[1]. Since the result is independent of the direction for the ray, the viewing direction may be used. Suppose that the location of Q is stored in the z-buffer as Z(q), the depth of the pixel q corresponding to the projection of Q onto the screen.

Q lies in the interior of S if and only if the number of times q is visited during the scan-conversion of the faces of S with a depth greater than Z(q) is odd.

Furthermore:

A point Q, projecting on a pixel q and lying on a clipping plane C, is inside a solid S if and only if q is visited an odd number of times while scan-converting the faces of S after they have been clipped using C.

Note that all points Q of C that correspond to pixels of the screen may be classified during a single pass over S. We use the above property to construct a pixel mask (one bit per pixel), Mp, for the cross-section region where the clipping plane intersects any given solid. The process is illustrated in Figure 5.

When the outward normal of a cut-volume face points away from the viewer, it corresponds to a potential front face of the solid resulting from the cut. Therefore, we use the term **front clipping plane** when referring to the planes containing such a back face of the cut-volume. We need to fill only the cross-sections of front clipping planes, because other (back) clipping planes are never visible.

To make the classification results consistent with the mathematical definition of the regularized difference between the original solid and the cut-volume [8], the clipping of faces coincident with the clipping plane must be performed using a "less than" depth test for clipping planes and a "less or equal" depth-test otherwise[2].

The technique assumes that the solids are not clipped by the back plane of the viewing volume. If the depth span of the object is known, it suffices to temporarily adjust the back plane. However, changes to the z-resolution may produce side-effects. It is also possible to set the perspective such that the back plane coincides with the horizon,[3] to guarantee that no object is clipped by the back clipping-plane.

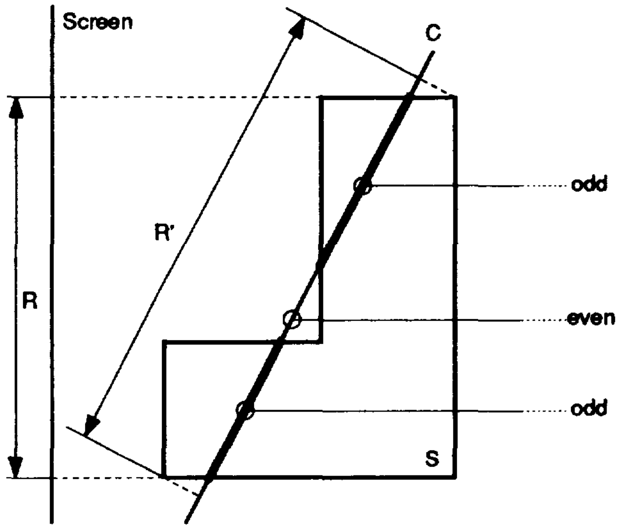

Figure 5. ***Parity-based mask construction:*** *The pixel-mask, Mp, for the intersection of plane C with solid S is computed by toggling the mask during the scan-conversion of the faces of S behind C. R is the projection of S onto the screen and R' the projection of S onto C.*

A high-level algorithm for rendering assemblies clipped by a cut-volume composed of a single half-space is presented below. It renders the cross-section through each solid using the color (surface properties) of that solid[4]. If C is a back clipping plane, the standard surface clipping approach may be used, otherwise, we proceed as follows.

```
Single-plane solid-clipping:
01 Activate C as a front clipping plane
02 For all pixels do Z=0, I=0, Mp=0
03 For each solid S do
04    Render all the faces of S toggling Mp
05    Deactivate C
06    Shade C and reset Mp where Mp==1
```

[1] Tangential intersection cases, where the ray touches a primitive's boundary without crossing it, must be treated properly by the hardware scan-conversion, so as to ensure the correct parity at all pixels [9]. Cases where a one-dimensional subset of the ray lies on a face are ignored by scan-conversion procedures without compromising the parity condition.

[2] Scan-conversion inaccuracies, which produce inconsistencies when displaying overlapping coplanar faces, may be addressed by introducing tolerances in the depth-test [11].

[3] The screen lies exactly between the viewpoint and the horizon-plane of all the vanishing points.

[4] Interference areas may exhibit color mixing unless the interference highlighting technique of the next section is used.

Line 02 resets the z-buffer (Z), the frame buffer (I), and the pixel mask (Mp). During the rendering of the faces of S (Line 04), the portions cut away by C or by the clipping planes of the viewing volume are discarded. The remaining portions are scan-converted and for each surface point s projecting on some pixel q, the following operations are performed: (1) toggle the parity mask Mp(q), (2) if the depth of s is smaller than the depth stored at q, update the z-buffer and the frame buffer at q. Note that both the front and the back faces of S must be scan-converted for the mask computation, although only the front faces need to be rendered.

The cross-section filling of Line 06 is performed using the color and surface properties of S, to distinguish the contribution of each solid to the cross-section. C is deactivated (Line 05) to prevent self-clipping.

The standard depth-test for hidden surface removal is used during the rendering of the faces of S (Line 04) and of the cross-section C (Line 06) to ensure that only visible faces in a scene are rendered. Consequently, **convex cut-volumes** may be produced using several passes through this algorithm for different clipping planes.

To render the cross-section using the standard scan-conversion with hidden-surface removal, a suitable face F_C on C must be constructed. As the clipping plane is manipulated interactively, F_C must be adjusted to always contain the cross-section area. We use a rectangle in C enclosing the orthogonal projection, R′, of S onto C (Figure 5).

Polyhedral cut-volumes with concave edges defined as arbitrary Boolean combinations of half-spaces may be needed to better expose the internal structure of tight assemblies. An example is shown Figure 6. The remainder of this section presents an extension of the solid-clipping technique for such cut-volumes.

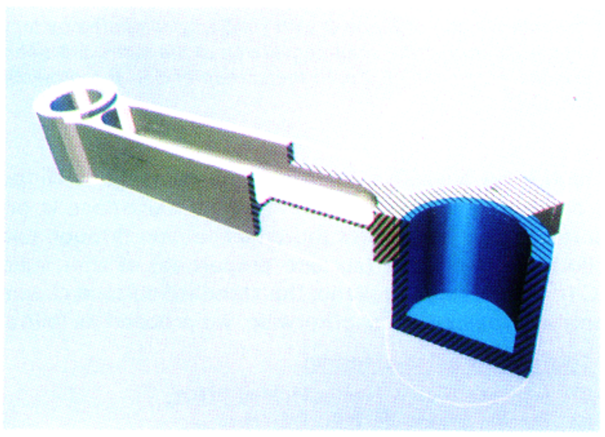

Figure 6. ***Solid-clipping with non-convex cut-volumes***: *Three clipping planes*, C_1, C_2, *and* C_3, *are used to define a compound cut-volume*, $C_1 \cap (C_2 \cup C_3)$.

Although, the metaphor of a "cut-volume", v, interactively manipulated by the designer to remove obstructing portions of the assembly may be more intuitive than the notion of a "clipping volume", v′, used to delimit the assembly through an intersection operation, both formulations are equivalent, since v′ is the complement, $\overline{v}$, of v, and for any solid S, we have: $S - v = S \cap v'$.

Given a Boolean expression of v, it is straightforward to extract a disjunctive form[5] for v′. For example, if the linear half-space volumes are denoted v_i, the cut-volume $v = (v_1 \cup v_2) \cap (v_3 \cup v_4)$ yields the following disjunctive form of two products: $(\overline{v_1} \cap \overline{v_2}) \cup (\overline{v_3} \cap \overline{v_4})$ for v′.

The intersections of S with these convex clipping-products are processed one-by-one using the algorithm below. The image of the union of these intersections is composed via the standard z-buffer test.

<u>**Solid-clipping algorithm for a clipping-product:**</u>

```
01 For allpixels do Z=0, I=0, Mp=0
02 For each solid S do
03    For each clipping-product P do
04      Activate all the front clipping planes of P
05      Disable writing into the depth and frame buffers
06      Render all the faces of S toggling Mp
07      Select rendering color for S
08      Enable writing into the depth and frame buffers
09      Activate all the front and back clipping planes
10      Render the front faces of S
11      For each front clipping plane C in P do
12        Deactivate C
13        Shade C and reset Mp for pixels where Mp==1
14        Activate C
15      Deactivate all planes of P
```

In Line 06, the front and back faces of S are clipped against all the front clipping planes of a product and then scan-converted. Each time a pixel q is visited during that scan-conversion, its mask bit Mp(q) is toggled. The frame and z-buffers are never updated during that scan-conversion (see Line 05). After the execution of Line 06, the mask Mp corresponds to a cut-volume composed of only the front cutting planes of that product (see Figure 7). When the cross-sections are displayed for that product (Line 13), this mask is used in conjunction with the other front and back clipping planes to delimit the contribution of each front clipping plane.

Each front clipping plane is temporarily deactivated (Line 12) prior to display (Line 13) to avoid self-clipping. The portions of the front faces of S that lie within the clipping-product and are not hidden by previously rendered objects are rendered into the z-buffer and the frame buffer (Line 10). The rendering in Line 13 is performed using the standard z-buffer test.

An efficient implementation of the solid-clipping with composite cut-volumes requires: (1) a standard z-buffer, (2) an application-controlled set of clipping planes, (3) one bit-plane for the mask, and (4) facilities for programming the scan-conversion so as to toggle the bit-plane for each surface point and to use the mask as a condition for rendering. All these facilities are supported by commercially available graphics hardware for a limited number of application-controlled clipping planes.

[5] The disjunctive form is a union of products, each product being the intersection of half-spaces.

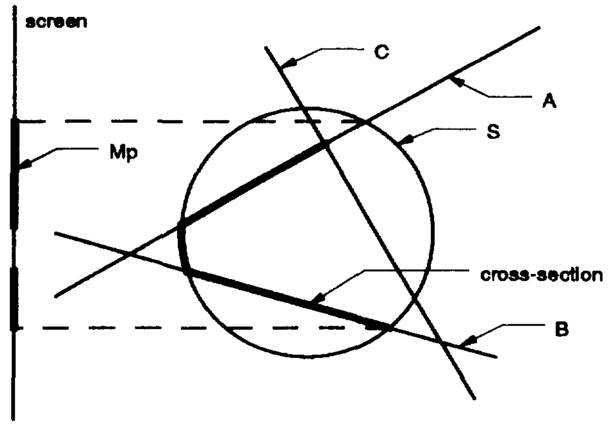

Figure 7. ***Masking for a clipping-product:*** *Two front clipping planes A and B and one back clipping plane C bound a clipping-product. Mp is constructed by scan-converting S clipped by A and B. The visible cross-sections are obtained by rendering A (clipped to B and C) and B (clipped to A and C) over pixels where Mp is 1.*

3. Interferences

Usually, a mechanical assembly must be free from interferences[6], but may contain lower-dimensional contact regions [7] between its components.

Intersections between pairs of solids may be computed in various ways. A geometric approach evaluates the boundary of the regularized Boolean intersection of the two solids. The existence of a single vertex in the intersection suffices to indicate interference. Efficient Null Object Detection techniques may be used, especially if the solids are in CSG form [10, 15]. Hardware architectures for testing interferences between triangulated boundaries have also been proposed [16]. Although asymptotically efficient computational geometry techniques for finding the minimum distance between two polyhedra are available [2], these numeric approaches are too expensive for interactive inspection and should be reserved for the final stages of the assembly verification.

Two hardware-assisted graphics techniques are relevant to interference detection: (1) a discretized (ray casting) approach reduces interference detection to a series of one-dimensional interval-intersection tests and is supported by special-purpose ray-casting hardware [3] and (2) the ability to automatically select and report which of the scan-converted objects interfere with an application-defined block provides a mechanism for eliminating unnecessary interference calculations. (Solids that are clearly disjoint from any solid S because they are disjoint from a box containing S may be efficiently identified that way.)

Geometric intersection techniques are too expensive. Ray-casting can be efficiently parallelized, but interactive performance on an affordable platform has not been demonstrated. Boxing techniques provide only a necessary condition for interference. Consequently, the approach described in this section constitutes an important tool for detecting and displaying interferences. The first portion of this section focuses on an extension of the solid-clipping technique to highlight interferences in the cross-sections (Figure 9). The second portion presents a technique for automatically locating the beginning of interference regions along a user-specified search direction. This search facility is used interactively for two purposes: (1) to quickly and reliably establish that a particular region is free of interferences or (2) to automatically locate the first interference region and position the clipping plane at its beginning to facilitate the visual inspection of the extent of the interference. Subsequent interferences are located automatically by starting the search past the current interference region.

3.1 Highlighting interference areas

The algorithm for highlighting the interference is presented below in its simplified version for a **clipping product restricted to a single front clipping plane** C. The successive steps are illustrated in Figure 8. The algorithm computes a parity pixel mask, Mp, for the cross-section of the current solid, a cumulative (union) pixel mask, Mu, for the union of the cross-sections of all previously processed solids, and an intersection-mask, Mi. The cross-section of the solid, restricted to (Mp AND NOT Mu), is rendered with the solid's colors. The interference area is rendered at the end in a highlighted mode over Mi.

```
Algorithm for highlighting interferences:
01 For all pixels do Z=0, I=0, Mu=0, Mi=0, and Mp=0
02 For each solid S do
03   Activate C as a clipping plane
04   Scan S toggling Mp and rendering where Mu==0
05   Disable writing into z-buffer
06   Deactivate C
07   Render C where Mp==1 and Mu==0
08   For all pixels in R do
09        If (Mu==1 && Mp==1) Mi=1
10        If (Mp==1) Mu=1 and Mp=0
11   Enable writing into the z-buffer
12 Disable writing into z-buffer
13 Select color and style for the interference
14 Render R' on C for pixels where Mi==1
15 Enable writing into the z-buffer
16 Disable writing into the frame buffer
17 Render R' on C for pixels where Mu==1
```

In Line 04, all the front and back faces of S are clipped by C and then scan-converted. For each access to a pixel during that scan-conversion the pixel's parity mask, Mp, is toggled. Furthermore, if at that pixel the mask Mu is not set, the z-buffer and frame buffer are updated. (Note that this update is not necessary for the back faces of S.) Testing Mu prior to update avoids overwriting previously computed cross-sections for which the z-buffer has not yet been properly set.

[6] The interference between two solids A and B is their regularized intersection: $(A \cap^{*} B)$. Regularization removes lower dimensional parts, thus, the regularized intersection is the closure of the interior of the intersection [8].

[7] The contact between two solids is $((A \cap B) - (A \cap^{*} B))$, the set theoretic difference between their set theoretic intersection and their regularized intersection.

In Line 07, the R′ portion of C is rendered over pixels in the Mp mask, but out of the Mu mask, to fill the cross-section contribution of S. However, the z-buffer is not yet updated to the cross-section depth, so as to avoid depth-conflicts when filling in the interference region, Line 14. The z-buffer is correctly set in Line 17, without altering previously computed colors in the frame buffer. This technique of delaying the update of the z-buffer is used to make sure that when the interference area of the cross-section is filled, the surface depth is not compared to previously computed z-values from pixels on the same cross-sectioning plane. Such comparisons, when performed with limited numeric accuracy, produce inconsistent pixel colors across the overlap area.

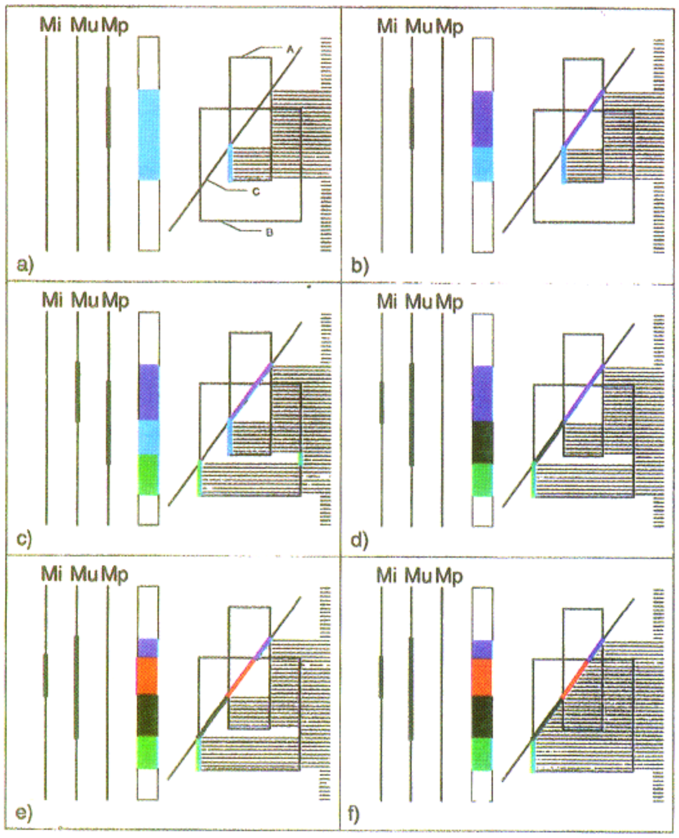

Figure 8. ***Highlight construction:*** *A 2D slice through the scene is used to explain the steps of the interference-highlighting algorithm. The interfering solids A and B are intersected by the clipping plane C. In the drawing the contents of the z-buffer is indicated by thin horizontal lines. The contents of the pixel-masks are shown using the thin vertical lines on the left. Asserted bits are shown by heavy lines. The contents of the frame buffer is indicated using colors in the vertical window on the left of each figure. Color lines on the contours of A, B, or C indicate, for each pixel, which surface has contributed to the frame buffer. (a) Solid A is scan-converted into the frame buffer and the z-buffer; the parity mask is constructed in Mp. (b) The contents of Mp is unioned into Mu. (c) Solid B is scan-converted into the frame buffer and the z-buffer; the parity mask is constructed in Mp. (d) Mi is asserted where Mp and Mu overlap. The contents of Mp is unioned into Mu. (e) The clipping plane is scan-converted into the frame buffer. Regions of interference are marked in red. (f) The clipping plane is scan-converted into the depth buffer.*

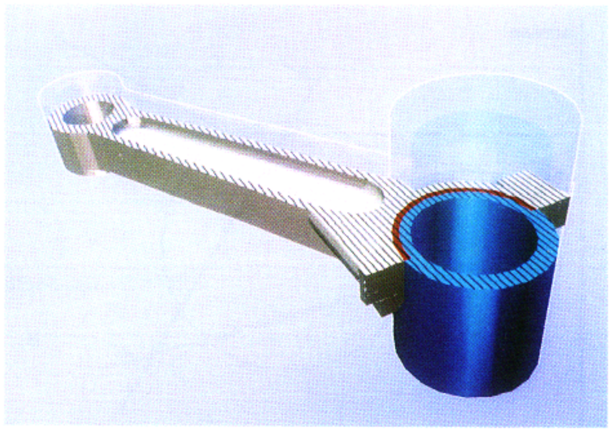

Figure 9. ***Interference highlight:*** *The solid-clipping technique of Figure 4 is enhanced by highlighting (in red) the cross-section regions where pairs of solids interfere. The portion of the assembly removed by the cut-volume is displayed in transparent mode.*

An elegant alternative was independently invented by Akeley [1]. It exploits the numeric increment operation on three stencil bits to implement our mask-combine operations (Lines 09 and 10). For each solid, Mp is computed as in our approach. Then the 3-bit counter (Mi,Mu,Mp) is incremented for pixels where Mp is set. The counter clamps to preserve Mi in case of overflow (i.e. when more than three solids are intersected by the same portion of the cross-section plane).

Using several parallel cross-section planes and only rendering interference areas, one can produce stacks of 2D cross-sections that indicate the extent and the shape of the 3D interference volume (see Figure 10). An algorithm for rendering **only the interference part,** i.e. the cross-section where the mask Mi was set may be obtained by eliminating the shading operations Lines 04, 07, and 17. It was used to produce the stacks of Figure 10.

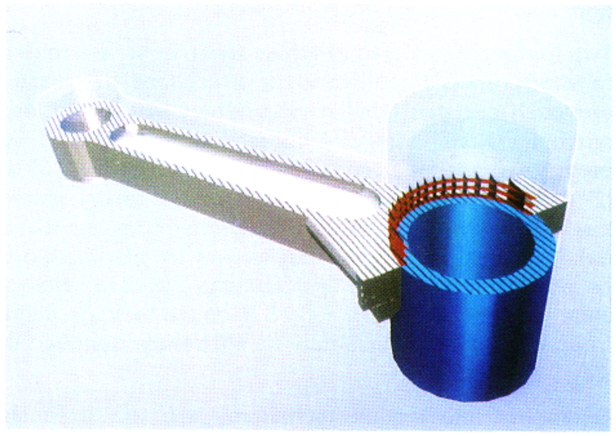

Figure 10. ***Interference stacks:*** *The interference visualization technique of Figure 9 is further enhanced with stacks of parallel cross-sections through the 3D interference region.*

3.2 Locating interference regions

This subsection is devoted to the automatic detection and location of interferences and contacts along a user-defined search direction and within a given search interval.

Without loss of generality, the search direction is chosen orthogonal to the cross-sectioning plane C. The search interval is confined to a slice between C and another plane C′ parallel to C. The location of C′ may be specified by the user or computed automatically from a bounding box, so as to extend past the entire assembly. The positions of C and C′ are indicated by the starting and ending parameters Z_{start} and Z_{end} along the search direction D.

Using a stack of parallel cross-sections evenly distributed between Z_{start} and Z_{end} and testing if any of them contains an interference region will not guarantee the detection of interferences, since these may occur between two consecutive cross-sections. The cost of testing a sufficient number of cross-sections to reduce the size (in depth) of possibly missed interferences is prohibitive.

Instead of such a discrete probing, the technique presented here uses the procedure "IntersectionFreeSlice" to compute a sufficient but not necessary condition for interference. If the answer is negative, the designer may be reassured immediately. Otherwise, the following algorithm recursively subdivides the search interval (Z_{start}, Z_{end}) until a user-defined maximum level (i.e. minimal slice thickness), L, is reached (in which case, the beginning of a possible interference region is returned) or until all branches of the search tree that correspond to positive test result have been explored (in which case there is no interference and Z_{end} is returned). The minimal slice thickness, the depth resolution, and the z-scaling factors control the accuracy of the test and define the ability to differentiate between interference and contact. The command Search(Z_{start}, Z_{end}, Ml), where Ml defines the maximum recursion level, starts the search. Ml may be adjusted to ensure the desired accuracy. The parameters, Zs, Ze, and level define the current status of the recursion.

Algorithm for interval location:

```
Search (Zs,Ze,level)
  If (IntersectionFreeSlice(Zs,Ze)) return $Z_{end}$
  If (level==L) return Zs
  Zm=(Zs+Ze)/2
  Zf=Search(Zs,Zm,level+1)
  If (Zf!=$Z_{end}$) return Zf
  Else return Search(Zm,Ze,level+1)
```

A 2D bounding box around the discovered interference is used to position an arrow highlighting the potential interference region. The clipping plane C is automatically placed at the beginning of that interval, so that the user can inspect the area, then move C past the current interference, and finally resume the search.

The "IntersectionFreeSlice" test is implemented in the following algorithm by generating a mask Mp for the projection of the intersection of the current solid S with the slice and by testing if this mask intersects the Mu mask for the union of the projection of previously processed solids. The approach is based on the following property.

If the projections of the slices through the solids are disjoint, there is no interference within the slice.

IntersectionFreeSlice:

```
01 Activate C as a front clipping plane
02 For all pixels do Mu=0, Mi=0, Mp=0
03 For each solid S do
04   Scan-convert S toggling Mp
05   Activate C' as a back clipping plane
06   Scan-convert S forcing Mp=1
07   For all pixels in R do
08       If (Mu==1 && Mp==1) return 0
09       If (Mp==1) Mu=1 and Mp=0
```

To avoid missing thin interferences that fall between pixels, it suffices, as part of the shading of a solid, to draw the edges of each solid in lines 3 pixels wide. Each edge must be drawn twice to maintain the parity condition. (We simply draw the edges of each face after shading it.) On the other hand, to distinguish contact regions from true interferences, we apply a two-dimensional discretized morphological shrinking operation [12], i.e. a 3x3 filter over all pixels, to the mask Mi so as to remove interferences that are thinner than two pixels.

By acting on the scaling factor (i.e. the space distance corresponding to the inter-pixel resolution), one can adjust the thresholds between clearance, non-invasive contact, and true interference. By performing the "IntersectionFreeSlice" test twice (once with drawing the edges and once with eroding the mask) one can distinguish clearance (if both test return false), from contacts (if the results of both tests are different), from interferences (if both tests return true). However, searching true interferences (through mask erosion) for regions with oblique contact areas between overlapping faces of different objects forces the adaptive subdivision to visit all the branches of the search tree down to a depth corresponding to a slice thickness for which there is no interference between the projections of the solids.

The interference search automatically positions the cross-sectioning plane at the beginning of an interference region. The user examines the interference by moving the viewpoint and the clipping plane. The interfering objects may be selected by a graphic pick and the corresponding CAD models which require engineering changes may be identified. Facilities for interactively hiding some models or for producing exploded views also help decide which of the interfering parts must be redesigned.

The search algorithms described above require extensions to the functions supported by currently available graphics libraries and may also involve some hardware modifications. For example, Line 08 of the "IntersectionFreeSlice" algorithm requires a feedback from the buffer to the application. Such a feedback exists for reporting enclosing boxes around pixels traversed by the scan-conversion, but does not take into account any result of testing mask values for these pixels. This step is handled today by the application software which must inspect each pixel of Mi. Similarly, the erosion operation is also currently performed in software, which considerably reduces the performance of the search algorithm.

Nevertheless, except for regions where two or more objects are in contact, the search algorithm only visits a few branches of the search tree, and thus its software implementation requires the inspection of only a small number of pixel-masks within a limited domain (R).

Conclusion

Simple algorithms for displaying cross-sections through solids, for highlighting interference areas, and for automatically detecting interferences and contacts between solids have been presented. Because the additional cost for filling the cross-sections and for highlighting interference areas does not significantly exceed the original rendering cost, these algorithms exhibit realtime performance for small assemblies—with the exception of interference detection. They provide the engineering visualization techniques needed to replace the expensive clay models, traditionally used during the design-inspection phases, by electronic "virtual" solid models. The algorithms have been integrated in an experimental system developed by the Interactive Geometric Modeling group at IBM Research and have been successfully tested on industrial assembly models.

Acknowledgements

We are very grateful to Kurt Akeley for his comments on this paper and for allowing us to compare both his and our implementations. We also wish to thank Dan Brokenshire for pointing out the limitations of the standard clipping technique for solid modeling applications and for participating in the early phases of this work.

References

[1] Kurt Akeley, Silicon Graphics Inc.. Private communication subsequent to the SIGGRAPH review process. March 1992.

[2] David Dobkin and Herbert Edelsbrunner, Space Searching for Intersecting Objects. ACM & IEEE Sum. on Foundations of Computer Science, IEEE Computer Society Press, New York, NY, 387-392, 1984.

[3] John Ellis, Gershon Kedem, Rich Marisa, Jay Menon, and Herbert Voelcker, Breaking Barriers in Solid Modeling. CIME, pages 28-34, February 1991.

[4] Dave Epstein, Friderik Jansen, and Jarek Rossignac, Z-buffer Rendering from CSG: The Trickle Algorithm. Research Report, RC 15182, IBM T.J. Watson Research Center, Yorktown Heights, NY, December 1990.

[5] Jack Goldfeather, Steve Molnar, Greg Turk, and Henry Fuchs, Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning. IEEE Computer Graphics and Applications, 9(3):20-28, May 1989.

[6] Starbase Reference Manual. "set_capping_planes" command. Hewlett Packard.

[7] Martti Mäntylä, An Introduction to Solid Modeling. Computer Science Press, Rockville, Maryland, 1988.

[8] Aristides Requicha and Robert Tilove, Mathematical Foundations of Constructive Solid Geometry: General Topology of Regular Closed Sets. Production Automation Project, Tech. Memo. No. 27a, Univ. of Rochester, June 1978.

[9] Jarek Rossignac, Accurate scanconversion of triangulated surfaces, in A. Kaufman, editor, Advances in Computer Graphics Hardware VI, Springer-Verlag, Berlin, 1992.

[10] Jarek Rossignac and Herbert Voelcker, Active Zones in CSG for Accelerating Boundary Evaluation, Redundancy Elimination, Interference Detection and Shading Algorithms. ACM Transactions on Graphics, 8(1):51-87, January 1989.

[11] Jarek Rossignac and Jeffey Wu, Correct Shading of Regularized CSG Solids Using a Depth-Interval Buffer. Eurographics Workshop on Graphics Hardware, Lausanne, Switzerland, September 1990.

[12] Jean Serra, Image Analysis and Mathematical Morphology. Academic Press, New York, 1982.

[13] Graphics Library—Reference Manual, Iris 4D VGX. Silicon Graphics, Inc., 1990.

[14] Robert Tilove, Line/Polygon Classification: A Study of the Complexity of Geometric Computation. IEEE Computer Graphics and Applications, 1(2):75-88, April 1981.

[15] Robert Tilove, A Null Object Detection Algorithm for Constructive Solid Geometry. Comm. ACM, 27(7):684-694, July 1984.

[16] Fujio Yamaguchi, A unified approach to interference problems using a triangle processor. Proceedings SIGGRAPH'85, 19(3):141-149, 1985.