Hidden contours on a frame-buffer

Jarek R. Rossignac
Maarten van Emmerik (*)

Interactive Geometric Modeling
IBM, T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA.
Phone: 914-784-7630, Fax: 914-784-6273, Email: jarek@watson.ibm.com

(*) van Emmerik is now back in the Netherlands with Intergraph

To comply with drafting practices and because shaded images do not always reveal the internal or hidden structures of 3D models, designers need wireframe images with hidden lines dashed and noncontour tesselation edges removed. Software techniques for wireframe rendering of polyhedra that require the viewpoint-dependent identification of the visible portions of intersection and contour (i.e. silhouette) edges are too slow for interactive applications. Hardware support in contemporary graphics pipelines is unavailable or at best limited to the identification of contour edges. In this paper, new hardware assisted techniques for hidden-line removal and determination of contour edges are presented. The techniques do not require any face/edge adjacency information and can be implemented easily on any platform that supports a hardware z-buffer.

Categories and Subject Descriptors:
I.3.1 [Computer Graphics]: Hardware Architecture — raster display devices;
I.3.3 [Computer Graphics]: Picture/Image Generation — display algorithms;
I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism — visible line/surface algorithms

Key Words and Phrases: Hidden Lines Removal, Contour, Silhouette, Profile.

Proceedings of the 7th Workshop on Computer Graphics Hardware, Eurographics, Cambridge, UK, September 1992.

1. Introduction

1.1 Motivation

Recent developments in graphics hardware enable real-time shading of 3D objects at relatively low cost. Shaded images clearly show the shape and color of visible surfaces, but do not provide any information about hidden features or internal structures (Figure 1). Shaded cross-sections (Rossignac et al. 1992) and exploded views may only be used to clarify the internal structure, but not to detect the hidden features. Wireframe images (which may be superimposed on shaded pictures) can show these hidden features, but are difficult to interpret, unless "hidden-lines" techniques are applied to differentiate—through line style or color—between the hidden and the visible lines. Furthermore, commonly used polyhedral approximations of curved surfaces exhibit a high density of "tesselation" edges, which should only be displayed when contributing to the "silhouette" of the surface.

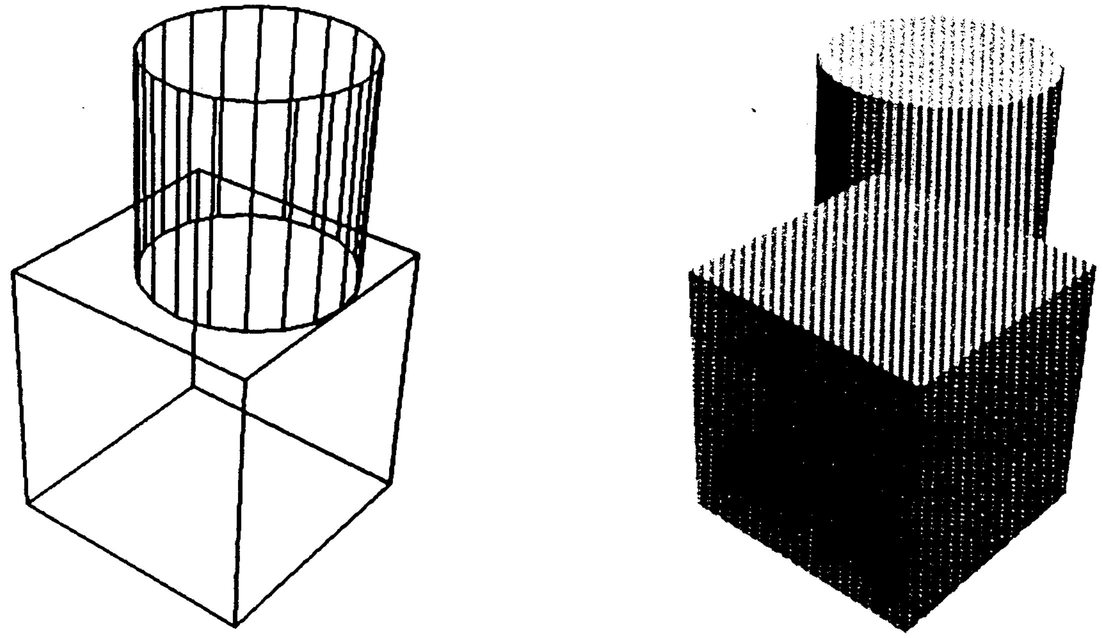

Figure 1. The wireframe representation does not disclose which object is in front, whereas the shaded representation does not disclose the extent of the cylinder.

1.2 Previous work

2D image processing technique have been proposed (Takafumi and Takahashi 1990) to enhance shaded images of 3D shapes by highlighting visible shape features and contour edges. Technique for identifying and displaying visible parts of lines are called "visible-line determination" or "hidden-line removal"—abbreviated HLR (Foley et al, 1990). An edge segment is visible if the line segments between any of its points and the viewpoint do not intersect any face of the model (except at its end-point). An edge segment is hidden if it does not contain visible segments. Early visible-line algorithm were limited to convex polyhedra (Roberts 1963). Efficient solutions for plotting a surface defined as a function of two variables were also proposed (Williamson, 1972; Butland, 1979; Skala, 1985; Boller, 1985). Several algorithms were developed for more general collections of 3D polygons (Appel 1967; Galimberty and Montari, 1969; Loutrel, 1970). An overview of various algorithms that take advantage of edge and face coherence may be found in (Sutherland et al. 1974). The additional complexity of curved surfaces was addressed in (Kamada and Kawai, 1987; Rankin, 1987; Li, 1988; Elber and Cohen, 1990).

1.3 Contribution

The above techniques are currently not supported by commercially available graphics hardware systems for several reasons. First, because they require an extensive amount of calculations and auxiliary datastructures. Second, because they exploit edge/face adjacency information, not always available to the application and certainly not available to the processors in the graphics hardware pipeline. In this paper, a new technique that exploits the graphics hardware capabilities of contemporary commercial systems is presented. The techniques allows real-time elimination—and thus display in a different style or color—of hidden lines using a hardware z-buffer. A different technique for the real-time display of contour edges is also presented and algorithms for combining these two techniques are proposed.

1.4 Terminology

The curved faces used in CAD are often tesselated for rendering purposes. The tesselation of a face is a piecewise planar approximation of the face by a connected mesh of facets. Each facet is bounded by a variable number of edges that are either tesselation edges or intersection edges. **Tesselation** edges are shared by two facets of the same surface tesselation. **Intersection** edges correspond to edges where (curved) surfaces intersect or where the faces of the objects meet.

The **silhouette** (also called profile or contour) of a smooth curved surface is the locus of surface points where the surface normal is orthogonal to the viewing direction. When perspective transformation is used, the viewing direction at point P is defined by the line that joins the viewpoint and point P. However, polyhedral or tesselated models may be transformed by the perspective transformation prior to the establishment of hidden and contour edges, which can then be carried out using an isotropic projection model. Thus perspective deformations need not be further discussed.

Contour edges of a polyhedron are simultaneously adjacent to a front-facing face and to a back-facing face. (Assuming that in the isotropic projection the viewing direction is along the positive z-axis, a face is front-facing if the z-component of its outward normal is negative and is back-facing if the z-component of its outwards normal is positive.)

2. Four techniques for hidden-line removal

Four techniques for displaying tesselated wireframes with hidden edges suppressed or dashed are shown in Figure 2. The first technique (TEHS) displays all of the visible edges and suppresses the hidden edges. The second technique (TEHD) displays the visible edges in one style (say as solid lines) and the hidden edges in a different line style (e.g. dashed or with a different color or brightness). The third technique (CEHS) only displays the visible intersection and contour edges and suppresses all other tesselation edges. The fourth technique (CEHD) displays the visible contour and intersection edges as solid curves and the hidden contour and intersection edges in a different line style.

2.1 Background

The edges are displayed by writing into a frame buffer, i.e. memory bit planes associated with the pixels of the screen. The depth of faces is stored in a z-buffer also composed of bit-planes associated with pixels. We use the terms z-buffer and framebuffer to define these two buffers. The algorithms for generating the images require four basic display functions which have to be provided for each object:

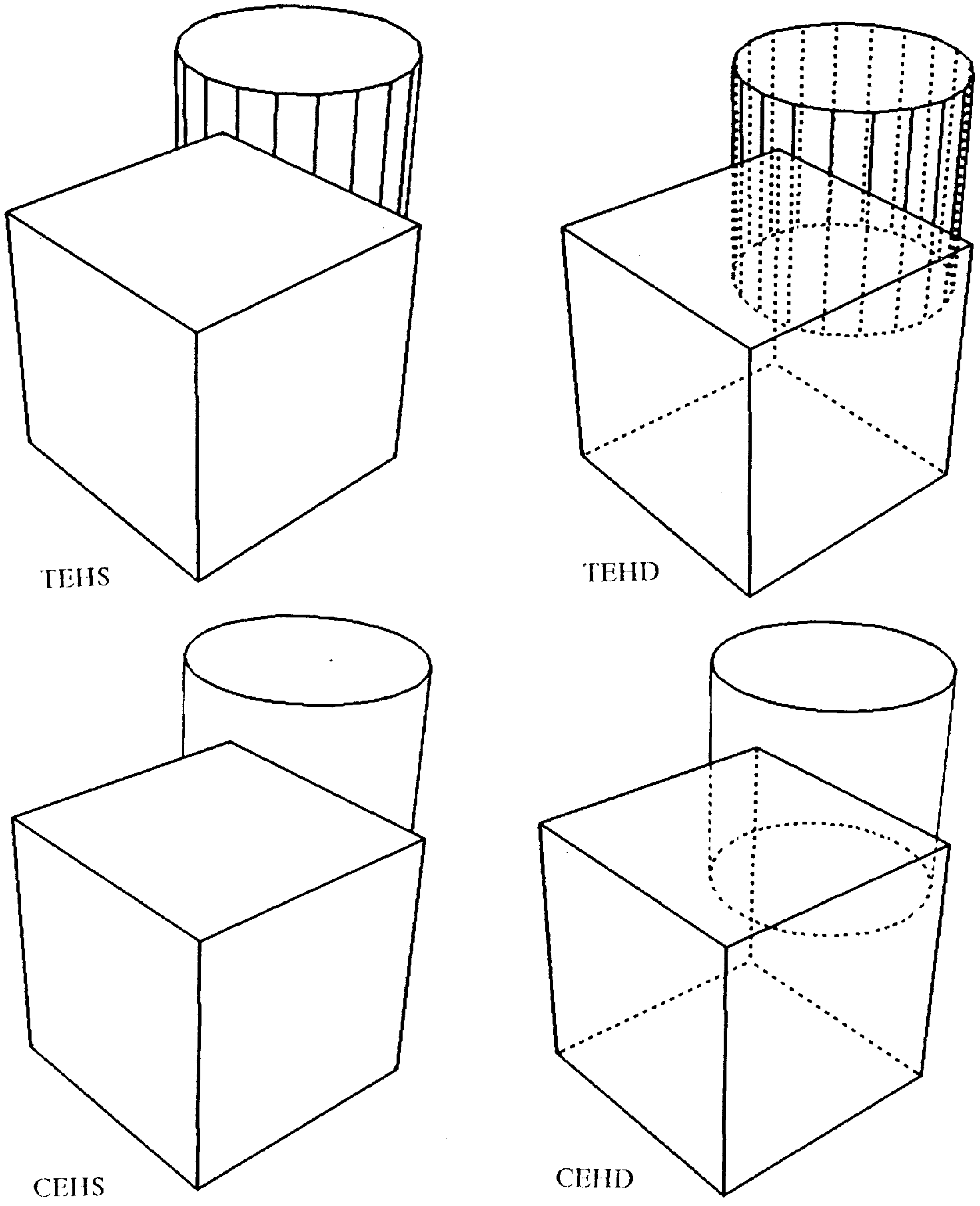

Figure 2. Four techniques for hidden-line removal: Tesselation Edges with Hidden-lines Suppressed (TEHS), Tesselation Edges with Hidden-lines Dashed (TEHD), Contour Edges with Hidden-lines Suppressed (CEHS), and Contour Edges with Hidden-lines Dashed (CEHD).

```
DisplayTessellationEdges (Object or collection)
DisplayIntersectionEdges (Object or collection)
DisplayAllEdges (Object or collection)
Fill (Object or collection)
```

The function 'DisplayTesselationEdges' draws the object (or collection of objects) as a tesselated wireframe. The function 'DisplayIntersectionEdges' draws only the intersection edges of the object(s), for example the edges approximating the circles at the bottom and top of a cylinder. The function 'DisplayAllEdges' combines the effect of 'DisplayTesselationEdges' and 'DisplayIntersectionEdges'. Finally, the function 'Fill' scan-converts all the faces of the object(s) and stores the associated depth values in the z-buffer. The 'Fill' function fills the z-buffer with the depth of the visible faces at all pixels covered by the projection of the solid, but does not write into the frame buffer.

In addition to the above display functions, we are using the following graphic commands easily implemented on most graphics libraries:

```
Move (Forward|Backward)
LineStyle (Solid|Dashed)
Zbuffer (On|Off)
ZbufferClear
```

The function 'Move' translates all objects towards or away from the viewer, and may be simply implemented as a temporary incremental modification of the z-translation component in the world-to-screen viewing matrix. (Moving forward brings the object closer to the viewer). The function 'LineStyle' provides the selection between two line styles used to distinguish visible and hidden edges (e.g solid and dashed). The functions 'Zbuffer' and 'ZbufferClear' respectively (de)activate and clear the z-buffer.

The algorithms for generating the various hidden line display modes will be discussed below. For simplicity, it is assumed that the z-buffer and the frame buffer have been cleared before the algorithms start or contain information that is supposed to affect the outcome of the rendering process.

To illustrate how our algorithms process a 3D scene we use a 2D cross-section through the scene by a plane that includes the view point (Figure 3.). In this 2D view (b), dots correspond to the edges of the 3D model and lines correspond to faces of the 3D model.

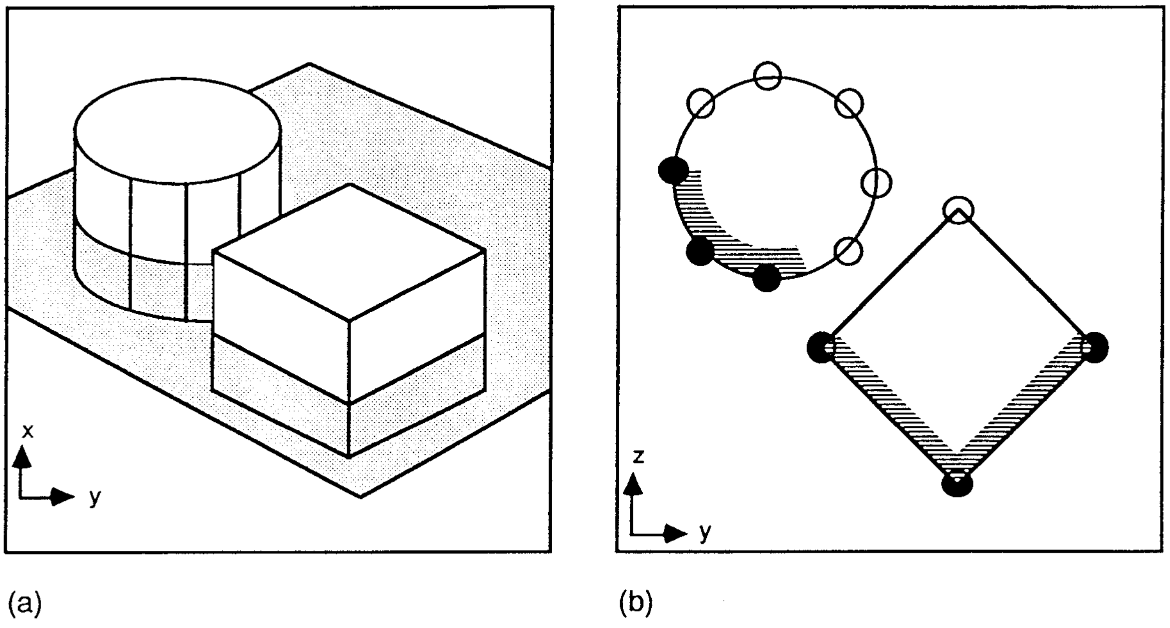

Figure 3. A 3D scene (a) and a top view of the cross-section by a plane that includes the view point (b).

The result of updating the z-buffer during the scan-conversion of a face is shown in 2D (b) by hatching the area just behind the line corresponding to that face. After all the faces were scan-converted, unhatched lines correspond to hidden faces.

The result of drawing the edges of the 3D solid is shown in 2D by circles. Black circles represent visible edges, whereas white circles represent edges that are not drawn, because they are behind objects previously scanned into the depth-buffer. Gray circles will show edges drawn in a different line-style or color to indicate hidden lines.

Some of the edge-drawing steps used in the techniques presented here require that object's edges be slightly displaced along the viewing direction prior to drawing. The 3D image in Figure 4a shows a cube with edges in their original position. The corresponding 2D cross-section (Figure 4b) shows the edges drawn as circles. Edges displaced towards the viewer are shown in 3D (Figure 4c) as cylinders. In the corresponding cross-section (Figure 4d) the circles are drawn displaced toward the viewer.

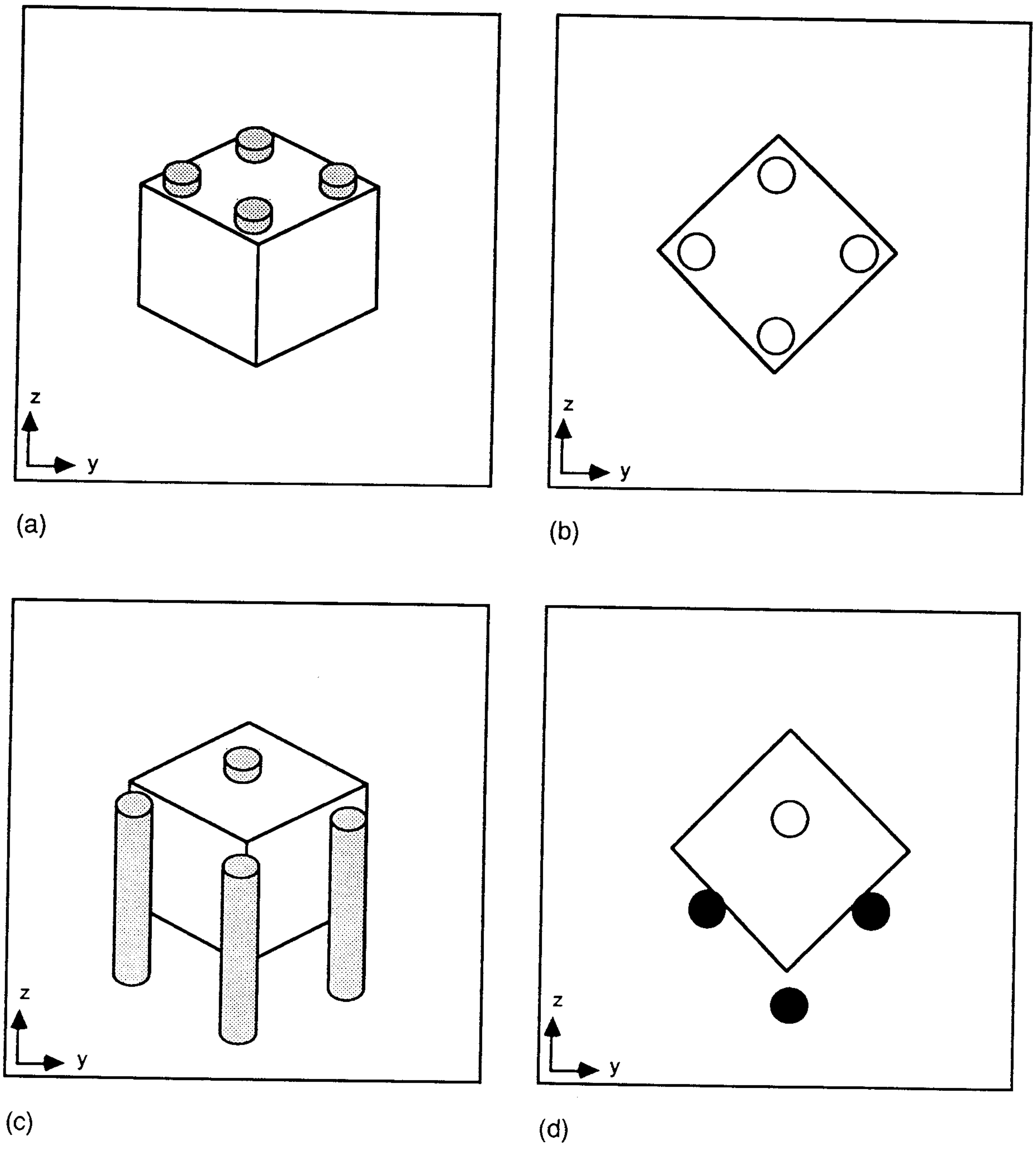

Figure 4. Original position of the edges (a, b) and edges after translation towards the view point (b, c).

When contour edges are drawn using the normal line thickness, their projections on the screen are partially covered by the projections of the faces they bound (Figure 5a). To ensure that a visible contour edge is shown, we draw it in a thicker lines-style, which colors pixels of the background, not covered by the object's faces (Figure 5b). We color half of that circle in black to indicate that the edge is partly obstructed.

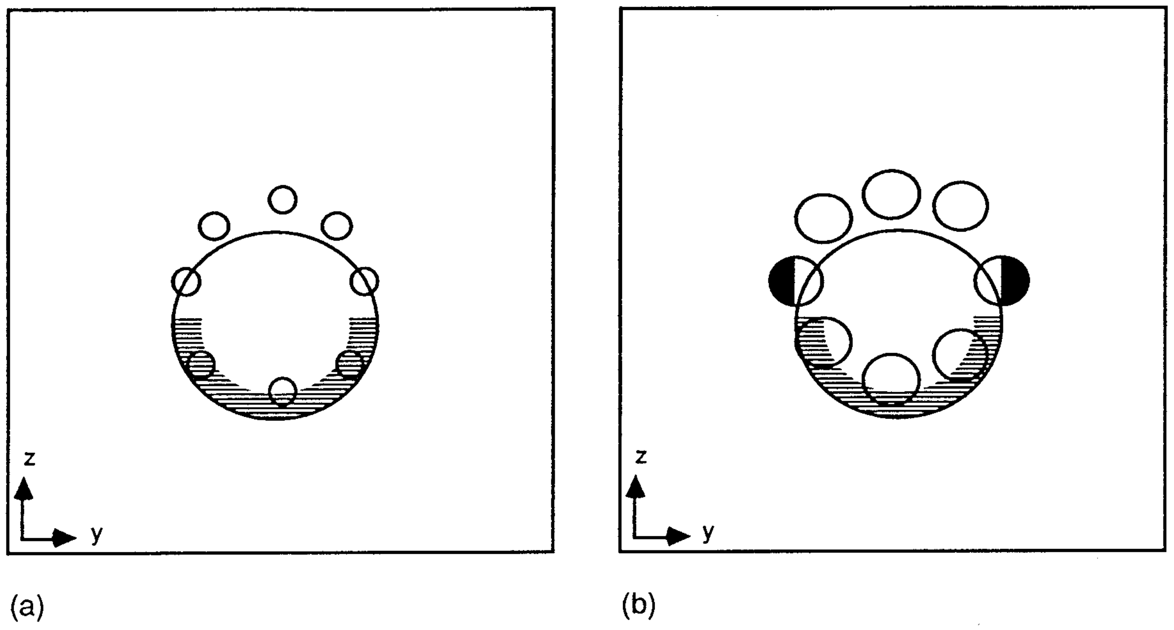

Figure 5. Contour edges drawn with normal line width (a) and with a thicker line width (b).

2.2 Hidden lines removed (TEHS)

The first method (TEHS stands for Tesselation Edges with Hidden lines Suppressed) is described by the following sequence of commands, draws only the visible tesselation edges including non-contour ones.

```
1. Fill (AllObjects);
2. Move (Forward);
3. DisplayAllEdges (AllObjects);
4. Move (Backward);
```

The technique is illustrated in Figure 6. First, all objects are drawn into the z-buffer (A). The hatched area represents the current (minimum) value in the z-buffer. For simplicity, it is drawn behind the surface. After the filling operation, the object is moved towards the user and the tesselated wireframe is drawn (b). As a result, only the visible lines, shown as filled bullets that represent tesselation edges, are displayed. Note that instead of moving the object forward after filling, one could also move the objects backwards prior to the filling operation.

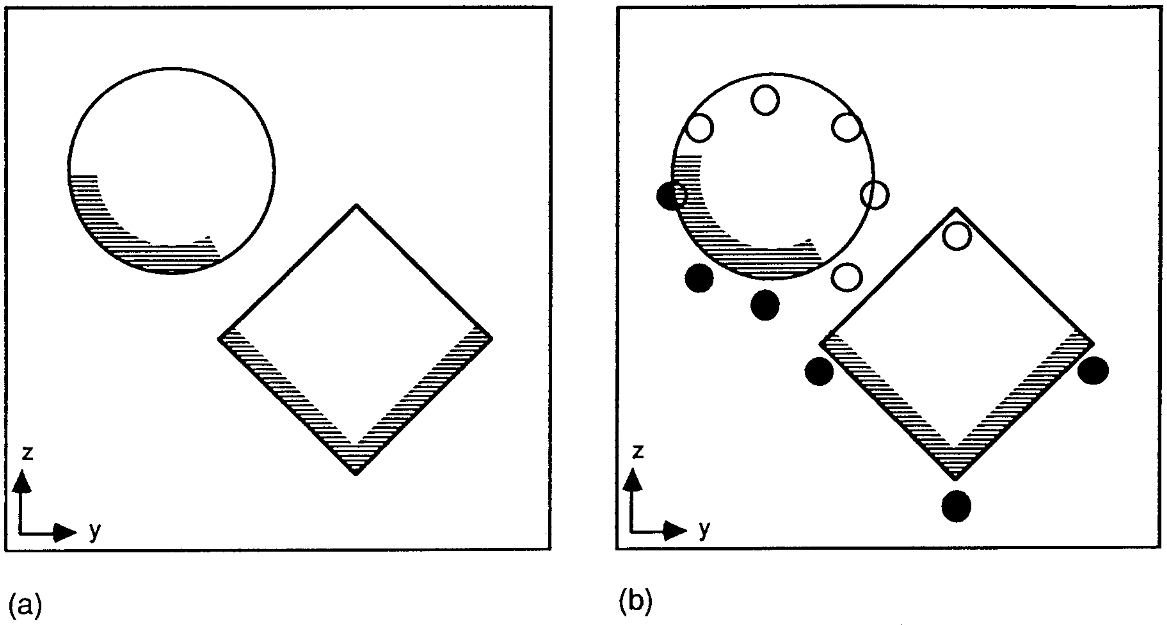

Figure 6. The cross-section after step 1 of the TEHS method (a) shows the state of the z-buffer. The edges are drawn displaced forward (step 2-4) and only visible edges are displayed (black circles in b).

2.3 Hidden lines dashed (TEHD)

The TEHD method (short for Tesselation Edges with Hidden lines Dashed) draws visible tesselation edges as solid lines and hidden tesselation edges in dashed lines. Figure 7 illustrates the process detailed in the algorithm below. We start by drawing all of the edges dashed without performing any z-buffer tests or updates. Then, we simply execute TEHS.

```
1. ZBuffer (Off);
2. LineStyle (Dashed);
3. DisplayAllEdges (AllObjects);
4. LineStyle (Solid);
5. Zbuffer (On);

6. Fill (AllObjects);
7. Move (Forward);
8. DisplayAllEdges (AllObjects);
9. Move (Backward);
```

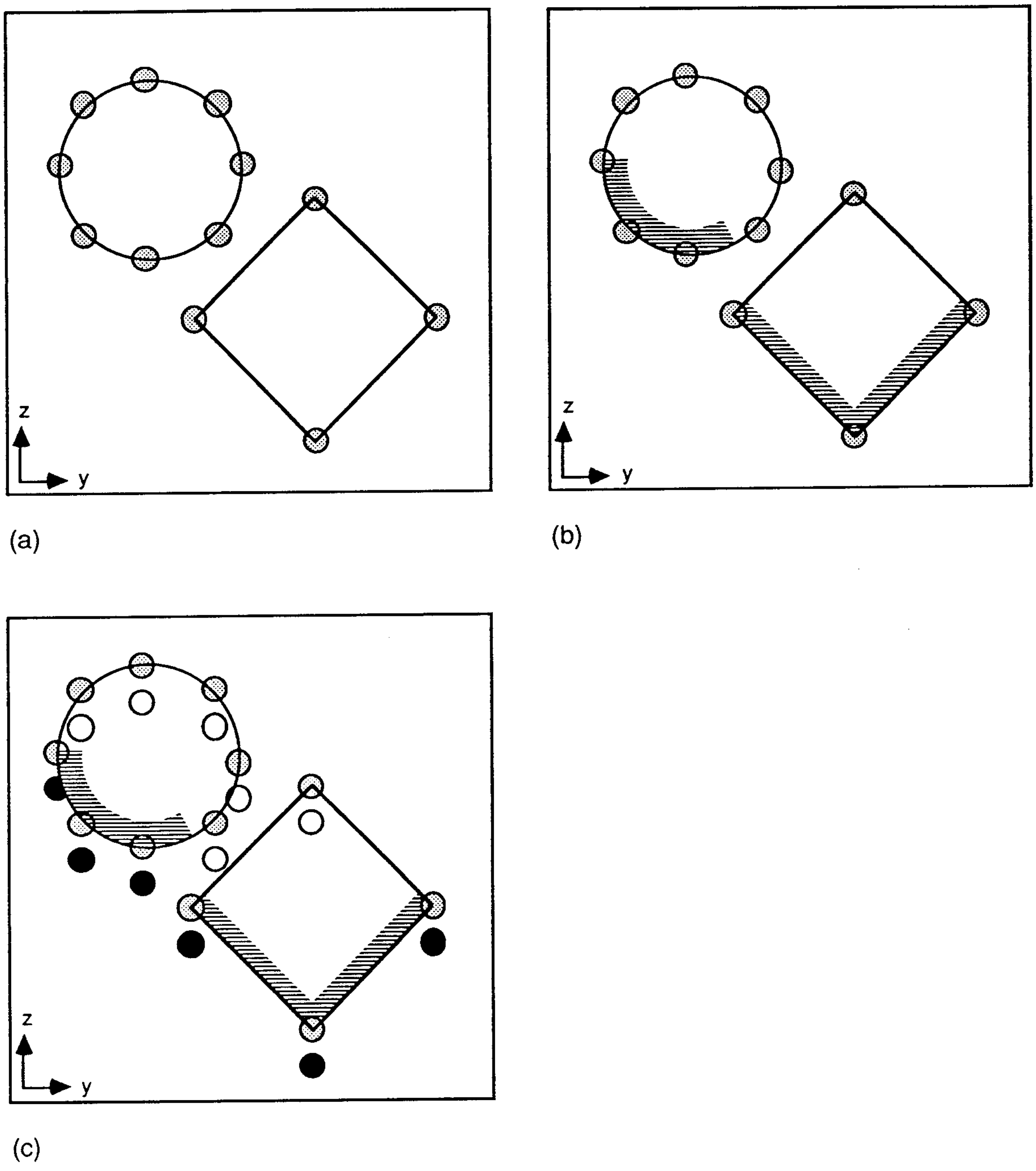

Figure 7. Step 1-5 in the TEHD display the tesselation edges in a dashed line style (a). Subsequently, the objects are scan converted in step 6 (b) and the tesselation edges are displayed in solid line style after moving towards the viewpoint in step 7-9 (c).

2.4 Visible contours only (CEHS)

The CEHS method (Contour Edges with Hidden Lines Suppressed) draws only the visible contour and intersection edges (see Figure 2). The first part of the algorithm is almost identical to the one for suppressing hidden lines (TEHS), except that the object is moved backwards instead of forwards. As a result, all tesselation edges that lie inside the 2D projection of the displayed object are suppressed. Visible contour tesselation edges are partly covered by abutting faces (Figure 8). Using a line width of more than one pixel for drawing the tesselated wireframe ensures that visible contour edges appear properly on the screen. (A line width of one pixel would give unpredictable results.) Once the visible contour edges are drawn, the object is moved forward and the visible intersection edges are displayed as previously.

```
1. Fill (AllObjects);
2. Move (Backward);
3. Linewidth (Thick);
4. DisplayTesselationEdges (Object);
5. Linewidth (Thin);
6. Move (Forward);

7. Move (Forward);
8. DisplayIntersectionEdges (AllObjects);
9. Move (Backward);
```

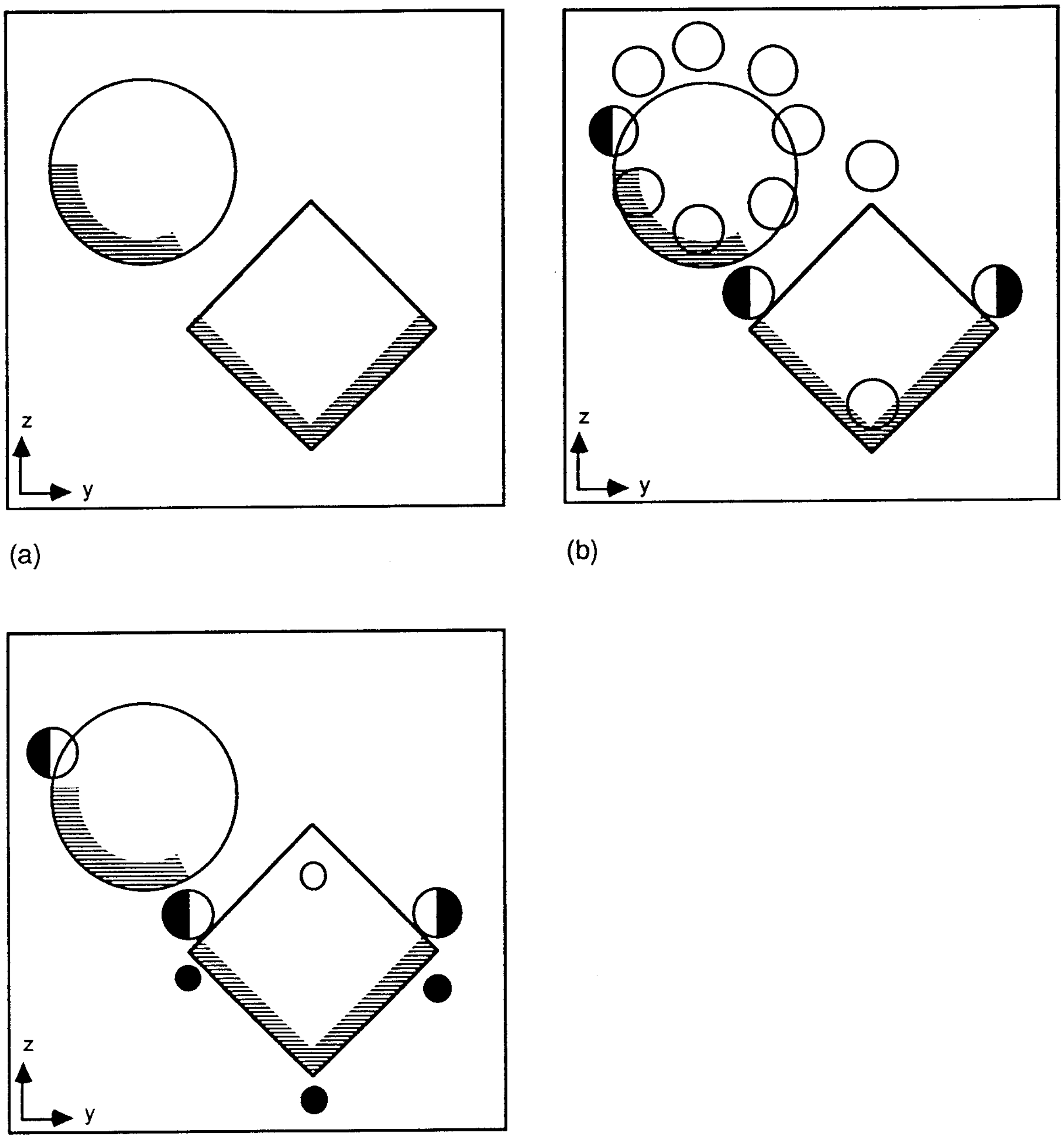

Figure 8. CEHD method after step 1 (a), step 2-6 (b) and step 7-9 (c).

2.5 Hidden contours dashed (CEHD)

The CEHD method (Contour Edges with Hidden Lines Dashed) draws contour and intersection edges using different styles for hidden and for visible edges.

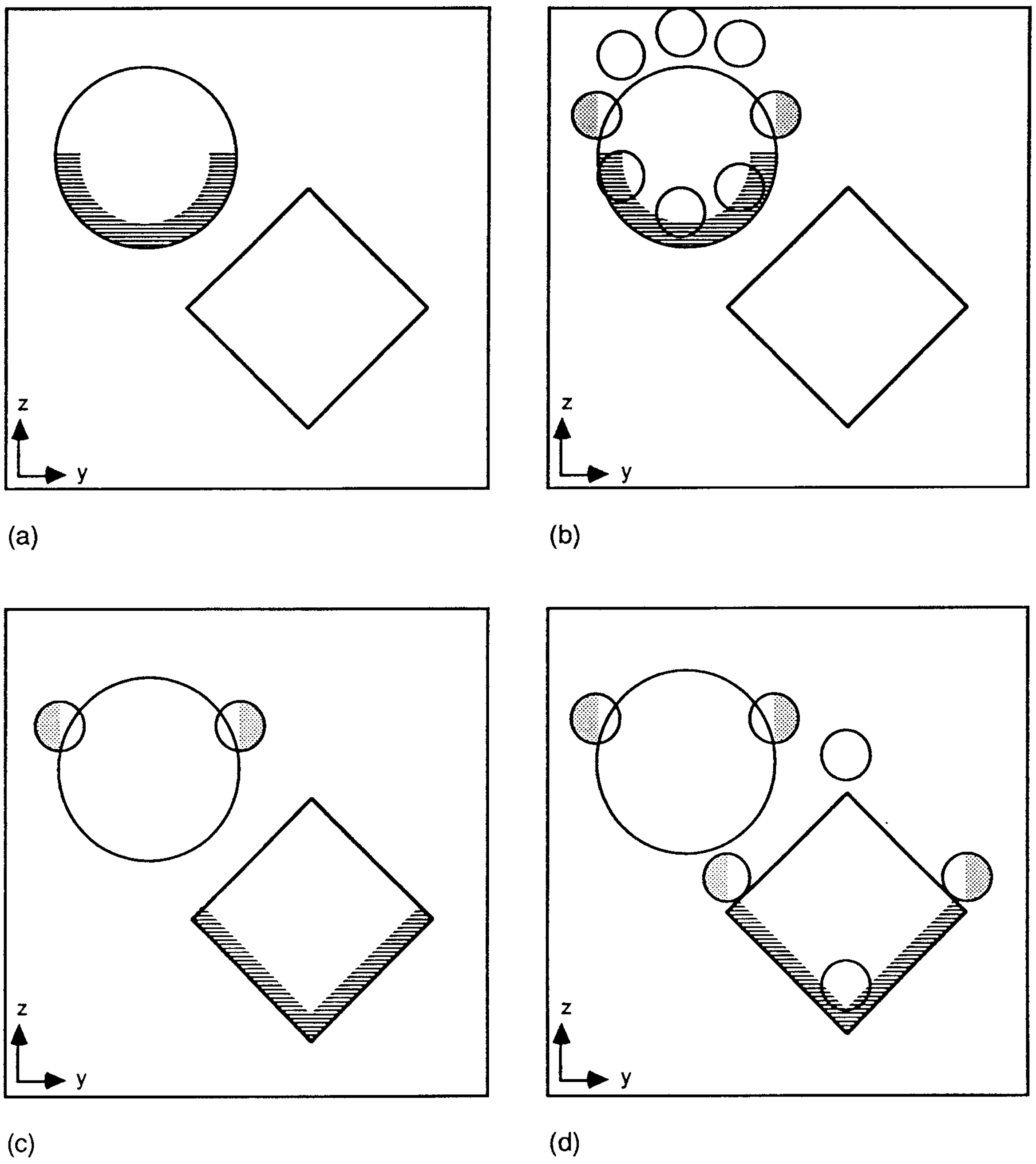

Figure 9. In the first part of the CEHD method, all objects are processed one by one. The cylinder is scan-converted in step 4 (a). The tesselation edges of the cylinder are drawn (b) in thicker dashed style translated backward (step 5-9), so as to only display the locally visible contour edges. After the z-buffer is cleared, the block's faces are scan-converted by step 4 (c). The tesselation edges of the block (which for this example include the intersection edges) are drawn in thicker style dashed translated backward (step 5-9) so as to only display the contour edges in dashed style (d) and the z-buffer is cleared again.

The first part (Lines 1-11) of the algorithm processes all of the primitives one by one and draws their 'locally' visible contours in dashed style using CEHS. The local visibility is defined here independently of the other primitives in the scene. (Figure 9).

The second part (Line 12) draws all intersection edges, still as dashed.

The third part (Lines 13-19) draws, in solid, the visible contours for the entire scene, using CEHS (Figure 10).

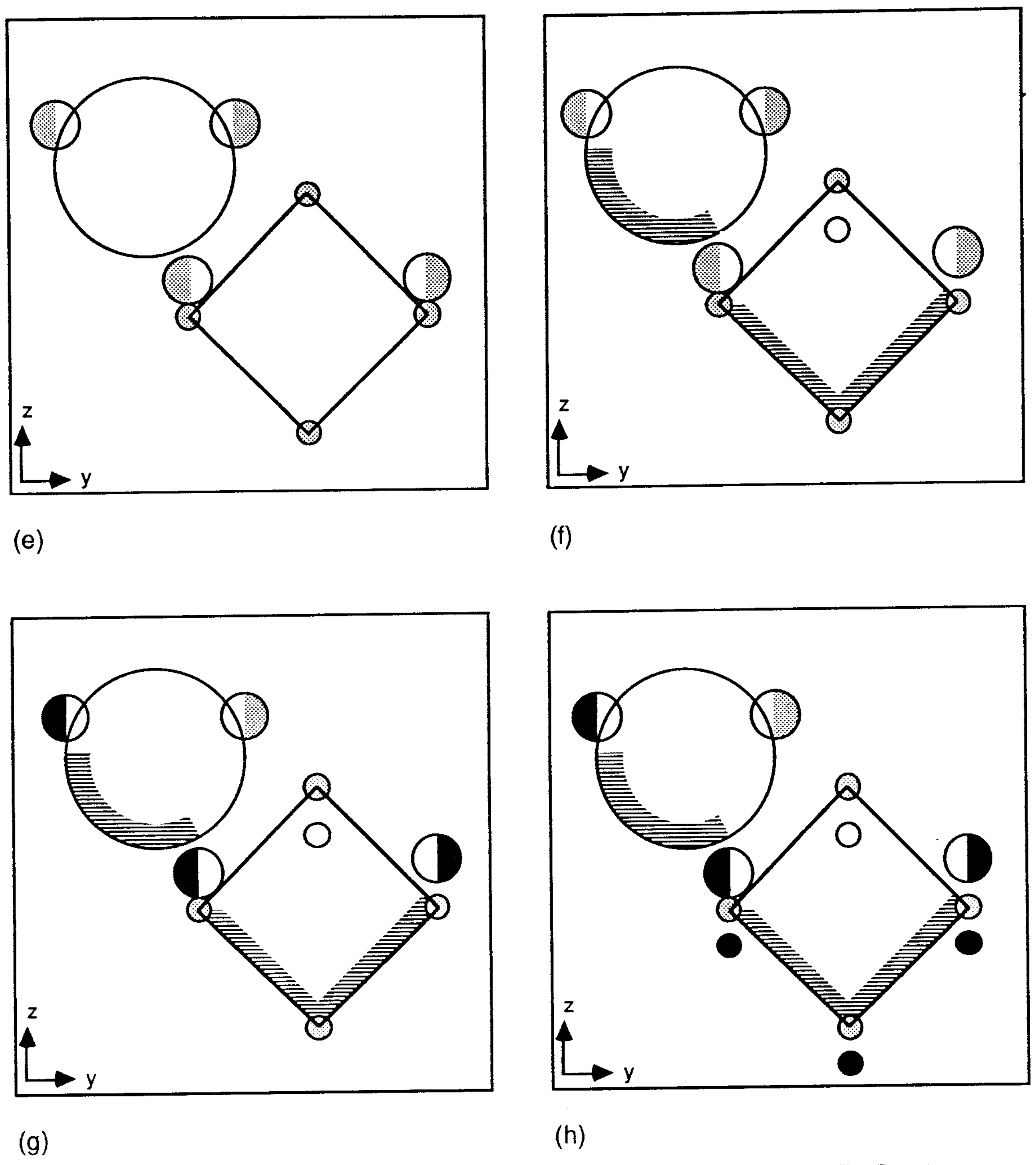

Figure 10. In the second part of the CEHD method, all of the objects in the scene are processed simultaneously. First, all intersection edges are drawn in dashed style by step 12 (e). Then all the objects are scan-converted (step 14) to fill the z-buffer (f). In steps 15-19, all the tesselation edges of all the objects are drawn in solid thick line-style displaced back (g) to display in solid line-style the visible contour edges. Finally, in step 21, all the intersection edges are drawn in solid style displaced forward (h) to display the visible intersection edges properly.

The fourth part (Lines 20-22) draws the visible intersection edges using as in TEHS.

```
1. LineStyle (Dashed);
2. FOREACH Object IN AllObjects DO
3.    BEGIN
4.    Fill (Object);
5.    Move (Backward);
6.    Linewidth (Thick);
7.    DisplayTesselationEdges (Object);
8.    Linewidth (Thin);
9.    Move (Forward);
10.   ZBufferClear ();
11.   END

12. DisplayIntersectionEdges (AllObjects);

13. LineStyle (Solid);
14. Fill (AllObjects);
15. Move (Backward);
16. Linewidth (Thick);
17. DisplayTesselationEdges (AllObjects);
18. Linewidth (Thin);
19. Move (Forward);

20. Move (Forward);
21. DisplayIntersectionEdges (AllObjects);
22. Move (Backward);
```

A drawback of the above CEHD solution is the removal of 'locally hidden' contour edges of individual primitives. (For example see the torus in Figure 13.) Although this problem can only occur with nonconvex primitives, it may require special treatment. A possible approach is to replace the temporary move backward of the CEHS technique by newly available dedicated support of contour edges selection from the (hardware) graphics pipeline. This solution requires that the application be capable of sending "polyhedron edges" to the graphics pipeline (i.e. edges associated with the two unit outward normal vectors for abutting faces). Although other alternatives have been investigated by the authors, they all require minor extensions of the functionality of the rendering pipeline.

3. Resolution

The algorithms proposed here are based on a small temporary offset (translation along the z-axis of the viewing coordinate system). A translation is necessary, because, due to inconsistent round-off errors in line drawing and in the scan conversion algorithms, one cannot assume that the depth of an edge agrees with the depth an abutting face at any given pixel. Thus depth-equality tests cannot be used to differentiate the visible edges from hidden ones in TEHS.

The **offset resolution** (i.e. the amount of translation used for this temporary offset) must be carefully selected.

Too small a translation will not consistently offset the edges from the abutting surfaces. Figure 11 (left) illustrates this effect through discontinuities in the visible lines.

Too large a translation, on the other hand, allows hidden edges to "pierce through" front faces in the vicinity of contour lines or thin walls (see Figure 11 (right)).

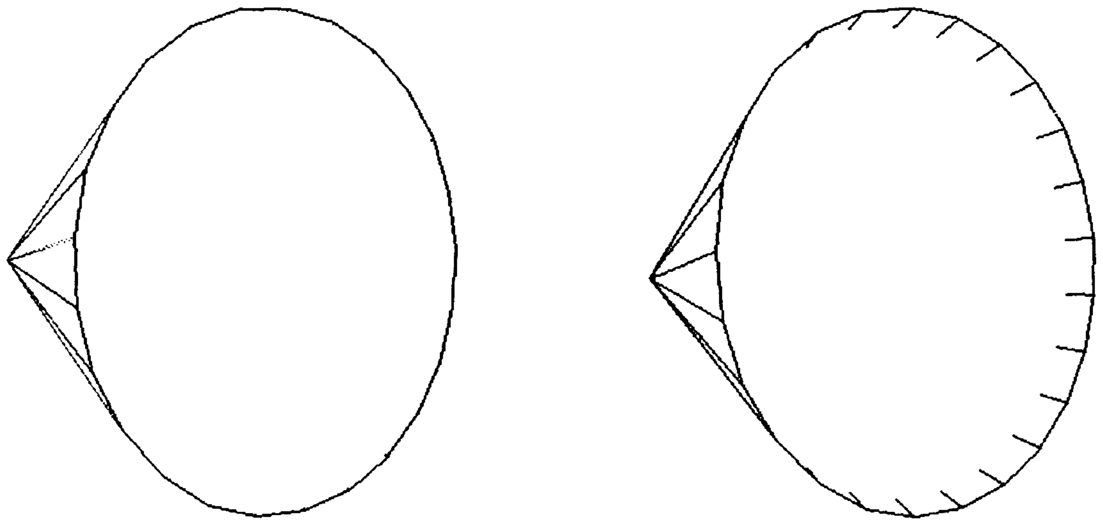

Figure 11. Errors due to a too small (left) and too large (right) transformation in z-direction.

Errors with edges parallel to the viewing plane as shown in Figure 11 can not be completely prevented if a translation in z-direction is used at all. However, the occurrence of these errors can be minimized by selecting the smallest value for the z-translation that suffices to ensure that all visible edges are in front of or on the visible front-faces of the objects.

The depth-resolution of the z-buffer is the minimum strictly positive depth difference between two values that can be stored in the z-buffer.

Face scan-conversion errors close to noncontour edges may significantly exceed the depth-resolution of the z-buffer, (Rossignac, 1991). Consequently, it may not suffice to set the offset resolution to be greater than the depth-resolution.

The best results are obtained by scaling the scene in z so as to achieve the maximum relative depth resolution and by letting the user adjust the offset resolution interactively, if necessary.

4. Conclusion

Two z-buffer techniques are presented here for improving the performance of wireframe rendering of solid models on commercially available frame buffers. The first one uses a temporary forward offset to discriminate between visible and hidden edges. The second one uses a temporary backward offset to discriminate between contour and noncontour tesselation edges. Four algorithms based on combinations of these techniques are presented. They support four popular drafting styles. The first two display all of the visible edges optionally showing hidden lines dashed, but do not differentiate between intersection and tesselation edges. The other two eliminate noncontour tesselation edges for clarity.

All four algorithms are easy to implement using widely available graphics libraries and exhibit graphics performance comparable to z-buffer shading of the scene.

Figures 12 and 13 use a simple scene made of tesselated solid primitives to demonstrate the output of TEHS and CEHD techniques.

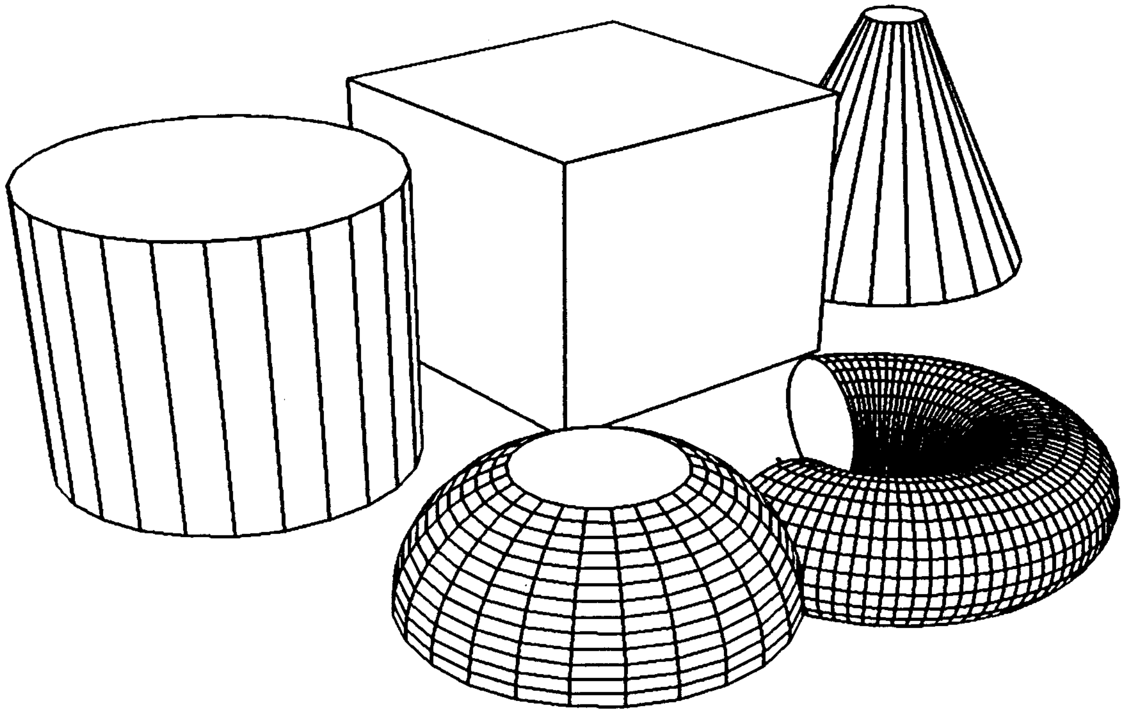
Figure 12. Example of a more complex scene rendered with hidden tesselation edges suppressed (TEHS).

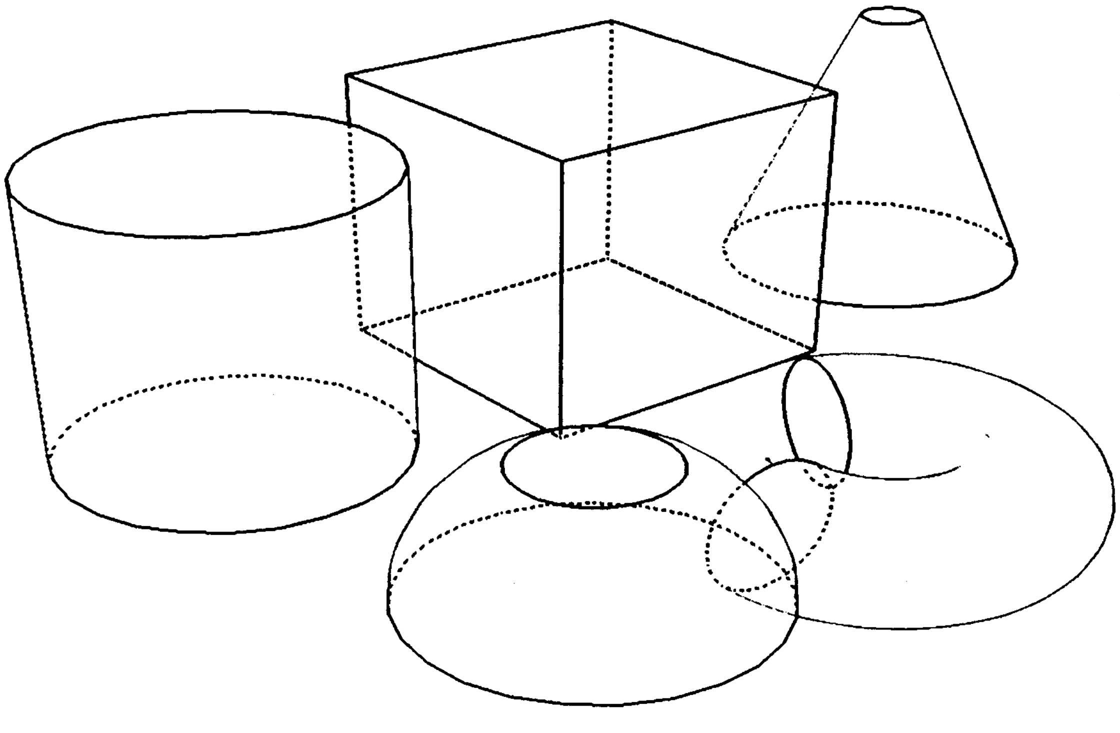
Figure 13. Example of the scene in Figure 12 rendered with hidden contour-edges dashed (CEHD).

5. References

Appel A, 1967, "The Notion of Quantitative Invisibility and the Machine Rendering of Solids", Proceedings of the ACM National Conference, pp. 387-393.

Boller, D.J., 1985, "Efficient Hidden Line Removal for Surface Plots Utilizing Raster Graphics", in *Fundamental Algorithms for Computer Graphics,* Ed. R.A. Earnshaw, Springer Verlag, Berlin, NATO, ASI Series, vol. F17, pp. 603-615.

Butland J, 1979, "Surface Drawing Made Simple", Computer-Aided Design, 11(1), pp. 19-22.

Elber G., Cohen E., 1990, "Hidden Curve Removal for Free Form Surfaces", Computer Graphics (Proceedings Siggraph '90), 24(4), pp. 95-104.

Foley J.D., van Dam A., Feiner S.K., Huges J.F., 1990, "Computer Graphics: principles and practice", Second Edition, Addison-Wesley.

Galimberti R., Montari U., 1969, "An Algorithm for Hidden-Line Elimination", CACM, 12(4), pp. 206-211.

Kamada T., Kawai S., 1987, "An enhanced treatment of Hidden Lines", ACM Transactions on Graphics, 6(4), pp. 308-323.

Li L, 1988, "Hidden-line algorithm for curved surfaces", Computer-Aided Design, 20(8), pp. 466-470.

Loutrel, P.P., 1970, "A solution to the Hidden-Line Problem for Computer-Drawn Polyhedra", IEEE Transactions on Computers, 19(3), pp. 221-229.

Rankin J., 1987, "A Geometric Hidden-Line Processing Algorithm", Computers & Graphics, 11(1), pp. 11-19.

Roberts L.G., 1963, "Machine Perception of Three Dimensional Solids", Lincoln Laboratory, TR 315, MIT, Cambridge, MA.

Rossignac J., 1991, "Accurate Scanconversion of Triangulated Surfaces", in *Advances in Computer Graphics Hardware VI,* Ed. A. Kaufman, Springer Verlag, Berlin.

Rossignac J., Megahed A. Schneider B.O., 1992, "Interactive Inspection of Solids", Computer Graphics 26(4) (Proceedings Siggraph '92).

Saito T, Takahashi T, 1990, "Comprehensible Rendering of 3D-Shapes", Computer Graphics 24(4) (Proceedings Siggraph '90), pp. 197-206.

Skala, V., 1985, "An Interesting Modification to The Bresenham Algorithm for Hidden-Line Solution", in *Fundamental Algorithms for Computer Graphics,* Ed. R.A. Earnshaw, Springer Verlag, Berlin, NATO, ASI Series, vol. F17, pp. 593-601.

Sutherland I.E., Sproull R.F., Schumacker R.A., 1974, "A Characterization of Ten Hidden-Surface Algorithms", ACM Computing Surveys, 6(1), pp. 1-55.

Williamson H., 1972, "Algorithm 420 Hidden-Line Plotting Program", CACM, 15(2), pp. 100-103.