9 Accurate Scanconversion of Triangulated Surfaces

Jarek R. Rossignac

ABSTRACT Scanconverting a planar face produces depth-values for pixels totally or partly covered by the projection of that face. State-of-the-art hardware-supported scanconversion techniques use subpixel adjustment and extended precision calculations to achieve an acceptable depth-accuracy despite numeric round-off errors. Unfortunately, this depth-accuracy only holds for the interior pixels of the face. During the scanconversion of the boundaries of polyhedral solids or of the tesselations of curved surfaces, significantly larger depth-errors may occur at pixels traversed by the projection of the bounding edges. These errors are due to the use of the wrong surface equations resulting from an erroneous classification of pixels with respect to the projections of faces. They may lead to logical mistakes of serious consequences for hidden-surface removal and for solid-modeling applications. To address this problem, a new scanconversion technique is presented, which exploits surface data and face/face adjacency information to infer face-projections. For simplicity, the exposition is confined to triangular faces of manifolds, where each edge is adjacent to two triangles. At pixels covered by the projection of an edge, the surface depth computed in the standard manner is compared to the depth of the surface supporting the adjacent triangle. Pixel classification is obtained by taking into account the result of this comparison and the orientations of both faces.

9.1 Introduction

The race for graphics performance has led to the development of a variety of hardware-supported graphics architectures [1]. Most common ones include a rasterizer, which receives 3D vertex coordinates in image space and associated color values; and which produces interpolating depth and color values for all of the pixels entirely or partly covered by the projection of the face bounded by these vertices [15, 34]. This process will be called scanconversion throughout this paper. For simplicity of exposition, and as it is often done in practice, we consider that faces are split into triangles before scanconversion. Efficient face triangulation is addressed in [3]. Figure 9.1 illustrates the scanconversion terminology.

The depth-values computed by the rasterizer are often used for selecting at each pixel the visible faces, i.e. the faces closest to the viewpoint. The selection is made possible by associating with each pixel a z-buffer memory holding the depth of the closest surface point encountered so far for this particular pixel [9].

Additional pixel-memory may be exploited for storing logical values or several depth-values and lead to important new applications of the graphics pipeline. Examples of such applications include correct transparency [23], shading from CSG [13, 16], or solid cross-section and interference visualization [26], which are visited in Section 2. These applications exploit topological properties of the model and rely on the accuracy of the depth computation at each pixel. Section 3 demonstrates three types of scanconversion

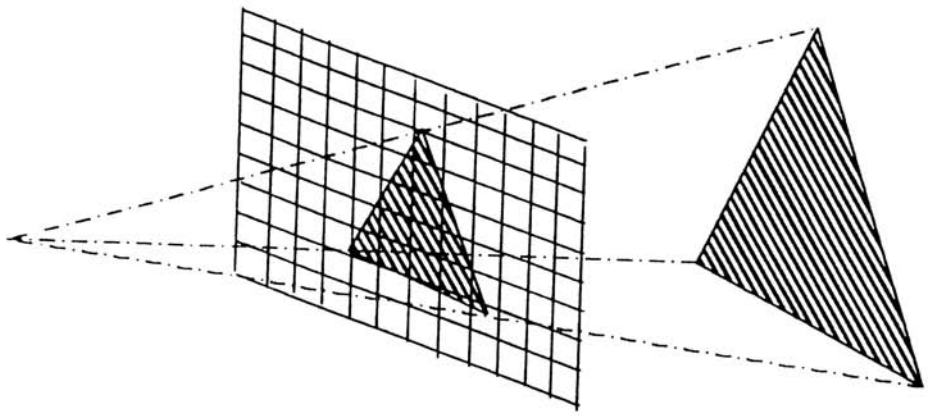

FIGURE 9.1. Face scanconversion: Given the location of the viewpoint (left) and of the screen (center) in the model-space, perspective geometric transformations and clipping are applied to compute the projection of a face (right) onto the screen. Pixels, which correspond to rectangular portions of the screen, that are (partly) covered by this projection are visited during the scanconversion process in an order prescribed by the scanconversion algorithm. At each pixel, the depth of the corresponding surface point and the color components are computed and used to conditionally update pixel memory. The viewing direction is usually associated with the z-axis and z-values are often called depth.

errors that may jeopardize the logical assumptions underlying the algorithms used for these applications. Previously known solutions for reducing errors of the first two types: depth-calculation round-off and classification inconsistency are outlined. Section 4 focuses on the third type, the edge round-off error, which has not been previously addressed. A simple solution, based on depth-comparisons between the surfaces of adjacent triangles is presented.

9.2 Applications

This section reviews several applications that exploit scanconversion data and pixel memory. It motivates the rest of this paper by summarizing the requirements imposed by the underlying algorithms on the scanconversion.

9.2.1 Visible Surface Selection

Z-buffer pixel-memory is usually exploited for selecting, at each pixel, the surface-points that should be displayed. The selection process, sometimes called hidden-surface removal, compares the depth produced by scanconversion with previously produced depth values stored in the z-buffer. Errors in evaluating the depth of two nearby surfaces may lead to incorrect selections and to the display of hidden portions of surfaces.

Depth-calculations are performed with fixed precision [17], which depends on the overall z-scaling factor in the world-to-screen mapping transformation. For example, let point A of face Fa lie slightly in front of point B on face Fb, where both A and B project onto the same pixel. The depths of both points will be undistinguishable if their difference is sufficiently small so that they are both rounded to the same z-value. In that case, the color displayed will depend on the order of surface traversal and on the particular depth-comparison test used (less-than or less-or-equal). The result is thus predictable. Unfortunately, no matter how close, two points may produce two different rounded z-values. For example, 135.49, the depth of A, may be rounded-off to 135 and 135.51, the depth of B, to 136. Since this situation typically occurs at some pixels and not at others, the overall ordering information resulting from depth comparisons is inconsistent with a geometric ordering of two planar faces in depth at each pixel. Images where the colors of two almost coplanar faces are mixed in seemingly random interference patterns are characteristic of this rounding effect.

It is commonly accepted that contemporary z-buffer rendering will not reliably select the visible face among faces that are closer to each other in depth than the z-resolution. On the other hand, there is an implicit assumption that the visual feedback obtained by graphics hidden-surface removal is consistent with the displayed geometry, within this tight tolerance of the depth-resolution. Specifically, applications strive to set the maximum z-scaling factor for a particular scene assuming that when the depth of the two points, A and B, are further apart than the z-resolution (or some fixed and small multiple of it), the ordering will be always correct. Failure to meet this assumption makes the reliability of z-buffer techniques questionable for the visual inspection of geometric models.

In conclusion, visible surface selection algorithms require that the maximal depth error encountered during the scanconversion of a surface be tightly bounded and defined in terms of the resolution of the z-buffer.

9.2.2 Sorting for Transparency

In order to correctly render a set of translucent surfaces, it is necessary to order them along the viewing direction, so as to account for the higher contribution of the foremost surfaces. A depth-interval buffer (double z-buffer) may be used to achieve this ordering [23]. A possible approach is briefly outlined below, skipping details for sake of simplicity.

Using the maximum-depth selection test (i.e. a greater than depth-comparison test), one computes in the z-buffer Z1 the depth of the furthest surface and initializes the intensity buffer I1 with the color contribution of that surface. Then the process described below is repeated until all surface layers are processed. (The term layer refers to a traversal of the surface as one moves along the viewing direction.)

The depth of the z-buffer Z2 is reset to zero and the color in the intensity buffer I2 is reset to background. The entire scene is scanconverted again. Z2 and I2 are only updated when the depth computed by the scanconversion falls between Z1 and Z2. The result of this pass produces in Z2 and I2 the surface information for the surface layer directly in front of the previously processed layer. The resulting color information is then merged with the content of I1 according to prescribed surface translucency properties.

Depth-errors produced during scanconversion will result in wrong surface ordering and thus in incorrect pictures. Consequently, this approach to transparency rendering imposes on scanconversion the same constraints as the visible-surface selection algorithm discussed above.

9.2.3 Shading from CSG

Models of solids are often constructed in CAD systems by combining primitive shapes through regularized Boolean operations. The combining expression may often be captured in a CSG graph [28, 35]. Several techniques have been proposed for producing shaded images of the resulting solids directly from CSG graphs. They bypass the difficult and expensive boundary evaluation process, which computes trimming curves of the solid's faces by intersecting primitives' surfaces. Ray-casting amounts to scanconverting all of the faces of a model simultaneously. Sorting and coherence techniques may be used to improve performance [2]. At each pixel, the face-point closest to the viewer is displayed. Ray-casting may be adapted to CSG by replacing the computation of the closest intersection point along a ray with the Boolean merge of line intervals [30]. Optimization techniques for software ray-casting on CSG with polyhedral primitives are discussed in [6-8, 18, 20]. A customized hardware solution is offered with the Ray-Casting Engine (RCE) prototype developed at Duke and Cornell [12]. The RCE combines a series of parallel ray-primitive classifiers with a matrix of parallel interval-merging units to which the CSG structure is mapped. Although the RCE is currently limited to quadric half-space primitives, its extensions to cubic implicit half-spaces are underway. Other hardware-based solutions are discussed in [21, 22].

Software scanconversion can easily be combined with point-in-solid tests against CSG graphs to produce shaded images of CSG solids [27]. To further improve performance, scanconversion hardware has also been applied to CSG rendering, as proposed in [19]. Approaches in [13, 16] use a generate-and-test paradigm illustrated in Figure 9.2. Tentative surface points are produced through scanconversion and stored in an auxiliary z-buffer, Z2. Then, a more complicated selections than hidden-surface removal is performed, because tentative points that are not on the boundary of the CSG solid must be discarded.

Both approaches in [13] and in [16] test the tentative points for inclusion in a product of the disjunctive formulation of the CSG expression. This test is performed by establishing whether the tentative point lies inside all other positive primitives of that product and outside all negative primitives of that product. This test is implemented through scanconversion as follows. Given a surface-point, r, whose depth is stored in the z-buffer of some pixel, x, and given a solid primitive, C, the point-in-primitive test establishes whether r lies inside C or not by scanconverting all of the faces of C and by toggling the

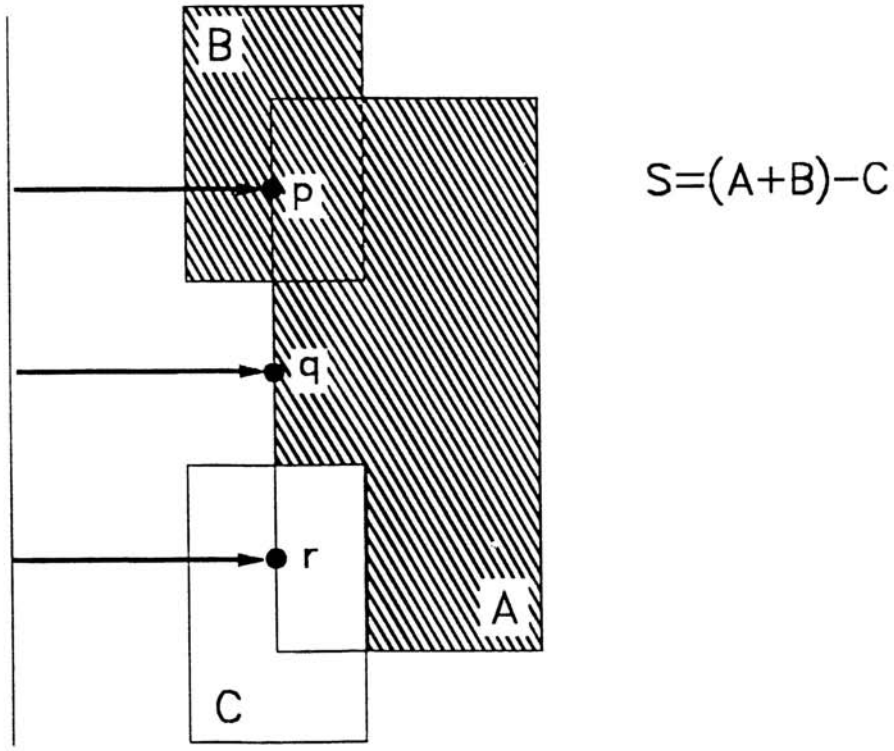

FIGURE 9.2. Rendering from CSG: Among the points p, q, and r on the boundary of primitive A, only q is displayed. Point p is eliminated by a standard hidden-surface test, because it lies behind some point on the boundary of B. Point r is eliminated by a point-in-primitive test, because it lies inside primitive C.

parity-bit associated with pixel x each time the surface-point of C that projects on x has greater depth than r. The approach is illustrated in Figure 9.3. This point-in-primitive test against primitive C is performed simultaneously for all points stored in the memory of the different pixels. There is no restriction on the nature and shape of C, except for having a valid description of its boundary suitable for accurate scanconversion.

The above technique is based on a topological property of the primitive's boundary that ensures the classification alternating rule: Any line crosses the primitive's boundary an even number of times and the classification alternates at each crossing as one moves along the line. When primitives' boundaries are bounded (i.e. not infinite) and when classification at infinity is known (usually outside), classification of any point r may be obtained by casting a half-ray from r to infinity and by counting the parity of the number of times the boundary of the primitive is crossed by the half-ray. The half-ray is cast along the viewing direction, and thus, the parity of the number of its intersections with the boundary of a primitive C may be computed by scanconverting the faces of C and counting only points that lie behind r.

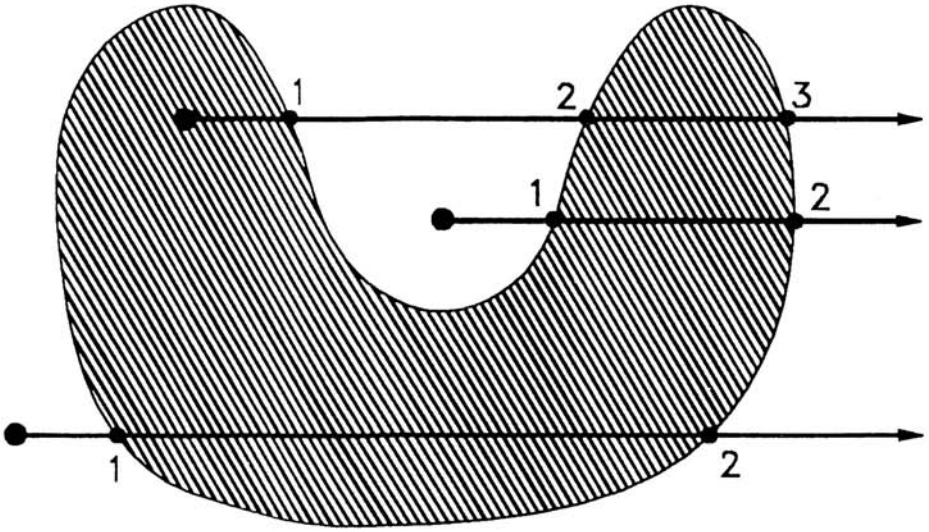

FIGURE 9.3. Point/primitive classification: Among the three points (large dots), only the top one lies inside the hatched area. Notice that there is an odd number of boundary points of the hatched area behind (i.e., to the right of) that top point. (We assume that the viewer is positioned on the left.)

In conclusion, graphics hardware-assisted point-in-primitive tests cannot be trusted unless scanconversion preserves the alternating parity rule.

In addition to its dependency on the consistency of scanconversion with the parity rule, direct rendering from CSG is very sensitive to scanconversion depth round-off errors–and this, much more that hidden-surface removal or transparency–for the following reasons. Often, the faces of CSG primitives overlap. Round-off errors occurring during the world-to-screen transformations combined with scanconversion round-off may result in significantly different z-values for two logically identical surfaces at the same pixel. Fail to recognize that points with slightly different depth-values lie on the same surface results in arbitrary decisions regarding point-in-primitive classification, and thus lead to incorrect pictures. Small discrepancies may be caught using a depth-tolerance during the scanconversion. For example, in [29] the test $Z2 \leq Z \leq Z3$ is preferred to $Z2 \leq (Z - \epsilon) \leq Z3$. (Here Z denotes the depth generated by scanconversion and Z2 and Z3 are the corresponding contents of two depth-buffers.) The use of such a tolerance amounts to testing a point slightly behind the scanconverted point, as shown Figure 9.4.

The ϵ tolerance must be chosen so as to exceed depth-calculation errors during scanconversion. On the other hand, too large a tolerance will have side-effects that we call: ϵ-regularization. Standard 3D regularization is a process that removes zero-thickness walls

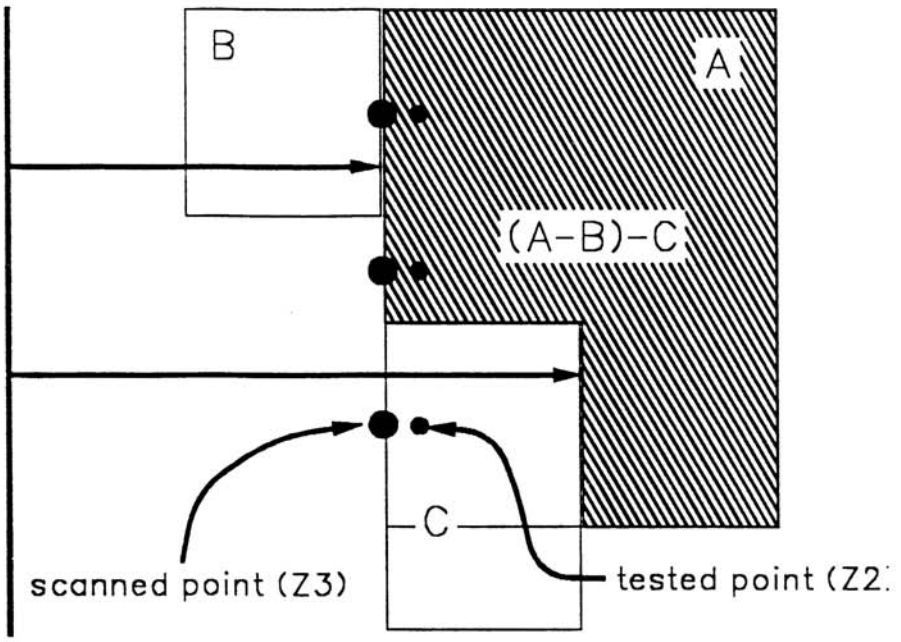

FIGURE 9.4. Coincident faces: Large dots represent points on the boundary of A. The viewpoint is on the left. Since the front face of A coincides with a face of B and a face of C, standard depth-comparisons may not be used for point-in-primitive testing. Instead, a tolerance is used, which simulates the effect of testing a point slightly behind the front face of A against primitives B and C. These virtual points are shown as smaller dots.

and cracks from a solid model's representation [24]. (A pointset is regularized if it is equal to the closure of its topological interior. Representations of solids used in contemporary CAD systems are regularized.) ϵ-regularization is a special case of an opening of the solid by a line segment of length ϵ aligned with the z-axis. (An opening is a morphological operation used in image processing and porosity analysis for removing noise and small details [33].) To cope with the limited resolution of graphics implementations, we use ϵ-regularization understanding that it may remove object's features that are thinner that ϵ in depth, as shown Figure 9.5. These features not only contain thin walls and cavities, but also portions of the model that are close to silhouette edges. ϵ-regularization is consistent in spirit with the discrete nature of graphics–provided that ϵ is small.

In conclusion, ϵ-regularization is important for correctly processing singular, or nearly singular, geometric situations. However, to limit the side-effects, it is important to maintain as small a value for ϵ as possible and thus to control the possible extent of scanconversion depth-errors.

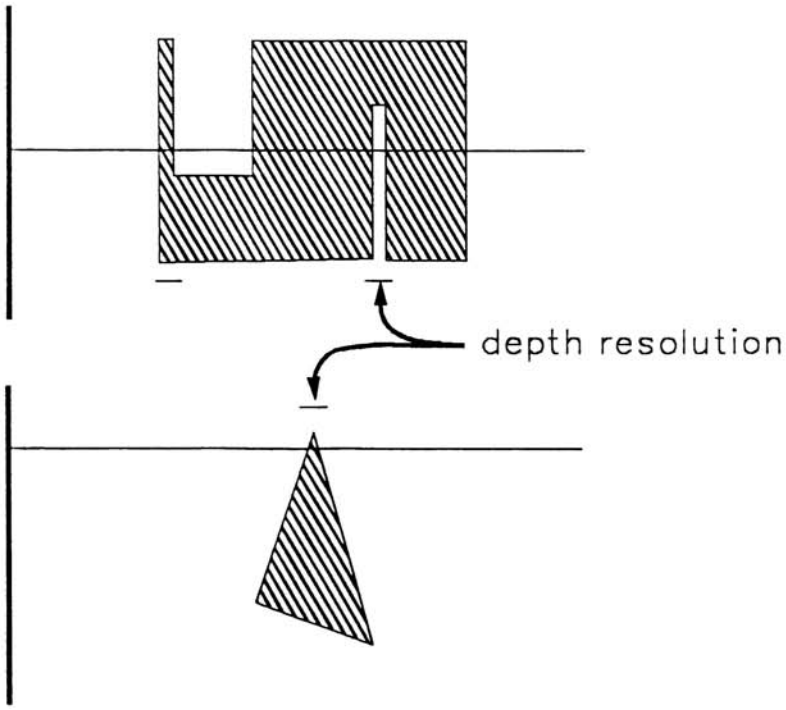

FIGURE 9.5. Cracks and walls: The use of a tolerance for depth-tests eliminates narrow constrictions, cracks, and chamfers silhouettes.

9.2.4 Cross-Section and Interference Visualization

Cross-sections through solids are important for visualizing complex assemblies and composite solids. Shaded images of such solids with portions removed (i.e. clipped away) may be produced automatically by combining the effect of the standard clipping planes with a capping (i.e.: cross-section filling) operation. To avoid the expensive computation of geometric intersections with the cross-sectioning planes, capping is done automatically by the graphics hardware using scanconversion and masking [26]. Tentative points on the cross-section planes are written into the z-buffer and then tested for inclusion in all of the solids by scanconverting these solids and performing the parity-based point-in-solid inclusion test discussed above. An extension of this capping technique for the display and automatic detection of interferences is also presented in [26]. It uses parity-preserving scan-conversion to perform the necessary point-in-primitive tests. A sufficient but not necessary condition is used to automatically establish that subsets of an assembly are free from interferences and to position a cross-sectioning plane at the beginning of the first potential interference encountered along a given search direction. Realtime implementations

of such automated facilities for checking the validity of a design prior to sending it to manufacturing are key factors for the designers' productivity. The limited pixel-resolution may be compensated during interference detection by using a thicker line-width while drawing the faces' outlines, so as to guarantee a sufficient–although not necessary–condition for disjointness. However, parity errors and scanconversion inaccuracies lead to pixel errors in the capping and to wrong decisions in the interference detection algorithms, and, if not controlled, make these hardware-based solutions unreliable.

9.3 Scanconversion Errors

To scanconvert a triangular face, the rasterizer typically receives, for each one of the three vertices, their x and y coordinates in the screen coordinate system and a series of scalar values, such as the depth (i.e. oriented distance to the viewer) or the color information (e.g.: red, green, and blue components). Most graphics architectures use between 24 and 32 bits per pixel for storing the depth values and 8 to 16 bits per pixel for storing color components.

For all of the pixels completely covered by the projection of the face and for some of the pixels traversed by the projection of the edges, scanconversion computes interpolated values of the depth and of the three color components. A detailed discussion of the interpolation process may be found in [11]. Numerous architectures for parallel implementations of scanconversion have been proposed (see [10, 32] for examples).

For the sake of efficiency, the computation of interpolated values are performed incrementally from one pixel to the next, using additions [14]. Since a triangular face is convex, the projection of the face on the screen may be decomposed into a series of adjacent spans of pixels. Spans are usually constructed of consecutive pixels in the horizontal direction. The first pixel of a span is defined by the leading edge of the triangle in the direction of the spans. The last pixel is defined by the trailing edge. Thus, scanconversion must establish which are the leading and trailing edges for each span and then compute at which pixels the span beginning and at which pixel it ends. Suppose the spans are accessed top-down in the y direction. The top vertex defines the starting pixel. The slopes (dx/dy) of the two adjacent edges to that pixel may be used for the incremental calculations of the span starting and ending pixels.

The linear interpolations of the scalar values (here we mention only depth and color components) are computed by first calculating their slopes (i.e., rates of change, or absolute variations, for unit displacements along the x and y direction, or for the displacement by one unit in y along an edge), and then by adding these constant slope-values when going from one pixel to the next one in a span and when going from one span to the next along the leading edge. These slopes may be easily precomputed from the vertex coordinates and from the associated scalar values.

In conclusion, scanconversion must establish, for each span, the starting and ending pixels as well as the values of the depth and of the three color components. Then, the precomputed slopes are used to incrementally produce interpolating values for the consecutive pixels of the span. Scanconversion errors arise from mistakes in establishing the

starting and ending pixels for a span, from errors in accurately computing the starting scalar values for each span, and from round-off errors in the slopes, the computed depth, and the color components. These errors and their consequences are discussed in the following subsections.

9.3.1 Depth-Calculation Round-Off

The limited precision fixed-point representation of the starting z-value and of the z-increment yield cumulative round-off errors. The problem is usually addressed by providing a sufficient resolution for storing the depth and increments so as to guarantee that the maximum round-off error is less than the resolution of the z-buffer. For example, if N bits are used for storing the depth information in the z-buffer, and if the screen is K pixels wide and K pixels large, using N+log(2K) bits for computing the running depth and the increment guarantees that the final depth is computed to the full resolution of the z-buffer.

9.3.2 Classification Inconsistency

Pixels traversed by the projection of an edge are only partly contained in the projection of the scanconverted triangle. Processing only pixels that are entirely covered by the face's projection would result in gaps between adjacent faces (see Figure 9.6) and could entirely miss faces whose projections are thinner than a pixel's width.

Processing all pixels even partly covered by the face would solve the gaps and the skinny triangle problems, but could jeopardize the parity calculations needed by many applications, as discussed earlier. If only one of the adjacent faces to a given edge is front-facing, the parity will be correct at the pixels traversed by the edge's projection provided that their classification be consistent for both faces. Such edges are called profile or silhouette edges. On the other hand, if the edge is not a silhouette edge, i.e., if both faces are either front-facing or back-facing, as shown in Figure 9.6, pixels covered by their common edge will be counted twice, yielding the wrong parity.

The problem is commonly addressed by using the center of the pixel to test its containment in the face's projection. Only pixels whose centers lie inside the face's projection will be visited during the scanconversion of the face, see Figure 9.7. Consequently, pixels covered by the projection of a non-silhouette edge will belong to only one of the two faces abutting on that edge. The resulting parity will be correct and there will be no gaps, but skinny faces may disappear.

This approach imposes additional constraints on the scanconversion algorithm, which must establish, for each scanline, which are the pixels whose centers fall between the projections of the leading and the trailing edges. Since floating point calculations are avoided during scanconversion, fixed-precision techniques must be applied for tracing the edge [5].

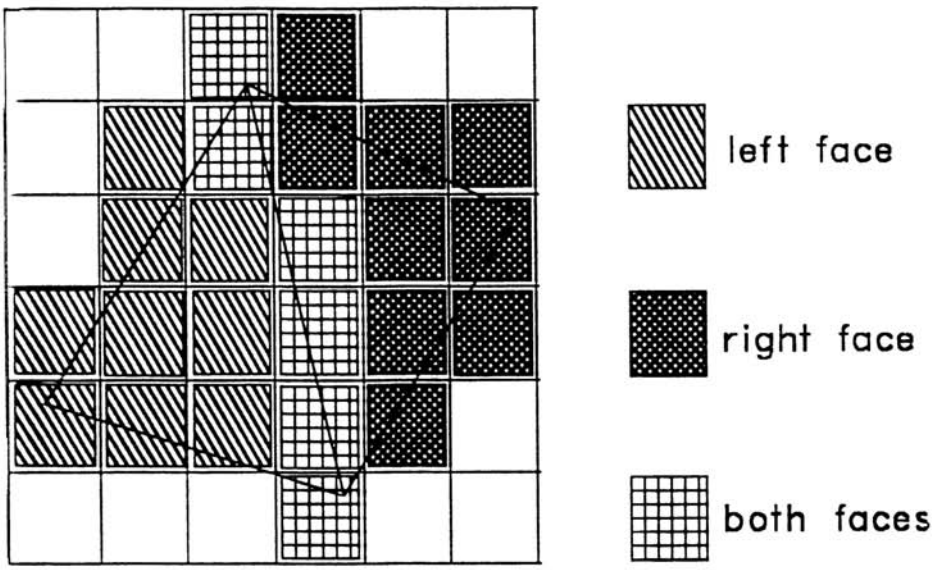

FIGURE 9.6. Shared pixels: Not processing pixels that are partly covered by both faces can produce cracks in the image. Processing all pixels partly or fully covered by a face's projection will produce a wrong parity for pixels covered by the shared edge.

9.3.3 Edge Round-Off

The technique outlined in the above subsection guarantees correct parity for the visited pixels when scanconverting the boundary of a solid or a tesselation of a curved surface. Pixel-classification may be performed by using a modified Bresenham's algorithm to establish the first pixel on the left (respectively right) of a leading (respectively trailing) edge for each span. In broad lines, the modified algorithm uses the edge's slope (dX/dY) and initial error (horizontal distance between the edge and the center of the pixel) and updates the error for each scanline. The error is smaller than 1.0 and is either negative (for leading edges) or positive (for trailing ones). Since the initial error, the slope, and the error increment are rounded off to the nearest fixed-point value, this calculation is not precise, and pixels whose center is close to the edge may be misclassified (see Figure 9.8). Note that this problem will not disappear even if floating point arithmetic is used, because it also has limited accuracy.

It may seem that such small errors are of no consequences, because misclassifying a pixel in such a manner amounts to perturbing slightly an edge in the real model or to moving slightly the object in the horizontal direction. And indeed, for pixels that were misclassified with respect to silhouette edges, the resulting image will be one of a slightly modified solid, but the z-values computed during scanconversion will be within the prescribed tolerances.

Unfortunately, misclassifying pixels with respect to non-silhouette edges leads to more

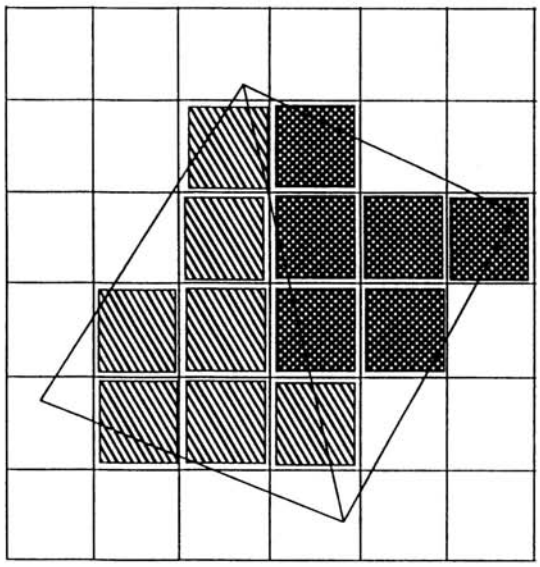

FIGURE 9.7. Pixel classification based on its center: Only pixels whose center lies inside a face's projection are visited. There is no gap between faces.

severe errors in the depth-calculation, because the wrong surface equation is used to compute the depth at these misclassified pixels (see Figure 9.9).

A two-dimensional view in Figure 9.10 illustrates these two situations. It also demonstrates that the depth-error at misclassified points may be significantly larger that the distance between the center of the pixel and the projection of the edge.

In conclusion, one cannot rely on edge tracing for accurate pixel classification, and arbitrarily small errors in pixel classification may result in arbitrarily large depth errors.

Of course, in practice, the slope of the plane is rounded and the distance between the pixel's center and the edge is bounded. Let M be the maximum slope (dZ/dX) that can be represented for a forward-facing face. Let E be the maximum distance along the X-axis between the center of the misclassified pixel and the edge. The maximal depth error at that pixel is 2ME.

In practice, E is 1/16 or larger, because the edge's vertices are rounded off to the nearest subpixel [31]. The value of M may be typically as large as the depth-buffer's span (for example $2 * *32$). Consequently, the maximal depth error may be as large as $2 * *29$.

Our experiments have demonstrated that very large depth-errors persistently appear along edges bounding a face with a steep slope. As a consequence, hidden objects appear to pierce through faces in front of them at isolated pixels along the edges, even though the distance separating the hidden objects from the visible surface considerably exceeds the depth resolution. Furthermore, sorting and point-primitive classification is unreliable, because ϵ-regularization cannot trap such big errors for acceptable values of ϵ.

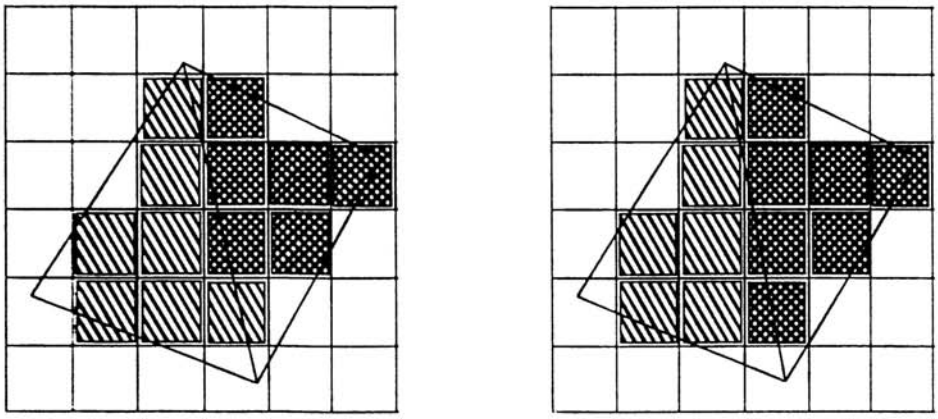

FIGURE 9.8. Round-off error for edge projections: A slight perturbation of the location of the bottom vertex (due to round-off) changes the classification of the bottom-right pixel.

9.4 A New Approach to Edge Round-Off

In this section, we describe a new approach, which eliminates the depth-error resulting from the misclassification of pixels with respect to the projections of edges.

9.4.1 Overview

The overall approach may be summarized as follows. The face F is a subset of a planar surface S. The extent of F (i.e.: the triangle) is delimited by edges Ei (i=1,2, or 3) that are the intersection of the above plane with planes supporting adjacent faces. There are three such planes, Si (i=1,2, or 3), one per edge. Using the outwards orientation of the three neighboring faces and the concavity of the three edges, one can define an unbounded convex polyhedral volume V such that F is the set-theoretic intersection of S with V. V may be expressed as the intersection of three half-spaces, each bounded by one of the Si planes. Edge convexity and face orientation may be used to establish whether to use a particular half-space or its complement for the corresponding face.

We propose to use the description of the volume V, in terms of the three half-spaces to perform pixel/face classification.

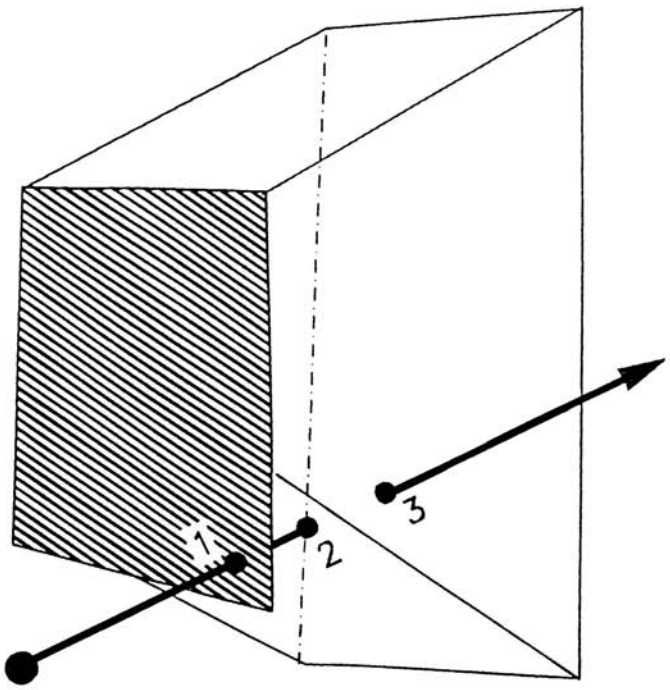

FIGURE 9.9. Using the wrong surface equation: The pixel (large dot on the left) was misclassified and is assumed to fall in the projection of the dashed front face. Therefore, the depth stored at this pixel will be that of point 1, instead of the correct depth of point 2.

9.4.2 Signs of Half-Spaces

The sign (orientation) of the half-spaces is viewpoint-independent and may be precomputed by testing whether the corresponding edge is convex or not. For example, when all the three edges that bound triangle T are convex, the volume defining F is the intersection of the three half-spaces that lie on the interior side of the corresponding faces. For concave edges, it suffices to use the complemented half-space, or equivalently, to invert the direction of the outward normal. This is illustrated in Figure 9.11.

The rule of using the complement of the half-space for concave edges may be derived from the following observation: face F must lie inside the resulting half-space.

9.4.3 Pixel Classification

In order to exploit the results developed above, we must be able to classify, during scanconversion, the points on S with respect to V. Since V is the intersection of planar halfspaces, it is convex (although it may be unbounded).

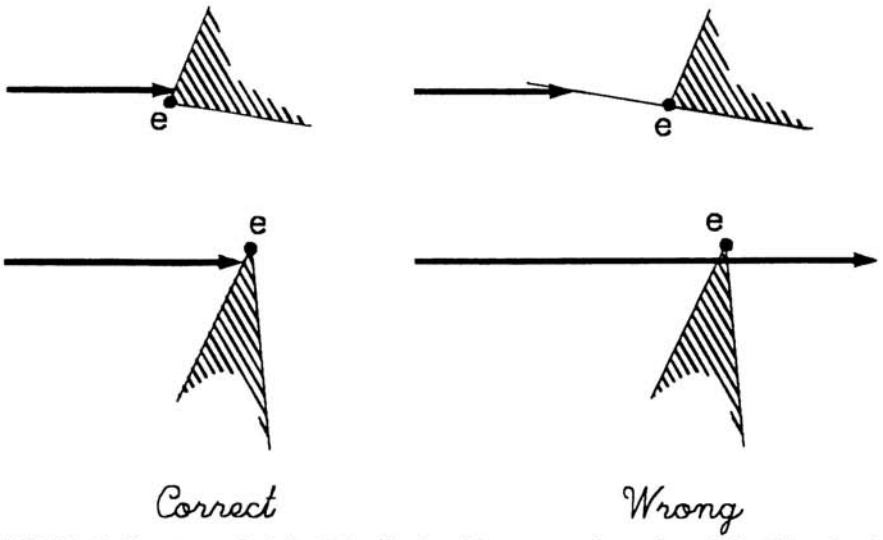

FIGURE 9.10. Two types of pixel misclassification: The top row shows that misclassifying the pixel against edge e will result in using the wrong surface equation. The error is proportional to the difference in the slope of both surface and to the distance between the pixel's center and the edge. The bottom row shows that the same misclassification with respect to a silhouette edge results in a misclassification with respect to an entire layer (or the whole solid) and produces a depth value that corresponds to a slightly translated layer.

Note that a point lies in a convex body if it is behind the furthest front-facing half-space and if it lies in front of all the back facing half-spaces. The property is illustrated in Figure 9.12.

This property leads to the following approach: while scanconverting F, keep track of the sign of the depth differences, $Z - Z_i$, between the surface S of F and the three other surfaces, Si. (Z is the depth of S and Zi are the depth values for the other surfaces.) Each depth-difference is multiplied by a coefficient Ki that takes values of 1 or -1, depending on the forward/backward orientation of the faces and on the concavity/convexity of the corresponding edges. The result is a function that is positive over the projection of the face. Figure 9.13 illustrates the change of depth-ordering between two surfaces as one moves across their intersection line during scanconversion.

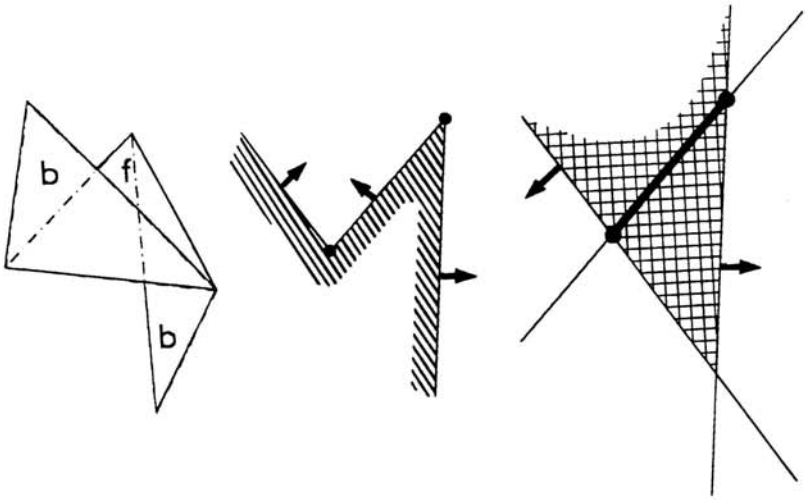

FIGURE 9.11. Half-space orientation for the clipping volume: For simplicity, only two of the neighboring triangles are displayed together with the forward-facing triangle marked f (left). They are both back-facing in the 3D picture. The front-most (backward-facing) triangle has a concave edge with f. The other one shares a convex edge with f. The central figure shows a 2D view corresponding to a cross-section through the three faces. The line-segment in the middle represents a cut through triangle F. The other two lines represent the other two triangles. The two dots represent one concave and one convex edge. Hatching defines the inner-side of the boundary, i.e.; the inside of the solid bounded by the three faces. Arrows represent the outward normal. Note that the left dot correspond to a concave edge and the right dot to a convex one. The figure on the right represents a cross-section through the volume V (hatched) and the cross-section through face f (thick line). Note that V lies inside the half-space of the face sharing a convex edge with f and inside the complement of the other half-space, which corresponds to the concave edge.

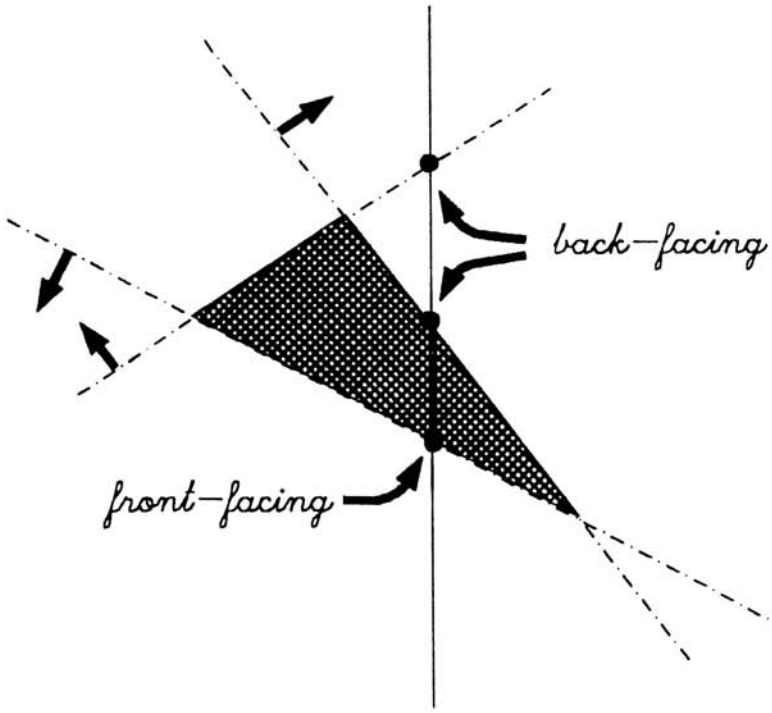

FIGURE 9.12. Pixel classification against intersection of half-spaces: The dashed area corresponds to the intersection of three convex half-spaces. Their outward normals are indicated by arrows. The vertical line corresponds to the viewing direction. Points inside the area are behind all of the the front-facing half-spaces and in front of all of the back-facing ones.

The three depth-difference functions may be computed incrementally at each pixel by adding a constant increment in X or Y, or along an edge. Each increment is the difference between the increment (i.e. slope) for the corresponding surface Si and the depth-increment for S.

The problem remains to locate the first span, i.e. the first row of pixels where these three functions are positive, and then to locate the beginning and the end of each span for the successive rows. Linear programming seems an overkill here, especially because we know the approximate location of the three vertices. It suffices to estimate how much error, in screen coordinates, may have been produced when computing the projection of the vertices. If we assume–and this is a reasonable assumption because the surface coefficients are derived from vertex data–that this error is always less than a pixel, it suffices to visit pixels entirely covered by the projection of the face as usual and add a special test for the pixels covered by the projections of the edges. (Pixels covered by the edges' projection will only be processed if the three functions are positive.)

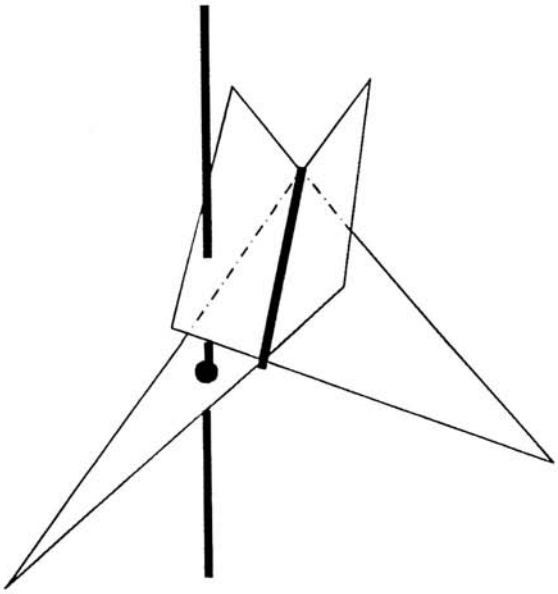

FIGURE 9.13. Classification based on order: On the left of the thick edge (intersection line between the two surfaces) the left-most face should be displayed. (Both triangles are front-facing and the edge is convex.) The viewing direction is indicated by the thick vertical line. On the right, the order of the surface-depth changes and the other face should be displayed.

9.4.4 Adjacency Information

Accessing the surface information of neighboring faces during graphics requires a datastructure which captures face/face adjacency information or an extended graphics representation for triangles.

The winged-edge datastructure [4] or the ELAG structure proposed by the author for storing adjacency information between triangles [25] are examples of very simple datastructures that provide easy access to the three auxiliary vertices. (ELAGs associate with each triangle a sorted set of references to the three neighboring triangles.)

Alternatively, the graphics format for triangles can be extended to include the normals to the three neighboring triangles. This solution ensures proper scanconversion of each triangle while maintaining the simplicity of the graphics dataflow architecture, where the descriptions of triangles are independently sent to the rasterizer and processed one by one.

9.5 Conclusion

The accurate evaluation of the depth of scanconverted faces is crucial for many applications. Contemporary rasterizers ensure very accurate evaluation of the surface depth for pixels entirely covered by the projection of a face, but may produce very large depth errors for pixels whose centers are close to the projection of an edge bounding at least one face with a steep slope. The errors are due to the use of the wrong surface equation for establishing the depth. The wrong surface is used when pixels are incorrectly classified with respect to the face's projections. Wrong classification may result from very small roundings of the edge coefficients, yet they may produce arbitrarily large depth errors. The proposed solution scanconverts a slightly enlarged area containing all of the pixels entirely covered by the projection of the face, plus neighboring pixels. The decision as to whether a pixel's center lies inside the projection of the face is carried out by testing the signs of three linear functions that are incrementally updated during scanconversion. These functions capture the depth-ordering between the surfaces containing the scanconverted face and the surfaces containing the three neighboring faces. Their slopes and initial values may be easily computed by subtracting slopes and depths of the appropriate surfaces. The correct signs of these functions (for the pixel to be inside the face) are established by considering the concavity of the corresponding edge.

Acknowledgements:

Thanks to Dan Brokenshire and to Mark Fowler for pointing out and discussing examples of scanconversion inaccuracy. Thanks also to Bengt-Olaf Schneider for his technical input and to Abe Megahed for providing software support for simulating various scanconversion techniques and for testing their accuracy.

9.6 References

[1] Akeley, Kurt. The Silicon Graphics 4D/240GTX Superworkstation. IEEE Computer Graphics and Applications, 9(4):71-83, 1989.

[2] Atherton, P.R. A Scan-line Hidden Surface Removal Procedure for Constructive Solid Geometry. ACM Computer Graphics, 17(3):73-82, July 1983.

[3] Bar-Yehuda, R. and Grinwald, R. Triangulating polygons with holes. In J. Urrutia, editor, Proc. Sec. Canadian Conf. in Computational Geometry, 112-115, Ottawa, Ontario, August 6-10 1990.

[4] Baumgart, B.G. Winged Edge Polyhedron Representation. AIM-79, Stanford Univ., Report STAN-CS-320, 1972.

[5] Borges, R. Line Algorithms for Raster Displays Rescued from Round-Off Errors. Comput. & Graphics, 15(2):155-160, 1991.

[6] Bronsvoort, W. An Algorithm for Visible-Line and Visible-Surface Display of CSG Models. The Visual Computer, 3(4):176-185, December 1987.

[7] Bronsvoort, W. Boundary Evaluation and Direct Display of CSG Models. Computer Aided Design, 20(7):416-419, 1988.

[8] Bronsvoort, W.F., Van Wijk, J.J., and Jansen, F.W. Two Methods for Improving the Efficiency of Ray Casting in Solid Modelling. Computer-Aided Design, 16(1):51-55, January 1984.

[9] Catmull, E. A subdivision algorithm for computer display of curved surfaces, PhD thesis. University of Utah, Salt Lake City, UT, December 1974.

[10] Clausen, U. Parallel Subpixel Scanconversion. In A.A.M.Kuijk and W. Strasser, (Eds.) , *Advances in Computer Graphics Hardware II*, p.155-166, EurographicSeminars, Springer-Verlag, Berlin, 1987.

[11] Denault, D., Ryherd, E., Torborg, J., Tosi, R., and Werner, R. VLSI Drawing Processor Utilizing Multiple Parallel Scan-line Processors. In: A.A.M.Kuijk and W. Strasser, (Eds.), *Advances in Computer Graphics Hardware II*, p.167-182, EurographicSeminars, Springer-Verlag, Berlin, 1987.

[12] Ellis, J.L., Kedem, G., Lyerly, T.C., Thielman, D.G., Marisa, R.J., and Menon, J.P. The Ray Casting Engine and ray representations. In J. Rossignac and J. Turner, editors, ACM Symp. on Solid Modeling Foundations and CAD/CAM Applic., 255-268, Austin, TX, June 5-7 1991.

[13] Epstein, D., Jansen, F., and Rossignac, J. Z-buffer Rendering from CSG: The Trickle Algorithm. T.J. Watson Research Center, Yorktown Heights, NY, RC 15182, December 1990.

[14] Field, D. Incremental linear interpolation. ACM Transactions on Graphics, 4(1):1-11, 1985.

[15] Gharachorloo, N., Gupta, S., Sprull, R., and Sutherland, I. A characterization of ten rasterization techniques. Computer Graphics, Proc. SIGGRAPH'89, 23(3):355-368, July 1989.

[16] Goldfeather, J., Molnar, S., Turk, G., and Fuchs, H. Near Real- Time CSG Rendering Using Tree Normalization and Geometric Pruning. IEEE Computer Graphics and Applications, 9(3):20-28, May 1989.

[17] Hughes, J. Integer and floating-point z-buffer resolution. Brown University, Providence, RI. 1989.

[18] Jansen, F.W. A CSG List Priority Hidden-Surface Algorithm. Proc. Eurographics '85 Conf., 51-62, Elseviers Science Publishers, Amsterdam, 1985.

[19] Jansen, F.W. A Pixel-parallel Hidden Surface Algorithm for Constructive Solid Geometry. In A.A.G. Requicha, editor, Eurograhpics '86,, North-Holland, Amsterdam, 1986.

[20] Jansen, F.W. Solid Modelling with Faceted Primitives, PhD thesis. Technische Universiteit Delft, The Netherlands, September 1987.

[21] Jansen, F.W. CSG Hidden-Surface Algorithms for VLSI Hardware Systems. In: Strasser, W., (Ed.) , *Advances on Computer Graphics Hardware I*, p.75-82,EurographicSeminars, Springer Verlag, Berlin, 1987.

[22] Jansen, F.W. and Sutherland, R.J. Display of Solid Models with a Multi-Processor System. Proc. Eurographics '87, Elseviers Science Publishers, Amsterdam, 1987.

[23] Mammen, A. Transparency and antialiasing algorithms implemented with the virtual pixel map technique. IEEE Computer Graphics and Applications, 9:43-55, July 1989.

[24] Requicha, A. A. G. Representations for Rigid Solids: Theory, methods, and systems. ACM Computing Surveys, 12(4):437-464, December 1980.

[25] Rossignac, J. Through the cracks of the solid modeling milestone. Eurographics'91 State of The Art Reports, Springer Verlag, 1991. (To appear)

[26] Rossignac, J., Megahed, A., and Schneider, B.O. Interactive Inspection of Solids: Cross-sections and interferences. ACM Computer Graphics, Proc. SIGGRAPH'92, 26(4), 1992.

[27] Rossignac, J.R. and Requicha, A.A.G. Depth Buffering Display Techniques for Constructive Solid Geometry. IEEE Computer Graphics and Applications, 6(9):29-39, September 1986.

[28] Rossignac, J.R. and Voelcker, H.B. 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.

[29] Rossignac, J.R. and Wu, J. Correct Shading of Regularized CSG Solids Using a Depth-Interval Buffer. In: R.L. Grimsdale and A. Kaufman, *Advances in Computer Graphics Hardware V*, EurographicSeminars, p.117-138, Springer-Verlag, Berlin, 1992.

[30] Roth, S.D. Ray Casting for Modelings Solids. Computer Grahpics and Image Processing, 18(2):109-144, February 1982.

[31] Schilling, A. A new simple and efficient antialiasing with subpixel masks. Computer Graphics, Proc. SIGGRAPH'91, 25(4):133-141, July 1991.

[32] Schneider, B.O. and Claussen, U. PROOF: An architecture for rendering in object space. In A. Kuijk, editor, *Advances in Computer Graphics Hardware II*, p.121-140, Springer-Verlag, Berlin, 1988.

[33] Serra, J. Image Analysis and Mathematical Morphology. Academic Press, New York, 1982.

[34] Sutherland, I.E., Sproull, R.F., and Schumacher, R.A. A characterization of ten hidden-surface algorithms. ACM Computing Surveys, 6(1):1-55, March 1974.

[35] Tilove, R.B. and Requicha, A.A.G. Closure of Boolean Operations on Geometric Entities. Computer-Aided Design, 12(5):219-220, September 1980.