3.3 Results

We implemented a specialization of the general algorithms in §3.1 and §3.2 for the example cases of §2.3, which are the cases of most practical use. Figures 7 to 9 show some pictures made with our implementation in the public-domain raytracer *rayshade*.

Figure 7 shows a variation of the famous Barnsley fern [1], a prototype of a hierarchical IFS, in this case with 6 transformations in 3 levels. Figure 8 shows a temple-like construction and is described by a CIFS-code with 4 transformations and a composite condensation set. The scene in figure 9 is described by a hierarchical code with also 3 levels and in total 4 transformations with a geometry for a branch in the lowest level and an alternative geometry for a leaf, as proposed in §3.1 for drawing trees à la [11]. All these scenes are composed of several hundreds of tousands of primitive objects, making it impossible to compute a list of primitive objects and raytrace these using e.g. space partitioning to improve efficiency.

The raytracing of these images takes about 15 minutes and 100 Kbytes of memory on an IBM RS/6000-320 system. The examples were chosen to show that quite complex pictures can be made from an extremely short description, a property of fractal-like objects. The raytracing can be done in reasonable time and with almost negligible memory usage.

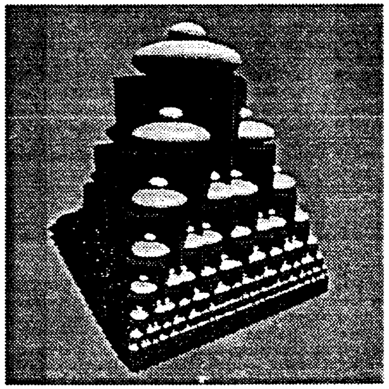


Figure 8: Temple: a CIFS with 4 transformations and composite condensation set.

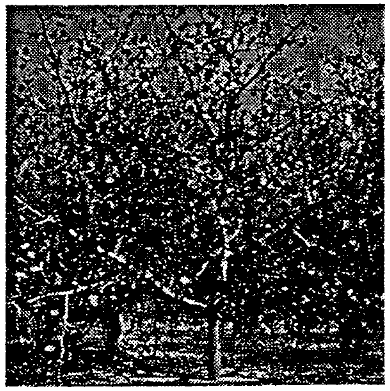


Figure 9: Trees: described by a code with 4 transformations in 3 hierarchical levels, a condensation set and alternative geometry.

4 Conclusion

In §2 we presented a formalism that can describe arbitrary linear graftals. IFS-codes, CIFS-codes, HIFS-codes emerge as simple examples. In §3 we presented an algorithm for raytracing objects described by this formalism, which is very general. We implemented a specialized version, suitable for the interesting cases of multilevel HIFS and

Coherence in scan-line algorithms for CSG

Eduard Gröller, Peter Brunner

Technical University Vienna
Institute for Computergraphics
Karlsplatz 13/186/2
A-1040 Vienna, Austria
Tel.: +43(1)58801-4582
FAX: +43(1)5874932
e-mail: groeller@eigvs4.tuwien.ac.at

Abstract

Scan-line algorithms for visibility calculation exploit various types of coherence properties. Several scan-line algorithms for Constructive Solid Geometry (CSG) are discussed. In one approach CSG primitives are represented by polygonal approximations. Another technique processes CSG primitives as general quadric surfaces. We investigate the handling of frequently occurring quadric surfaces (cube, cone, sphere, cylinder) as distinct cases. Thus the differing properties of such objects can be used more efficiently than a uniform approach would allow. A so called eBRep (extended Boundary Representation) is defined for the frequently occurring quadric surfaces. An eBRep is an exact representation of a quadric object and contains curved edges and faces. For each of the above mentioned quadric surfaces a different, geometry dependent eBRep is specified. A comparison between the polygon-based scan-line algorithm for CSG and our eBRep based approach is done. eBRep is a storage efficient exact representation of quadric surfaces, well suited for scan-line visibility determination.

Keywords: CSG, quadrics, scan line algorithm, extended BRep, curved edges

1. Introduction

Scan-line algorithms for visible surface determination have been one of the earliest examples that extensively use coherence properties ([Athe83], [Bron90], [FoDa90], [Gröl93], [Klei90], [Klei92], [PuMe87], [SuSp74]). Originally scan-line algorithms have been designed for hidden line/hidden surface removal of polygonal objects. Objects are specified in a 3D world-coordinate system. The position of the viewer and the image plane are defined in a camera-coordinate system whereby the eye point of the viewer is assumed to lie on the negative z-axis (z^-) and the image plane coincides with the xy-plane of the camera-coordinate system. The 3D object scene is then transformed into the camera-coordinate system and is processed sequentially according to descending y-coordinates.

The 3D objects are intersected with scan planes sp_i (planes with constant y-coordinate in the camera-coordinate system) that correspond to scan lines s_i. A scan line s_i itself is associated with a row of pixels in the final raster image. The endpoints of the 2D segments, resulting from the intersection of 3D objects with scan plane sp_i introduce a subdivision of scan line s_i into so called spans. Spans are processed from left to right in order of decreasing x-coordinates. For an illustration of scan planes, scan lines, segments and spans see Figure 1. By reducing the 3D visibility problem to 2D problems (segments in scan planes sp_i), 1D problems (spans on scan line s_i) and 0D problems (visibility on span endpoints) the scan-line algorithm uses the principle of locality to attain coherent regions that can be processed easily.

The surface of objects is defined through faces, edges and vertices. In a preprocessing step the edges are sorted according to their highest y-value and are placed in y-buckets, one y-bucket for each scan line. The y-bucket of scan line s_i consists of all those edges that start between scan plane sp_i and sp_{i+1}. During the algorithm two lists are maintained: The active-edge list (AEL) contains all edges that intersect the current scan line s_i. The AEL is sorted on decreasing x-coordinates of the intersection points between edges and scan plane sp_i. The active-face list (AFL) contains all those faces that cover the current span and therefore might be visible. Both lists are modified incrementally. The AEL of scan line s_i is determined by updating the AEL of scan line s_{i+1}. Edges that do not intersect the current scan line s_i anymore are removed from the AEL whereas the entries of the y-bucket of s_i (edges starting at s_i) are added to the AEL. The active-face list AFL is modified analogously between adjacent spans on the current scan line. Faces that did cover the previous span but are not covering the current span are eliminated from the AFL, newly covering faces are added to the AFL. Visibility at the current span is determined by sorting the faces of the AFL of the current span on z-depth (distance to the viewer) and drawing the pixels covered by the current span with the color of the face closest to the viewer. More elaborate shading models, e.g., Phong shading [FoDa90], can be used to determine the color value of a pixel. In case of non-penetrating objects sorting has to be done only once for a span. In case of intersecting objects or faces, sorting has to be done on both ends of the span. If the nearest faces on both ends are different the span is split at the intersection point of the corresponding segments, and the two resulting spans are processed recursively [Bron90].

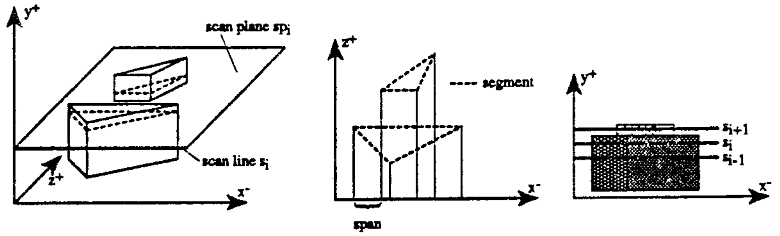


Figure 1: scan plane sp_i, scan lines s_{i+1}, s_i, s_{i-1}, segments, spans (figure similar to figure 3.2 in [Bron90])

The efficiency of the scan-line algorithm is due to the exploitation of various types of coherence:

Scan-line coherence: edges that intersect scan line s_{i+1} will also, with high probability, intersect scan line s_i. Additionally the (x-coordinate) order of intersection points of edges with a scan plane will change only slightly from scan plane sp_{i+1} to sp_i. Scan-line coherence is taken advantage of by maintaining and incrementally updating the active-edge list (AEL), by incrementally calculating the intersection points between edges and scan planes and by sorting the AEL with bubble sort (bubble sort is, despite its quadratic worst case behavior, well suited for almost sorted data sets)

Span coherence: active faces and their depth ordering will almost be the same between adjacent spans on a scan line. Efficient processing is achieved by an incremental update of the active-face list AFL and by sorting the entries of the AFL with bubble sort. Moreover the visible face usually remains the same on the entire span, so depth sorting of the AFL needs to be done only at the span boundaries. The color values of pixels covered by a single span can be calculated incrementally, e.g., with Phong shading.

Depth coherence: the ordering of active faces and active edges changes, as already pointed out above, gradually. This depth coherence property reduces the cost of sorting substantially (bubble sort).

Edge coherence: The intersection points between an edge and a scan plane sp_i can be easily calculated incrementally from the intersection point of the same edge with the previous scan plane sp_{i+1}. The visibility on a span can be usually determined by just processing the endpoints of the span.

Although scan-line algorithms can not account for complex global lighting effects as shadowing, reflection, transparency they are suited for fast visualization during the interactive modeling and design process of objects. After the specification is completed a non-interactive rendering technique that supports a more complex and more realistic shading model, e.g., ray tracing, may be used. CSG (Constructive Solid Geometry) is a powerful and versatile technique for modeling complex mechanical parts. Thereby an object is represented as a binary tree. Leaf nodes correspond to geometrical primitives (cube, sphere, cylinder, cone, ...)

whereas in the intermediate nodes set operations (union, intersection, difference) define how the objects represented by the two child nodes are combined. The CSG data structure does not contain any explicit surface information of the object represented by the root node of the CSG tree.

There are different possibilities for visualization CSG objects with scan-line algorithms. In one approach an approximate polygonal boundary representation (BRep) is derived from the CSG data structure. Visibility calculations are then done on the BRep structure. The drawbacks of this technique are: costly generation of the BRep structure, and the BRep is only an approximation to the original object. In section 2 a scan-line algorithm is presented that requires only the calculation of the polygonal BRep for the primitives (leaf nodes of the CSG tree). In section 3 a scan-line algorithm is described where an exact representation of the primitives through quadrics is used. Primitives are either treated as general quadrics [Klei92] or specific properties of frequent quadrics (cube, cone, sphere, cylinder) are exploited as was done in our approach. Algorithms of section 2 and 3 require a more elaborate handling of individual spans during visibility calculation.

2. A polygon based scan-line algorithm for CSG (method 1)

In this section a scan-line algorithm for CSG is presented, that works with polygonal approximations of the CSG primitives [Athe83], [Bron90]. This algorithm is (in the context of this paper) denominated method 1 as opposed to the quadrics based approach of section 3 which is denominated method 2. As no BRep of the composite object, represented by the entire CSG tree, is determined, the handling of individual spans during visibility calculation is more complicated. Penetrating primitives and intersecting faces have to be dealt with. Faces of primitives may not lie on the boundary of the composite object, so processing of the active-face list AFL at span boundaries must find the first visible face of the composite object which may not be the first entry of the AFL. This task is accomplished by Boolean classification of the faces in the AFL using the set operations in the intermediate nodes of the CSG tree. If at both endpoints of a span the AFL does have the same entries (same faces, same ordering) starting from the nearest face up to the first face on the composite object, i.e., the visible face, the span can be drawn with the color of the visible face. Otherwise the span is subdivided at the intersection point of those segments that correspond to the first two different faces, and the two subspans are handled recursively.

Top-down classification of the faces in the AFL performs a recursive traversal which starts at the root node of the entire CSG tree. Considering the specific situation (processing entries of the AFL according to their depth ordering) a much more efficient bottom-up classification can be done [Bron90]. Stepping from one face of the AFL to the next, the status of being inside or outside of the entire CSG object is changing for only one primitive. Bottom-up classification exploits depth coherence and Boolean combination coherence and proceeds from the respective leaf node upwards until the root node is reached or the status of an intermediate node is not affected by the changed status of the primitive. Progression up the CSG tree is enabled by adding to each CSG node a pointer to its parent node.

Classification is not only simplified by using coherence properties between successive entries of the active-face list AFL. Classification similarities between adjacent span endpoints are exploited as well. In [Bron90] some further techniques for reducing the classification cost are discussed: The size of the CSG tree can be reduced by eliminating those leaf nodes that do not have any effect on the current scan line. Partial back-face culling reduces the number of classifications. The unchanged order of active faces at span boundaries can be exploited. Often it is sufficient to classify the AFL only at one of the two span boundaries. For more details see [Bron90].

3. A quadrics based scan-line algorithm for CSG (method 2)

3.1. Introduction

In this section a scan-line algorithm for CSG is described where the CSG primitives are not represented by a polygonal approximation but are handled analytically as quadrics (see section 3.2). Primitives as quadric surfaces are represented through an extended Boundary representation (eBRep) that contains curved edges and faces. An eBRep is made up of considerably fewer vertices, edges and faces as a polygonal BRep and is chosen so that the basic structure of the scan-line algorithm of section 2 (method 1) has to be modified only slightly (different handling of individual spans). Compared to method 1 such an approach has several advantages: exact representation and shading of objects, low storage requirements, faster processing because of a small data base.

3.2. Quadrics

Quadric surfaces, or quadrics, are sets of all points in 3D space that satisfy a second-degree polynomial equation

$$F(x,y,z) = 0 \qquad (1)$$

Quadrics represent surfaces that include planes, spheres, cones, cylinders, ellipsoids, paraboloids and hyperboloids ([Klei90], [Klei92], [Mill87], [Mill88]). Natural quadrics are a subset of the quadrics and comprise planes, spheres, cylinders and cones. As pointed out in [HaHi80] 90-95% of mechanical parts can be defined through natural quadrics. Natural quadrics are also the most commonly used primitives in CSG modeling. The second-degree polynomial of a quadric is given as follows:

$$Ax^2 + By^2 + Cz^2 + 2Dxy + 2Eyz + 2Fxz + 2Gx + 2Hy + 2Jz + K = 0 \qquad (2)$$

Equation (2) can be written in matrix form as

$$\bar{p}Q\bar{p}^T = 0 \qquad (3)$$

with

$$\bar{p} = (x,y,z,1) \qquad \text{and} \qquad Q = \begin{pmatrix} A & D & F & G \\ D & B & E & H \\ F & E & C & J \\ G & H & J & K \end{pmatrix}$$

If the surface of a quadric Q is transformed by a matrix R the transformed surface is again a quadric and is defined by Q' with

$$Q' = R^{-1}Q(R^{-1})^T \qquad (4)$$

Primitives of a CSG model are typically defined in a local-coordinate system. An additional matrix specifies the position of the locally defined object in the world coordinate system [Roth82]. The algorithm presented in this chapter requires quadrics to be specified in the camera-coordinate system. Equation (4) is used to derive the transformed quadric Q' in the camera-coordinate system from quadric Q in the local-coordinate system. Quadrics can be handled in two ways: In the first approach quadrics are processed uniformly without regard of their specific shape [Klei92]. In the second approach, which we investigated in more detail, characteristics of the shape of a given quadric, e.g., of a cylinder, are exploited for speeding up processing. The first approach leads to short, somewhat inefficient program code. The second approach is more efficient, but produces longer program code as different cases have to be treated separately. The second technique is better suited to handle numerical instabilities as well. We implemented the second approach and give comparisons to method 1 of section 2.

3.3. Algorithm

The usage of quadrics to define primitives in the CSG model requires some modifications of method 1, i.e., the handling of spans has to be adapted to curved surfaces [Klei92]. To keep the modifications as small as possible quadric surfaces are partitioned to get an extended BRep (eBRep) with curved edges and curved faces. Method 1 assumes edges to be monotonous with respect to the y-axis, i.e., there is at most one intersection of an edge with a scan line and the endpoints are extremal with respect to the y-axis. With this assumption an edge is added to the active-edge list AEL when the endpoint with higher y-value is reached during the scan process and the edge is deleted from the AEL when the endpoint with lower y-value is passed (edge

coherence). To maintain this property with quadrics the y-extremes (points with minimal and maximal y-values) are introduced as vertices in the eBRep of quadrics.

Furthermore in method 1 segments, i.e., intersections of a scan plane with a BRep, are also monotonous with respect to the x- and z-axis. Monotony along the x-axis is important to decide when a face becomes active. Monotony along the z-axis is important for easy depth sorting of faces at span boundaries. (Curved) edges are added to the eBRep to assure monotony of segments along the x- and z-axis. In general edge coherence can be exploited for sorting monotonic edges by sorting the endpoints of these edges.

As the monotony of edges is not invariant under affine transformations the eBRep has to be constructed in the camera-coordinate system. The quadrics of CSG primitives are therefore transformed from the local-coordinate system into the camera-coordinate system using equation (4) of section 3.2. The incremental update of curved edges from one scan line to the next is a little bit more complicated than with linear edges and will be explained later on.

A further modification of method 2 is due to the fact that two curved segments might intersect twice within one span. This situation may occur if the bounding boxes of two curve segments overlap which is called depth-overlap [Klei92].

Depth-overlap may cause different faces to be visible within one span although the active-face list AFL is the same at both span boundaries. In this case spans are recursively subdivided until no depth-overlap occurs or spans are shorter than the width of a pixel. In [Klei92] span subdivision is done only if there are in fact intersection points within a span, which might be somewhat more efficient.

The intersection calculation between segments belonging to two faces with different depth-order on both ends of a span has to be extended to handle curved segments as well. Three cases may occur: straight segment - straight segment intersection, straight segment - curved segment intersection, curved segment - curved segment intersection. Intersecting two straight segments is trivial. Intersecting a straight segment and a curved segment is done by substituting the formula of the straight line into the quadratic equation of the curved segment. The resulting second-degree equation is solved to give two values. Only one of these solutions is the desired intersection point. Intersection of two curved segments is done with Newton iteration. The middle of the span is taken as starting point for the iteration process. The iteration is stopped as soon as subpixel precision is achieved.

Finally shading is done by calculating the exact normal vector $\bar{n} = (\partial F/\partial x, \partial F/\partial y, \partial F/\partial z)$ for each pixel. No approximate normal vector interpolation as in method 1 is done.

3.4. Extended BRep for Quadrics (eBRep)

[Klei92] presents a quadrics based scan-line algorithm for CSG where primitives are processed as general quadric surfaces. In our approach we also use quadrics based primitives but we handle frequently occurring quadrics (cube, cone, sphere, cylinder) separately.

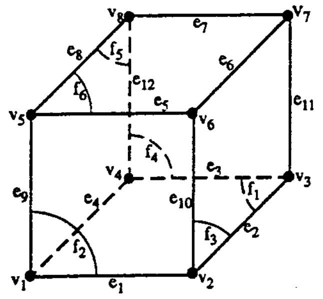


Figure 2: (e)BRep of a cube

In this section extended BReps (eBReps) of the CSG primitives cube, sphere, cylinder and cone are discussed. The formulas for the incremental update of curved edges are given. The eBRep of the cube is the usual BRep and consists of 8 vertices, 12 straight edges and 6 planar faces (see Figure 2).

The eBRep of a sphere (or ellipsoid that results from transforming a sphere) consists of two vertices (v_1,v_2), four curved edges (e_1, e_2, e_3, e_4) and four curved faces (f_1, f_2, f_3, f_4) (see Figure 3). v_1 (v_2) is the point on the quadric surface with maximal (minimal y-value) and is calculated by using the normal vector $\vec{n}$ of the quadric F(x,y,z)=0 which is given as $\vec{n} = (\partial F/\partial x, \partial F/\partial y, \partial F/\partial z)$. Vertices v_1 and v_2 are then the solutions of the system of equations F(x,y,z)=0, $\partial F/\partial x = 0$ and $\partial F/\partial z = 0$. Edges e_1, e_2, e_3 and e_4 are chosen so that their intersection points with a scan plane partition the resulting 2D quadratic curve (circle, ellipse) into four monotonic segments. The incremental update of these intersection points is explained later on. Edges e_1, e_2 (silhouette edges in z-direction) are defined by F(x,y,z)=0 and $\partial F/\partial z = 0$. Edges e_3 , e_4 (silhouette edges in x-direction) are defined by F(x,y,z)=0 and $\partial F/\partial x = 0$.

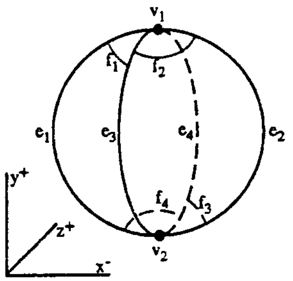
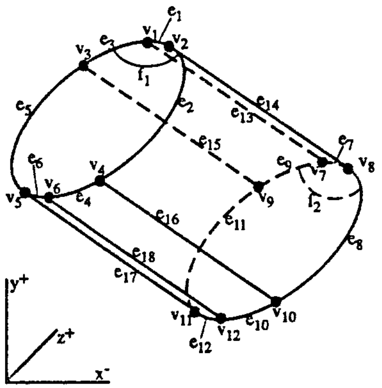


Figure 3: eBRep of a sphere



faces f_3, ..., f_8 are on the curved cylinder boundary

Figure 4: eBRep of a cylinder

The eBRep of a cylinder consists of 12 vertices, 12 curved edges, 6 straight edges, 2 planar faces and 6 curved faces (see Figure 4). A cylinder is defined by a quadric surface F(x,y,z)=0 (curved boundary), a top plane p_{top} and a bottom plane p_{bot}. Vertices v_1 and v_6 are the points on face f_1 (contained in p_{top}) with maximal and minimal y-coordinates. They are calculated as follows: The equation of p_{top} is used to eliminate the z-coordinate in F(x,y,z)=0. The resulting

expression is solved for x giving an equation of the type $_1x_2 = u(y) \pm \sqrt{v(y)}$. Vertices v_1 and v_6 are then determined by v(y)=0. Vertices v_7 and v_{12} on face f_2 (contained in p_{bot}) are calculated analogously. The silhouette edges e_{14}, e_{15}, e_{16}, e_{17} are calculated the same way as silhouette edges of the sphere. Vertices v_2, v_3, v_4, v_5 (v_8, v_9, v_{10}, v_{11}) are the intersection points of the silhouette edges with p_{top} (p_{bot}). Edge e_{13} (e_{18}) is the connection between vertices v_1 (v_6) and v_7 (v_{12}).The remaining curved edges e_1,...,e_6 (e_7,...,e_{12}) lie on the intersection of quadric F(x,y,z)=0 and p_{top} (p_{bot}) and connect the vertices on face f_1 (f_2). Faces f_3,...,f_8 make up the curved cylinder boundary. Special cases arise if p_{top} and p_{bot} are parallel or perpendicular to the xz-plane. In these cases a simplified eBRep can be used.

The eBRep of a cone consists of 7 vertices, 6 straight edges, 6 curved edges, 1 planar face and 6 curved faces (see Figure 5). A cone is bounded by a quadric F(x,y,z)=0 and a bottom plane p_{bot}. Vertices v_1 and v_6 (points with maximal and minimal y-value) on face f_7 (contained in p_{bot}) are calculated the same way as v_1, v_6 of the eBRep of a cylinder. Edges e_7 and e_{12} are connecting v_1 and v_6 to the apex v_7 of the cone. Edges e_8, e_9, e_{10}, e_{11} are again silhouette edges. They are calculated the same way as silhouette edges on a sphere. Their intersection points with p_{bot} define the vertices v_2, v_3, v_4, v_5. The remaining curved edges e_1,...,e_6 connect vertices on the intersection curve between quadric F(x,y,z)=0 and plane p_{bot}. There are some special cases that simplify the eBRep of cones: If p_{bot} is parallel to the scan planes sp_i (xz-plane) then vertices v_1, v_6 and edges e_1, ..., e_6, e_7, e_{12} are omitted. Depending on the position of a cone there might be fewer than four silhouette edges. The number of silhouette edges (one, two or four) can be deduced from the type of intersection between scan plane sp_i and the conic quadric, which is given as equation (2):

$AB < D^2$	ellipse	4 silhouette edges
$AB = D^2$	parabola	1 or 2 silhouette edges
$AB > D^2$	hyperbola	1 or 2 silhouette edges

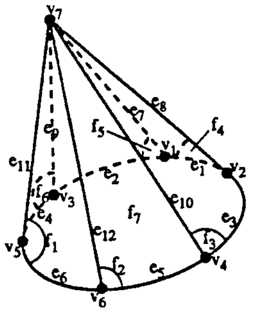


Figure 5: eBRep of a cone

Monotony does not change along edges e_{13}, e_{18} in Figure 4 and e_7, e_{12} in Figure 5, so these edges could even be omitted (although thus compromising on the topological validity of the model). As silhouette edges are not transformation invariant quadric surfaces have to be transformed into the camera-coordinate system before the eBRep can be calculated.

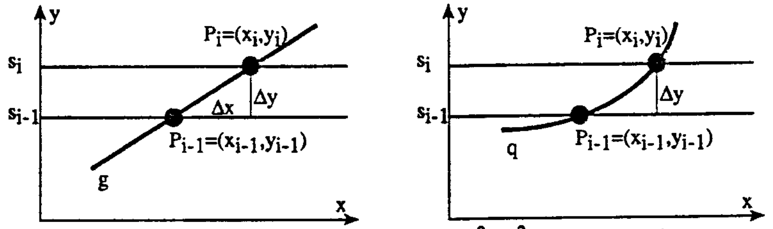
3.5. Incremental update of curved edges

Scan-line coherence can be exploited by incrementally updating edges from scan line s_i to s_{i-1}. The incremental update of a straight edge can be easily accomplished. If the x-coordinate of a point P_i on scan line s_i is given, the x-coordinate of the Point P_{i-1} on the following scan line s_{i-1} can be determined with only one subtraction (see Figure 6). The incremental update of a quadric curve is more complicated and requires four subtractions and one square root evaluation (see Figure 6). $P_i=(x_i,y_i)$ is a point on the quadric edge q: $Ax^2+By^2+Cxy+Dx+Ey+F=0$ on scan line s_i. The x-coordinate of the point $P_{i-1}=(x_{i-1},y_{i-1}=y_i-\Delta y)$ on the next scan line is incrementally calculated as follows:

$$^{(1)}x^{(2)}{}_{i-1} = k_i \pm \sqrt{l_i} \quad \text{with}$$
$$k_{i-1} = k_i - \Delta k, \qquad \Delta k = -\frac{C}{2A}\Delta y$$
$$l_{i-1} = l_i - \Delta l_i$$
$$\Delta l_{i-1} = \Delta l_i - \Delta^2 l, \qquad \Delta^2 l = \Delta y^2 (\frac{C^2-4AB}{2A^2})$$

Δk and $\Delta^2 l$ are calculated for each curved edge only once. If s_n is the first scan line that intersects the curved edge q then the values k_n, l_n, Δl_n are initialized with:

$$k_n = -\frac{Cy_n + D}{2A}$$
$$l_n = y_n^2(\frac{C^2-4AB}{4A^2}) + y_n(\frac{CD-2AE}{2A^2}) + (\frac{D^2-4AF}{4A^2})$$
$$\Delta l_n = (2y_n - \Delta y)\Delta y(\frac{C^2-4AB}{4A^2}) + \Delta y(\frac{CD-2AE}{2A^2})$$



g: $Ax+By+C=0$
$y_{i-1} = y_i - \Delta y$
$x_{i-1} = x_i - \Delta x$ with $\Delta x = -\frac{B}{A}\Delta y$

q: $Ax^2+By^2+Cxy+Dx+Ey+F=0$
$y_{i-1} = y_i - \Delta y$
$^{(1)}x^{(2)}{}_{i-1} = k_i \pm \sqrt{l_i}$

Figure 6: Incremental update of straight and curved edges

4. Implementation and results

A test system was implemented [Brun92] on a VAX-cluster (Micro VAX 2000, 3100, 3200) under the VMS operating system in VAX Pascal. Tests show that method 2 requires significantly less storage than method 1 (see Figure 7). This holds although an edge or face of method 2 requires about two to three times as much storage space as an edge or face of method 1.

Comparisons of the calculation times show that method 2 is slower than method 1 for small CSG models but is more efficient for larger object scenes. For small scenes the overhead of setting up and manipulating curved edges is not made up by a significantly smaller data base.

Only with larger CSG models method 1 has to handle (e.g., sort) large lists of faces and edges. In this case the small data base due to the usage of eBReps in method 2 allows an image generation faster by about a factor of two compared to method 1 (see Figure 8).

			method 1		method 2	
image	#prim	resolution	#edges	#faces	#edges	#faces
'sphere'	1	200x200	480	320	4	4
'm-sphere'	3	400x400	972	646	20	14
'h-virus'	19	400x800	1488	850	266	128
'atomium'	26	400x400	5120	3185	314	164
'molecule'	104	400x400	20480	12740	1344	684

Figure 7: number of edges and faces of method 1 compared to method 2

	method 1			method 2				
image	BRep	scan	total	eBRep	scan	total	α	β
'sphere'	0:01	0:06	0:27	0:00	0:09	0:37	0.6	0.7
'm-sphere'	0:03	0:27	0:41	0:00	0:44	1:10	0.6	0.7
'h-virus'	0:04	0:30	0:53	0:00	0:19	0:42	0.8	1.2
'atomium'	0:14	0:46	1:13	0:00	0:23	0:35	1.3	2.1
'molecule'	0:56	4:02	6:38	0:01	2:31	2:59	1.5	2.2

BRep, eBRep: time for constructing the boundary representation
scan : time used for the scanning process
total : BRep+scan+shading time
α : ratio: scan of method 1 / scan of method 2
β: ratio: total of method 1 / total of method 2

Figure 8: time requirements of method 1 compared to method 2 (in min:sec)

Method 1 is characterized by an approximation of CSG primitives that are handled in a general uniform way. With method 2 primitives are represented exactly, each type of primitive (cube, sphere, cone, cylinder) is handled separately. Therefore the program code of method 2 is a little bit more complicated than the program code of method 1. As natural quadrics occur frequently in CSG models the special treatment of these primitives is worth the additional programming effort. Comparisons with a ray tracer developed at our department [GePu88] show that the scan line algorithms for CSG presented in this section are, on the average, faster by a factor of 40. This is an immediate consequence of the fact that the scan-line algorithms make extensive use of given coherence properties.

References

[Athe83] Peter R. Atherton, "A Scanline Hidden Surface Removal Procedure For Constructive Solid Geometry", Computer Graphics 17(3), July 83, pp. 73-82.

[Bron90] Bronsvoort, W.F., "Direct Display Algorithms For Solid Modelling", PhD thesis, Delft University Press, 1990.

[Brun92] Brunner, P., "Scanline Algorithmen für CSG-Objekte", master's thesis, Institute for Computergraphics, Technical University Vienna, September 1992.

[FoDa90] Foley, J., van Dam, A., Feiner, St., Hughes, J., "Computer Graphics Principles and Practice", Addison-Wesley, 1990.

[Gröl93] Gröller, E., "Coherence in Computer Graphics", PhD dissertation, VWGÖ Verband der wissenschaftlichen Gesellschaften Österreichs, September 1993.

[HaHi80] Hakala, D.G., Hillyard, R.C., Nourse, B.E., Malraison, P.J., "Natural quadrics in mechanical design", Proceedings of Autofact West, 1980, pp. 363-378.

[GePu88] Gervautz, M., Purgathofer, W., "RISS-Ein Entwicklungssystem zur Generierung realistischer Bilder", Informatik Fachberichte 182, 1988, pp. 61-79.

[Klei90] van Kleij, R., "Implementation of a solid modelling system with quadratic surfaces", Reports of the Faculty of Technical Mathematics and Informatics no. 90-41, Technical University Delft, 1990.

[Klei92] van Kleij, R., "Efficient display of quadric CSG models", Computers in Industry 19, Elsevier Science Publishers, 1992, pp. 201-211.

[Mill87] Miller, J.R., "Geometric Approaches to Nonplanar Quadric Surface Intersection Curves", ACM Transactions on Graphics 6(4), October 1987, pp. 274-307.

[Mill88] Miller, J.R., "Analysis of Quadric Surface Based Solid Models", IEEE Computer Graphics & Applications, 8(1), January 1988, Pages 28-42.

[PuMe87] Pueyo, X., Mendoza, J.C., "A new scan line algorithm for the rendering of CSG trees", Proceedings of EUROGRAPHICS´87, North-Holland, Amsterdam 1987, pp. 347-361.

[Roth82] Roth, S.D., "Ray Casting for Modeling Solids", Computer Graphics and Image Processing, Vol. 18, 1982, pp. 109-144.

[SuSP74] Sutherland, I.E., Sproull, R.F., Schuhmacher, R.A., "A Characterization of Ten Hidden-Surface Algorithms." ACM Computing Surveys 6(1), March 1974, pp. 1-55

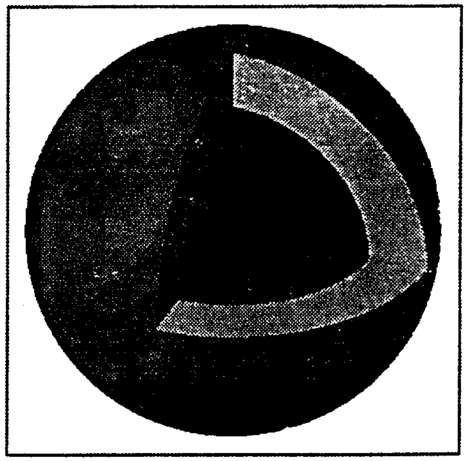


Figure 9: image 'm-sphere'

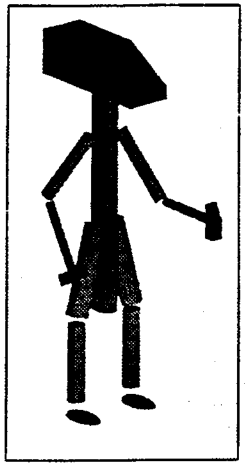


Figure 10: image 'h-virus'

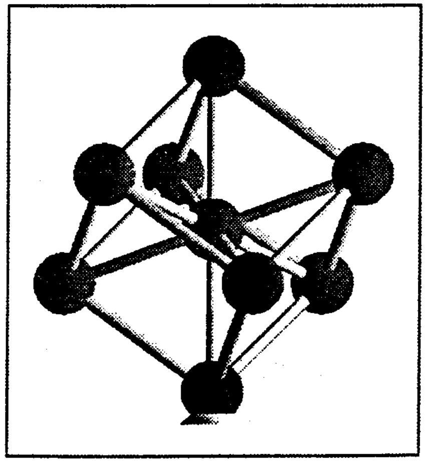


Figure 11: image 'atomium'

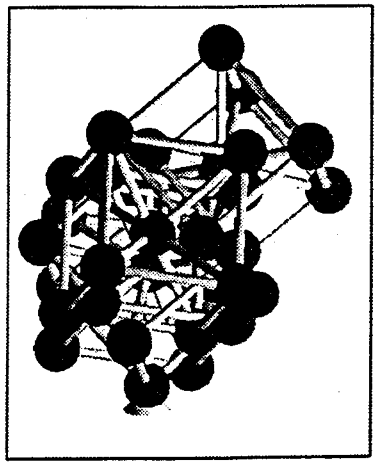


Figure 12: image 'molecule'