Contour-Edge Based Polyhedron Visualization

Lutz Kettner*, UNC, Chapel Hill.
kettner@cs.unc.edu

Abstract

We present a new approach to computing visibility for three-dimensional polyhedral surfaces based on contour edges. The advantages are supported by our extensive experimental study of contour edges [11]. We describe preprocessing possibilities for contour edges, a fast silhouette computation algorithm, a new object-space hidden-surface removal algorithm for three-dimensional polyhedral surfaces, and its specialization for terrains.

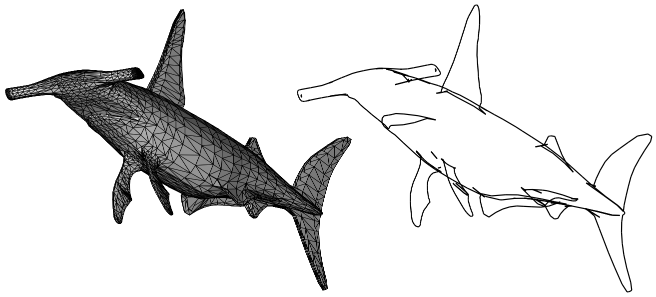


Figure 1: Contour edge example: a rendering showing all edges and the projected contour edges.

1 Introduction

Given a polyhedron in three-dimensional space and a view point, an edge of the polyhedron is called *contour edge* if one of the two incident facets is directed towards the view point, and the other incident facet is directed away from the view point (Figure 1). Algorithms on polyhedral surfaces can exploit the fact that the number of contour edges is usually much smaller than the number of edges.

In a classification of ten hidden-surface removal algorithms Sutherland introduced in 1974 the distinction between image-space and object-space algorithms [15]. Object-space methods are characterized to compute the exact answer to the hidden-surface removal problem, which link them to techniques used for visibility problems addressed in computational geometry. A recent survey by Dorward gives an extensive overview of object-space hidden-surface removal algorithms [6]. The result of an object-space hidden-surface removal algorithm is an exact description of the visible parts of a three-dimensional polyhedral surface. A common representation is the *visibility map* containing the projections of all visible parts including their incidence relations.

Object-space algorithms have various advantages over image-space hidden-surface removal algorithms. They are device independent, especially device resolution independent, and for high-resolution printers they may well be faster than image-space algorithms. Furthermore, they are useful for shadow computations, form-factor computations for radiosity, and viewshed analysis in GIS applications. The output can keep useful symbolic information that can be used, for example, to determine the drawing style for different edges.

Their disadvantages are best illustrated by the advantages of the popular image-space algorithm, the Z-buffer algorithm [4]. The Z-buffer algorithm is easy to implement, is robust, makes no assumptions on the input data, is easily made parallel, and is easily and effectively supported in hardware. A drawback of the Z-buffer algorithm is that it scan-converts each polygon before deciding on visibility. Conventional applications use therefore object-space preprocessing steps to reduce the number of polygons, e.g., back-face culling, view-frustum culling, or culling with large occluders.

Object-space hidden-surface removal algorithms can be classified as worst-case optimal algorithms with runtime $O(n^2)$, as intersection-sensitive algorithms, which run faster on cases where objects do not intersect quadratic times in the projection, and as output-sensitive algorithms. For terrains, consisting of n triangles, it is particularly easy to compute a depth order in $O(n)$, but the worst-case complexity for hidden-surface removal remains $\Theta(n^2)$, see the overview in [5]. The depth order is used in two output-sensitive algorithms for terrains [13, 9].

Contour edges in visibility computations were introduced by Appel in 1967 [1]. Hamlin and Gear described a sweep-line algorithm using contour edges to reduce the number of depth-comparisons [8]. Séquin and Wensley extended this to a broader class of input geometries [14].

We study in this paper a new approach to hidden-surface removal which is intersection-sensitive to contour edges (in contrast to edges or polygons in other methods). Let n_c be the number of contour edges, and let int_c be the number of their intersection in the projection. We compute the silhouette of a polyhedral surface in $O((n_c + int_c) \log n_c)$, and we solve the hidden-surface removal problem for terrains in $O((n_c + int_c) \log n_c + n + k)$ where n is the number of edges and k is the size of the output. We also discuss the problem in extending this approach to general scenes of polyhedral surfaces. The sensitivity to contour edges makes our algorithms particularly suitable for application domains where we can expect a low ratio of contour edges and intersections, e.g., polyhedral approximations of smooth surfaces in CAD/CAM, medical imaging, surface reconstruction, or GIS. We give ev-

*This research is part of the author's thesis at ETH Zürich, Switzerland [10]. Support from the Swiss Federal Office for Education and Science (Projects ESPRIT IV LTR No. 21957 CGAL and No. 28155 GALIA) is acknowledged.

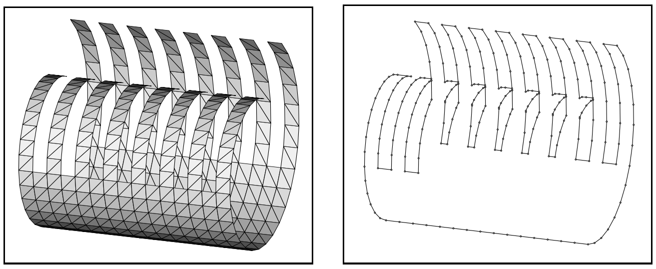


Figure 2: The silhouette of a `bracelet` example.

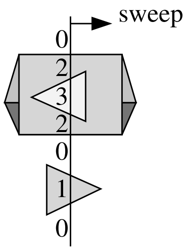


Figure 3: An example of the quantitative invisibility ν.

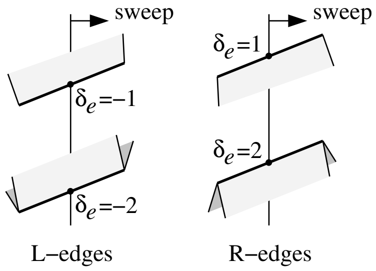


Figure 4: The classification into L-edges, R-edges, and their change δ_e in quantitative invisibility.

idence in [11].

Here, a polyhedral surface is an oriented 2-manifold, possibly with boundary. Each edge is incident to one or two facets. All polygonal facets have a non-zero area, are planar, and have an out-facing surface normal. A facet is *front facing* if its normal points towards the viewer, otherwise it is *back facing*. An edge is a *contour edge* if it is either a *border edge*, i.e., on the manifold boundary, or if it is incident to a front-facing and a back-facing facet.

2 Contour-Edge Preprocessing

A view-independent data structure of nearly linear size can be created in polynomial time which can report all contour edges for a given view point efficiently. For orthogonal projections the query time is $O(\sqrt{n}\log n + k)$ time, where n is the total number of edges and k is the number of contour edges reported. For perspective projections the query time is $O(n^{2/3}\,\mathrm{polylog}\,n + k)$. These results follow easily from results on simplex range searching, see [10] for the derivation.

A balanced aspect-ratio tree (BAR-tree) has been used in [2] for reporting contour edges. The authors report a significant performance improvement. Furthermore, they have investigated the incremental update of the contour edges for a moving view point.

3 Silhouette Computation

The *silhouette* of a polyhedron is the boundary of its projection. The silhouette consists of those edges in the projection that separate the visible facets from the background. This definition includes boundaries of holes through which the background can be seen (Figure 2). The silhouette is always a subset of the contour edges and it can be computed using only the contour edges. The exact solution provided by an object-space algorithm is useful in shadow casting.

We compute the silhouette with a Bentley-Ottmann type sweep-line algorithm that computes all intersecting pairs of projecte contour edges in the viewing plane [3]. We annotate the sweep-line status with the *quantitative invisibility*, defined in the next paragraph; moreover, the algorithm does not report the intersections but it reports those contour-edge fragments that form the silhouette.

The *quantitative invisibility* of a feature has been defined by Appel [1] as the number of facets hiding this feature from the viewer. We annotate the sweep-line status with the quantitative invisibility ν of the background (Figure 3), i.e., the number of faces stacked upon each other in this region. The quantitative invisibility ν is always greater than or equal to zero. It changes at contour edges and it changes only at contour edges. A condition for a projected edge in the sweep-line algorithm to be part of the silhouette is that the quantitative invisibility is zero for one of its two incident faces. Moreover, we observe that ν changes by plus/minus one at border edges and by plus/minus two at contour edges that are not border edges. See Figure 4 for the definition of L-edges, R-edges, and the $\delta_e \in \{-2,-1,1,2\}$, the amount of change of the quantitative invisibility at this edge.

Using δ_e we can easily compute the quantitative invisibility ν for all new edges at each event in the sweep-line algorithm. Altogether, no additional effort is necessary for the silhouette computation compared to the line-sweep algorithm.

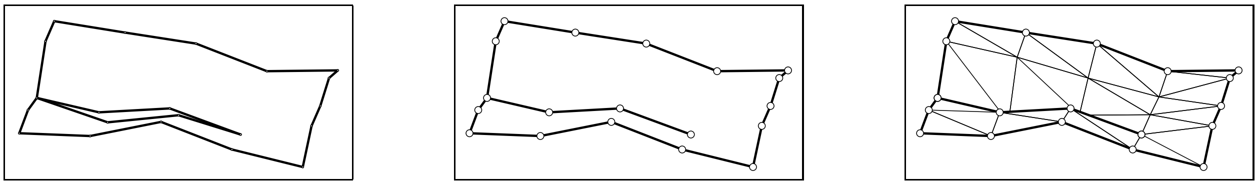
Theorem 3.1 (silhouette computation). *Given a set of n_c contour edges that intersect int_c times in the projection, the silhouette of a polyhedral surface can be computed with a Bentley-Ottmann type sweep-line algorithm in time $O((n_c + int_c)\log n_c)$.*

4 Hidden-Surface Removal

In 1967 Appel described the use of contour edges in an object-space hidden-line removal algorithm [1]. Considering the projection of the edges and facets into the viewing plane, the visibility of an projected edge can only change at a crossing with a projected contour edge. Contour edges had also been used for object-space hidden-surface removal algorithms [8, 14] which are essentially line-sweep algorithms that compute the visibility map in a single pass. Contour edges are only used in classifying intersections, not to reduce the complexity.

In our approach we extend these ideas to a three-pass algorithm (Figure 5): First, we extract the contour edges and project them into the viewing plane. Second, we compute the visible part of the projected contour edges. They form a nice partition of the viewing plane. Third, we fill each region in this partition with the visible facets. Since the visibility can only change at contour edges this step is easy given a visible facet in each region. The core part of the hidden-surface removal problem is the computation of the visible contour edges and adjacent visible facets in phase two. However, we will merge the third with the second phase later.

We use again a Bentley-Ottmann type sweep-line algorithm to compute all intersecting pairs of projected contour edges in the viewing plane. Here, we annotate the contour edges with their visibility and report the arrangement of vis-



(i) All contour edges. **(ii)** Visible contour edges. **(iii)** Filled with facets.

Figure 5: Illustration of the three-pass algorithm using a small synthetic terrain as an example.

	g	h		
1	Rv	Rv		$\beta_h \leftarrow \beta_g$
2a	Rv	Ri		if h is in front of β_g: $\beta_h \leftarrow \beta_g$, $\beta_g \leftarrow \mathrm{f}_h$
2b				if h is behind β_g: nothing changes
3	Ri	Rv		nothing changes
4	Ri	Ri		nothing changes
5	Rv	Lv		impossible at intersection events
6a	Rv	Li		if h is in front of β_g: see main text
6b				if h is behind β_g: nothing changes
7	Ri	Lv	symmetric to 6	
8	Ri	Li		nothing changes
9a	Lv	Rv		if h is in front of g: $\beta_h \leftarrow \mathrm{f}_g$
9b				if h is behind g: $\beta_g \leftarrow \mathrm{f}_h$
10	Lv	Ri		nothing changes
11	Li	Rv	symmetric to 10	
12	Li	Ri		nothing changes
13–16	L*	L*	symmetric to 1–4	

Table 1: Intersection events; v = visible, i = invisible edge.

ible contour edges in a suitable data structure, e.g., a planar map. Let f_e be the incident front-facing facet of a contour edge e. If the contour edge e is visible, we also annotate it with a pointer β_e referring to the visible facet on the other side of e than f_e, i.e., β_e refers to the next facet behind e. This annotation provides the required labeling of the regions in the planar map with visible facets needed for phase three.

The line-sweep algorithm proceeds from left to right. Symbolic perturbation enables us to simplify the event handling to two simple cases: intersection events with two contour edges intersecting in the projection and start/stop events for vertices of the polyhedral surface with incident contour edges.

The necessary updates of visibility, f_e, and β_e at an intersection event are given in Table 1. The classification into L- and R-edges and the visibility of the two projected edges, g and h, amount to 16 cases, wherefrom six cases are symmetric to other cases. Only three cases need to resolve to an actual depth comparison to decide on the required update. Among these, case 6a remains as an open problem for our approach to hidden-surface removal for general polyhedral surfaces.

Case 6a in Table 1 represents the situation where a hole opens behind two visible contour edges. The next visible facet in this hole needs to be found. But we give an example in [10] showing that there is not enough information in the arrangement of projected contour edges to solve this problem (independent of the algorithm). Thus, we would like to investigate if the class of objects suitable for the contour edge approach, namely polyhedral approximations of smooth surfaces, requires also only a low number of facet locations. For a restricted yet useful class of polyhedral surfaces, namely terrains represented as triangulated irregular networks (TIN), a better result is shown in the next section.

We continue with the discussion of the start/stop event handling. If we cannot conclude from incoming, visible edges that the vertex at this start/stop event is visible as well we compare the depth of the vertex with the facet currently visible at this position. Therefore, we fill all regions left to the sweep line with visible facets (phase 3 of our algorithm) in parallel while the sweep line proceeds. To locate the facet efficiently we fill the regions using again a sweep-line algorithm with a search tree on all facets intersecting the sweep line.

If a vertex is visible, the annotated information for its outgoing contour edges is updated with a rotational sweep around the vertex. The rotational sweep starts with the visible facet from vertically above the event point and rotates over all outgoing edges. For each outgoing edge, visibility is tested and the visible facet below the outgoing edge is determined. To illustrate the rotational sweep, we intersect the neighborhood of the vertex with an infinitesimal sphere. The rotational sweep is equivalent to a line sweep in the parameter space of the sphere. Since we allow border edges, the arrangement in the parameter space can be an arbitrary arrangement of segments. The sweep is straightforward, except that we might have holes in the surface with the same problems as for the intersection events. Ignoring the costs for holes, the total runtime of all rotational sweeps is in the order of $O(n \log n)$ (note that even invisible non-contour edges can be touched during this sweep).

Given a polyhedral surface of n edges and the set of n_c contour edges that intersect int_c times in the projection for a given viewing transformation, the visibility map can be computed with a Bentley-Ottmann type sweep-line algorithm in time $O((n_\mathrm{c} + int_\mathrm{c}) \log n_\mathrm{c} + n \log n + k \log k)$ plus the time needed for the extra visible facets locations, where k is the size of the output.

5 Specialization for Terrains

The computation of the visible contour edges simplifies considerably for a polyhedral terrain represented as a triangulated irregular network (TIN). Let the terrain be a function in z over the xy-plane with a right-handed coordinate system. Let us assume a perspective projection where the $-z$ direction projects to a finite vanishing point in the projection plane. Any sweep-line direction that moves towards this vanishing point but does not cross it does not create intersection events of type case 6a from Table 1 (see [10] for the proof).

Furthermore, the rotational sweep at start/stop events also simplifies considerably. We start with the visible facet above the visible vertex and trace the visible surface until we encounter a contour edge (an R-edge) that is closer to the viewer and continue with the visible surface beginning at this contour. Note that the priority queue can give us the contour edges in the correct sorted order and that all the additional non-contour edges traced in this sweep are visible edges. Thus the rotational sweep requires constant time per contour edge plus constant time per visible edge.

Boundary edges can be handled easily if we change the modeling of the problem slightly. We extend each boundary edge of the terrain with an infinite curtain hanging from this edge parallel to the z-axis to $z = -\infty$. With this conceptual change of the terrain model, the surface of the terrain will not be visible from the underside.

Theorem 5.1 (terrain hidden-surface removal). *Given a terrain and the set of n_c contour edges for a given viewing transformation, possibly including border edges, that intersect int_c times in the projection, the visibility map can be computed with a Bentley-Ottmann type sweep-line algorithm in time $O((n_c + int_c)\log n_c + k \log k)$, where k is the size of the output (see [10]) for the proof).*

6 Results and Conclusion

We have presented applications of contour edges for three-dimensional polyhedral surfaces. Contour edges are also a prominent visual feature on their own. Necessary implementation details have been symbolic perturbation and exact geometric predicates based on exact arithmetic.

We have implemented two prototypes: the silhouette computation and the hidden-surface removal for terrains under orthogonal projections. The terrains are required to have a boundary which projection to the xy-plane is convex. This further simplifies boundary edge handling. Symbolic perturbation simplified our second implementation considerably. In addition, in our examples it is roughly twice as fast as the explicit handling of degeneracies used in the first prototype. We give a few runtime results to indicate the order of magnitude for our algorithm implemented in C++ using generic programming and templates for flexibility and efficiency [7, 12].

We give the running time for the computation of the visible contour edges only and the computation of the hidden-line image for terrains based on our first implementation. The test data consists of a data set from Albis and Türlersee, `Albis`, and a data set from the Matterhorn mountain in Switzerland, `Matter`, both with tolerances from 10 meters to 1 meter. The test machine is a Linux PC with a Pentium II 300 MHz processor. The following table lists for each data set the number of edges, the expected number of contour edges and their intersections in the projection, and the runtime range observed in seconds per frame.

data set	n	$\overline{n_c}$	$\overline{int_c}$	visible contour	hidden line
`Albis 10`	5256	399	158	0.02–0.07	0.14–0.20
`Albis 5`	13819	845	393	0.05–0.13	0.30–0.38
`Albis 2`	34582	1602	829	0.11–0.24	0.71–0.87
`Albis 1`	58821	2141	1159	0.19–0.53	1.1–1.3
`Matter 10`	18924	1819	1323	0.08–0.14	0.40–0.48
`Matter 5`	31227	2542	1874	0.12–0.27	0.60–0.76
`Matter 2`	51076	3302	2397	0.18–0.40	1.0–1.1
`Matter 1`	65965	3629	2770	0.24–0.50	1.1–1.4

Acknowledgments

The author wishes to thank Emo Welzl for the freedom, the encouraging guidance and the strong support throughout these studies. I wish to thank Jack Snoeyink for reading and commenting on previous versions and Bernd Gärtner, Michael Hoffmann, Andreas Rosenthal, Sven Schönherr, and Michael Schutte for the many fruitful discussions in the meetings in Berlin that have let to these results.

References

[1] A. Appel. The Notion of Quantitive Invisibility and the Machine Rendering of Solids. In *Proc. ACM National Conf.*, pages 387–393, 1967.

[2] G. Barequet, C. A. Duncan, M. T. Goodrich, S. Kumar, and M. Pop. Efficient Perspective-Accurate Silhouette Computation. In *Proc. 15th Annu. ACM Sympos. Comput. Geom.*, pages 417–418, 1999. 8th Annual Video Review of Computational Geometry.

[3] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28(9):643–647, September 1979.

[4] E. Catmull. A Subdivision Algorithm for Computer Display of Curved Surfaces. Technical Report UTEC-CSC-74-133, Dept. Comput. Sci., Univ. Utah, Salt Lake City, 1974.

[5] M. de Berg. Visualization of TINs. technical report UU-CS-97-22, Department of Computer Science, Utrecht University, The Netherlands, 1997.

[6] S. E. Dorward. A Survey of Object-Space Hidden Surface Removal. *Int. J. of Comput. Geom. & Appl.*, 4(3):325–362, 1994.

[7] A. Fabri, G.-J. Giezeman, L. Kettner, S. Schirra, and S. Schönherr. On the Design of CGAL, the Computational Geometry Algorithms Library. *Software – Practice and Experience*, 1999. To appear.

[8] Jr. G. Hamlin and C. William Gear. Raster-Scan Hidden Surface Algorithm Techniques. In *Computer Graphics (Proc. SIGGRAPH '77)*, volume 11, pages 206–213, 1977.

[9] M. J. Katz, M. H. Overmars, and M. Sharir. Efficient hidden surface removal for objects with small union size. *Comput. Geom. Theory Appl.*, 2:223–234, 1992.

[10] L. Kettner. *Software Design in Computational Geometry and Contour-Edge Based Polyhedron Visualization.* PhD thesis, ETH Zurich, Switzerland, 1999. DISS. ETH No. 13325.

[11] L. Kettner and E. Welzl. Contour Edge Analysis for Polyhedron Projections. In W. Strasser, R. Klein, and R. Rau, editors, *Geometric Modeling: Theory and Practice*, Focus in Computer Graphics, pages 379–394. Springer Verlag, 1997.

[12] D. R. Musser and A. A. Stepanov. Algorithm-oriented Generic Libraries. *Software – Practice and Experience*, 24(7):623–642, July 1994.

[13] J. H. Reif and S. Sen. An efficient output-sensitive hidden-surface removal algorithms and its parallelization. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 193–200, 1988.

[14] C. H. Séquin and P. R. Wensley. Visible Feature Return at Object Resolution. *IEEE Computer Graphics and Application*, 5:37–50, May 1985.

[15] I. E. Sutherland, R. F. Sproull, and R. Schumaker. A Characterization of Ten Hidden-Surface Algorithms. *ACM Computing Surveys*, 6(1):1–55, 1974.