

Project: Hidden-Surface Removal for Terrains

Susan Hert Lutz Kettner Abhishek Tiwari Akash M. Kushal

Max-Planck Institut für Informatik, Saarbrücken

1 Introduction

Algorithms for hidden-surface removal can be classified into image-space and object-space algorithms according to Sutherland, Sproull, and Schumaker [?]. Katz, Overmars, and Sharir give in [?] an object-space algorithm that is particular suitable for terrains represented as a triangulated surface.

The task was to implement the algorithm specialized for terrains following [?]. CGAL, the Computational Geometry Algorithms Library, was used as foundation.

2 The Algorithm

The algorithm follows the following steps,

- As a first step, the method sorts the objects by depth order and stores them in this order in the leaves of a binary tree \mathcal{T} .
- Then it recursively computes for each internal node of the tree the union of the projection of the objects in the sub-tree \mathcal{T}_δ of \mathcal{T} rooted at δ . This is called U_δ .
- It proceeds to compute the visibility maps of the objects by doing a preorder tree traversal. For each node it computes V_δ which is the visible portion of U_δ , i.e. the subset of U_δ consisting of those points that are not contained in the projection of any nearer object.
- The V_δ s are computed as follows,

$$V_{root} = U_{root},$$

$$V_{leftchild(\delta)} = V_\delta \cap U_{leftchild(\delta)},$$

$$V_{rightchild(\delta)} = V_\delta - U_{leftchild(\delta)}.$$

- The individual visibility maps (V_δ s) are glued together to get the final visibility map.

The time complexity of the algorithm is $O((U(n) + k)\log^2 n)$

The algorithm is explained in detail in [?].

3 Implementation Details

The program takes as input a polyhedral surface in OFF format and a vector for the viewing direction.

3.1 Validity check for terrains

This part of the code checks whether the input polyhedral surface is a valid terrain or not.

First all the facet normals of the polyhedron are checked to be along one of the 6 coordinate directions. In case this is not the positive z direction the terrain is reoriented so that the normals are now oriented along the positive z direction.

Definition: A *polyhedral terrain* is the graph of a piecewise linear continuous function $z = \Sigma(x, y)$.

Theorem: A polyhedral surface with no holes and a convex boundary is a terrain.

The convexity check involves checking the

- **Local convexity condition** The boundary must be locally convex at each vertex. There must be a right turn at each vertex of the boundary of the terrain while traversing the vertices in the clockwise direction.
- **Global convexity condition** Let o be the center of gravity of all the vertices of the boundary of the polyhedron, and let p be the center of gravity of some side of the boundary of the polyhedron. Then, the boundary of the polyhedral surface is convex iff
 - o is on the negative side of all its sides, and
 - the ray emanating from o and passing through p intersects only the side containing p .

For checking for the presence of holes, number of edges on the outer boundary of the polyhedron is found and compared with the total number of boundary half edges. The latter number should be double the former in case of no holes.

3.2 Depth sorting

Topological sort is done on the facet graph of the terrain to obtain the depth order among the facets. The algorithm for Depth First Search is used for this purpose. The depth first search is performed on the graph of facets of the polyhedral terrain.

The facet graph is read directly from the Polyhedron datastructure of CGAL. No new graph is constructed for this purpose. The edges of this graph point from a facet to a neighbouring facet which is farther away in the viewing direction.

To find the depth order among neighboring facets, cross product of the border half-edge and the viewing direction is taken. If the z component of this cross product is negative then the face lying to the left of this half-edge is the nearer facet. An edge of the facet graph point from the nearer facet to the farther facet.

3.3 Constructing a balanced binary tree and calculating unions

The projections of the facets of the terrain thus sorted by depth order are stored in the leaves of a balanced binary tree, from left to right. Each node of the binary tree stores the U_δ and V_δ maps.

- **Projecting facets on Viewing Plane:** Each facet of the terrain has to be projected along the viewing direction. The plane perpendicular to the viewing direction is called the viewing plane. The projected facets are stored in the Nef_polyhedron data structure of CGAL.

- **Choose coordinate axes.** The co-ordinate axes for the viewing plane have to be suitably chosen. For this 3 cases arise according to the orientation of the view vector.

Case 1: View vector not parallel to the global z-axis. The new x-axis must be perpendicular to both the global z-axis and the view vector. Hence, it is taken as the cross product of view vector and the global z-direction. As the negative of the view vector is the normal of the viewing plane, the new y-axis is the cross product of the new x-direction and the view vector.

Case 2: View vector along negative z-axis. The new x- and y- axes are the same as the global x- and y- axes.

Case 3: View vector along positive z-axis. The new x-axis is the negative of the global x-axis. The new y-axis is the same as the global y-axis.

- **Project facet on the viewing plane.** Each point of the facet is projected on the viewing plane. The x- and y- components of each point in the viewing plane are found by taking the dot products with the new x- and y- directions on the viewing plane.

As a result of this projection, the order of the vertices of the projected facet may get reversed. In such a case, while traversing the vertices in the given order, the facet will not lie to the left of each side. Therefore, the Nef polyhedron obtained will be the complement of the required facet.

To identify this case, the dot product of the viewing vector with the normal of the facet is computed. If the dot product is positive, the order of the vertices is reversed. A new Nef polyhedron is thus obtained from the ordered vertices projected on the viewing plane.

- **Constructing a Balanced Binary Tree:** A balanced binary search tree is constructed from the Nef_polyhedra for the projections of the individual facets of the terrain in its leaves. This tree is constructed recursively starting from the leaves. Each node δ of this binary tree contains U_δ (the union of all the leaves in the subtree \mathcal{T}_δ rooted at δ), V_δ (the visible part of U_δ), pointers to the left child and the right child.
- **Computing unions:** As the tree is constructed recursively, the U_δ at each node is computed simultaneously from the union of the U_δ s of the left child and the right child. For computing the unions the member function 'join' of Nef_polyhedron is used.

3.4 Computing visibility maps

The visibility maps are constructed by doing a preorder traversal on the balanced binary tree constructed above. The visibility map of the right child is nothing but the intersection of the parent's visibility map and the union stored in the left child. The visibility map of the left child is the difference of the visibility map of the parent and the union stored in the left child.

Once the visibility maps for the children of a node δ have been computed, the V_δ is no longer needed. Also the $U_{leftchild(\delta)}$ and $U_{rightchild(\delta)}$ are no longer needed. Hence, these are deleted, once the node δ has been visited.

3.5 Visualising Output

The individual visibility maps of the facets of the terrain are obtained in the V_δ s of the leaves of the binary tree at the completion of the algorithm. The individual visibility maps are stored as planar Nef polyhedra. These can be glued together to get the final visibility map for the terrain, in the given viewing direction.

The individual visibility maps for the facets thus obtained cannot contain any holes. Also, they will be formed of x-monotone pieces. Hence, these pieces can be triangulated easily and written in OFF format to a file.

The individual Nef polyhedra for the visibility maps can also be visualized using the Window stream interface. This interface is provided by LEDA windows. To verify the correctness of the results obtained, this interface was used to visualize the final output.

The input terrain was visualised in 'geomview' in different orientations. The geomview global coordinates transformation matrix was used to get the viewing vector. The transformation matrix was printed using the command, (`write transform - g0`). The negative of the third column of this matrix was taken as the viewing vector for the terrain.

The unions of the edges(boundaries) of the final visibility maps are taken. This output is rendered in the Window stream window. This is compared with the visibility map of the orthographic projection of the terrain in the 'geomview' window. Both these maps match within a rotation in the viewing plane.

3.6 Generating Random Terrains

The algorithm used to generate the random terrains of different sizes for input to the program was

- A grid of the required size is taken and the 4 end points of the grid are randomly displaced vertically by an amount proportional to the side length of the base of the grid.
- The midpoints of the sides of the square are assigned height values equal to the average of the two end points plus a random disturbance again proportional to the side length.
- The midpoint of the square is also assigned a height value equal to the average height of the 4 points of the square plus a random disturbance proportional to the side length.
- The above steps are carried out recursively until all the points in grid are assigned height values.
- Then the grid points are triangulated to get a terrain.

These terrains are then output to an OFF file which can be used as input for the main program.

4 Using the Program

Syntax:

```
$ hsr <filename>
```

The input file is a polyhedral surface in OFF format. Subsequently, the program will prompt for the viewing direction. The user should enter the x, y, and z components of the viewing direction in that order.

5 Results

Random terrains were generated in OFF format and the program's running time was noted to verify the theoretically expected complexity. Two different Kernels in CGAL were used - Extended homogeneous and Filtered Extended homogeneous. The timing results are shown in the tables below.

Trial	Number of faces	Time taken(in Seconds)	
		Filtered Extended homogeneous	Extended homogeneous
1	128	3.77025	9.66545
2	128	3.23978	9.79559
3	512	13.6102	41.1298
4	512	13.5592	41.2378
5	2048	58.3779	174.247
6	2048	58.46	174.205
7	8192	247.66	736.098
8	8192	248.343	736.811
9	32768	1094.3	3149.67
10	32768	1096.79	3148.3

Table 1: Timing Results

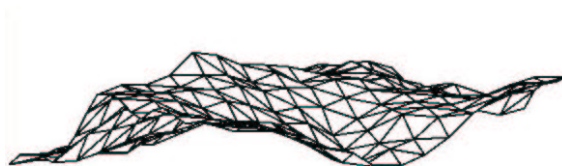


Figure 1: Terrain with 512 Triangles

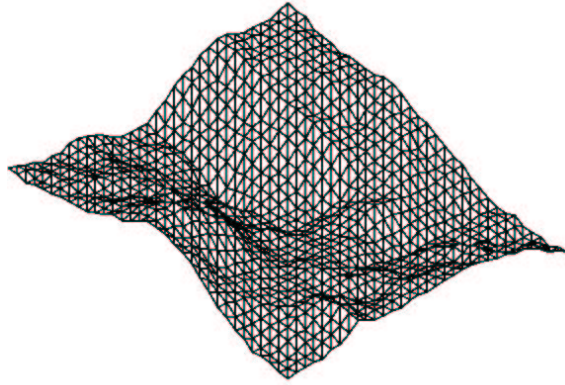


Figure 2: Terrain with 2048 triangles

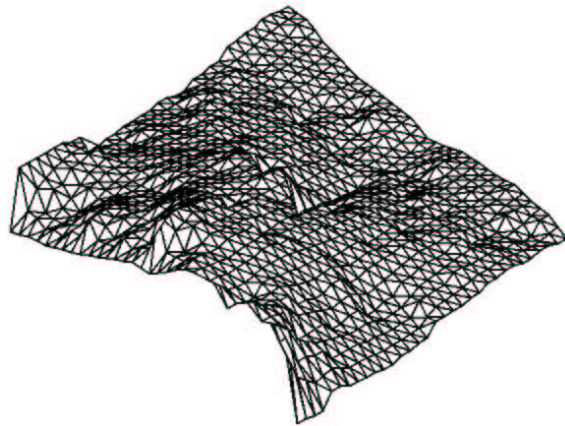


Figure 3: Terrain with 2048 Triangles

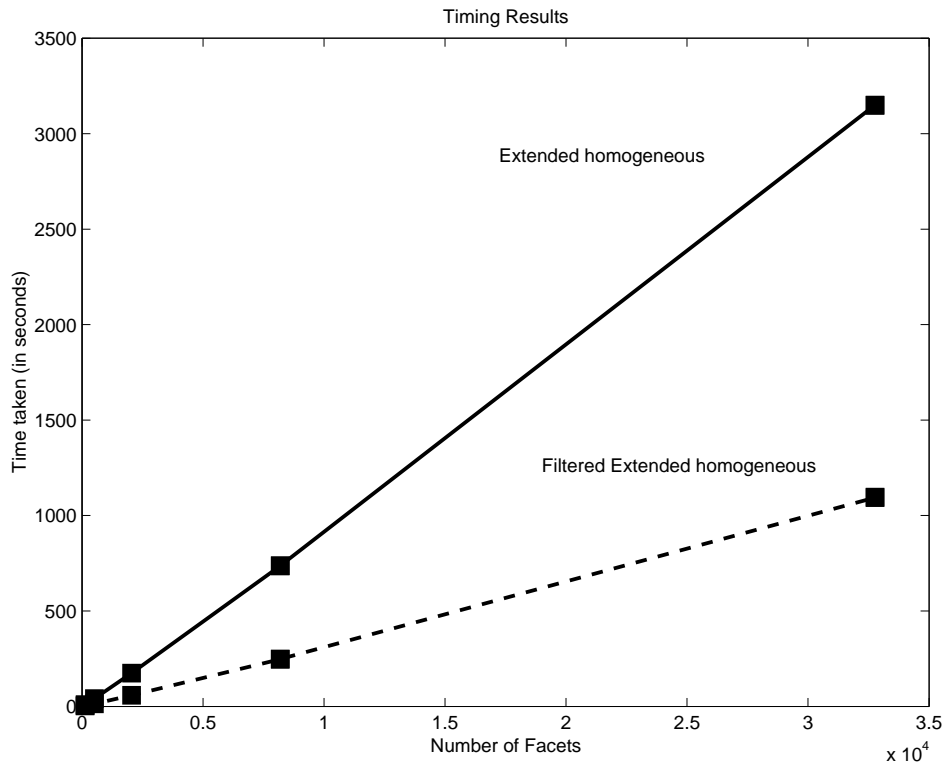


Figure 4: Comparison between Extended homogeneous and Filtered Extended homogeneous Kernel Implementations

6 Conclusion

From the above results we can conclude that Filtered Extended homogeneous kernel implementation was approximately 3 times faster than the Extended homogeneous implementation. The running time was experimentally verified to be $O(n \log n)$.