Precomputed visibility ordering for a constrained viewpoint

Chrysanthou, Yiorgos

Precomputed visibility ordering for a constrained viewpoint

Y. Chrysanthou

Department of Computer Science,
QMW University of London,
Mile End Road,
London E1 4NS, UK.

e-mail: yiorgos@dcs.qmw.ac.uk

Abstract

An algorithm for determining an invariant visibility ordering valid from any point on a predefined area is presented. The application for which this algorithm was initialy developed is the ordering of polygons in respect to an area light source and it is presented in that context.

1 Determining the Order in Respect to an Area

One of the principle uses of the BSP tree is for ordering the scene polygons, either for visible surface determination from the viewpoint or from the point light source for shadow calculations. Ordering from area light sources is not as easy. We are not dealing with just one point but a finite area that might fall on both sides of a plane defined at a BSP node.

Previous researchers dealing with area light sources realised this problem but being unable to find a satisfactory solution, their methods resulted in unnecessary processing (see *Section* ??):

- Campbell and Fussell [1] did not attempt at all to find an order, assuming that it is not possible, their method requires two passes over the polygon set. One for building the unified shadow volumes, umbra and penumbra, from an unsorted set of polygons using merging and the second for calculating the shadow boundaries by inserting the polygons in these volumes.
- Chin and Feiner [2] simplify the problem of ordering in respect to the area source to one of ordering from a point by splitting the light source whenever it is found straddling the plane of a scene polygon. Thus they can combine the two passes mentioned above into one but performed multiple times, once for each source fragment.
- Gatenby and Hewitt [5], use the BSP tree to find the polygons lying between the source and a receiver but not by getting an absolute order. Rather they take the union of the polygon sets derived by traversing the BSP tree from each source vertex to the receiver and get an over-estimation of the set of polygons between the receiver and the whole of the source.

The reason traversing a BSP from an area light source (or viewing area) is not the same as for traversing the tree from a viewpoint is because the viewing area cannot necessarily be unambiguously classified against the root plane at each node. Take for example the simple scene of Figure 1(a). Traversing the tree of Figure 1(b) front-to-back from different points on A gives different orderings: a1 gives {2, 1, 3} while a2 gives {3, 1, 2}. But the different orderings do not imply that there is a cycle or that an order valid for all points on A cannot be found. In fact because we are dealing with oriented polygons and we are only considering a limited viewing space, commonly there will be an invariant ordering, for example {1, 3, 2} here.

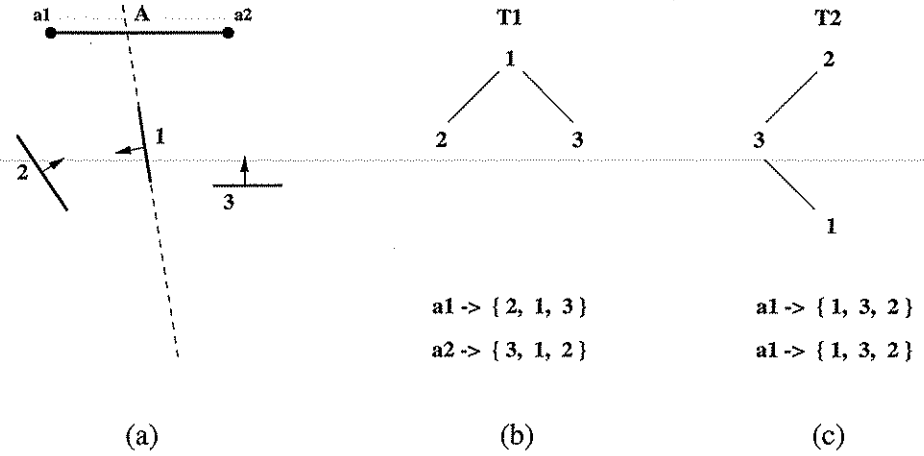

Figure 1: (a), (b) When the plane of an internal node cuts the area (A) different orderings are produced for different points on the area. (c) If the cutting node is placed at the leaves then the ordering is the same for every point on A

As we can see in Figure 1(c) a different tree can be constructed that when traversed gives that ordering from any point on A. The reason why a tree like T_1 does not work is because different points on A lie in different subspaces of polygon 1 and hence produce different orderings on the children of 1. In T_2 this still holds but since both children of 1 are now empty their ordering is irrelevant.

So an apparent solution to the problem of ordering from a viewing area, is to push all polygons that intersect the viewing area to the leaves of the tree and the traverse it from any point on the area.

1.1 Definitions

An interesting study related to this problem can be found in the list priority algorithms for visible surface determination that use pre-processing to induce priority relations among the scene polygons. Of particular interest is the work of Schumacker [7, 8]. He applied graph theory to find an invariant order for a set of polygons (a *cluster*). Along the same lines was also the work of Fuchs [4] and Naylor [6].

Before proceeding let us refrase some adapted definitions and theorems from the above literature. In what follows s_i and s_j denote polygons:

Definition 1 *Actual visibility obstruction: s_i can have an actual visibility obstruction on s_j, with respect to a viewing vector v, if the vector goes through points t_i on s_i and t_j on s_j, and t_i is closer to the origin of v than t_j, and both polygons are front facing w.r.t. the origin of v.*

The actual obstruction gives us the visibility priority of two polygons from a particular viewpoint. This can be generalised by the following definition to give the priority relation of two polygons from any point in space.

Definition 2 *Potential visibility obstruction: s_i can have a potential visibility obstruction on s_j if $\exists$ a vector v that relates s_i and s_j under actual visibility obstruction.*

Potential visibility obstruction makes no assumptions on the position of the viewpoint so any ordering obtained under that relation is valid for the whole space. This relation between two polygons can be determined using the following theorem.

Theorem 1 *s_i can have a potential visibility obstruction on s_j if and only if $\exists$ a point on s_i in the front half-space of s_j and $\exists$ a point on s_j in the back half-space of s_i.* (See [6] for proof of the theorem).

1.2 Small Viewing Areas

In the research mentioned above, the aim was either to find an invariant order for the whole of 3-D space, or one order for each viewpoint (or sets of viewpoints). Here we are dealing with a sub-problem since we are only interested in the set of viewpoints lying on a pre-defined planar polygonal space. Note that in what follows we assume this area to be convex. For a concave area, it will still hold if we use its convex hull.

Let us now state clearly what it is we will try to achieve:

Given a set of polygons $S = \{s_1, ..., s_n\}$ and a viewing area A we want to build a tree T^* that when traversed from left to right[1] will give a front-to-back priority ordering $<\sigma_1, ..., \sigma_k>$ valid from any point on A, where $\forall\sigma_i$, $\sigma_i \subseteq s_j$, some $1 \leq j \leq n$. For convenience we will call such a tree *ordered.*

This ordered tree is equivalent to a linear list of the polygons as seen front-to-back from A. As we will see later it is not always possible to generate this tree without splitting A since cycles may be present. In general, however, under the conditions we will be assuming it will be feasible. Any unbreakable cycles will be reported.

The method has two stages. First a BSP tree T' is constructed in the conventional way but only with the polygons in S that have A totally in their front half-space. The polygons that intersect A with their plane are stored in a list L_o while those that embed A or have A totally behind their plane are ignored. We will refer to the polygons that have A totally in-front as *area-facing* and to those that cut A with their plane as *offending.* Clearly traversing T' from any point on A gives the same ordering. Any point on A is in-front of the plane defining each node so traversing the tree from left-to-right is equivalent to a front-to-back traversal.

The second step is to construct T^* by inserting the polygons in L_o into T'. It is easier to understand why T^* is ordered by thinking of T' as an ordering of subspaces C_i (cells) as seen from A. Any additional polygons inserted into T' will further subdivide the cells but cannot change their relative priorities. Thus it suffice to show that polygons within each C_i are ordered.

Since the area we are considering is the light source, which is relatively small, the number of polygons cutting it with their plane are expected to be few compared to those facing it. So the polygons in L_o will probably reach different cells or maybe few in the same cell. When an offending polygon is the only one to reach a leaf node of T' then we just create a tree node and add it there. For cells with more than one offending polygons, we use a graph theoretical approach, described below, to order them and produce a subtree to replace the cell.

Each polygon in the cell forms a node of the graph and has *arcs* connecting it to all other graph-nodes that it has priority over. The priority relation is determined using *Theorem 1.* We will denote this relation using $\rightsquigarrow$, $s_i \rightsquigarrow s_j$ meaning that s_i has a potential visibility obstruction (PVO) on s_j. Once the graph is built it is searched using a depth first search (DFS) to order the nodes and check for cycles. If cycles are detected a second DFS is performed on the transpose of the graph (the arcs are reversed) to locate the cycles and reduce it to its strongly connected components [3]. In the scenes we used for *Chapter* ?? this second search was never needed. If the cycle is a trivial one involving only two polygons (Figure 2) then it can be resolved by cutting one of the polygons along the plane of the other. A method for dealing with other cycles is suggested in *Section 1.4.*

Building a tree using the arcs in the graph is an $O(n^2)$ process, where n is the number of polygons involved, but as these are very few this is not detrimental to the speed of the whole algorithm.

Such a graph theoretical approach was used by Schumacker [7] to order the polygons in each cluster. It was also used in [6] for generating a total ordering on the whole set of polygons but it was found impractical due to the large number of cycles involved. Our problem, however, is not as demanding since we are only interested for visibility priority from a limited area.

The tree we have constructed here excludes any back-facing polygons, with respect to A. This is not a requirement of the algorithm and if the tree is to be used for other operations then the

[1] Assuming that the left is the front subtree and the right is the back

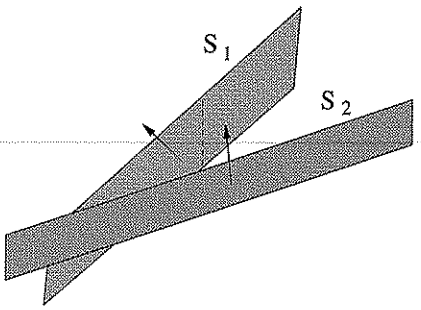

Figure 2: Cycle of two polygons

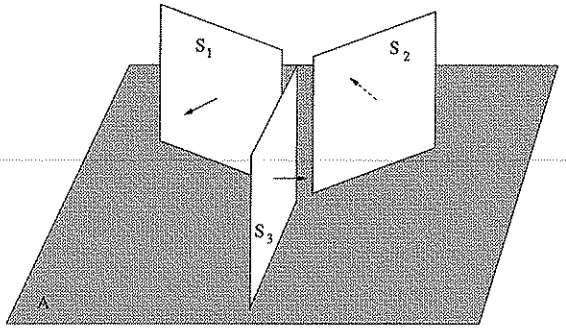

Figure 3: Cycle of more than two

complete set of polygons can be included. In this case the back-facing polygons are marked so as to traverse the right subtree before the left, when getting the order with respect to A.

The order for visible surface determination from any point in space, can be obtained by a normal back-to-front BSP traversal (classifying the viewpoint at each node) on the nodes of T' and right-to-left on the offending subtrees.

To use such a complete ordered tree from a different viewing area A', all that is needed is to identify the offending polygons for A' and push them down to the leaves. They do not have to be reinserted at the root of the tree but rather from the node where they lie. The priorities on the previously-offending subtrees are still valid but if the addition of the now-offending polygons creates cycles they will have to be rebuilt.

1.3 Maintaining the Order During Interaction

For interaction we assume the use of the algorithm described in *Section* ??: the polygons of the transformed object are removed from the tree and they are added back at their new positions. When removing polygons from leaf-nodes or from nodes with one non-empty subtree it can be shown trivially that the ordering in the tree is not affected. But while putting two subtrees together in the *restore* function or as the new polygons are added, area-facing polygons might reach an offending subtree. To maintain the assumptions we made earlier, area-facing polygons must go before any offending polygons in the tree. So when inserting an area-facing polygon, if it meets a subtree with offending polygons then the area-facing one is placed at the root of the subtree and the offending polygons are pushed down past it one by one.

1.4 A Generalisation of the Method

As mentioned earlier the visibility is restrained to a particular area, even if the area is large, the fact that it is limited and planar places an extra constraint on the visibility relation which can be used to resolve most of the cycles.

The arcs in our graph denote the potential visibility obstruction. This relation is very broad, we are not interested in whether there exists a point in space for which two polygons can have an actual visibility obstruction but what we are interested is if there exists such point on A. We will give a new name to this relation in the following definition:

Definition 3 *Distributed visibility obstruction: two polygons, s_i and s_j, are related under distributed visibility obstruction with respect to a planar polygon A, if $\exists$ a point p on A such that s_i and s_j are related under actual visibility obstruction with respect to that point.*

Obviously, distributed obstruction is a subset of potential visibility obstruction. It cannot create any extra arcs that are not present already. But by validating the existing arcs against this new relation their number can be reduced and cycles resolved. The existence of distributed visibility obstruction (DVO) can be determined by the following theorem:

scene	N_pol	F_area	O_area	Max_O	N_cycles
office1	139	58	17	4	0
office2	211	96	19	3	0
office3	333	147	28	4	0
office4	751	336	65	4	0
office4*	751	327	70	6	0

Table 1: Visibility from the light source

Theorem 2 *Given 3 polygons A, s_i and s_j, s_i can have distributed visibility obstruction on s_j if s_i has potential visibility obstruction on s_j and $\exists$ a point on s_j behind all of the extremal penumbra planes from A and s_i.*

Proof: By definition (see *Section* ??), the extremal penumbra planes have the "source" totally in the front half-space and the "occluder" totally in the back half-space. Here the source is A and the occluder is s_i. If s_j falls in front of any penumbra plane then $\{A, s_j\}$ and $\{s_i\}$ are linearly separable by that plane. This implies that there can be no line going through a point on A and a point on s_j that can be intersected by s_i. Thus there is no point on A from which s_i and s_j can have an actual obstruction. □

Using the same reasoning we can also show that DVO between A, s_i and s_j can exist only if A has some intersection with the subspace defined by the intersection of the extremal penumbra planes of s_i and s_j and the front half-space of s_i. This penumbra structure is similar to the *obstruction polyhedron* defined in [6].

The test for distributed obstruction is more expensive than checking for potential obstruction. If we are expecting only few cycles then we can build the arcs in the graph using only PVO, and only apply DVO to verify them after the polygons involved in the cycle are found. Also we can create fewer splits if we verify an obstruction before cutting a polygon involved in a cycle of size two, mentioned in the previous section.

We might find examples, such as the one in Figure 3, where cycles occur that can not be resolved by splitting the polygons or otherwise. In this case the actions taken depend on the application. For example in the area sources algorithm described in *Chapter* ??, we can assign the same priority number to all elements of a cycle. When deciding the shadow relations, polygons with the same numbers will be compared both ways (see the inner loop in Figures ?? and ??).

If this is to be used for rendering with the viewpoint moving on A, then we can find the polygon(s) whose removal will break all the cycles and use their planes to split A. The BSP nodes of these polygons are marked and their planes are checked when crossing from one fragment of A to another. The subtrees of the node whose plane is crossed are then switched (left goes right and vice-versa).

To use a tree constructed using DVO from a different viewing area A' the offending subtrees should be rebuilt. Also it can not be used for rendering from any viewpoint not on A.

1.5 Results

For evaluation of the method the office scenes were used. A relatively large light source was placed at the center of each room, near the ceiling and the tree was built with respect to that source. In Table 1 we give statistics on the total number of polygons (*N_pol*) in the scene, the number of polygons that have the area totally in their front half-spaces (*F_area*) and number of polygons that cut the source with their plane (*O_area*). The next field (*Max_O*) gives the maximum number of offending polygons in any cell of T'. As we can see this number is very small, not exceeding 6 in any of the scenes. Finally we give the number of cycles found. For determining the visibility relations between the offending polygons, we used only potential visibility obstruction (*Theorem 1*). We can see that no cycles were present in any of the scenes.

scene	N_pol	F_area	O_area	Max_O	N_cycles*
office1	139	51	41	8	0
office2	211	74	64	11	0
office3	333	110	108	31	0
office4	751	240	241	66	0
office4*	751	227	259	46	1

Table 2: Visibility from a light source with half the dimentions of the ceiling

scene	N_pol	F_area	O_area	Max_O	N_cycles*
office1	139	26	91	32	0
office2	211	39	143	64	0
office3	333	59	225	96	0
office4	751	129	497	227	17
office4*	751	122	509	191	25

Table 3: Visibility with the ceiling as the source

The corresponding trees for these scenes office1, office3 and office4 are shown in Figures 4 and 5. To show clearly the double structure of the tree, the BSP (T') nodes are marked with white squares (these are from the area-facing polygons) while the graph theory based nodes are marked with black squares (the offending polygons).

To see how the algorithm behaves when we use large sources (or viewing areas), two more sets of experiments were run: one with a source having sides equal to half that of the ceiling and one with using the ceiling as the source. The data of these experiments are shown in Table 2 and Table 3. The columns in these tables are the same as in Table 1, apart from the last column (*N_cycles**). This corresponds to the number of cycles in the tree with the arcs created on distributed visibility obstruction (*Theorem 2*) rather than potential obstruction.

As the size of the source becomes larger the number of polygons intersecting it with their planes (offending) grows rapidly (*O_area*). Also the number of offending polygons reaching the same cell of T' grows (*Max_O*), making their ordering, using only potential obstruction impossible. This is the same problem recorded by Naylor in [6]. However, using the extra constraint provided by DVO, we can see that almost all of the cycles are resolved. Only in experiments with office4 and office4* do cycles remain. The reason for that is that the area used for visibility (the source) here is much larger than in the other experiments and in particular it is very large compared to its distance to the other objects in the scene. In further experiments using the same data but pushing the ceiling to twice the height we found that all the cycles disappeared.

The tree for office4 in Table 2 is shown in Figure 6 and for office1, office3 and office4 of Table 3 in Figures 7 and 8. As we can see the double structure of the tree is almost lost in the trees for Table 3. Also in the latter tree cycles were present. These cannot be seen on the tree because we stored all polygons in each cycle on the same node. So the nodes of the tree are ordered but some of them hold polygons that form a cycle.

We will not give the timings for building the tree as the purpose of the experiment was not to build the tree fast but to show that an ordering in respect to an area was attainable. Also in applications the tree would be built in the pre-processing stage so it would not affect the run-time performance. But to give an idea of the times involved we can say that for Table 1, where the reference area is a normal light source, the timings were less than building a normal scene BSP tree. This is because the back-facing polygons were excluded. For Table 2 it took slightly longer, while for Table 3 it took many times longer than for building a normal tree. These results were predictable since, as we said in *Section 1.4*, ordering the polygons in a cell is an $O(n^2)$ operation and in Table 3 n is very large.

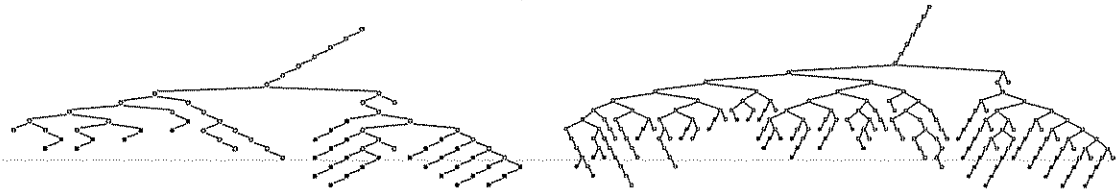

Figure 4: Trees of office1 and office3 from the area light source

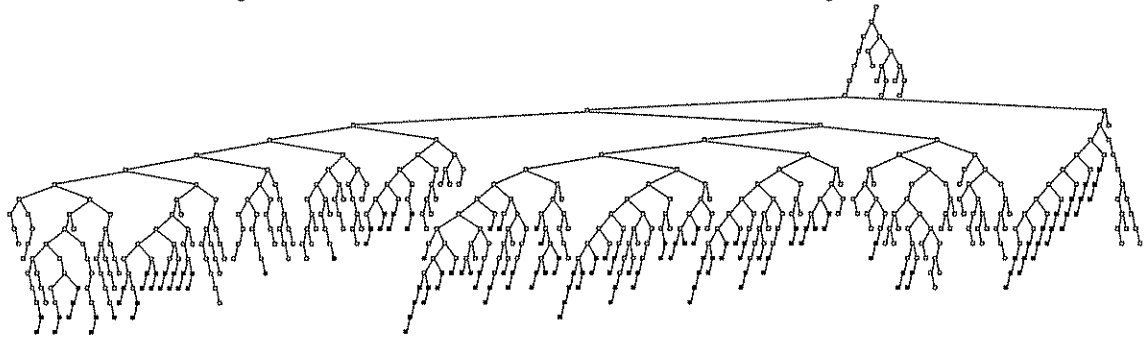

Figure 5: Tree of office4 from the area light source

Interpolating from the above results we can draw the following conclusion: that even for complex scenes an ordering from an area light source can be derived using this method with few or even no splittings of the source.

References

[1] A. T. Campbell. *Modelling Global Diffuse Illumination for Image Synthesis.* PhD thesis, Department of Computer Science, University of Texas at Austin, December 1991.

[2] N. Chin and S. Feiner. Fast object-precision shadow generation for area light sources using BSP trees. In *ACM Computer Graphics (Symp. on Interactive 3D Graphics)*, pages 21–30, 1992.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[4] H. Fuchs, Z. M. Kedem, and B. Naylor. Predetermining visibility priority in 3-D scenes (preliminary report). *ACM Computer Graphics*, 13(3):175–181, August 1979.

[5] N. Gatenby and W. T. Hewitt. Radiosity in computer graphics: a proposed alternative to the hemi-cube algorithm. In *Second Eurographics Workshop on Rendering*, 1991.

[6] B. F. Naylor. *A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes.* PhD thesis, University of Texas at Dallas, May 1981.

[7] R. Schumacker, B. Brand, M. Gilliland, and W. Sharp. Study for applying computer-generated images to visual simulation. Technical Report AFHRL-TR-69-14, NTIS AD700375, U.S. Air Force Human Resources Lab., Air Force Systems Command, Brooks AFB, TX,, September 1969.

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

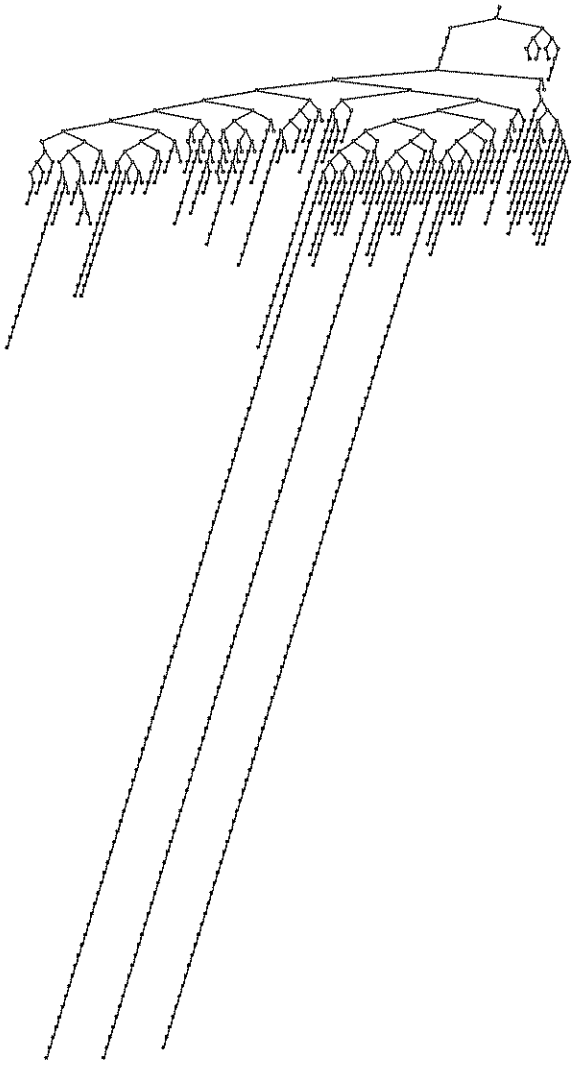

Figure 6: Tree of office4 with light having half the dimensions of the ceiling

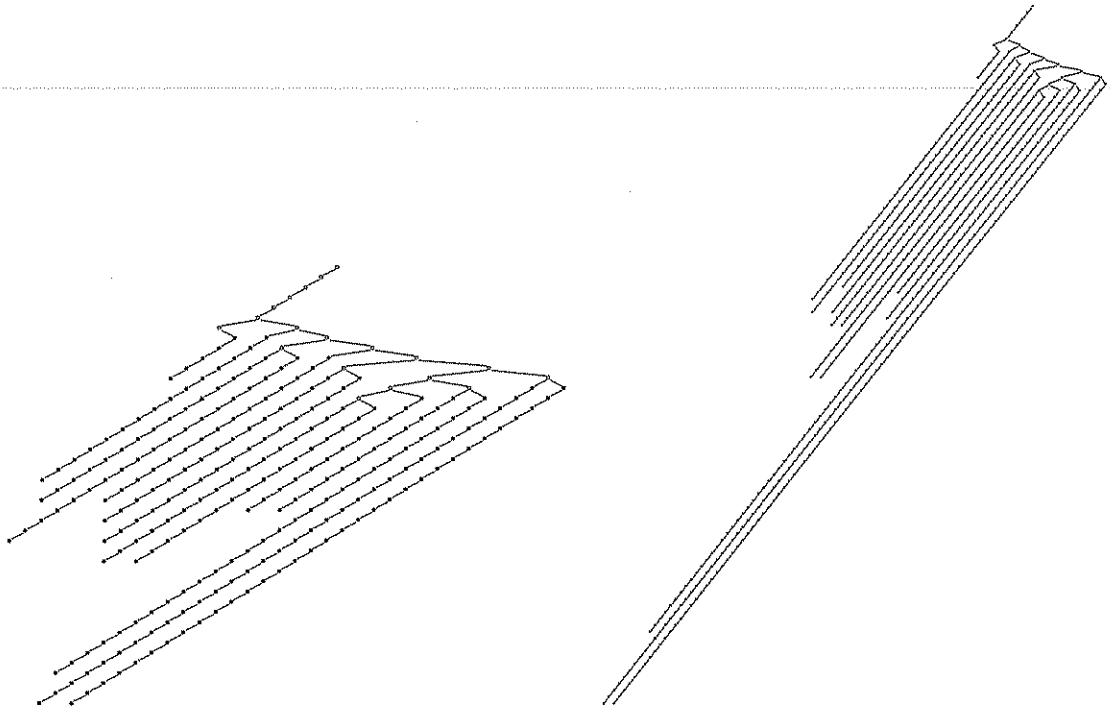

Figure 7: Trees of office1 and office3 from ceiling

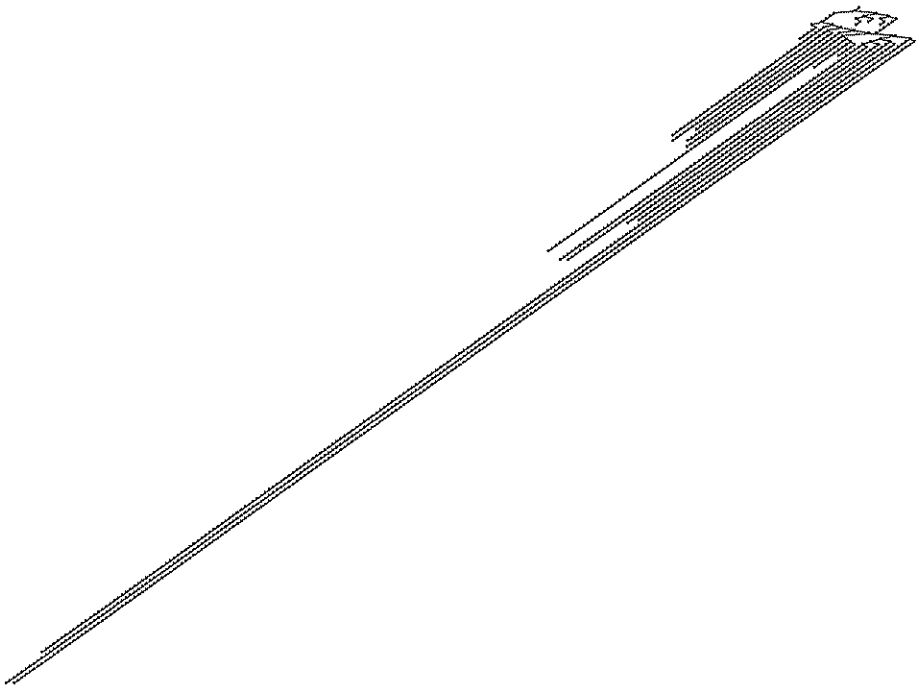

Figure 8: Tree of office4 from ceiling