Shadow Volume BSP Trees for Computation of Shadows in Dynamic Scenes

Yiorgos Chrysanthou* and Mel Slater*

Queen Mary and Westfield College,
University of London.

ABSTRACT

This paper presents an algorithm for shadow calculation in dynamic polyhedral scenes illuminated by point light sources. It is based on a modification of Shadow Volume Binary Space Partition trees, to allow these be constructed from the original scene polygons in arbitrary order and to support for fast reconstruction after a change in scene geometry. Timings using sample scenes are presented that indicate substantial savings both in terms of computation time and shadows produced.

KEY WORDS: shadows, BSP Trees, SVBSP Trees, dynamic modification.

1 INTRODUCTION

An algorithm is presented for rapid updating of shadows in dynamic environments, where objects are transformed in near real time with induced changes to shadows computed and displayed. The algorithm employs shadow volumes (SV). This was a term used by Crow [6] to denote the semi-infinite volume enclosed by the shadow planes (SP) formed by the triangle of the edge vertices and the light source position, for each edge of a polygon and culled by the polygon plane. Reviews of shadow algorithms for static scenes may be found in [1, 19, 15]. An algorithm for shadows in dynamic scenes is described in [5]. This uses a Shadow Tiling and a Binary Space Partition (BSP) tree. BSP trees were developed by Fuchs, Kedem and Naylor [8] as a visible surface determination, based on Schumacker's results [14]. A BSP tree is a hierarchical subdivision of space into homogeneous regions, using the planes defined by the scene polygons as partitions. It is mainly suitable for static scenes but under the right circumstances it can deal with moving objects [7, 18, 5]. Thibault, Naylor, and Amanatides [11, 17] employed the BSP tree to represent arbitrary polyhedral solids and for set operations on polyhedra in representation and rendering for CSG. They also showed that a BSP tree can be constructed incrementally.

Based on these results Chin and Feiner [3] introduced the Shadow Volume Binary Space Partition (SVBSP) tree algorithm for point light sources that can be used to compute shadows efficiently for polyhedral scenes. The algorithm used a BSP tree to order the input polygons in increasing order of depth from the light source, and to incrementally build a BSP tree representation of a single merged shadow volume for the whole scene. In the process of building the tree, the scene polygons are split and labeled as *lit* or *shadowed*. The algorithm operates in object space so that the shadows need not be regenerated if the viewing parameters are changed. The method is therefore suitable for walk-through applications. However, it is not suitable for interactive modification of objects in the scene, since any change in an object's position could destroy the ordering and may require the reconstruction of the shadow tree. Similar structures were used in [10] for image representation and in [4, 2] for determining illumination discontinuities from area light sources.

The algorithm presented here employs a generalization of the SVBSP tree that does not require the construction of a BSP tree of the original scene polygons, and that does support near real-time incremental changes to the SVBSP tree and therefore to shadows in response to object transformations. This method also results in computation of a smaller number of shadows compared to the original method. An application of the algorithm on a VR system is described in [16].

2 BUILDING THE UNORDERED SVBSP TREE

The standard SVBSP tree is built from an ordered set of polygons so there is no question as to which polygon is closer to the light source when the SVs of two polygons intersect. Building the tree using only the shadow planes is sufficient. For the unordered SVBSP tree the scene polygons themselves must be added to convey that infor-

*Department of Computer Science, QMW University of London, Mile End Road, London E1 4NS, UK. e-mail: yiorgos@dcs.qmw.ac.uk, mel@dcs.qmw.ac.uk

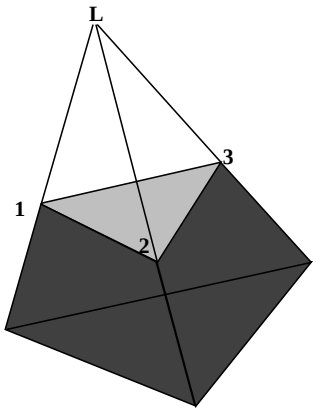

Figure 1: Full shadow volume

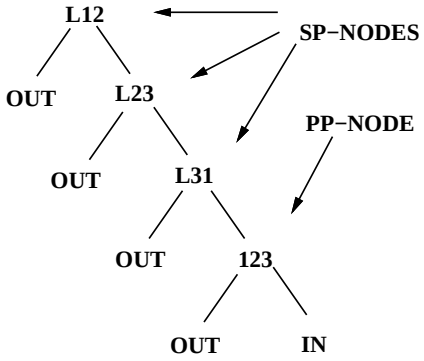

Figure 2: Shadow volume as a BSP tree

mation, so the SV of each individual polygon is complemented with the polygon plane. Since nodes containing shadow planes and nodes containing polygon planes are treated differently by the algorithm they will be distinguished by calling the former SP-Nodes (Shadow Plane Nodes) and the latter PP-Nodes (Polygon Plane Nodes), Figure 2 .

The algorithm uses a copy of the scene polygons in the tree for calculating the shadows which are then stored as detail ontop of the actual scene polygons.

The tree is built incrementally by inserting the light-facing polygons into an initially empty tree, a single OUT node (Figure 3). The first polygon just replaces that node with its SV (Figure 4). Subsequent scene polygons are filtered into the tree by comparing them at each level against the plane at the root of the tree and recursively inserting them into the appropriate subtree. If they straddle the root plane then they are split and each piece is treated separately. When an OUT node is reached it is replaced by the SV. If the polygon was split its SV is built using the shadow planes of the original polygon (polygon 4 in Figure 6). This is necessary for dynamic modification and it also means that the SV needs to be calculated only once even if a polygon is split into many pieces.

A face onto which a shadow is cast is referred to as a *target* face. When a PP-Node is encountered, if the inserted polygon is classified as behind its plane then it is marked as shadowed and stored there (face 3 in Figure 6). If it is classified as in front then it takes note of the face at the root as a potential target and it is inserted into the front subtree. If it reaches an OUT node then a shadow is cast on the face stored as potential target (face 2 in Figure 6). If it comes in front of more than one potential target, only the last one is remembered and used (polygon 5 in Figure 7 comes in front of 2 and then in front of 4, a shadow is casted only on 4).

To cast a shadow onto a target, the original scene polygon of the target is clipped against the relevant SV.

3 USING THE TREE FOR DYNAMIC SHADOW COMPUTATION

In an interactive application where the scene geometry changes, the tree can be used to maintain the correct shadows.

During the building of the tree, each inserted polygon constructs a list of pointers to the locations it occupies on the tree. When an object is transformed, its polygons and their shadow planes on the tree are found using the location lists and are marked. After all relevant polygons have been marked, a recursive function is called that will iterate through the SVBSP tree once and remove all marked nodes. The result of this will be a valid SVBSP tree for the scene, but now without the transformed object. The object can then be reinserted into the tree using the algorithm described in section 2, to get the shadows at its new position.

3.1 REMOVING THE MARKED NODES

The function used for removing the marked nodes works on the whole SV of polygons rather than on single nodes. There are 3 possible positions for each polygon and its shadow volume to consider:

(a) In the IN region, behind a PP-node (no shadow planes were attached here, just the polygon). This is the simplest case, the polygon is just removed (polygon 3 in Figure 7).

(b) At the leaves, subdividing an empty subspace. Again this is simple, the SV is replaced by an OUT node. Care must be taken if the PP-Node had a non-null target. This occurs when it is in front of some other PP-Node during insertion and it is now casting a shadow on this. In this case the shadow must be removed. For example when deleting polygon 5 in Figure 7 , the front (left) subtree of node labeled 4.2 should be replaced by OUT and the shadow on polygon 4 should be deleted (the arrows there show the target relation).

(c) Splitting a non-empty subspace, the SV forms the root of a larger subtree. This is the only relatively complex case. Removing it would result in unconnected subtrees and these must be put together to form a new tree to replace the old one. If the deleted polygon was casting a

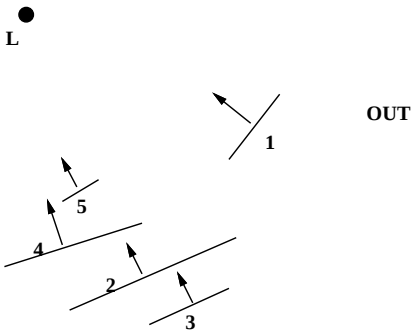

Figure 3: Initial scene

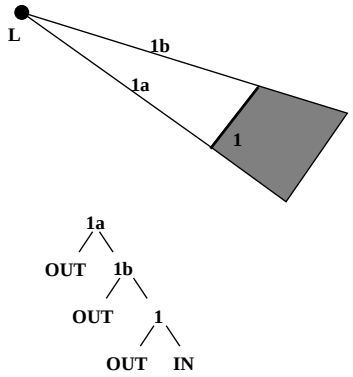

Figure 4: Insert poly 1

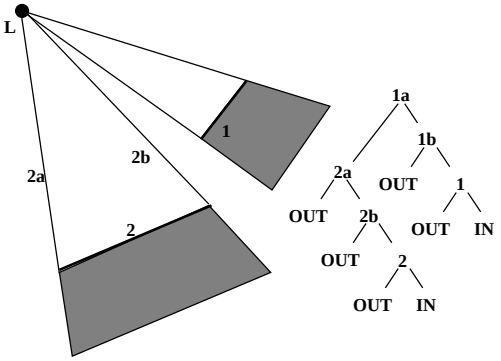

Figure 5: Insert poly 2

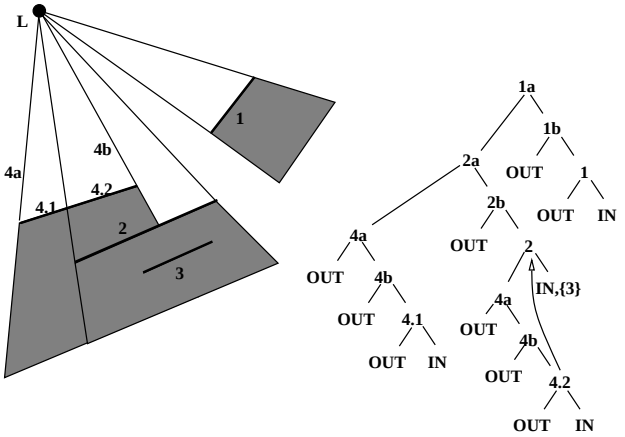

Figure 6: Insert poly 3 and 4

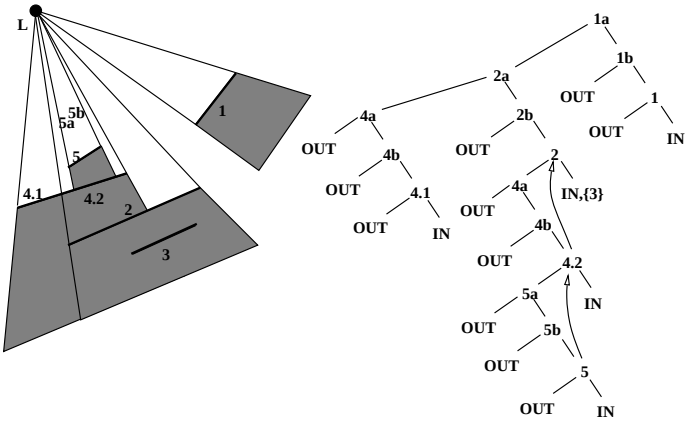

Figure 7: Insert poly 5

shadow then that must be replaced by shadows from polygons that had the deleted one as target. These can only be in the front subtree of the deleted PP-node. For example if polygon 4 in Figure 7 is deleted then the shadow from 4 to to 2 should be replaced by a shadow from 5. Any polygons that were in shadow, in the IN region behind the deleted polygon, must also be inserted into the new unified subtree.

3.2 JOINING THE SUBTREES

Naylor in [11] described an algorithm for merging BSP trees that could be used in case (c) above. Given two BSP trees for merging, tree1 is inserted into tree2. To achieve this tree2 is split into tree2.front and tree2.back by filtering the root of tree1 into it. These two new trees are then recursively merged into the corresponding subtrees of tree1 until they reach the leaves. Experimental results have shown this method to be very slow for our purposes. The main reason is that it operates on a closed subspace and it would require the shadow planes involved in the merging to be clipped and bounded. Also it is very general, it doesn't utilise the fact that all the shadow planes emanate from the same point (the light source).

A more specialised algorithm is used here. The largest of the trees to be merged is found, say tree1, and any possible marked nodes on this are removed. The inserted tree, tree2, is then treated as a set of shadow volumes. The polygon node (PP-Node) of the shadow volume forming the root of tree2 is found and filtered down tree1 along with its front and back subtrees. The filtering is done in a similar manner to a polygon. The fact that all shadow planes go through the light source position, ensures that anything enclosed by a polygon's shadow volume can be split by another shadow plane, only if the polygon itself is split. This means that the front and back subtrees need to be checked for intersection with a plane only if the polygon is split by that plane. If the PP-Node meets another fragment of its own original polygon they can join up under its shadow volume (this is possible since the shadow planes used by the fragments are those of the original). When it reaches an OUT node its SV is attached. After the 'root' SV and the subtrees of its PP-Node have been inserted, the algorithm is called recursively to insert the front subtrees of its SP-Nodes.

Note that the subtrees involved here existed in non-intersecting subspaces separated by the deleted planes so there is no shadow relation between them. Also, if polygons split or come together during the merging, the shadows on them or the shadows they cast do not change.

scene	scene polygons		S-SVBSP		U-SVBSP after BSP		U-SVBSP no BSP	
	initial	after BSP	time (sec)	shadow pol	time (sec)	shadow pol	time (sec)	shadow pol
1	133	178	.46	332	.45	237	.28	172
2	211	286	.72	526	.74	384	.51	292
3	313	579	1.25	892	1.16	677	.63	389
4	745	1830	4.65	3820	3.70	2458	1.51	1063

Table 1: Timings for initial building

scene	object moved	object polygons		absolute time (sec)		compared to U-SVBSP (%)		compared to S-SVBSP (%)	
		number	% of scene	first move	next move	first move	next move	first move	next move
1	computer	34	19	0.08	0.055			17	12
		20	15	0.03	0.025	12	9	7	5
2	computer	34	19	0.12	0.060			16	8
		20	15	0.07	0.030	14	6	10	4
2	bookcase	84	30	0.25	0.135			34	19
		54	26	0.19	0.095	37	18	26	13
3	computer	51	9	0.21	0.070			17	5
		20	6	0.05	0.030	8	5	4	2
4	computer	34	2	0.25	0.100			5	2
		20	3	0.12	0.060	8	3	3	1
4	comp. & desk	197	11	0.42	0.210			9	4
		56	8	0.16	0.080	11	5	3	2

Table 2: Transformation timings

4 FURTHER DISCUSSION

When a target object is being continuously transformed, for example as a result of being dragged during an interactive application, the functions described in sections 3.1 and 3.2 are only relevant for the very first transformation. After the first deletion and re-insertion, the faces will end up at the leaves and in subsequent frames can be deleted in constant time.

In the standard SVBSP tree the smaller objects tend to be higher up the tree because they tend to be closer the light source. This is the order that is obtained from the scene BSP tree traversed from the light position. Also their polygons may be widely distributed in the tree (Figure 10). Moreover these smaller objects are the ones most likely to be selected and transformed during an interaction.

In the method described here, the polygons may be grouped together according to the object to which they belong and given to the SVBSP tree in that order. Therefore there is greater probability that the polygons belonging together will be grouped together in the SVBSP tree (Figure 11). Also the smaller objects can be inserted last. For Figure 11 the objects were inserted in depth-first order in the scene hierarchy (Figure 9).

A small proportion of shadow planes in the tree are responsible for most of the splitting. Removing these, when their polygons have moved, could be an expensive operation. This can be avoided by leaving these nodes in the tree as marked and not removing them, if their subtrees are found to be too large. They are removed eventually when later transformations make their subtrees sufficiently small.

More than one light source can be modeled by creating a separate SVBSP tree for each. The input for subsequent sources are the initial scene polygons and their shadows.

5 RESULTS

The algorithm was implemented in C on a SUN SparcStation 2 with 16MB memory. The scenes used consisted of a room, which contained bookcases with two books, and desks with computers on top, the number of initial polygons ranging from 133 to 745.

Table 1 shows the numbers of polygons in four test scenes, including the number of polygons after the creation of a scene BSP tree. It also shows the time for creating the SVBSP trees (excluding the time to create the scene BSP tree). The unordered SVBSP tree is created in two alternative ways, using the initial set of polygons (column marked *U-SVBSP no BSP*) and using the polygons after they have been split by the scene BSP tree (column marked *U-SVBSP after BSP*). In both cases however the order of insertion is determined by the scene hierarchy. The different ordering is partly responsible for the difference in timing between the two methods. Timings include the calculation of the shadow geometry. The results suggest that even when (unnecessarily) using the polygons from the scene BSP tree, the unordered tree takes no

more time to build, and results in less shadow polygons than the method describe in [3].

Table 2 gives timings for transformations of various objects. The different versions of *computer* are for different computers in the scene. In each case the number of polygons along with the proportion of the total scene accounted for by the object being transformed is shown. The timings for transformations differentiate between the first move and subsequent moves. The subsequent transformations always take less time, for the reasons given in Section 4. The column marked *compared to S-SVBSP* gives the proportion of time taken for the transformation in comparison with recreating the complete standard SVBSP tree and the column marked *compared to U-SVBSP* the proportion of time against recreating the complete unordered SVBSP tree.

Two sets of experiments were performed for each scene. In the first set (first row for each scene), a BSP tree was built for representing the scene and the resulting polygons were used as input for building both SVBSP trees. This was done for getting a measure of the performance when the input polygons for the SVBSP trees are the same. In the second set of experiments (second row), the unordered SVBSP tree was built from the scene polygons, which is why the same object has less polygons and it takes less time to move. The standard SVBSP tree used for comparison is the same throughout.

6 CONCLUSION

This paper has presented a method for shadow generation for dynamic scenes. It is based on the generalised SVBSP tree algorithm that uses full shadow volumes and adds the polygons in any order. The resulting tree preserves all the benefits of the ordered SVBSP and yet it can be rapidly modified in response to changes in the scene polygons due to object transformations. As a result it is possible to interactively manipulate objects in near real time, while maintaining correct shadows, even on a standard workstation without a 3D graphics accelerator, such as the SparcStation 2 running X Windows. The algorithm only produces shadow umbras, which although better than no shadows at all, still leaves a lot to be desired in terms of realism. Future work involves extending the algorithm to soft shadows, produced by area light sources. This was considered, for static scenes, by Chin and Feiner in [4].

ACKNOWLEDGMENTS

Y. Chrysanthou's work is supported by the Keddy Fletcher-Warr Studentship (University of London) and the Overseas Research Students' Fee Support Scheme.

References

[1] P. Bergeron. A general version of crow's shadow volumes. *IEEE Computer Graphics & Applications*, 6(9):17–28, 1986.

[2] A. T. Campbell, III and D. S. Fussell. An analytic approach to illumination with area light sources. Technical Report R-91-25, Dept. of Computer Sciences, Univ. of Texas at Austin, August 1991.

[3] N. Chin and S. Feiner. Near real-time shadow generation using BSP trees. *ACM Computer Graphics*, 23(3):99–106, 1989.

[4] 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.

[5] Y. Chrysanthou and M. Slater. Dynamic changes to scenes represented as BSP trees. *Computer graphics Forum, (Eurographics 92)*, 11(3):321–332, 1992.

[6] F. Crow. Shadow algorithms for computer graphics. *ACM Computer Graphics*, 11(2):242–247, 1977.

[7] H. Fuchs, G. D. Abram, and E. D. Grant. Near real-time shaded display of rigid objects. *ACM Computer Graphics*, 17(3):65–72, 1983.

[8] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *ACM Computer Graphics*, 14(3):124–133, 1980.

[9] D. Gordon and S. Chen. Front-to-back display of BSP trees. *IEEE Computer Graphics & Applications*, 11(5):79 –85, 1991.

[10] B. F. Naylor. Partitioning tree image representation and generation from 3d geometric models. In *Proceedings of the Graphics Interface '92*, pages 201–212, 1992.

[11] B. F. Naylor, J. Amandatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. *ACM Computer Graphics*, 24(4):115–124, 1990.

[12] M. S. Paterson and F. F. Yao. Binary partitions with applications to hidden surface removal and solid modeling. In *Proceedings of the 5th Annual ACM Symposium on Computational Geometry*, pages 23–32, 1989.

[13] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. *Discrete Computational Geometry*, 5:485–503, 1990.

[14] 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.

[15] M. Slater. A comparison of three shadow volume algorithms. *The Visual Computer*, 9(1):25–38, October 1992.

[16] M. Slater, M. Usoh, and Y. Chrysanthou. *The influence of shadows on presence in immersive virtual environments*, pages 8–21. Springer Computer Science, 1995. Virtual Environments '95.

[17] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partition trees. *ACM Computer Graphics*, 21(4):153–162, 1987.

[18] E. Torres. Optimization of the binary space partition algorithm (BSP) for visualization of dynamic scenes. *Computer graphics Forum, (Eurographics 90)*, 9(3):507–518, 1990. C.E. Vandoni and D.A. Duce (eds.), Elsevier Science Publishers B.V. North-Holland.

[19] A. Woo, P. Poulin, and A. Fourier. A survey of shadow algorithms. *IEEE Computer Graphics & Applications*, 10(6):13–31, 1990.

Figure 8: Scene 2

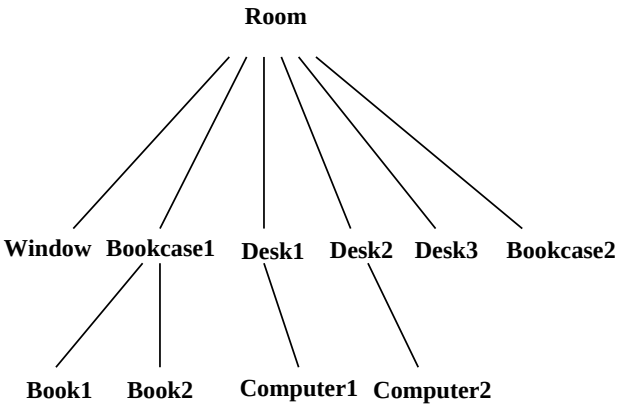

Figure 9: Model hierarchy

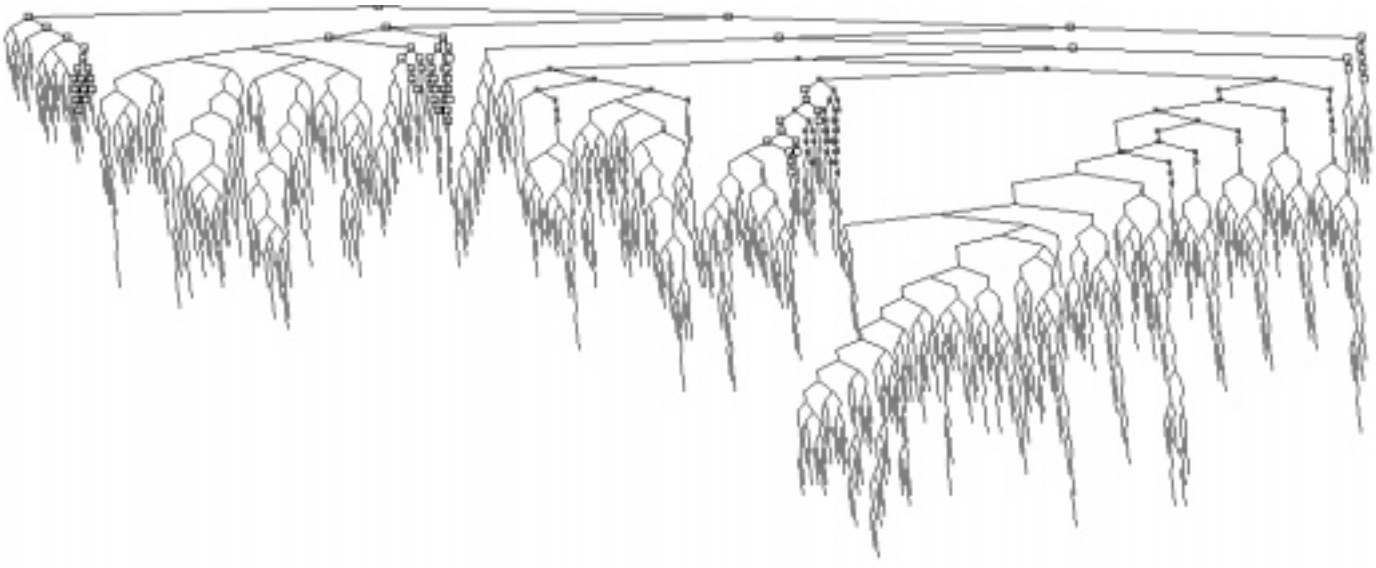

Figure 10: Position of computers in standard SVBSP

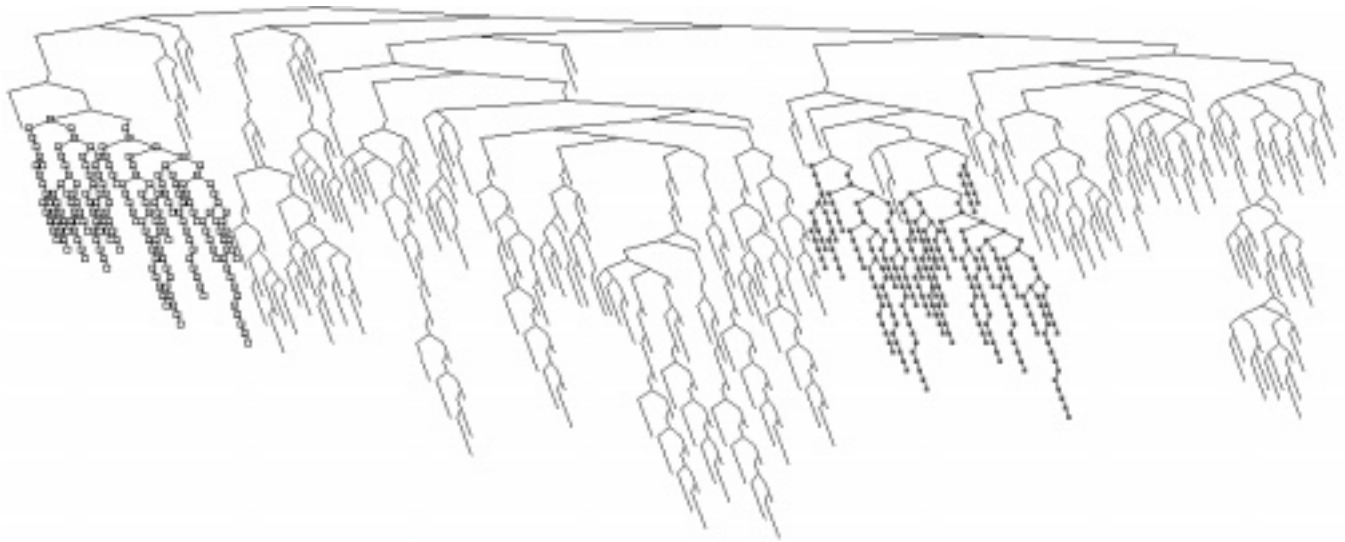

Figure 11: Position of computers in unordered SVBSP

The faces of computer1 in the tree are marked by a □ and those of computer2 by *