Volume 14, (1995) number 1, pp. 59–66

COMPUTER GRAPHICS *forum*

An Algorithm for Perspective Viewing of Objects Represented by Octrees

Walid G. Aref† and Hanan Samet*

† Matsushita Information Technology Laboratory, Two Research Way, Princeton, New Jersey 08540.
Email: aref@mitl.research.panasonic.com
* Computer Science Department and Center for Automation Research and Institute for Advanced Computer Studies, The University of Maryland, College Park, Maryland 20742. Email: hjs@cs.umd.edu

Abstract

A new algorithm is presented for viewing three-dimensional objects, represented by an octree, from an arbitrary location. The algorithm generates a perspective view of the objects while eliminating hidden surfaces. The viewer can be located anywhere inside or outside the objects. The algorithm presented in this short note fixes an artifact that is generated by a previously published algorithm due to Meagher when the viewer is located in certain regions in space. The new algorithm traverses the octree in a back-to-front order and recursively chooses correct orders for visiting the sons of non-leaf nodes.

Keywords: hidden-surface removal, perspective projection, display techniques, octree, three-dimensional representation.

1. Introduction

The region octree[1,2] is an extension of the region quadtree data structure[3,4] to represent three-dimensional data. It is useful for applications involving solid modelling and medical imagery. The region octree represents an object by its interior (in contrast to the boundary model). It yields a decomposition of space into cubical cells whose sides have a length that is a power of 2. The region octree is particularly useful for the purpose of mass computation and display.

The display of octree-encoded objects is usually achieved either by traversing the tree in a particular order[5–9] or by ray tracing and beam tracing techniques[10–17]. Our focus in this paper is on display methods that are based on traversals. Note that since the octree data structure is inherently a spatial ordering of the three-dimensional space, there exist traversal orders that result in the correct performance of hidden-surface elimination[18] without having to sort the blocks explicitly. The octree traversals presented in the literature access the blocks corresponding to the visited nodes in a back-to-front or a front-to-back order.

Doctor and Torborg[5] present a back-to-front algorithm for displaying an octree-encoded object by parallel projection. They use a quadtree for storing intermediate results. Their techniques depend heavily on the property that the octree blocks are aligned in parallel with their corresponding quadtree blocks. This property makes it difficult to adapt their techniques directly to perspective projection. In particular, they suggest that prior to the display process, a geometric transformation be performed on an octree-encoded object to generate a perspective projection. This is impractical because it involves creating a new octree that represents an object whose parallel projection is a perspective projection of the original object. The transformation is costly since virtually every block in the original octree will have to be subdivided many times to generate the new octree. Moreover, the transformation distorts the surfaces of the displayed objects. A further shortcoming of this algorithm is that in order to generate a view from an arbitrary location, the objects must first be rotated, which is an expensive process.

Meagher[6,8] presents two back-to-front viewing algorithms for a parallel projection when the viewer is

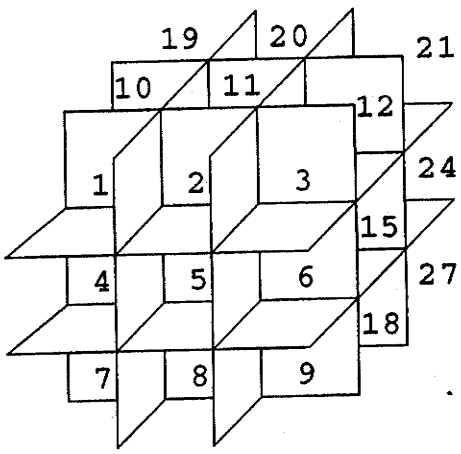

Figure 1: *A partition of space into 27 regions with respect to an object in the central region which is labelled 14 but is not visible.*

located arbitrarily in space. The two algorithms differ in the way the octree is traversed. The first algorithm uses depth-first search while the second algorithm uses breadth-first search. The general strategy is to traverse the octree representing the objects in the appropriate order for the location of the viewer. Meagher also shows how to perform a perspective projection.

Unfortunately, Meagher's approach only works when the viewer is constrained to be in certain positions. For example, consider Figure 1 which is a partition of space consisting of 6 infinite planes (2 parallel to the *xy* plane, 2 parallel to the *yz* plane, and 2 parallel to the *xz* plane). These planes are all separated by the same distance. Consider an object contained in the region bounded by these 6 planes. Meagher's algorithm only works when the viewer is in one of the 8 corner regions of the space in the Figure.

Figure 2 illustrates how Meagher's algorithm can err. Figure 2b is an example of the application of Meagher's algorithm to the objects of Figure 2a for a particular viewpoint. Notice the presence of artifacts in Figure 2b consisting of narrow strips corresponding to faces that are supposed to be hidden. Figure 2c is the correct view from the same viewpoint.

In order to understand the cause of this error, consider the object in Figure 3a and the viewpoint V. V is in region 2 of Figure 1. In particular, V is at a point such that the *yz* plane through it is perpendicular to the front face of block 7 (i.e. the face containing the label) and passes through this face. Meagher's algorithm visits the blocks of the object in the order 1, 2, 3, 4, 5, 6, 7, 8, 9. The perspective projection generated by it is shown partially in Figure 3b. However, the correct order of visiting the blocks of the object is 1, 2, 3, 4, 5, 8, 6, 9, 7, and the perspective projection generated by it is shown partially in Figure 3c. The removal of the artifacts requires that Meagher's technique be extended significantly and is the subject of this paper. In essence, we avoid these artifacts by visiting the children of each non-leaf node in an order that depends on the position of the non-leaf node in space relative to the viewer.

Meagher also presents a front-to-back algorithm for the parallel projection of three-dimensional objects represented by octrees[7,9]. A front-to-back algorithm is also used by Veenstra and Ahuja[19] to generate the line drawing of a set of objects represented by an octree. They use parallel projection from any specified viewpoint. The artifacts shown in Figure 2 are also expected to appear when both of the above algorithms are extended to handle perspective projection for a viewer at an arbitrary location.

In this paper we present a new back-to-front algorithm. It differs from the algorithms of Doctor and Torborg and Meagher in two ways. First, we perform perspective projection directly on the octree-encoded objects without the need for specific geometric transformations or any preprocessing of the object-encoding. Second, we permit the viewer to be at an arbitrary position inside or outside an object in contrast to Meagher's algorithm. Our algorithm does not require special processing to avoid the artifacts shown in Figure 2.

2. Algorithm

We adopt the same terminology and notation used by Samet[14]. Figure 4 shows the coordinate system that we are using relative to a cube. Let L and R denote the resulting left and right halves, respectively, when the *x* axis is partitioned. Let D and U denote the resulting lower (down) and upper halves, respectively, when the *y* axis is partitioned. Let B and F denote the resulting back and front halves, respectively, when the *z* axis is partitioned. Figure 5 illustrates the symbolic labellings corresponding to the partitions. Octants are labelled by using a concatenation of these symbols as shown in Figure 6 (octant LDB is not visible). The objects are located in a space such that for all points $x > 0$, $y > 0$, $z > 0$, and the origin corresponds to its smallest address. The viewer is located at an arbitrary point inside or outside an object and can move to any other point in the object space.

The expression of the algorithm is aided by the following definitions. The set O of octants is defined as O = {LDB, LDF, LUB, LUF, RDB, RUB, RUF}. Given blocks B_1 and B_2 which do not overlap, B_1 is said to be *visibly prior* to B_2 with respect to a viewer at location *V iff* B_1 covers part or all of B_2 when viewed from *V*. Similarly, blocks B_1 and B_2 are said to be *visibly neutral* with respect to a viewer at location *V iff* neither B_1 or B_2 cover any part

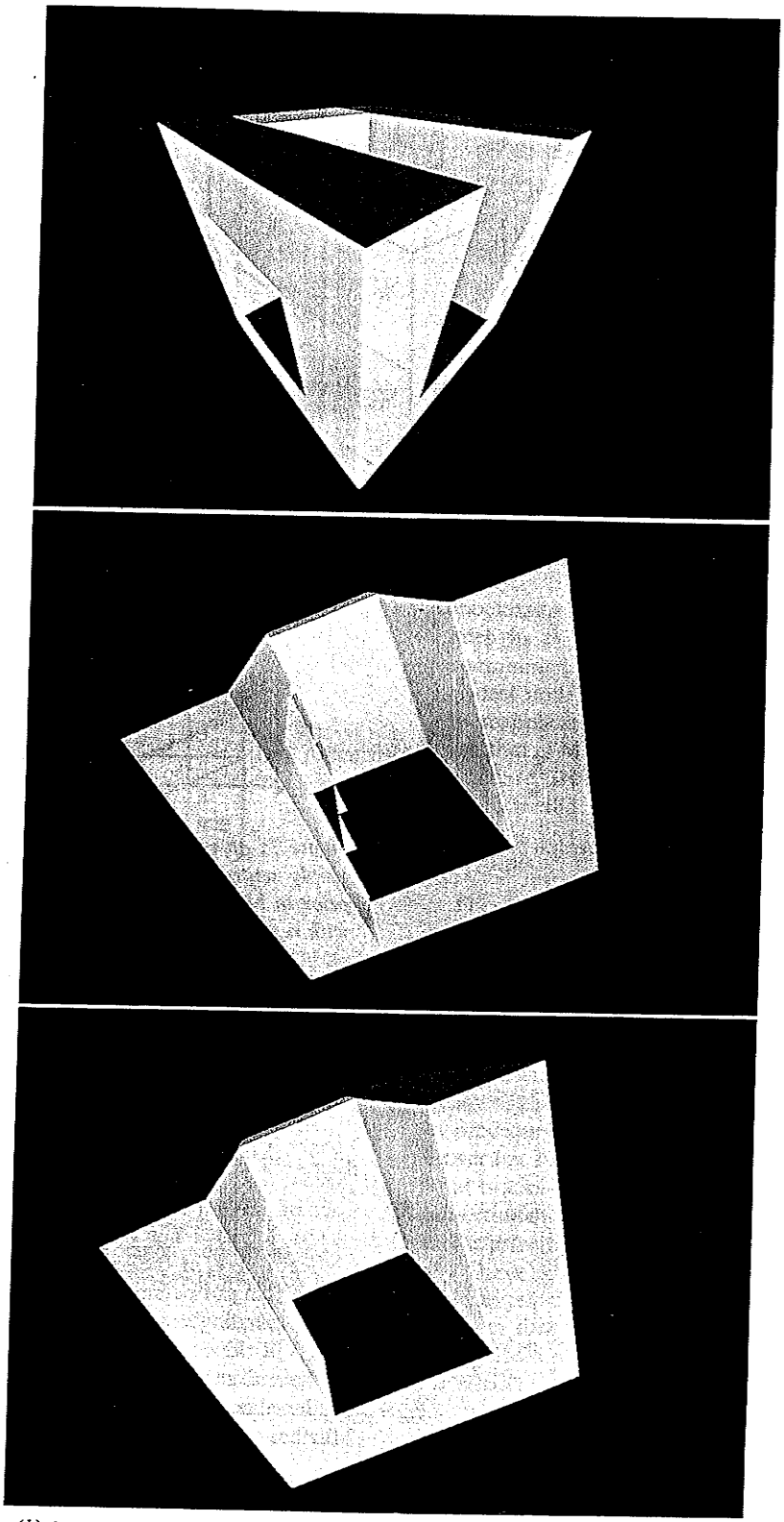

Figure 2: *(a) An object, (b) its perspective projection using Meagher's algorithm (notice the artifacts in the left front of the image), and (c) its correct perspective projection.*

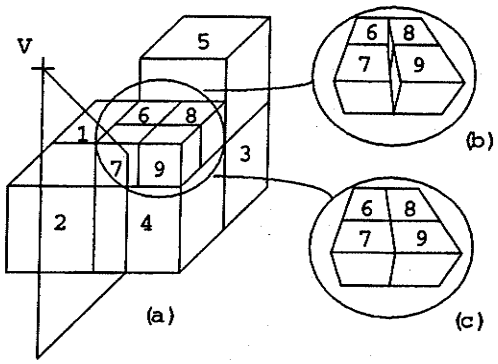

Figure 3: *(a) An object, (b) the perspective projection of the circled subfigure of (a) using Meagher's algorithm and (c) its correct perspective projection.*

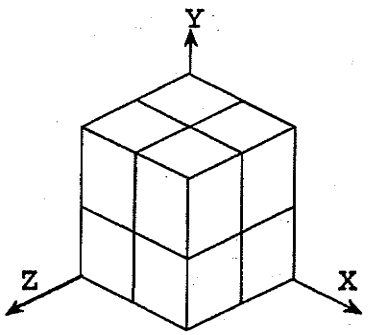

Figure 4: *A three-dimensional coordinate system.*

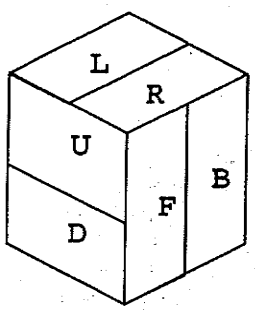

Figure 5: *Three orthogonal partitions of a cube.*

of each other when viewed from V. For example, blocks 1 and 2 in Figure 7 are visibly neutral with respect to viewer V, while block 3 is visibly prior to block 1. Note that if block 3 is projected into the projection plane after projecting block 1, then we will get a consistent view (i.e., the front blocks cover the back blocks).

An *octant order* is a list containing each element of O. We use R_i to refer to the ith element of octant order R where i ranges between 0 and 7. Octant order R is said to be *view-consistent* with respect to a viewer at location V if

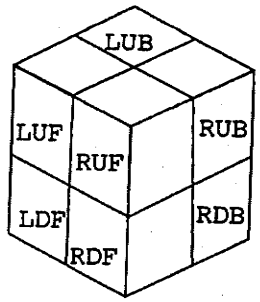

Figure 6: *Labelling of octants in a cube based on the partitioning defined in Figure 8 (octant LDB is not visible).*

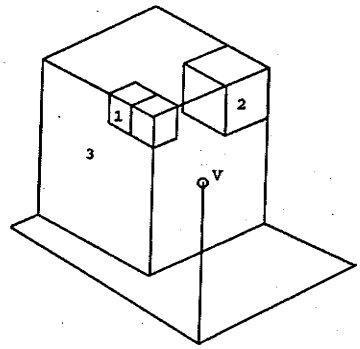

Figure 7: *An example of blocks that are visibly prior and visibly neutral with respect to viewer V. Block 3 is visibly prior to block 1, while blocks 1 and 2 are visibly neutral with respect to each other.*

for all $i > j$, the block corresponding to octant R_i is visibly prior to the block corresponding to octant R_j, or the blocks corresponding to octants R_i and R_j are visibly neutral. For example, the octant orders (LDF, RDF, LDB, RDB, LUF, RUF, LUB, RUB) and (LDF, LUF, RDF, LDB, RDB, RUF, LUB, RUB) are view-consistent for the block and viewer V in Figure 8. However, octant order (LDF, RDB, RDF, LDB, LUF, RUF, LUB, RUB) is not view-consistent with respect to the block and V in the same Figure since the block corresponding to octant RDB is visibly prior to the block corresponding to octant LDB while RDB appears before LDB in the octant order. Notice that if the blocks are projected into the projection plane in a view-consistent order, then the hidden-surface problem is solved without further effort.

Assume that the objects are represented by a pointer-based octree data structure[14]. However, the algorithm can be adapted easily for pointerless representations (e.g., the linear octree[14]). An octree node, say Q, is a record

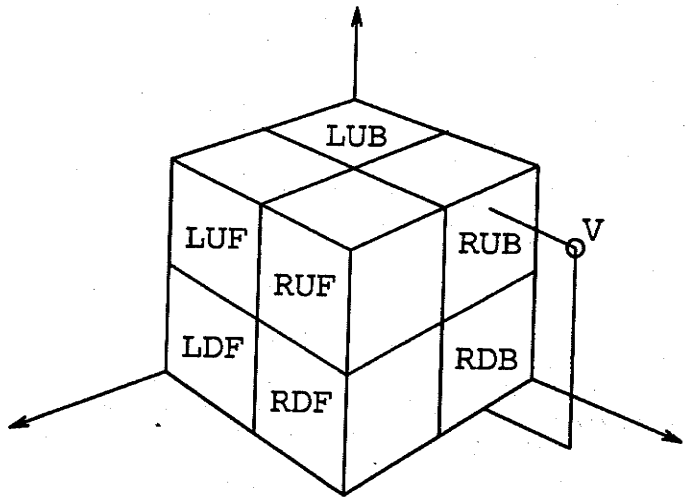

Figure 8: *A viewer V in the RUB octant relative to the centre of the block.*

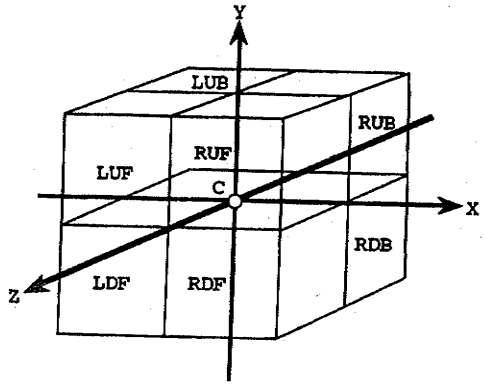

Figure 9: *An illustration of how a view-consistent octant order is selected. The three orthogonal planes intersect at the centre C of the block. Octant LDB is not visible.*

structure that has the following fields: CENTRE is a record of type *point* representing the centre of Q's corresponding block, say b, while SIZE is the side length of b. If Q is a leaf node, then Q's record has the field VALUE which stores whether Q is black or white. If Q is a gray node, then Q has eight additional fields representing a pointer to each of Q's sons (b's suboctants).

A point in the three-dimensional space is represented by a record of type *point* with three fields X, Y, and Z corresponding to the values of the x, y, and z coordinates, respectively, of the point. The coordinate values of the centre of each block are also represented by a record of type *point*.

The viewing algorithm is encoded by procedure VIEW_A and proceeds as follows. For each non-leaf node Q, determine a view-consistent order, say D, for Q's suboctants with respect to the viewer. Next, visit Q's son nodes in the sequence given by D. The process of determining a view-consistent order is reapplied to each non-leaf node in the tree starting at the root node. It is this reapplication step that differentiates our approach from that of Meagher[6]. When a leaf node is encountered, procedure DISPLAY_BLOCK determines via table-lookup methods which of the 1, 2, or 3 faces of the corresponding block are to be projected into the projection plane.

The appropriate view-consistent octant order to be applied at each non-leaf node, say Q centered at C, is determined as follows. Assume the existence of three mutually perpendicular infinite planes that pass through C as shown in Figure 9. The three planes divide the space into eight infinitely-sized regions. We label each of these regions by its octant label relative to C. The viewer can be in any of these regions, say R. This region is determined by procedure GET_OCTANT_ORDER. Being in R, the viewer has to see a consistent view of the scene (i.e., the front blocks cover the back blocks). This is achieved by projecting the octants of Q in an order that maintains the visible priorities among them. In other words, the octants of Q have to be projected in a view-consistent octant order to achieve a correct perspective projection.

Of course, when the viewer is located in R, there can be several ways to properly project the octants, all of them being view-consistent octant orders. We only need to follow one of them and hence we only store one of them for each region. These view-consistent octant orders are obtained by accessing array ORDER which is given in Table 1. It has 8 rows – one for each region. Every row in the array represents a valid view-consistent octant order when the viewer is located in the corresponding region.

View-Consistent Octant Order	Ordered Elements of the Octant Order							
	0	1	2	3	4	5	6	7
LDB	RUF	RUB	RDF	RDB	LUF	LUB	LDF	LDB
LDF	RUB	RUF	RDB	RDF	LUB	LUF	LDB	LDF
LUB	RDF	RDB	RUF	RUB	LDF	LDB	LUF	LUB
LUF	RDB	RDF	RUB	RUF	LDB	LDF	LUB	LUF
RDB	LUF	LUB	LDF	LDB	RUF	RUB	RDF	RDB
RDF	LUB	LUF	LDB	LDF	RUB	RUF	RDB	RDF
RUB	LDF	LDB	LUF	LUB	RDF	RDB	RUF	RUB
RUF	LDB	LDF	LUB	LUF	RDB	RDF	RUB	RUF

Table 1: *ORDER* [*octant, octant order position*]

Notice that the orders in the array are indexed by octant labels (which are also the region labels, as shown in Figure 9). It is interesting to observe that this array is also used by Meagher[6]. The difference is that he uses a single order for the entire octree while the order that we apply may differ at each non-leaf node depending on the position of the node's block relative to the location of the viewer.

3. Concluding Remarks

A compact and simple algorithm for generating the perspective view of a set of octree-encoded objects from an arbitrary location in three-dimensional space has been presented. It is based on a recursive computation of a view-consistent octant order of visiting the sons of a non-leaf node. Leaf nodes are projected in the same order as they are encountered during the traversal process. In Aref and Samet[20], a proof has been given that this will result in the correct performance of hidden-surface elimination without explicit processing (i.e., sorting). This is because the octree data structure is itself a spatial ordering of the three-dimensional space. The worst-case running time of the algorithm is linearly proportional to the total surface area of the octree-encoded objects.

Future work includes the application of our techniques to develop a front-to-back display algorithm since the same artifacts shown in Figure 2 are expected to appear. Viewing polyhedral objects represented by PM octrees (see, e.g., Samet[4] and the references cited therein) is another possible extension of this work.

Acknowledgements

The first author conducted most of this research while at The University of Maryland, College Park. This work was supported in part by the National Science Foundation under Grant IRI-9017393. We have benefitted greatly from the comments of Robert F. Sproull.

References

1. G.M. Hunter, *Efficient computation and data structures for graphics*, PhD thesis, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, NJ (1978).
2. D.R. Reddy and S. Rubin, Representation of three-dimensional objects, Technical Report CMU-CS-78-113, Computer Science Department, Carnegie-Mellon University, Pittsburgh, April (1978).
3. A. Klinger, Patterns and search statistics. In J.S. Rustagi, editor, *Optimizing Methods in Statistics*, pp. 303–337. Academic Press, New York (1971).
4. H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA (1990).
5. L.J. Doctor and J.G. Torborg, Display techniques for octree-encoded objects. *IEEE Computer Graphics and Applications*, **1**(3), pp. 29–38, (1981).
6. D. Meagher, Octree encoding: A new technique for the representation, manipulation, and display of arbitrary 3-d objects by computer, Technical Report Electrical and Systems Engineering Report IPL-TR-80-111, Rensselaer Polytechnic Institute, Troy, NY, (1980).
7. D. Meagher, High speed display of 3-d medical images using octree encoding. In *IEEE Computer Society Tenth Workshop on Applied Imagery Pattern Recognition*, (1981).
8. D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, **19**(2), pp. 129–147, (1982).
9. D. Meagher, The octree encoding method for efficient solid modeling. Technical Report Electrical and Systems Engineering Report IPL-TR-032, Rensselaer Polytechnic Institute, Troy, NY, 1982.

10. N. Dadoun, D.G. Kirkpatrick, and J.P. Walsh, Hierarchical approaches to hidden surface intersection testing. In *Proceedings of Graphics Interface*, pp. 49–56, Toronto, (1982).

11. A.S. Glassner, Space subdivision for fat ray tracing. *IEEE Computer Graphics and Applications*, **4**(10) 15–22, (1984).

12. F.W. Jansen, Data structures for ray tracing. In F.J. Peters, L.R.A. Kessener, and M.L.P. van Lierop, editors, *Data Structures for Raster Graphics*, pp. 57–73. Springer-Verlag, Berlin, (1986).

13. M.R. Kaplan, The use of spatial coherence in ray tracing, In D.F. Rogers and R.A. Earnshaw, (eds), *Techniques for Computer Graphics*, pp: 173–193. Springer-Verlag, New York, (1987).

14. H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading, MA, (1990).

15. M. Tamminen, O. Karonen, and M. Mäntyläx, Ray-casting and block model conversion using a spatial index, *Computer-Aided Design*, **16**(4), pp. 203–208, July (1984).

16. T. Whitted, An improved illumination model for shaded display, *Communications of the ACM*, **23**(6): 343–349, (1980).

17. G. Wyvill and T.L. Kunii, A functional model for constructive solid geometry, *Visual Computer*, **1**(1), pp. 3–14, (1985).

18. J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*. Addison-Wesley, Reading, MA, second edition, (1990).

19. J. Veenstra and N. Ahuja, Line drawings of octree-represented objects, *ACM Transactions on Graphics*, 7(1), pp. 61–75, (1988).

20. W.G. Aref and H. Samet, Perspective viewing of objects represented by octrees. Technical Report CS-2757, University of Maryland, College Park, MD, (1991).

21. Revised report on the algorithmic language ALGOL 60, *Communications of the ACM*, **3**(5), pp. 299–314, (1960).

Appendix: Pseudo-code for the Algorithm

The viewing algorithm is given using pseudo-code which is a variant of the ALGOL[21] programming language. The algorithm is encoded by procedure VIEW_A. It takes as its input a pointer to the root of the octree, say *P*, representing the set of objects and the viewer's position *V*. The origin of the space is assumed to be the extreme left (L), down (D), and bottom (B) point in the octree space. Nodes of *P* are visited using a view-consistent traversal. For each leaf node, say *L*, VIEW_A project *L*'s block into the projection plane using procedure DISPLAY_BLOCK which is not given here.

VIEW_A makes use of an additional auxiliary procedure called GET_OCTANT_ORDER. It also uses the two-dimensional array ORDER, described in Section 2. ORDER contains 8 rows, indexed by an octant label, each corresponding to a view-consistent octant order.

Procedure GET_OCTANT_ORDER is invoked for each non-leaf node *Q* in the octree. It returns the octant, say *D*, of Q that is nearest to the viewer. Row *D* of array ORDER yields a view-consistent octant order for visiting the sons of Q. GET_OCTANT_ORDER is invoked with the points corresponding to the center of *Q* and the viewer's position. It ensures that the visible priorities among the sons are preserved with respect to the viewer. It is enough to perform the decision on the basis of the center of *Q*'s block and the viewer's location.

```
/*
recursive procedure VIEW_A(P,V);
/* Given an octree rooted at P and a viewer
   at V, traverse P in a view-consistent
   order and display the perspective
   projection of its corresponding object
   with respect to a viewer at V. All
   coordinate values are with respect to an
   origin at the extreme LDB point in the
   octree space. */
begin
   value pointer node P;
   value pointer point V;
   octant D;
   integer I;
   global octant array ORDER[LDB...RUF,0:7];
   if GRAY(P) then
      begin /* Gray node */
      D←GET_OCTANT_ORDER(CENTRE(P),V);
      for I→0 step 1 until 7 do
       VIEW_A(SON(P,ORDER[D,I]),V);
      end
   else if BLACK(P) then DISPLAY_BLOCK(P,V);
/* Black node */
/* No action is required for a white node.*/
end;
octant procedure GET_OCTANT_ORDER(C,V);
/* Given a universe rooted at point C,
return the label of the octant that contains
point V. */
begin
   value pointer point C,V;
   return(if X(V)<X(C) then
      if Y(V)<Y(C) then
         if Z(V)<Z(C) then 'LDB'
         else 'LDF'
```

```
else if Z(V)<Z(C) then 'LUB'
   else 'LUF'
else if Y(V)<Y(C) then
   if Z(V)<Z(C) then 'RDB'
   else 'RDF'
      else if Z(V)<Z(C) then 'RUB'
         else 'RUF');
end;
```