TREE DATA STRUCTURES FOR GRAPHICS AND IMAGE PROCESSING

R. Gillespie and W.A. Davis

Department of Computing Science
University of Alberta, Edmonton

ABSTRACT

This paper presents a systematic development of hierarchical, or tree-based, representations of digital pictures which have been proposed for use in computer graphics and image processing. Starting with the one dimensional case, the development proceeds to quadtrees (the 2-D case), then to octrees (the 3-D case), and finally to hextrees (the 4-D case), and the relationships shown between successive representations. A brief summary of the algorithms and results for quadtrees is presented. Since less research has been conducted for octrees, there are fewer results to report. The details of an algorithm for projecting an octree onto a quadtree are presented; this algorithm is necessary for producing a display of an octree on an output device. In addition, the applicability of the octree representation to three dimensional scenes, is discussed. Finally, a hierarchical data structure for representing four dimensional 'objects' where the fourth dimension is time, will be given.

RÉSUMÉ

Cette communication décrit de façon systématique les diverses représentations hiérarchiques ou arborescentes des images numériques qui ont été proposées pour le traitement, par ordinateur, de l'information graphique et des images. L'étude présente d'abord le cas de la représentation unidimensionnelle, puis aborde successivement les têtra-arborescences (le cas 2-D), les octo-arborescences (le cas 3-D) et, enfin, les hexa-arborescences (le cas 4-D) et les corrélations entre les représentations successives. Cette analyse est suivie d'une présentation sommaire des algorithmes et des résultats relatifs aux têtra-arborescences. Comme les octo-arborescences ont fait l'objet de beaucoup moins de recherches, les résultats signalés sont moins nombreux. L'algorithme servant à projeter une octo-arborescence sur une têtra-arborescence est exposé en détail; cet algorithme est nécessaire à l'affichage d'une octo-arborescence sur un périphérique de sortie. On traite en outre de la possibilité d'appliquer la représentation sous forme d'octo-arborescence aux images tridimensionnelles. L'exposé se terminera par la description d'une structure hiérarchique des données pour la représentation des objets à quatre dimensions, lorsque la quatrième dimension est le temps.

1. Introduction

A hierarchical data structure to represent a picture is a tree whose nodes correspond to, and represent, some portion of the picture. The picture may reconstructed in a conventional form by traversing the tree. This paper surveys some of the tree data structures which have been used in graphics and image processing. A uniform development of tree structures to represent images in one, two, three, and four dimensions is made. After each structure is defined, the possible uses for it are discussed, as well as how the tree structure compares to other possible representations.

The representation of images in a computer graphics or image processing system is of primary importance. Unfortunately, many designers do not devote much thought towards what sort of data structures to use; *ad hoc* decisions are generally made. However, interest in hierarchical data structures for graphics and image processing has been increasing.

Recursive algorithms to process images have existed for some time. Warnock [9] eliminates hidden surfaces by recursively subdividing a picture until each area is 'simple' according to his criteria. The same principle is used for most tree representations of pictures: the original picture is recursively subdivided until each area is 'simple' according to some criteria. The most common criterion is that all the picture elements in an area be the same, or at least similar, colour. A tree data structure can be constructed such that each of its nodes corresponds to some portion of the picture. A node is a leaf if the portion which it represents is 'simple'. Non-terminal nodes have sons each of which corresponds to some smaller portion of the area which their father represents. The root of the tree corresponds to the entire picture. Horowitz and Pavlidis [4] demonatrate the usefulness of such a data structure in graphics and image processing.

A different sort of hierarchic structure for graphics is given by Mallgren and Shaw [8]. In their structure nodes do not represent areas of the scene. Each node is either a graphic transformation (rotation, scaling, translation, etc.) or a picture object. A picture object consists of a set of graphic operations which produce a picture of some scene, and can only be found at a leaf of the tree. All internal nodes are graphic transformations which are to be applied to their sons, which may again be transformations. As the tree is traversed several transformations may be encountered and applied to the picture objects found at the leaves.

The best prospect for an image processing hierarchic structure is a pyramid structure [1, 14]. Here, an image is simplified by averaging regions of pixels over the entire picture, thus producing a smaller, blurred picture. In [14] the pixels in a two by two region are averaged, whereas in [1] the regions are chosen according to some global criteria. In either case, the 'pyramid' is the complete set of successively smaller pictures.

The vast majority of the work in the area of tree data structures has focused upon the two dimensional case. More recent interest has been shown for three dimensional extensions of two dimensional structures. The remainder of this paper will develop tree data structures to represent images in various spaces. The discussion starts with linear pictures (one-space), proceeds through two- and three-space, and finishes with four-space, where the fourth dimension is time. Each successive data structure is developed as an extension of the previous structure through the added dimension. Section 2 deals with the one dimensional case, Section 3 the two dimensional case, Section 4 the three dimensional case, Section 5 the four dimensional case, and Section 6 draws the conclusions.

2. One Dimension - Bitrees

Consider a one dimensional picture, say the projection of a two dimensional picture onto an axis. For convenience, assume that the picture consists of k=2**n 'units' (where '**' denotes exponentiation). Objects in the picture are drawn with only one colour, BLACK, and the rest of the picture is filled with EMPTY spaces. A binary tree (bitree) may be constructed to represent this picture as follows:

1) every node in the bitree represents a linear region of the picture whose size is 2**(n-d) units, where d is the depth of the node (the root is at depth 0);
2) every node *n* in the bitree either is a leaf or has exactly two sons. One son represents the left, or Western, half of the region which *n* represents. The second son represents the right, or Eastern, half.
3) every node *n* in the bitree has a colour associated with it. If every point in the region represented by *n* is BLACK, then *n* is coloured BLACK. If every point in the region is EMPTY, then *n* is coloured EMPTY. In both cases *n* is a leaf. Otherwise, the region is a mixture of BLACKs and EMPTYs, so *n* is coloured MIXED and is given two sons to represent the two halves of the region.

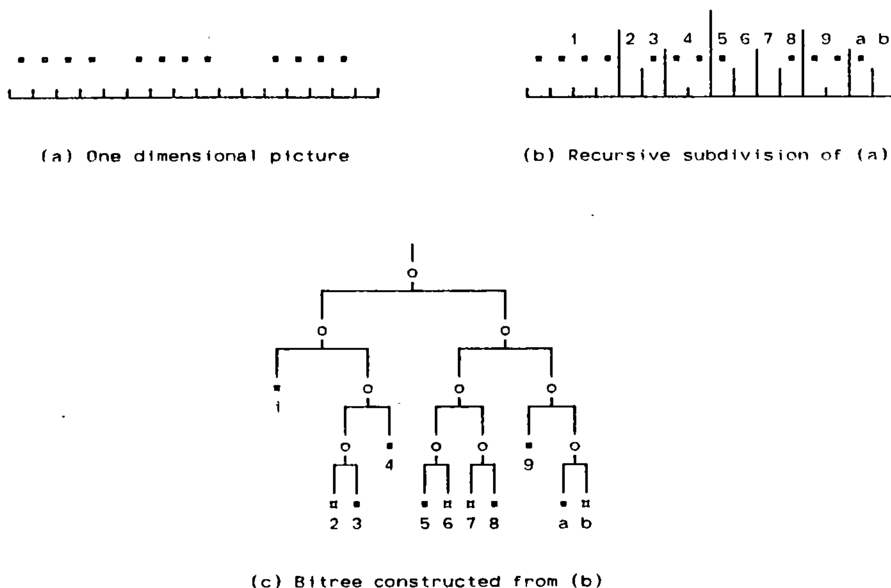
Note that no node will have two sons that are leaves and both coloured the same, since this is more compactly represented by colouring the father node and removing the sons from the tree. See Fig. 1 for a linear picture and the corresponding bitree.

It should be apparent from Fig. 1 that the bitree representation of a linear picture is not economical. A picture can be represented much more efficiently by vector end-points, or even, in some cases, as an array of colour values. Since the bitree representation requires more space than other representations, it is of little practical use. This, plus the minimal utility for linear pictures, accounts for the lack of interest in the bitree representation. Bitrees are of theoretical interest, however, as a simple basis for the other, higher dimensional tree structures.

3. Two Dimensions - Quadtrees

Now, consider two dimensional pictures. The basic unit of a two dimensional picture is a unit square area called a pixel. Again, assume that each dimension is k 'units' in size and that pixels are either BLACK or EMPTY.

There are several ways to represent such a two dimensional picture, the simplest being a two dimensional array of colours. A raster display on a monitor is such an array. Unfortunately, such arrays tend to be sparse and waste a great deal of space. A tree representation, the quadtree, is more space-efficient, as well as being



(a) One dimensional picture

(b) Recursive subdivision of (a)

(c) Bitree constructed from (b)

Fig.1 A one dimensional picture and the bitree which represents it. The vertical bars in (b) indicate the recursive subdivision points. The higher the bar is, the closer to the root the representative bitree node is. The numbers on (b) and (c) are labels indicating quadtree terminal nodes.

relatively easy to manipulate. A quadtree is defined by:

1) every node *n* in the quadtree represents a square area of the picture whose sides are of length 2**(n-d), where d is the depth of *n*;
2) every node *n* either is a leaf, or has four sons each of which represents one quadrant of the area *n* represents. The area of *n* is split up by dividing in half along each axis;
3) every node *n* has a colour associated with it. If every pixel in the area represented is BLACK, then *n* is a leaf and is coloured BLACK. If every pixel in the area is EMPTY then *n* is a leaf and is coloured EMPTY. Otherwise, if the area is a mixture of BLACKs and EMPTYs, then *n* is coloured MIXED and is given four sons to represent the four quadrants of the area.

Note that the introduction of an additional dimension means that the internal nodes in the tree have twice as many sons as in the one dimensional case. Fig. 2 shows the relationship between the subquadrants (labelled NW, NE, SW, SE), and the sons of a quadtree node. Fig. 3 shows a two dimensional picture and the corresponding quadtree.

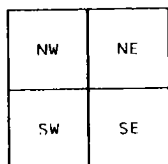
From Fig. 3 it can be seen that the quadtree representation can be more space efficient than a matrix of colour values. Recall that in the one dimensional case the tree representation was less space efficient than an array representation. By moving to two dimensions the array representation has grown exponentially. If the picture being represented consists of a few regions of contiguous, same-colour pixels then the quadtree representation will not grow nearly as fast. This difference becomes even more significant in the higher dimensional cases.

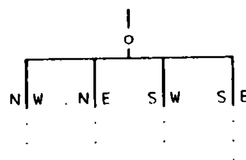
Of course, there are representations of two dimensional pictures other than pixel matrices and quadtrees, e.g., boundary codes and polygon edges. Since any one representation has its own advantages, it is useful to be able convert from one representation to another.

Samet [12] gives an algorithm for constructing a quadtree from the pixel array representation of a picture. This algorithm has a running time which is linear in the number of pixels in the picture. The reverse process, from a quadtree to a pixel array, is equally straightforward. The conversion of a quadtree to a pixel array is necessary for producing the display of a quadtree.

In [11] Samet gives an algorithm for converting from boundary code representations to quadtrees in time O(p * log d), where p is the perimeter and d the diameter of the object being represented. Dyer, Rosenfeld and Samet [3] give a complementary algorithm for converting a quadtree representation of a picture to a boundary code representation in time O(p).

A two-dimensional polygonal object may be represented by a sequence of vectors which describe the polygon boundary. Usually such vectors are stored as vector end-points. In [6] Hunter and Steiglitz present an algorithm for constructing a quadtree from such a polygon-vertex description of the boundary of a region in time O(v+p+q), where v is the number of vertices, p the perimeter of the region, and q is a resolution parameter. In [5] an algorithm is presented to perform a general linear transformation of a picture represented by a quadtree in time O(n + s*p + mq), where m is the





(a) Quadrant labels (b) Quadtree node labels

Fig. 2 The labels of quadrants of a square area, and the corresponding quadtree node sons.

number polygon edges in the picture, n is the number of nodes in the quadtree, and s is a 'scaling factor'. The resolution factor in both algorithms is a fixed constant for any particular application, but it is useful to know what the effect of changing the resolution is.

Dyer [2] presents an image processing application for quadtrees. The Euler number of an image region is the number of connected components minus the number of holes. Dyer gives an algorithm to find the Euler number of a region represented by a quadtree. The the running time of the algorithm proportional to the number of leaves of the quadtree.

As a means for storing two dimensional pictures quadtrees are both space- and time-efficient. The efficiency of tree structures is even more apparent in three dimensions.

4. Three Dimensions - Octrees

The transition from two to three dimensions is analogous to the transition from one to two dimensions. A unit cube is called a voxel (for volume element). The additional dimension is divided in two by recursive subdivision, and so cubes are subdivided into eight equal octants. One face of the universe cube is arbitrarily chosen to be the front, and the four octants adjacent to this face are labelled Front. Similarly, the other four octants are labelled Back. Fig. 4 shows the subdivision of a cube, the labelling of the eight octants, and the correspondence of octants to the sons of a tree node.

The definition of a tree, the octree, to represent three dimensional scenes is also analogous to the one and two dimensional cases. Assume that the voxels in the picture are either BLACK or EMPTY. Each node of the octree is either a leaf, or has eight sons each of which represents an octant of the father. A node is a leaf if all the voxels in the cube of space represented are the same colour. Fig. 5 shows a simple three dimensional picture and the octree which represents it.

A cube of three-space can be viewed as the various instances of a square frame of two-space in an interval of the third dimension, depth. As the frame moves through the third dimension, each pixel of the square will create several voxels. The colours of the voxels created will be independent of the colour of the original pixel. Although this view of three-space is unorthodox, it will be helpful when dealing with four dimensions.

Relatively little research has been conducted on octrees. Jackins and Tanimoto [7] discuss simple rotations of octrees (by multiples of 90 degrees), and an algorithm for the translation of octrees. Both algorithms are roughly linear in the number of nodes in the octree. Srihari [13] discusses various hierarchical representations of three dimensional pictures, including octrees. In addition, he discusses a structure similar to the octree except that the recursive subdivision need not divide a volume into cubes but may divide into rectangular parallelpipeds of unequal size. A similar approach is also discussed by Rubin and Whitted [10].

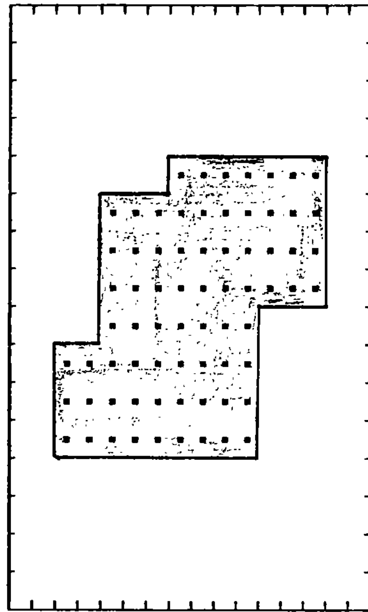
Regardless of the representation used for a three dimensional picture, it is still necessary to produce two dimensional images on a display screen. The primary problem in displaying three dimensional objects is the elimination of hidden surfaces. Many algorithms for the removal of hidden surfaces need to sort the surfaces of a scene by depth. Then 'rays' are extended from the viewpoint to find the closest surface at each pixel. Newman and Sproull [9] give examples of such algorithms. For most conventional picture representations, the sorting of surfaces is non-trivial, and the time required grows exponentially as the complexity of the scene increases [9]. An octree, on the other hand, does not suffer from this combinatorial explosion. Due to the organization of an octree the surface sorting is inherent in the structure of the tree. The surfaces which are hidden can be easily removed by a 'ray tracing' algorithm which traverses the octree from front to back. An algorithm follows which produces the quadtree represented by projecting an octree onto a plane coincident with its front face. A display of the quadtree can then be generated easily.

4.1. Octree to Quadtree Projection Algorithm

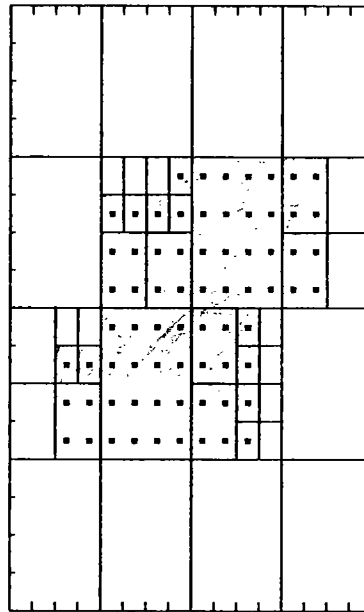
This algorithm begins with an octree, o, and constructs a quadtree, q, which is the orthogonal projection of o onto the front face plane. In any octree a BLACK front sub-octant will obscure any equal or smaller sized sub-octants behind it, for instance, BLACKs in the FNE octant will obscure any octants behind them in the BNE octant. If the octree o is traversed from front to back the hidden cubes can be discovered quickly and discarded.

A quadtree node will have 6 fields associated with it: a pointer to the father node, a colour (which can be BLACK, EMPTY, or MIXED), and four son pointers (labelled NW, NE, SW, SE). An octree node will have 10 fields associated with it: a pointer to the father node, a colour, and eight son pointers (labelled FNW, FNE, FSW, FSE, BNW, BNE, BSW, BSE). These are defined in Pascal as follows:

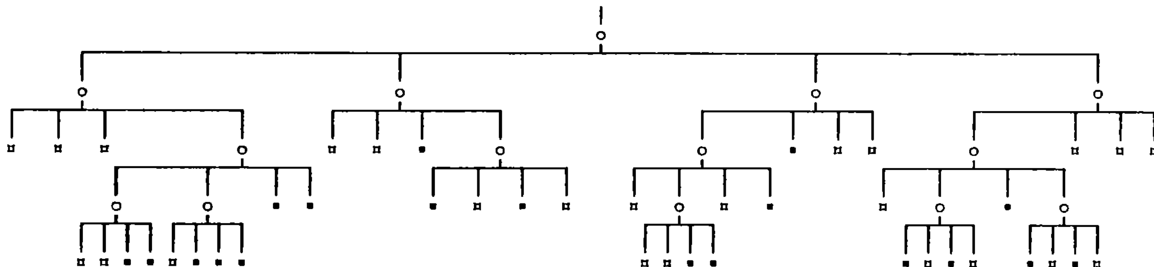
```
type
  Octants = (FNW, FNE, FSW, FSE,
             BNW, BNE, BSW, BSE);
  Quadrants = (NW, NE, SW, SE);
  Colours = (BLACK, EMPTY, MIXED);
  Oref = ↑Octree;
  Qref = ↑Quadtree;
  Octree =
    record
      colour: Colours;
      father: Oref;
      son: array( Octants ) of Oref,
    end;
  Quadtree =
    record
      colour: Colours;
```



(a) A two dimensional picture



(b) Recursive subdivision of (a)



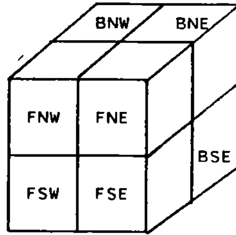
(c) Quadtree constructed from (b)

Fig 3. A two dimensional picture and the corresponding quadtree. The pixel array in (a) has 256 entries, while the quadtree in (b) has just 57 nodes. However, the quadtree nodes do take more space to store.

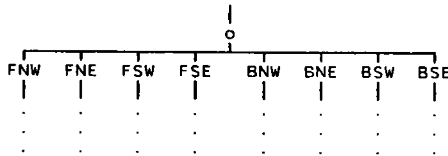
```
    father: Qref;
    son: array( Quadrants ) of Qref;
end;
```

The recursive traversal of the octree to produce the quadtree proceeds as follows. If the original octree is one solid colour (BLACK or EMPTY) then the quadtree will be one node of that same colour. However, if the octree is not a solid colour then the front sons are recursively projected towards their fronts, which is also the front of the top level octree. Any EMPTYs left after projecting the front sons can then be filled in by the back sons. Note that BLACKs created by the front sons will obscure anything behind them.

The projection algorithm is implemented by a procedure Project, which takes two parameters. The first parameter, o, is an octree which is being projected towards its front face. o is either the original octree or some descendant of the original. The second parameter, q, is a pointer to a quadtree which is to be constructed from o. Since o can be a 'backwards' son of the original picture it is possible that at least part of it may be obscured. q is used to pass such information along during the construction process. If q arrives as a **nil** pointer, then that quadrant of the quadtree has not been constructed yet, so a node is created for q and coloured EMPTY. However, if q isn't **nil** and the colour of q is MIXED or EMPTY then at least part of q remains which may be coloured BLACK. Finally, if q isn't **nil** and is coloured BLACK, then q has already been completely

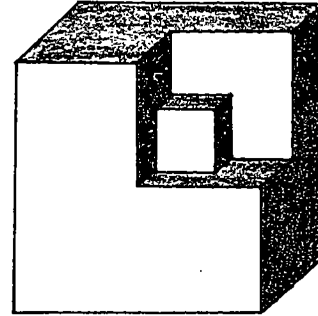


(a) Octant labels

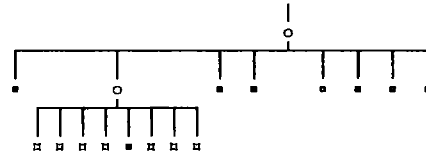


(b) Octree node son labels

Fig. 4 Labels of the octants of a cube, and the corresponding labels of the sons of an octree node. Note that octant BSW is hidden in (a).



(a) Three dimensional picture



(b) Octree representation of (a)

Fig. 5 A simple three dimensional picture and the corresponding octree.

coloured by octants which were further towards the front of the father octree, and so sub-octree o is completely obscured.

Now assume that q has not been previously filled (i.e. it is currently EMPTY or MIXED). If o is completely BLACK then the EMPTY quadrants of q are filled in with BLACK. If o is completely EMPTY then q is not changed at all. If o is MIXED (all sons are not the same colour), then the projection algorithm proceeds recursively by calling Project to project the sons of o onto the sons of q, starting with the front octants and proceeding to the back octants. The following is an algorithm for the procedure:

```
procedure Project( o : Octree, var q : Qref );
  if q is nil then
    create a new node for q, and colour it EMPTY;
  else
    if q↑ is coloured BLACK then
      return;
    endif
  endif

  (*
   * now, q is not nil, and if we reach this point
   * then q has quadrants (or sub-quadrants) which
   * have not been filled yet.
   *)
  if o is not coloured MIXED then
    if q↑ is coloured MIXED then
      project o onto the sons of q↑;
    else
      colour q↑ the same as o;
    endif
  else
    project the front sons of o onto sons of q↑;
    project the back sons of o onto sons of q↑;
    if all sons of q↑ are same colour, not MIXED then
      merge the sons into one node in q↑;
    endif
  endif
end
```

It should be clear that since this algorithm is merely a traversal of the octree o, the algorithm is linear in the number of nodes in the octree. However, not all nodes of the octree need be traversed since traversal occurs only if the corresponding area of the quadtree q has not been filled yet. Thus, in the average case, when there are back volumes which are occluded, the algorithm will not need to traverse the entire octree.

4.2. Applicability of Octrees

Octrees are quite suitable for use in three dimensional graphics. Usually they use less space than other representations of the same scene. Algorithms to manipulate octrees should be forth-coming in the near future as more people investigate tree data structures. In particular, if octrees are to become usable for graphics users and programmers algorithms must be developed to convert existing picture data bases from polygon representations to octrees.

However, some questions remain. For instance, it is not clear, at present, what effect applying linear transformations to octrees will have on the accuracy of the representation. Since an octree is an approximation to

the object being represented, there may be information lost (due to rounding errors) if several transformations applied.

5. Four Dimensions

The final extension, to the fourth dimension, is not as simple as the previous cases. To begin with, the fourth dimension, time, is not measured in the same units as the other three dimensions. As well, people have trouble conceiving what a four dimensional picture is.

A picture in four dimensions may be thought of as the various instances of a three dimensional scene as it changes, continuously, through time. However, for the purposes of storage time must be divided into arbitrary discrete steps. Time is split into k 'units' to be consistent with the other three dimensions, although time is measured in different units. The basic unit of four-space is called a tixel (time element) and consists of a unit of time and a voxel of three-space (during that unit of time).

The recursive subdivision for the tree representation divides the time dimension in half at each step. The 'earlier' half of the four-space will be referred as Past, and the 'later' half will be referred to as Future (in the same way that three space was divided into Front and Back). A hyper-volume of four-space is divided into 16 equal sized sections, hence the tree structure name is the hextree. The hextree is defined much like the lower dimensional trees: a node represents a hyper-volume of tixels and is a leaf if all the tixels of that volume are the same colour.

The meaning of 'all the tixels being the same colour' should be clarified. A tixel is a unit hyper-volume of four-space, i.e. a unit cube of three-space during a unit time interval. Thus, for all the tixels in a hyper-volume to be the same colour, the voxels of that section of three-space must remain the same colour throughout that interval of time. For example, if the voxels in a 4 by 4 by 4 volume of three-space remain the same colour for 4 time steps, then that volume of four-space need not be divided by the recursive subdivision. This is analogous to the three dimensional case discussed in section 4, when a voxel was considered to be the instances of a pixel through an interval of depth.

Real-time computer animation requires that several image frames be displayed in rapid succession. Generally, the difference between successive frames is small. Thus, if the frames are stored as a sequence of two- or three-dimensional scenes then there will be a great deal of redundant information, and wasted space, between consecutive frames. The hextree representation, on the other hand, takes advantage of any voxels which remain constant for more than one unit of time by merging them into one node. And so, a hextree will save space.

To generate a display of a hextree at an instant of time it is necessary to select the hyper plane of that particular time. Such a selection requires only to traverse the hextree constructing an octree which represents the three-dimensional scene at that time. Such an algorithm is similar to the octree projection algorithm given in Section 4.

Hextrees may provide an effective, structured approach to computer animation. They are particularly more efficient than producing a series of pictures for display and discarding each after it has been displayed.

6. Conclusions

This paper has presented a sequence of hierarchical data structures for use in computer graphics and image processing. Each structure was shown to be directly related to other structures, and that the space-efficiency of the structures increased with the number of dimensions.

In order for these tree data structures to become wide-spread there will have to be changes made to the programming techniques in current use by graphics programmers. The computer language most widely used at present is FORTRAN. It is well known that FORTRAN suffers from various flaws. The most serious flaws, for the purposes of implementing trees, is the lack of data structuring and recursion. And so, FORTRAN needs to be abandoned if tree data structures are to gain acceptance. The language that replaces FORTRAN will need data structuring capabilities, recursion, 'number crunching', and various graphic instructions to make picture displaying easy.

The gains from using tree data structures are several. The tree structures save storage space and execution time over other representations. As well there is the gain of portability of picture data bases if the common implementations are used.

References

[1] Alexandris, N. and Klinger, A. Picture Decomposition, Tree Data Structures, and Identifying Directional Symmetries as Node Combinations. *Computer Graphics and Image Processing 8* (1978), 43-77.

[2] Dyer, C.R. Computing the Euler Number of an Image from Its Quadtree. *Computer Graphics and Image Processing 13* (1980), 270-276.

[3] Dyer, C.R., Rosenfeld, A. and Samet, H. Region Representation: Boundary Codes from Quadtrees. *Communications of the ACM* 23 3 (1980).

[4] Horowitz, S.L. and Pavlidis, T. Picture Segmentation by a Tree Traversal Algorithm. *Journal of the ACM* 23 2 (1976).

[5] Hunter, G.M. and Steiglitz, K. Linear Transformations of Pictures Represented by QuadTrees. *Computer Graphics and Image Processing 10* (1979), 289-296.

[6] Hunter, G.M. and Steiglitz, K. Operations on Images Using QuadTrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-1 2 (1979), 145-153.

[7] Jackins, C.L. and Tanimoto, S.L. Oct-Trees and Their Use in Representing Three-Dimensional Objects. *Computer Graphics and Image Processing 14* (1980), 249-270.

[8] Mallgren, W.R. and Shaw, A.C. Graphical Transformations and Hierarchic Picture Structures. *Computer Graphics and Image Processing 8* (1978), 237-258.

[9] Newman, W.M and Sproull, R.F. *Principles of Interactive Computer Graphics.* New York: McGraw-Hill, 1979.

[10] Rubin, S.M. and Whitted, T. A 3-Dimensional Representation for Fast Rendering of Complex Scenes. *SIGGRAPH '80 Conference Proceedings* (1980), 110-116.

[11] Samet, H. Region Representation: Quadtrees from Boundary Codes. *Communications of the ACM* 23 3 (1980).

[12] Samet, H. Region Representation: Quadtrees from Binary Arrays. *Computer Graphics and Image Processing 13* (1980), 88-93.

[13] Srihari, S.N. Hierarchical Representations for Serial Section Images. *Proc. 5th ICPR Vol2 Miami Beach* (1980), 1075-1080.

[14] Tanimoto, S. and Pavlidis, T. Hierarchical Picture Data Structures. *Computer Graphics and Image Processing 4* (1975), 104-119.