Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer

Technical Report · October 1980

CITATIONS
223

READS
2,681

1 author:

Donald Meagher
Octree corp
20 PUBLICATIONS **1,497** CITATIONS

SEE PROFILE

Technical Report IPL-TR-80-111

Octree Encoding: A New Technique For The Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer

Donald J. R. Meagher

October 1980

Image Processing Laboratory

Electrical and Systems Engineering Department
Rensselaer Polytechnic Institute, Troy, New York 12181

ABSTRACT

A new 3-D geometric modeling technique called Octree Encoding is presented. Arbitrary 3-D (or N-Dimensional) objects can be represented to any specified resolution. The memory required is shown to be on the order of the surface area of the object. Algorithms are included for translation, scaling, rotation, union, intersection, difference, and the generation of a hidden surface view of any number of objects. None of the algorithms require floating point operations, integer multiplication, or integer division (except the most general form of perspective display which currently requires integer division). Large numbers of simple calculations are typically generated, allowing implementation over many high-bandwidth processors operating in parallel.

ACKNOWLEDGEMENT

The author wishes to thank Professor Herbert Freeman for inspiring, directing and guiding this research.

Funding for the use of the computer facilities was provided by the Department of Electrical and Systems Engineering, Rensselaer Polytechnic Institute.

Table of Contents

List of Tables

List of Figures

1 INTRODUCTION

The digital representation of three-dimensional objects is becoming an increasingly important part of computer applications in Computer-Aided Design (CAD), medical systems, robotics, finite element modeling and automatic object recognition, to name just a few. Many schemes have been proposed for generating and encoding 3-D models in digital computers. Several are currently used in commercially available systems but all contain inherent limitations based on the nature of the 3-D geometric modeling techniques used.

1.1 EXISTING SCHEMES

A survey of 11 existing systems has been compiled by Baer, Eastman and Henrion [1]. All make use of some form of edge-face-vertex representation of polyhedral solids in which the faces may be planar or curved. One such system, PADL, allows the user to generate complex 3-D objects using three operations (union, intersection and difference) and two solid primitives (cylinders and cuboids). Another, TIPS [2], allows a more extensive set of primitives. A half-space technique is used in which "existence regions" select a section of the primitive object.

All are characterized by a limited range of objects which can be represented and processed. One system [1] cannot operate on concave objects, for example. In general, as more complex

primitive solids are added to a scheme, the magnitude and complexity of the required calculations increases. Simply adding one new primitive or generalizing its use may require an extensive development of mathematical tools and substantial modification of the software.

An alternate scheme has been proposed by Gomez and Guzman [3] in which triangles are fitted to a surface. If a triangle exceeds some error threshold, it is subdivided into four smaller triangles.

The popular 2-D chain code [4] has been extended to allow the encoding of 3-D objects [5,6]. A few additional shape recognition techniques which have been effective in 2-D have been applied to solve 3-D problems [7].

1.2 GOAL OF THE EFFORT

The goal of the effort described here is to develop a 3-D encoding scheme in which an arbitrary object can be represented to a specified resolution and to develop a set of computationally efficient algorithms for manipulation and display which can be implemented in parallel, low cost hardware processors. Specificially, the goals are to:

- represent any N-Dimensional object to any specified resolution in a common encoding format
- operate on any object or set of objects with:
  - translation
  - scaling

- rotation
- union
- intersection
- subtraction (difference)

- implement a computationally efficient solution to the N-Dimensional interference problem (detect two or more objects occupying the same section of N-Dimensional space)

- display any number of objects from any viewpoint on a raster graphics terminal with:
  - color
  - shading
  - shadowing
  - multiple illumination sources
  - transparent objects
  - orthographic or perspective view
  - hidden surfaces removed
  - smooth edges (anti-aliasing)
- allow full color half-tone surfaces
- allow all manipulation and display algorithms to operate in the near future in real time or close to real time on relatively inexpensive parallel, high bandwidth hardware.

It is believed that all of these goals have been achieved with the Octree Encoding scheme presented here. Most of the algorithms have been implemented and demonstrated. The last goal, however, requires further elaboration.

At the beginning of the effort a number of cost trends were analyzed to provide guidance in algorithm development. The plan

was to minimize the use of functions and operations which will continue to be relatively expensive in the future and to allow liberal use of low cost alternatives. An overview of the analysis of future costs and the resulting algorithm design goals is as follows:

COST ANALYSIS	ALGORITHM DESIGN GOAL
- memory costs are decreasing	- allow liberal use of memory
- floating point operations are expensive	- no floating-point operations
- integer multiplications and divisions are expensive	- no integer multiplications or divisions (if possible)
- parallel calculations may be less expensive than serial calculations	- algorithms that can be implemented in many parallel processors
- computation cost vs. problem complexity	- performance that degrades gracefully as complexity increases
- computation cost vs. quality	- allow user to trade off computation vs. precision (time vs. picture fidelity, for example)

1.3 ORIGINS

Williams provided an introduction to early object representation techniques in a survey of graphic data structures [8]. Clark [9] has proposed hierarchical geometric models as the basis for future hidden surface algorithms. Multidimensional binary trees and algorithms have been studied by Bentley for use in data base applications [10,11]. Franklin has developed the "variable grid" technique for hidden line and surface applications [12]. It is shown to be a linear growth algorithm (rather than quadratic) at the expense of pre-sorting. This has been extended into a hierarchical structure in Octree Encoding and forms the basis of the linear computational characteristics of the scheme.

A two-dimensional tree structure, the quadtree, has been developed for image processing applications. An early study was performed by Sidhu and Boute [13]. Tanimoto [14] has proposed a similar structure as a measure of image complexity. Hunter and Steiglitz [15] have provided a comprehensive introduction to quadtrees. Rosenfeld [16] has also presented an overview of quadtree efforts being conducted at the University of Maryland. Samet has developed quadtree algorithms to compute the perimeter [17], perform deletion [18], convert from boundary codes [19], to boundary codes (with Dyer and Rosenfeld) [20], from raster format [21], to raster format [22], from binary arrays [23], compute the Medial Axis Transformation [24], and compute a distance function [25].

It should be noted that no translation, scaling or rotation algorithms for quadtrees have previously been published. Those presented below are applicable to quadtrees without modification.

An algorithm for calculating the volume of N-Dimensional convex polyhedra has been presented by Cohen and Hickey [26]. It performs a depth-first subdivision of parallelopipeds.

1.4 OCTREE ENCODING

Octree encoding utilizes a hierarchical N-dimensional binary tree to represent an N-Dimensional object. For a 3-D object, this is a 3-D binary tree. Each branch node is the parent of 8 children nodes at the next lower level forming an 8-ary tree or "octree". Each node represents a region of the universe and has one or more values which defines the region. If the value of the node completely describes the region, it is a terminal node or leaf. If not, an ambiguity exists and the node contains 8 children which represent the 8 subregions or octants of the parent node.

There are several advantages to this data structure. First, there is a single primitive shape, the cube. An arbitrary object can be represented to the precision of the smallest cube. Also, only a single set of manipulation and analysis algorithms is required for all objects. New techniques are not needed to handle more complex or sophisticated shapes.

A second advantage arises because of the hierarchical structure. The root node represents the entire object. In like

manner, the nodes at a level together with the higher nodes completely describe the entire object to the resolution of that level. Thus, algorithms can operate at a level appropriate for the task at hand and, hopefully, avoid the bulk of the data which is contained at the lower levels. In an application it might allow, for example, many objects to be represented at a high level (coarse resolution) in the main memory of a computer while the higher resolution detail resides on secondary storage until needed. This arrangement is very efficient for interference detection.

An additional advantage of the hierarchical structure lies in the ability of an algorithm to perform partial calculations which are then passed to the next lower level. In this respect it is similar to the FFT. Substantial reductions in computation can result.

Another advantage of the structure is that all objects are spatially pre-sorted at all times. By traversing the tree in the proper sequence, for example, regions of space will be visited in a uniform direction in space. Thus, the hidden surface display algorithm requires no searching or sorting. The trees representing the objects to be displayed are simply traversed in a specific order, depending on the view direction.

Boolean operations benefit from the structure in that the algorithms simply traverse the two (or more) input trees in order while generating the output tree.

None of the algorithms developed to date (union, intersection, subtraction, translation, scaling, rotation and

numerous display functions) require floating point operations or integer multiplications or division except the most general form of perspective display which currently requires integer division. Thus, the algorithms can be implemented in relatively inexpensive, high bandwidth hardware.

In addition, the processing of each node generates zero to 8 independent sub-calculations. Thus, a system can be envisioned in which large number of very simple processors perform the operations.

An important capability of a 3-D octree is its ability to act as a structure to position color and intensity information on the surfaces of objects. It becomes the 3-D equivalent of a 2-D image. Each resolution point on the surface (or interior, if needed) of an object would contain a color half-tone value. This combined with the hidden surface feature could be very important in medical and movie applications.

The main disadvantage of the encoding technique is the large memory requirements. Hunter and Steiglitz [15] present a proof that the quantity of memory required to store (and the calculations to process) a quadtree 2-D object is on the order of the perimeter of the object. A proof is given below which shows that the memory and processing computation required for a 3-D object is on the order of the surface area of the object. Depending on the object and the resolution, this can still represent a large memory storage requirement.

In a particular situation, if ideal primitive objects (cubes, spheres, cylinders, etc.) are sufficient to accomplish the task,

Octree Encoding is probably not the technique to employ because of the amount of memory required. If, however, the objects are to be deformed or have complex surfaces, the octree techniqes could hold important advantages.

1.5 APPLICATIONS

The following are a few potential areas of application:

- High Speed Interference Checking in CAD Systems - Because of the computational efficiency (point checking in log time) of interference detection, especially when large numbers of objects are involved, real time interference checking on very large data bases becomes possible. This could be valuable, for example, in a CAD system for building design. The pipe layout designer could be informed immediately when an interference problem was encountered rather than employing off-line checking.

- NC Tape Verification - The entire NC machine as well as the cutter and workpiece could be modeled in an Octree system. Correct operation could be checked automatically or interactively.

- Robotics - The ability of these techniqus to manipulate large numbers of very complex 3-D objects at high speed can be applied to modeling the robot workspace and proposed action commands. Also, a number of the difficult calculations required in robot arm control are greatly simplified by Octree techniques. A simple algorithm has been

developed to rapidly determine the shortest distance between two complex objects, for example. Also, region growing for clearence determination is simple and efficient in an Octree environment.

- Medical Imaging - The ability to efficiently display 3-D gray scale objects from any viewpoint with hidden surfaces removed could be important in medical applications. The techniques presented below could be used, for example, to generate 3-D color images of selected organ systems from multiple CAT or CAT-like scans. The generation of interior cut-away views would be a basic feature. Given sufficient hardware, such examinations could be conducted interactively.

- Cinematographic models - The Octree technique could probably be used very effectively in the generation of film sequences involving object models. Instead of constructing physical models containing thousands or tens-of-thousands of parts, the object would reside in the computer. This could also be applied to generation of animated movies.

- Object Recognition - Substantial effort has been applied to the automated recognition of 3-D objects. Applications include industrial automation, identity verification, aircraft identification, and so on. The hierarchical nature of the Octree technique could facilitate object recognition. A view of an objet could be compared to an Octree model of an object at a high level to determine if a coarse fit exists before expending the larger amount of computation required to perform a comparison at a lower

level.

1.6 STATUS OF IMPLEMENTATION

Program OCTREE has been written to develop, verify and demonstrate the Octree Encoding algorithms presented below. It currently is implemented in Fortran on the Prime 750 computer at the Image Processing Laboratory at Rensselaer Polytechnic Institure. A DeAnza IP5000 color imaging system is used for display.

The following functions have been implemented and verified to operate properly:

OBJECT GENERATION

- 2-D object entry via lightpen or graphics tablet
- 2-D object entry from equations
- 3-D object entry via:

  3 views

  multiple 2-D slices (lightpen or graphics tablet)

  equations for 2-D slices

OBJECT MANIPULATION

- 2-D and 3-D

  translation

  scaling

  rotation (90 deg., 180 deg., reflection about axis)

  rotation (0 to 90 deg.)
- N-Dimensional

union

intersection

subtraction

DISPLAY

- 2-D objects (area or edges)

- view of 1 to 10 objects in color with hidden surfaces removed from arbitrary location (within 90 deg. in 3 dimensions):

orthographic

orthographic (subpixel for anti-aliasing)

perspective

perspective (subpixel for anti-aliasing)

2 N-DIMENSIONAL OBJECT REPRESENTATION AND STORAGE

2.1 DEFINITIONS

In this section, definitions will be given for basic terms which will be used throughout the report.

2.1.1 GRAPH

A graph G(N,E) is a finite, nonempty collection N of nodes and a set E of unordered pairs of distinct nodes called edges. Two nodes connected by an edge are adjacent nodes. If an edge has an associated direction, it is a directed edge. The direction is from the tail node to the head node. A graph containing only directed edges is a directed graph or digraph. The number of edges which have a node as their tail node is the outdegree of that node. The number of edges which have a node as their head node is the indegree of that node. A graph which has no paths which originate and end in the same node is called acyclic.

A tree is an acyclic directed graph in which all nodes have indegree 1 except one node, the root, which has indegree 0. Any node which has outdegree 0 is called a terminal node or leaf. Nodes with outdegree greater than 0 are branch nodes. The level is defined as the distance in edges from the root. The root is at level 0.

The root is assumed to be at the top of the node structure and all other nodes exist below the root. All nodes that are reachable from a particular node are called the descendants of that node. All nodes from which a particular node can be reached are the ancestors of that node. Descendants which are one level below a node are the children of that node. The original node is the parent of the child node.

If a node does not actually exist in a tree but can be inferred from an existing terminal node which would be (or will be) one of its ancestors, it is called an implied node. Loosely, operations on a tree which use implied nodes are said to process the implied tree rather than the actual tree.

Every branch node is a root of one or more subtrees. The subtrees immediately below a node form a set of disjoint trees called a forest. The degree of a node is the number of subtrees that exist for that node. If the outdegree of every branch node is <= m, the tree is an m-ary tree. If the outdegree of every branch node is m, the tree is a complete m-ary tree. For a binary tree or a complete binary tree, for example, m=2.

A positional m-ary tree is an m-ary tree in which the children have m distinct positions. The position of a node is indicated by a value from the child number set {0,1,2,...,m-1}.

Every node is uniquely identified by a node address which is a string over the child number set. The root is represented by the empty string. The node address of a child is the child number prefixed by the address string of its parent.

2.1.2 UNIVERSE

All objects exist within the universe. It is a finite section of N-dimensional space defined by N orthogonal axes and 0<=x(i)<=e where x(i) is a displacement in dimension i, (x(1), x(2),...,x(N)) is a point in the universe, e is the length of an edge of the universe and N is the order of the universe.

Note that all edges of the universe have the same length forming a square for N=2, a cube for N=3 and an N-dimensional hypercube for N>3. The origin of the universe is the point of intersection of the axes. Negative displacements from the origin are not allowed. The space beyond the universe is the void. No object can exist in the void. Any part of an object moved into the void is annihilated. An augmented universe is one in which one or more adjacent (empty) universes are added to the primary universe. Augmented universes are used to facilitate algorithm initialization.

2.1.3 OBJECTS

Before encoding, objects are called real objects. They may be real world objects or a mathmetical description of an ideal shape. An object encoded in the Octree format is known as the encoded object or simply the object.

If a single encoded object is used many times to generate new, transformed objects, the original object is the model and the new objects are instances.

An object as defined in the context of this report can have any number of dimensions. A one-dimensional object is one or more segments of the axis forming a one-dimensional universe. A 2-D object occupies area, a 3-D object occupies volume, a 4-D object can be thought of as occuping spacetime, and so on.

An object is always of the same order as the universe in which it is defined and is composed of discrete units of N-dimensional space. All objects in a third order universe must occupy volume, for example. A 2-D object could not exist here. The smallest object in such a universe would be the smallest resolvable unit of space. No object or part of an object can occupy a point (zero volume).

Other than the above, there are almost no restrictions on objects. They can be concave as well as convex, have any number of holes including interior holes, and can be composed of multiple disjoint parts.

Each object is defined over the entire universe. It has a property value defined at each point in the universe. For a typical small object (relative to the universe) most of the space in the universe has the property of being empty.

An encoded object B is thus defined as a family of ordered pairs $B(k)=(P,E(k))$ where P is a finite set of properties and E(k) is the set of disjoint object elements or <u>obels</u> which exactly fill the universe at level of resolution k in a manner described below. In Octree Encoding, the obels which constitute an object are represented by the nodes in a tree structure. The tree contains all members of the family of objects of increasing

resolution up to some maximum level of resolution.

The "property" as used above could be a simple description of the obel such as "empty", "partial", or "full", indicating that the obel is entirely free of the object, partially filled with the object or is completely occupied by the object, respectively. Or it could be a much more complex description containing such items as material type, color, function, density, thermal conductivity, etc.

It should be noted that a characteristic could be included as an additional dimension of an object rather than as an obel property if it were desirable to perform sorting or searching operations over that characteristic. It could necessitate a substantial increase in resource utilization such as memory, however.

2.2 OBJECT REPRESENTATION

Take a positional m-ary complete tree in which each node represents a section of N-dimensional space. Divide the section in each dimension represented by each parent into p equal subsections or segments numbered 0 to p-1. This forms p**N disjoint sections of N-dimensional space j(i)(e/p) <= x(i) <= (j(i)+1)(e/p), j(i)=0,1,2,...,p-1 & i=1,2,...,N where i is the dimension, x is a displacement from the origin of the parent and e is the length of an edge of the parent.

The sections of space are the obels of an object and are represented by the m=p**N children of the parent.The root of the

tree represents the universe and contains all N-dimensional space. The level of the obel is the level of the corresponding tree.

The child number is defined to be:

$$\sum_{i=1}^{N} p^{**}(i-1)*s(i)$$ where s(k) is the segment number in the k-th dimension

The child number set is thus {0,1,2,...,p**N-1}.

For the remainder of this report, only a value of p=2 will be considered. A binary complete tree in each dimension will therefore be used forming an N-dimensional binary tree (a positional m-ary complete tree in which m=2**N).

Thus, a one-dimensional binary tree is simply a binary tree. A 2-D binary tree is a quadtree. A 3-D tree has 8 children and will be called an octree. A 4-D tree has 16 children and could be called a hexadecatree.

The representation of 3-D objects by N-dimensional trees of any order will be called volume encoding. Octree Encoding is thus a form of volume encoding.

This report will only consider terminal nodes which are completely defined by the corresponding properties. In other words, the space occupied by a terminal node is homogeneous. More complex forms are possible such as half filled obels separated by a diagonal plane. This would complicate the processing of terminal nodes in the manipulation algorithms. Perhaps families of compatible forms could be developed with high speed manipulation based on table driven processors generating

subtrees.

The tree address of a node identifies a particular node. In addition, it gives the series of traversal commands to locate the node relative to the root. It also identifies the section of space represented or covered by the node.

A node address in a 1-D tree is a binary string. The number of bits used is equal to the level of the node. The value is the number of the section of the 1-D universe covered by the node, numbered from 0 at the origin to 2**k-1 where k is the level. The covered section of a higher order universe can be likewise determined by independently considering the individual bit for each dimension in the child number values of the node address string.

2.2.1 ANALYSIS

Several advantages of the tree structured encoding are immediately obvious:

(1) Objects can be manipulated at many levels of resolution. Operations requiring only coarse information can process the entire object at a high level with a relatively small volume of data and computation. All of the data concerning the object need not be examined. When additional detail is required, it is available either over the entire object or in selected sections. High level information on a large number of objects could be kept in main memory while the lower level data are maintained in lower cost secondary

storage.

(2) Computations can be reduced because partial calculations at a level can be passed to the children. The object manipulation and display algorithms presented below require only integer calculations. In addition, no multiplications or divisions are required, except for the most general form of true perspective display (currently requires integer division).

(3) Many independent sets of calculations are generated by the processing and display algorithm, raising the possibility of implemention on large numbers of very simple high bandwidth hardware processors.

(4) Octree encoding is a true volume representation of a 3-D object. There are no zero volume entities such as points or lines or areas. This eliminates the special cases of tangent objects, zero volume surfaces, etc. possible with other techniques. All objects are guaranteed to be well-formed and physically realizable.

(5) There is no restriction on objects except that they possess N-dimensional space within the resolution of the system.

(6) All objects are spatially pre-sorted. This holds significant savings for operations such as hidden surface removal which require a spatial ordering.

(7) There is an advantage in the ability to attach a property to each obel. In three dimensions, assigning a color to each obel on the surface of objects allows "object

processing" as a 3-D extension of image processing.

A future extension to the tree structure would be generalization to a graph structure in which sibling and neighbor edges would be present. This would facilitate traversal over the surface of an object. Threading nodes across a surface is also possible, as is the threading of features such as vertices.

The main disadvantage of the tree form of representation lies in the use of memory. Compared to other techniques, a large amount of storage is required to represent ideal primitive objects such as cylinders and spheres. Simple manipulation of a small number of such objects may also be costly in computation compared to more analytical techniques. For situations where such manipulations are sufficient, this scheme may be inefficient. For situations where simple primitive shapes cannot easily represent the objects or where large numbers of objects are being manipulated, the tree structure could be a very cost effective alternative. In some situations, it may be the only practical alternative currently available.

2.2.2 MEMORY REQUIREMENTS

The actual volume of storage required to represent an object is a function of the size and shape of the object, its position and orientation when digitized, the level of resolution, etc. Of particular significance is the result that the surface area of a 3-D object sets an upper bound on the memory required.

Theorem. For a 3-D connected object, the size of the

encoding is on the order of the surface area of the object.

Proof. A function g(n) is defined to be on the order of f(n) or O(f(n)) if there exists a constant c such that g(n)<=c*f(n) for all but some finite (possibly empty) set of non-negative values for n.

Consider a 3-dimensional universe defined to level n. The edge of an obel at any level is e. Without loss of generality, the edge of an obel at level n is defined to be 1. An edge at level k is 2**n/2**k or 2**(n-k). The surface area of an object within the universe which intersects 8 obels (any level) and continues into a ninth must have a surface area >4e+2. The minimum object intersects 8 obels at and around the common point at which all 8 touch and continues along an edge for the entire length of an edge until it enters obel 9.

The minimum object which can touch all 9 (without intersecting all 9) is a linear run of minimum level obels for a distance e. It has a surface area of 4e+2. Thus, for an object to actually intersect all 9, it must be larger than this and have a larger surface area.

Let S be the surface of an object. Let m be the number of cubes at level k which could be required to represent the object.

In a worst case situation, the surface area would cover a maximum length run of minimum level obels, which sets a limit on the number of obels at level k which can be intersected:

$$m<8(S/(4e+2)+1)$$

The 1 accounts for (actually, more than accounts for) the four obels which could be intersected along with obel 9 at the

far end of the run.

m<8(S/4e+1)

m<8(S/(4*2**(n-k))+1)

m<2S/2**(n-k)+8

Let L be the total number of obels or nodes required to represent an object:

$$L<\sum_{k=0}^{n}(2S/2^{n-k}+8)$$

let r=n-k and reverse the order

$$L<2S*\sum_{r=0}^{n}(1/2^{r})+8*\sum_{r=0}^{n}(1)$$

L<4S+8(n+1)

or

L=O(S+n)

or

L=O(S) for fixed n

2.3 OBJECT STORAGE

A minimum usable scheme requires two data items per node, a property value and a pointer to its children (if a branch node). Additional data items which could be used are parent pointers, multiple property values, average subtree properties, sibling pointers and object feature pointers.

Parent pointers are probably not necessary because in any real application the number of levels is limited. With a 32 level octree, for example, objects could be represented to a

resolution of 0.001 inch in a universe enclosing 311,482.8 cubic miles. In such a situation, depth-first traversal algorithms could keep parent pointers in a small stack.

The binary object property will be used in most of this report. A terminal node will be either FULL (corresponding obel completely occupied by the object) or EMPTY (corresponding obel completely disjoint from the object). Branch nodes are PARTIAL meaning partially full. It is assumed that additional properties such as color values are simply attached to FULL terminal nodes. Additional data storage locations may be appended to FULL nodes for this purpose. The algorithms below can be easily extended to process such trees if an operation is defined to combine properties for the various transformations.

Before implementing a node storage format, two major design considerations should be evaluated:

(1) Depth-First vs. Breadth-First Traversal

(2) Sequential vs. Linked Allocation

2.3.1 DEPTH-FIRST VS. BREADTH-FIRST TRAVERSAL

A depth-first algorithm generally traverses a tree downward from parent to child, returning to the parent when all lower nodes have been processed. Breadth-first traversal processes nodes at one level before working at the next lower level.

In an application, tree traversal is not necessarily restricted to either depth-first or breadth-first but system performance may be optimized by allocating memory according to

the predominant traversal strategy. This could, for example, minimize page faults in a virtual memory environment.

A depth-first algorithm tends to be spatially oriented. It has advantages when an overall spatial ordering is needed such as display with hidden surfaces removed. Examination of a selected point in space is also facilitated. This is important when high resolution information is needed in a limited section of the universe such as is necessary in detecting interference.

Depth-first operations typically use a stack either directly or via reentrant code to maintain tree location.

Breadth-first traversal is generally more object oriented. The entire object is processed at increasing levels of fidelity. Operations can be performed on an entire object at reduced precision before the bulk of the object data has been processed. The results can be used for decision making or passed to other operations in a pipeline arrangement. Many objects can be in process with results at a high level (low resolution) becoming available before the lower level detail of objects are even accessed in memory.

In operation, information is passed from one level to the next in a queue.

2.3.2 SEQUENTIAL VS. LINKED ALLOCATION

Sequential allocation as applied here makes use of the position of a value in a string to indicate its tree address. Explicit pointers are not needed. This is sometimes called heap storage. The allocation can be breadth-first or depth-first. The major advantage is in reduced memory use. The major disadvantage is that, similar to a magnetic tape, all earlier nodes must be read before a desired node can be located.

In linked allocation, each node contains the memory address of its children. It requires considerably more memory but is faster for selective depth searches. Also, dynamic memory management can be performed.

The use of linked allocation probably means the entire tree is kept in main memory although schemes could be developed to bring in lower levels from secondary storage as needed.

In a real system, the performance of one allocation method relative to the other will depend on the algorithms and the objects being processed. If a regular traversal in the same order as physical memory is performed over all nodes, sequential allocation would be preferred. If large numbers of branches were left unvisited, linked storage could probably show superior performance.

In any event, sequential would probably be used for bulk storage, perhaps with automatic link address generation in hardware when accessed.

2.3.3 SEQUENTIAL ALLOCATION SCHEMES

The first sequential allocation scheme to be investigated requires two bits per node. The four possible node values are defined as follows:

Bits	Value	Property
00	0	EMPTY (Terminal node which is empty)
01	1	PARTIAL (Branch node)
10	2	FULL (Terminal node which is full)
11	3	ESCAPE CHARACTER (special value follows)

The use of the escape character is left undefined. It indicates that a value follows which has special meaning.

Figure 1(a) shows the breadth first storage format. A heap storage format is used in which terminal nodes are allowed anywhere in the string. They are simply not continued. This does not allow storage addreses to be computed directly as with normal heap storage. Figure 1 (b), (c), and (d) are examples of depth-first sequental formats.

Individual property values can be added after FULL terminal nodes. Thus the 24 bits after a node value of 2 could be 3 bytes of intensity, one for each of the primary colors. Alternately, the escape character could be eliminated and a value of 3 could indicate a full node which is in the interior of the object and therefore does not carry a color. This would complicate some algorithms such as subtraction, however.

An alternate mapping would be as follows:

Bits	Value	Property
00	0	EMPTY
01	1	PARTIAL (<0.5 full)
10	2	PARTIAL (>=0.5 full)
11	3	FULL

At display time, only PARTIAL obels of type 2 (>=0.5 full) would be displayed at the bottom display level.

An even more compact representation requires less than two bits per node:

Bits	Property
0	EMPTY
10	PARTIAL
11	FULL

In this case, the escape character is eliminated or replaced by some illegal or seldom-used string. The code must contain a value which indicates the non-escape condition.

The property represented by the single bit value could, as an alternative, be PARTIAL or FULL if, based on experience, more compact files would result.

2.3.4 LINKED ALLOCATION SCHEMES

The simplest form of linked storage would require a single bit to determine the type of node (branch or terminal) and a multiple bit value containing either a pointer for branch nodes or a property value for terminal nodes. In a given situation, this might require a total of 25 bits but in a typical

application on an existing computer a full 4 byte (32 bit) storage location would probably be used. The most significant bit could be the branch/terminal bit so that sign checking instructions could be used. The value associated with a terminal node could directly contain a color or property value or point to a table of such values.

2.3.5 IMAGE CONVERSION

It is possible to convert a 2-D image array into a 3-D octree. It could then be manipulated in 3-dimensions.

Theorem. A square image (8 bits for each of 3 colors per pixel) converted into a 3-D octree (using 2 bits/node sequentially allocated plus color values) where pixels form an orthogonal plane of minimum level obels requires about a 22% increase in memory.

Proof. The image will be placed on a face of the universe (any orthogonal plane will do). Each nonterminal node will have 4 EMPTY terminal nodes and either 4 FULL terminal nodes or 4 branch nodes. Thus, the number of nonterminal nodes at level k is 4**k. The number of nodes at level k+1 is 8*4**k or the number of nodes at level k is 8*4**(k-1). The number of bits required for nodes at level k is thus 16*4**(k-1). The total number of bits required for nodes down to the pixel level, level n, is:

$$16*\sum_{k=1}^{n}(4**(k-1))$$

Note level 0 not included because no memory is required.

The number of pixels at level n is 2**n by 2**n. The number of bits for pixel storage is 24*4**n. The fraction of extra memory required is f.

$$f=16*\sum_{k=1}^{n}((4**(k-1))/24*4**n)$$

$$f=(2/3)\sum_{k=1}^{n}(4**(k-n-1))$$

let j=n-k+1

$$f=(2/3)\sum_{j=1}^{n}(4**(-j))$$

but, from the binomial series:

$$\sum_{i=0}^{inf}(x**i) = (1-x)**(-1) \text{ or } \sum_{i=1}^{inf}(x**i) = (1-x)**(-1)-1$$

thus, f=(2/3)((1-(1/4))**(-1)-1) = 2/9 or about 22%

Something of a worst case transformation of such an image would then be to uniformly distribute the pixels (now 3-D obels) throughout the universe. This, of course, would greatly increase the memory required. Consider, for example, a 1024 by 1024 pixel image. Calculations show that such a distribution would require about 27.5 million nodes or about 6.89 million bytes. This divided by the 3.15 million bytes in the image results in about a 219% increase in memory.

2.4 ALGORITHM ENVIRONMENT

The general environment which was assumed for the algorithms developed to date is as follows:

(1) Virtual memory - Because of the large tree structures required, it is believed that an addressing space in the millions of bytes is required.

(2) Breadth-First Implementation - A pipelined system is proposed. This would have the advantage of rough image generation very quickly following a user request, and improvement in fidelity with time. It is hoped that very often a decision can be made and a new command given before the vast bulk of the object information has been processed.

(3) Sequential Allocation of Memory - This is assumed because of the expected high page fault rate for a linked scheme and because of the increase in object complexity allowed by more efficient memory usage.

(4) Special Hardware - It is assumed that in future systems most of the algorithms will be implemented in hardware processors.

3 OBJECT MANIPULATION ALGORITHMS

Algorithms for the generation of quadtree encoded objects and their manipulation via Boolean operations have been published elsewhere [15,16,19,20,21, and 23]. The extensions required to apply them to Octree Encoding is straightforward and are not presented here for brevity. Algorithms in this section will cover geometric manipulation of N-Dimensional objects.

3.1 DEFINITIONS

Most of the algorithms which follow can be applied to objects defined over any number of dimensions. Unless otherwise noted, the number of dimensions in use will be represented by N and must remain fixed. In general, an attempt is made to indicate a particular dimension by K (1<=K<=N).

A node is assumed to be an actual node within a tree structure. Its node address is the storage location in memory. An obel is a data item used within an algorithm which represents an actual node or an implied node (descendent of a terminal node). The level of an obel is the tree level of the corresponding node or implied node.

The status of an obel is a variable over {EMPTY,PARTIAL,FULL,IGNORE}. The first three values correspond to node values. The last, IGNORE, enables the algorithm to mark an obel as of no further significance.

An overlay is a section of an augmented universe composed of

a number of obels at a common level. An overlay represents a part of the "old" (input) universe which completely overlaps or "overlays" an obel in the "new" (output) universe. The value of an obel in the new universe is completely determined by the obels in its corresponding overlay or their descendents.

The "case" of an overlay is determined by the position of the overlay relative to the origin of the obel in the new universe which it covers. There is a separate case value in each dimension. The case of an overlay is used to generate the overlays at the next lower level and generate new node values.

A transform record contains the transform parameters passed to an algorithm. For the translation algorithm, it contains a movement value for each dimension, for example.

3.1.1 ALGORITHM DATA FORMATS

The standard ALGOL formats have been extended to include compound records which are records containing one or more fields, each of which can be a single data item or another record of a more primitive format. Fields are numbered from 1.

In addition to integer, Boolean and string the following data formats are defined:

vector - a record containing N integer fields

obel - a three field record as follows:

1) field: STATUS

   format: string

   value: over {EMPTY,PARTIAL,FULL,IGNORE}

2) field: NODE_ADDRESS

format: integer

value: storage address of node for STATUS=PARTIAL

undefined otherwise

overlay - a record containing a level field and a number of obel fields (number dependent on algorithm):

1) field: LEVEL

format: integer

value: level of obels in overlay

2) field: OBEL_0

format: obel

value: obel in first overlay position (position 0)

...

m) field: OBEL_n

format: obel

value: obel in n-th overlay position (n determined by algorithm and number of dimensions)

overlay pointer - a three field record defined as follows:

1) field: TYPE

format: string

value: over {POINTER,IGNORE}

2) field: OBEL_NO

format: integer

value: pointer to an obel in an overlay if TYPE=POINTER

undefined otherwise

3) field: CHILD_NO

format: integer

value: pointer to a child of obel defined by OBEL_VALUE
if TYPE=POINTER,
undefined otherwise

3.1.2 PROCEDURES

This section will define several common procedures. The more ad hoc procedures will be presented with specific algorithms.

Procedure names are kept unique throughout the report. Occasionally an existing procedure will be slightly modified. The name will be changed by appending 2,3,4, and so on. Thus, CASE becomes CASE2, for example.

The following procedures access the corresponding field within a data format:

```
string<-STATUS(obel)
integer<-NODE_ADDRESS(obel)
integer<-LEVEL(overlay)
string<-TYPE(overlay_pointer)
integer<-OBEL_NO(overlay_pointer)
integer<-CHILD_NO(overlay_pointer)
```

Additional data access procedures are defined as follows:

```
VALUE<-VECTOR_VALUE(VECTOR,FIELD);
    value vector VECTOR;
    value integer FIELD;
    integer VALUE;
```

Returns as VALUE the number in field number FIELD (1

to N) of VECTOR.

OBEL<-OVERLAY_OBEL(OVERLAY,OVERLAY_NO);

value overlay OVERLAY;

value integer OVERLAY_NO;

obel OBEL;

Returns as OBEL the obel in overlay position OVERLAY_NO (0 to n) of OVERLAY.

OVERLAY_SET(OVERLAY,OVERLAY_NO,STATUS,NODE_ADDRESS);

overlay OVERLAY;

value integer OVERLAY_NO,NODE_ADDRESS;

value string STATUS;

Sets the value of the obel in overlay position OVERLAY_NO (0 to n) of OVERLAY. The status is set to STATUS and the node address to NODE_ADDRESS.

BIT<-BIT_VALUE(VALUE,BIT_POSITION);

value integer VALUE,BIT_POSITION;

integer BIT;

Returns in BIT the binary value of VALUE in bit position BIT_POSITION. The LSB of VALUE is bit position 1.

The following utility procedures are defined:

CHILD<-CHILD(OBEL,CHILD_NO);

value obel OBEL;

value integer CHILD_NO;

obel CHILD;

Child number CHILD_NO of OBEL is returned. Note: If STATUS(OBEL)='PARTIAL' the tree is consulted to determine

CHILD. Otherwise, the status of OBEL is returned in CHILD.

Q<-INSERT(TOKEN);

Q<-DELETE(TOKEN);

The record TOKEN is inserted into or deleted from the queue. TOKEN can be of any record format.

Q_EMPTY;

Predicate function with value TRUE when the queue is empty and FALSE otherwise.

VALUE<-EVAL(OVERLAY);

value overlay OVERLAY;

string VALUE;

The overlay OVERLAY is evaluated with the results being returned in VALUE over {EMPTY,PARTIAL,FULL}. VALUE is computed as follows:

'EMPTY' if TYPE='EMPTY' or 'IGNORE' for all overlay obels

'FULL' if TYPE='FULL' or 'IGNORE' for all overlay obels

'PARTIAL' otherwise

Note: TYPE='IGNORE' for all overlay obels is an illegal condition.

OUTPUT(VALUE,LOC);

value string VALUE;

integer LOC;

VALUE is output to the new tree being generated at location LOC. It is over {EMPTY,PARTIAL,FULL} and is

assumed to be the next value in the new tree. For breadth-first algorithms it is the next value in a breadth-first sequence. For depth-first algorithms it is the next value in a depth-first sequence. The structure of the new tree is via sequential allocation but is easily changed to linked allocation.

REDUCE(ROOT);

value obel ROOT;

Working from the bottom of the output tree, removes any redundant nodes (nodes with all terminal children of the same type).

3.2 TRANSLATION

The goal of the translation algorithm is to convert a tree representing an N-dimensional object and a movement vector into a new tree representing the translated object. The movement vector specifies a translation value to some precision, i.e. at some level for each dimension. An informal description of the algorithm follows.

The process begins by generating an augmented "old" universe composed of the universe containing the original object and a number of empty universes. It is put together so that the "new" universe containing the translated object is covered (completely enclosed) by the old (augmented) universe. The translation vector specifies the alignment of the new universe relative to the old.

Beginning with the root of the new universe, the basic strategy is to traverse the implied tree for the new object and generate node values by simultaneously traversing the tree representing the old universe. If a terminal value for a node in the new tree is generated, no descendants of that node need be considered. If an ambiguity exists and, therefore, the status cannot be resolved, a nonterminal node is generated and the children of that node are generated in like manner.

Information is passed from the parent to the children through a queue for breadth-first traversal or a stack for depth-first traversal. The algorithm presented below was developed for a sequential access breadth-first environment but a depth-first version is also presented. No change to the algorithm is needed for use in a linked allocation environment.

It is possible for the new object to extend to levels well below the original object. This happens when the movement vector specifies a move at a higher resolution (lower level) than the object. If an ambiguity continues to exist, the algorithm will continue generating nodes down to the lowest level of the movement vector before the terminal value is resolved. The maximum resolution of the new object could be determined by controlling the actual resolution of the translation vector. In some later algorithms such as rotation, however, the child generation can continue indefinitely.

To handle this situation, the concept of the minimum level of operation of an algorithm is introduced. It is the level at which all nodes are forced to be terminal nodes. A value is

generated based on information available at that level. It should be noted that this could also be used when it is not necessary for the new object to contain all of the lower level detail of the original object.

Several methods of forcing a value are possible. The value could be selected randomly. Or, some form of averaging based on known terminal values could be calculated, perhaps involving the checking of one or more lower levels.

In the algorithms presented in this report, all unresolved terminal nodes are set to a value of FULL. This is selected because of possible applications to the N-dimensional interference problem where worst case interference must be assumed. This does, however, lead to an increase in volume at the surface of an object. This error is limited to approximately the surface area of the object times the size of an edge of an obel at the minimum level and is minimized by a low minimum level. The error is cumulative, however, with each move and could become significant. For this reason, it is assumed that in an application, fresh instances will be generated from the original model object rather than repetitive incremental manipulation of a single object.

3.2.1 NODE GENERATION

There are a number of schemes which can be used to traverse the tree structures and generate node values. The algorithm presented here makes use of overlays. The value of a node or obel in the new universe is completely determined by the obels in its overlay record or their descendents. The overlay obels present a picture of the section of the old universe which covers the new obel. They can be at any level relative to the target (new) obel. The lower they are below the target, the more numerous they are but the more accurate the result. The output tree will be more nearly the final reduced tree. This comes at an increase in memory use and computation. The TRANSLATE algorithm presented here uses an overlay with 2**N obels at the same level as the corresponding obel in the new universe.

Figure 2 illustrates the overlay in one dimension. Distance is positive to the right. Three obels are shown. The new obel has an edge distance of e. Its lower end is the local origin. The overlay is made up of two adjacent obels of the same size from the old universe connected at the overlay center. The offset value is the distance from the local origin to the overlay center. The value of offset is limited as follows:

$$0 <= offset < e$$

Figure 3 shows the configuration of a 2-dimensional overlay. The concept is easily extended to N-dimensional overlays containing 2**N obels.

Theorem. For an N-dimensional universe let there be 2**N

obels (OBEL_0,OBEL_1, ..,OBEL_n where n=2**N-1) in overlay OVERLAY all of which meet at the point CENTER (an N-dimensional vector). Place a new obel (OBEL) of the same size within the universe at location ORIGIN. Define the vector OFFSET=CENTER-ORIGIN and restrict the location of the new obel to 0<=OFFSET<=e where e is the size of an edge of the new obel. Then the new obel is completely enclosed by the overlay or, in other words: .

INTERSECTION(OBEL,UNION(OBEL_0,OBEL_1,...,OBEL_n))=OBEL

Proof. If 2**N disjoint obels meet at a point CENTER then a distance of 2*E in each dimension from CENTER-e to CENTER+e is covered. Let A be the minimum point in the overlay (lowest value in each dimension) and let B be the maximum point. All points above A in each dimension and below B in each dimension are in the overlay.

A=ORIGIN+OFFSET-e

but OFFSET<e, and therefore

A<ORIGIN

and B=ORIGIN+OFFSET+e

but OFFSET>=0, and therefore

B>=ORIGIN+e

Assume that there is a point P within OBEL that is not within OVERLAY. This means:

INTERSECTION(P,UNION(OBEL_0,OBEL_2,...,OBEL_n))=null

Thus, ORIGIN<=P<=ORIGIN+e and A<P and P<=ORIGIN+e<=b or, P<=B. But, any point A<point<=B is within OVERLAY and therefore there can exist no such point as P which is in OBEL and outside

OVERLAY.

An obvious corrollary is that if all overlay obels are terminal with the same status value, the overlaid obel also has that value. This forms the basis of the EVAL procedure described above. If all overlay obels which are known to intersect the target obel are FULL or EMPTY, the new obel is FULL or EMPTY, respectively. Otherwise, the decision is passed to the next lower level.

3.2.2 OVERLAY INITIALIZATION

At initialization, the algorithm generates an augmented universe as the first overlay. It contains the universe with the old object plus 2**N-1 additional empty universes. The placement of the object universe in the overlay is illustrated in Figure 4. In (a) the translation value is positive. The object universe is placed in the higher universe position for that dimension and the offset value is set equal to the translation value. In (b) the translation value is negative. The object universe is put in the lower slot and the offset is set to the translation vector plus the edge value for the universe.

This function is performed in the algorithm by the following procedure:

```
OVERLAY_INIT(ROOT,TRANSLATE,OVERLAY,NEW_ROOT,LOC);
    value obel ROOT;
```

```
vector TRANSLATE;
overlay OVERLAY;
obel NEW_ROOT;
integer LOC;
```

ROOT is the obel representing the object universe. TRANSLATE specifies the translation values and returns the offset values. The initial overlay is returned in OVERLAY. A new tree is allocated. The root is returned in NEW_ROOT and the first data location in LOC.

For the translation algorithm the offsets are identical for all overlays on the same level. The offset at level k+1 is the same as at k unless it equals or exceeds the edge value at that level in which case the edge value is subtracted from the offset.

3.2.3 CASES

The case of an overlay is determined by the offset value. It is used in generating the node value and the overlays at the next lower level. For the translation algorithm four cases are defined:

CASE	OFFSET VALUE
0	OFFSET=0
1	0<OFFSET<e/2
2	OFFSET=e/2
3	e/2<OFFSET<e

where e is the obel edge value at that level.

There is, of course, a separate case value for each dimension. The case values are used to select the overlay obel children which make up the overlays for the children of the target obel. This is reflected in the overlay pointer data item. As a function of the target obel child, case (through procedure SELECT) and sub-overlay obel position, it contains a pointer to a specified overlay obel (OBEL_NO field) and a specified child of it (CHILD_NO field).

In some cases, a sub-overlay obel may not intersect the target obel. This occurs for a case 0 or 2 in any dimension. If two obels intersect in all dimensions, except one, they do not intersect at all. To account for this, the TYPE field is included in overlay pointer items. If it intersects the target, TYPE='POINTER'. If it does not intersect in at least one dimension, TYPE='IGNORE'. In evaluating overlays to generate node values, IGNORE values are not taken into account.

The cases are illustrated in Figure 5. In (a) the case 0 situation (offset=0) is shown. The four children of the two overlay obels are shown as well as the two children of the target obel. The sub-overlay configuration for the two children are shown below. The associated table shows that the new overlay

obel 0 for child 0 does not intersect the target obel and is therefore marked IGNORE. New overlay obel 1 is derived from child 0 of overlay obel 1. The remaining cases are presented in (b) through (d).

Two procedures used by the translation algorithms to process case information are:

```
CASE<-CASE(LEVEL,TRANSLATE);
    integer LEVEL;
    vector TRANSLATE;
    vector CASE;
```

The case value for each dimension is calculated and returned in CASE. It is calculated from the translation vector TRANSLATE and the level in LEVEL (used to determine edge size e). TRANSLATE values >=e are reduced by e. LEVEL is incremented by 1.

```
POINTER<-SELECT(OVERLAY,CHILD,CASE);
    value integer OVERLAY,CHILD,CASE;
    overlay pointer POINTER;
```

The one-dimensional overlay pointer values are returned in POINTER as a function of overlay number in OVERLAY (0 or 1), child number in CHILD (0 or 1) and case number in CASE(0 to 3). See Figure 5.

3.2.4 FORMAL DESCRIPTION OF TRANSLATION ALGORITHM (BREADTH-FIRST)

The procedure to translate an N-dimensional object in a breadth-first traversal follows. Procedure TRANSLATE_BREADTH accepts the tree to be translated, the translation vector and the minimum level value and returns the root of the new tree. Procedure SUB_OVERLAY generates the new sub overlay for a specified child of the target obel. The old overlay and the case vector must be specified. OVERLAY SELECT is used by SUB OVERLAY to generate the overlay pointer for the individual overlay positions by repeated application of the one-dimensional select table N times. Procedure SUM is used to combine overlay pointers over dimensions.

```
obel procedure TRANSLATE_BREADTH(ROOT,TRANSLATE,MINLEV);
/* Translate object ROOT by TRANSLATE. Resolve to a minumum
   level of MINLEV. Perform in a breadth-first operation. */
begin
    value obel ROOT;
    value vector TRANSLATE;
    value integer MINLEV;
    integer CHILD,LEV,NEXT;
    vector C;
    string V;
    overlay OLD,NEW;
    obel NEW_ROOT;
    /* Initialization: set level to zero, initialize overlay
       universe and insert into queue. */
    LEV<-0;
    OVERLAY_INIT(ROOT,TRANSLATE,NEW,NEW_ROOT,NEXT);
    Q_INSERT(NEW);
    while not Q_EMPTY do
        begin
        /* Delete overlay from queue and place in OLD. Process
           the children of OLD. */
        Q_DELETE(OLD);
        if not LEV=LEVEL(OLD) then C<-CASE(LEV,TRANSLATE);
        for CHILD<-0 step 1 until 2**N-1 do
            /* Generate overlay for this child & evaluate
               to obtain value of new obel. If at minimum
               level, convert partials to full. Output the
```

```
        new value and insert partials. */
    begin
        NEW<-SUB_OVERLAY(OLD,CHILD,C);
        V<-EVAL(NEW);
        if (V='PARTIAL' and LEVEL(NEW)=MINLEV) then
            V<-'FULL';
        OUTPUT(V,NEXT);
        if V='PARTIAL' then Q_INSERT(NEW);
                    end;
                REDUCE(NEW_ROOT);
                return(NEW_ROOT);
                end;
    end;
```

```
overlay_procedure SUB_OVERLAY(OLD,CHILD,CASE);
/* Generate the overlay for child CHILD from
   overlay OLD under case CASE. */
begin
    value overlay OLD;
    value integer CHILD;
    value vector CASE;
    obel NEW_OBEL;
    overlay NEW;
    overlay_pointer P;
    integer NEW_NO;
    /* Step through the 2**N obels in overlay for the child. */
    for NEW_NO<-0 step 1 until 2**N-1 do
        /* Determine the overlay_pointer for this overlay obel
           and use to generate the obel. Put into overlay NEW. */
        begin
            P<-OVERLAY_SELECT(NEW_NO,CHILD,CASE,N);
            NEW_OBEL<-CHILD(OVERLAY_OBEL(OLD,OBEL_NO(P)),
                CHILD_NO(P));
                    OVERLAY_SET(NEW,NEW_NO,if TYPE(P)='POINTER' then
                        STATUS(CHILD_OBEL)
                            else 'IGNORE';,NODE_ADDRESS(NEW_OBEL));
            end;
        LEVEL(NEW)<-LEVEL(OLD)-1;
        return(NEW);
end;
```

```
overlay_pointer procedure OVERLAY_SELECT(OVERLAY_NO,CHILD,
                                         CASE,K);
/* Recursively generate overlay_pointer for overlay OVERLAY_NO
   for child CHILD under case CASE for dimensions K and below. */
begin
    value integer OVERLAY_NO,CHILD,K;
    value vector CASE;
    integer OVERLAY_K,CHILD_K,CASE_K;
    overlay_pointer POINT;
    /* Pick out overlay, child and case for dimension K. */
    OVERLAY_K<-BIT_VALUE(OVERLAY_NO,K);
    CHILD_K<-BIT_VALUE(CHILD,K);
    CASE_K<-VECTOR_VALUE(CASE,K);
    /* Consult the one-dimensional selection table for overlay,
       child and case in this dimension. */
    POINT<-SELECT(OVERLAY_K,CHILD_K,CASE_K);
    /* If not at first dimension, add to lower dimension
       values. */
    if not K=1 then POINT<-SUM(POINT,OVERLAY_SELECT(OVERLAY_NO,
                                    CHILD,CASE,K-1),K);
    return(POINT);
end;
```

```
overlay_pointer procedure SUM(P,Q,K);
/* Compute overlay_pointer which combines overlay_pointer P at
   level K with overlay_pointer Q containing all lower
   dimensions. */
begin
    value overlay_pointer P,Q;
    value integer K;
    overlay_pointer R;
    /* Place obel and child numbers into bit position K. */
    OBEL_NO(R)<-2**K*OBEL_NO(P)+OBEL_NO(Q);
    CHILD_NO(R)<-2**K*CHILD_NO(P)+CHILD_NO(Q);
    if (TYPE(P)='IGNORE' or TYPE(Q)='IGNORE') then
        TYPE(R)<-'IGNORE'
        else TYPE(R)<-'POINTER';
    return(R);
end;
```

3.2.5 FORMAL DESCRIPTION OF TRANSLATION ALGORITHM (DEPTH-FIRST)

A version of the above algorithm modified slightly for depth-first traversal follows. TRANSLATE_DEPTH is externally equivalent to TRANSLATE_BREADTH. Internally, it uses procedure OBEL_MOVE to recursively translate the descendants of the root obel.

All other procedures are identical except CASE which is changed to CASE2. It is no longer called only once for each level. Rather, it is called each cycle as the level changes. It therefore no longer returns modified level and translation values. Internally, it must calculate the offet values from the original translation vector. It would typically be implemented via look-up table initialized by OVERLAY_INIT.

The following new procedure is used:

NODE_REDUCE(LOC,V);

value integer LOC;

string V;

The node values in the output depth-first tree generated by this instance of OBEL_MOVE are examined. If they are all terminal of the same type, they are removed and the parent is changed from PARTIAL to FULL or EMPTY. This value is returned as V. If they are not all terminal of one type, 'PARTIAL' is returned in V.

Procedure NODE_REDUCE performs the elimination of redundant nodes during algorithm operation rather than during a later pass. A stack would typically be used to keep track of the 2**N

children of each node. It would contain the value and the tree address. Nodes in a sequentially allocated tree could be eliminated by marking them as nonexistant. They could be compressed out later or simply ignored.

```
obel procedure TRANSLATE_DEPTH(ROOT,TRANSLATE,MINLEV);
/* Translate object ROOT by TRANSLATE. Resolve to a minimum
   level of MINLEV. Perform in a depth first operation. */
begin
    value obel ROOT;
    value vector TRANSLATE;
    value integer MINLEV;
    overlay NEW;
    obel NEW_ROOT;
    integer NEXT;
    /* Initialize overlay universe. Move recursively. */
    OVERLAY_INIT(ROOT,TRANSLATE,NEW,NEW_ROOT,NEXT);
    OBEL_MOVE(NEW,TRANSLATE,MINLEV);
    REDUCE(NEW_ROOT);
    return(NEW_ROOT);
end;
```

```
procedure OBEL_MOVE(OLD,TRANSLATE,MINLEV);
/* Recursively move the children of OLD by TRANSLATE to
   a minimum level of MINLEV. */
begin
    value overlay OLD;
    value vector TRANSLATE;
    value integer MINLEV;
    integer CHILD;
    vector C;
    string V;
    overlay NEW;
    /* Generate case for this level and move children of OLD. */
    C<-CASE2(LEVEL(OLD),TRANSLATE);
    for CHILD<-0 step 1 until 2**N-1 do
        /* Generate overlay for this child & evaluate
           to obtain value of new obel. If at minimum
           level, convert partials to full. Output the
           new value and move at lower level if partial. */
        begin
            NEW<-SUB_OVERLAY(OLD,CHILD,C);
            V<-EVAL(NEW);
            IF(V='PARTIAL' and LEVEL(NEW)=MINLEV) then V<-'FULL';
            OUTPUT(V);
            IF(V='PARTIAL') then OBEL_MOVE(NEW,TRANSLATE,MINLEV);
        end;
end;
```

3.2.6 ANALYSIS

The basic tree translation process is one of traversing the implied tree of the new object and, for each obel, interrogating the original tree for a reading of the status of the space it represents. If the scheme for reading the tree is able to provide a final value, a terminal node is generated. If not, a branch node is generated, indicating that its status remains unresolved and the obel is subdivided. The status of each child is then requested.

There are two costs which can be traded off. One is the cost of subdividing an unresolved obel. The second is the cost of generating a value for an obel from the old tree. In general, if a higher cost is allowed for one, a lower cost can be expected for the other. A more costly scheme for generating a value, for example, would presumably generate final values at higher levels of subdivision, allowing savings in subdivision cost.

Under this analysis, an optimum system would be one in which the sum of the two costs was, on the average, minimal.

The cost of subdividing stops when the offset value becomes 0. In this case the subtree of each node in the new tree is identical to the corresponding subtree in the old tree. It can simply be copied or, if linked allocation is in use, nodes could point back into the model (assumed to be permanent).

Unfortunately, this does not occur as often as might be desirable. Assume that moves are allowed to the lowest level of resolution in a tree with n levels. For a uniform distribution

of move values, the probability that the offset value will go to 0 at level k (1<=k<=n) is 2**(k-n). This shows that, on the average, half of the cases will never go to an offset of zero where it will save subdivision, 75% will not be resolved before the next to last level, 87.5% not before the second to last, and so on. It indicates that most new obels must be divided to the lowest levels before they are covered by a terminal obel in the old tree.

This, coupled with the probability that most objects will have vast areas represented by terminal nodes at the higher levels has led to the overlay scheme for determining obel status. It allows all obels at a particular level which cover a new obel to be examined at once. If they are all of the same terminal type, subdivision stops. The lower the level of the overlay relative to the target obel, the more accurate the result, allowing termination at a higher level. There is, of course more cost in performing the evaluation.

With the overlay technique, one node in the unreduced output tree is generated for each overlay evaluation (8 for each subdivision). Thus, the number of calculations is approximately proportional to the complexity of the resultant object (number of nodes in octree) before reduction.

3.3 SCALING

The scaling algorithm allows the scaling of an object independently in each dimension. The input is the root of an object, a scaling vector and a translation vector specifying the origin of the new universe in (or outside of) the old universe.

3.3.1 SCALING BY A POWER OF 2

Scaling an object by a power of two in all dimensions is accomplished by adding or deleting levels at the root. An object is halved in each dimension by adding one level at the top. The new root points to one branch node, the old root, and 2**N-1 empty terminal nodes. The scaled down universe can be located in any of the level-one obels in the new universe.

In a like manner, selecting one of the level-one nodes to be a new root doubles the size (in each dimension) of anything within it. To double the size of an arbitrary section of space, it could be translated to the origin and obel number 0 expanded to fill the new universe.

Objects can be expanded or reduced by any power of two by, in effect, repeated expansion or reduction by a factor of 2. These scaling operations can be accomplished by manipulating a very small number of bits at the top of the tree. The vast bulk of the universe is left unchanged.

3.3.2 SCALING BY AN ARBITRARY FACTOR

Scaling by a factor other than a power of 2 is considerably more difficult. The algorithm presented below provides for any scale factor between 0.5 and 1.0 independently in each dimension. Scaling by any factor in any dimension can be performed by repeated application of power of two scaling and this algorithm. Modification of the given algorithm to provide arbitrary scaling in one pass is straightforward.

The scaling algorithm is very similiar to the translation algorithm. The target obel, however, may be smaller than the overlay obels in one or more dimensions. In addition, one set of offset values cannot be used throughout the algorithm. The offset vector must be computed independently for each child of the target obels.

The basic geometry of a one-dimensional overlay and target obel is shown in Figure 6. The edge of an overlay obel is e and the edge of the target obel is h. It should be noted that the value of h can be different in each dimension. The offset is again the distance from the origin of the target obel to the center of the overlay. The value of h is limited to 0.5<=h/e<=1.0. The associated cases are shown in Table 1.

Most of the support procedures are used without change from TRANSLATE with the following exception:

```
CASE<-CASE3(LEVEL,CHILD,OFFSET_OLD,OFFSET_NEW);
    value integer LEVEL,CHILD;
    value vector OFFSET_OLD;
```

vector OFFSET_NEW;

vector CASE;

The case value for each dimension is calculated for child CHILD and returned in CASE. It is calculated from the offset vector OFFSET_OLD and the tree level in LEVEL. The new offset vector for this child is calculated and returned in OFFSET_NEW. In each dimension it is the old offset value plus, if the child value is one in this dimension, e/2+h/2. It is returned MODULO(e/2).

Also, the one dimensional selection table used by SELECT must be expanded to accomodate the larger number of cases.

3.3.3 FORMAL DESCRIPTION OF SCALING ALGORITHM

Procedure SCALE_BREADTH accepts the root of the tree to be scaled, the locaton of the new universe, the scale factor (0.5 to 1.0) in each dimension and the minimum level. The root of the scaled tree is returned. Procedure OVERLAY_INIT sets up the augmented universe which forms the first overlay. This and the offset vector are inserted into the queue.

For each overlay removed from the queue, the children of the target obel are processed. CASE3 returns the case vector for each child. The new sub-overlay is generated by SUB_OVERLAY and evaluated by EVAL. Partials are filled at the minimum level and the value is placed into the new tree structure. The sub-overlay and offset vector for partials are inserted in the queue. REDUCE compresses the tree and the root is returned.

```
obel procedure SCALE_BREADTH(ROOT,TRANSLATE,SCALE,MINLEV);
/* Scale object ROOT by SCALE. TRANSLATE, a point in ROOT will be
   the origin of the new object. Resolve to a minimum level of
   MINLEV. Perform in a breadth-first operation. */
begin
    value obel ROOT;
    value vector TRANSLATE,SCALE;
    value integer MINLEV;
    integer CHILD,NEXT;
    vector C,OFFSET_OLD,OFFSET_NEW;
    string V;
    overlay NEW,OLD;
    obel NEW_ROOT;
    /* Initialization: initialize augmented overlay universe and
       insert overlay and initial offset vector into queue. */
    OVERLAY_INIT(ROOT,TRANSLATE,NEW,NEW_ROOT,NEXT);
    Q_INSERT(NEW);
    Q_INSERT(TRANSLATE);
    while not Q_EMPTY do
    begin
    /* Remove overlay and offset vector for target obel
       and process children. */
        Q_DELETE(OLD);
        Q_DELETE(OFFSET_OLD);
        for CHILD<-0 step 1 until 2**N-1 do
            /* Generate case vector for this child of target
               obel. Generate sub-overlay and evaluate. If
```

```
        at minimum level, convert partials to full.
        Output the new value and if partial, insert
        new overlay and new offset vector. */
    begin
        C<-CASE3(LEVEL(OLD),CHILD,OFFSET_OLD,
            OFFSET_NEW);
        NEW<-SUB_OVERLAY(OLD,CHILD,C);
        V<-EVAL(NEW);
                    IF (V='PARTIAL' and LEVEL(NEW)=MINLEV) then
                        V<-'FULL';
                    OUTPUT(V,NEXT);
                    if V='PARTIAL' then
                        begin
                            Q_INSERT(NEW);
                            Q_INSERT(OFFSET_NEW);
                        end;
                end;
            end;
    REDUCE(NEW_ROOT);
    return(NEW_ROOT);
    end;
```

3.3.4 SHEAR TRANSFORMATION

One-dimensional shear transformations can be performed by the overlay structure shown in Figure 7. The new universe is a parallelogram covered by an overlay. The number of overlay obels required is a function of the maximum shear angle allowed. In general, the total rise divided by an edge of the universe, rounded up gives the number of extra overlay obels beyond 2 that should be used. The target obel is subdivided into children as shown in Figure 8.

The actual shear algorithm is a straightforward extension of previous algorithms and will not be presented here for brevity. Multidimensional shear can be handled by a more complex overlay structure and techniques similar to the rotation algorithm below.

3.3.5 NONLINEAR SCALING

An overlay structure as shown in Figure 9 will result in a nonlinear scaling transformation. The size of the new universe in the y direction relative to the old universe is a function of x. In Figure 10 the y value of a point in the new universe, y_new will be:

$$y\_new = y / y\_univ = y / (mx + b)$$

The scale factor in the y direction is thus proportional to 1/x. Such an inverse scaling could be utilized for true perspective viewing transformations.

Figure 11 illustrates the method of target child generation.

As in shear, the number of overlay obels required is a function of the slope allowed.

The lower edge of the new universe can be set at a slope different from the upper edge. This can be used to perform a shear transformation as well as scaling.

3.4 ROTATION

Rotation of an N-dimensional object involves N*(N-1)/2 rotation angles, one for each pair of dimensions and an N-dimensional point of rotation. Points of the rotated object are related to points in the original object by:

Xk_new=(Xk-Xk_0)*COS(ANG_k_m)-(Xm-Xm_0)*SIN(ANG_k_m)+Xk_0

Xm_new=(Xk-Xk_0)*SIN(ANG_k_m)+(Xm-Xm_0)*COS(ANG_k_m)+Xm_0

where k,m=1,2,...N, k not m, k<m, Xi is the coordinate of a point in dimension i, Xi_new is the coordinate of a point in dimension i in the new (rotated) universe, Xi_0 is the coordinate of the center of rotation in dimension i and ANG_i_j is angle (0 to 2*pi) of object rotation (positive i direction to positive j direction).

The function of a rotation algorithm is to generate a new tree representing a rotated object from a tree for the original object, a set of rotation angles and a point of rotation.

3.4.1 ROTATION BY 90 DEGREES

If the center of rotation is the center of the universe, rotation by 90 deg., 180 deg., or 270 deg. or reflection about either axis or across a line at 45 deg. or 135 deg. requires no calculation. It can be accomplished by a simple reordering of the subtrees. For linked allocation, the subtree ordering could be an attribute of a tree or even redefined for selected subtrees. Thus, such a transform could be performed on an entire object or part of an object by changing a few bits.

For rotation about an arbitrary point, the object could be translated by the vector from the point to the universe center, transformed and then translated back.

The reordering for a 2-D universe is as follows:

NEW SUBTREE ORDERING	TRANSFORM
2,0,3,1	90 deg. rotation
3,2,1,0	180 deg. rotation
1,3,0,2	270 deg. rotation
2,3,0,1	Reflection across axis 1
1,0,3,2	Reflection across axis 2
0,2,1,3	Reflection across 45 deg. line
3,1,2,0	Reflection across 135 deg. line

The reordering for N-Dimensional space is a simple extension of the 2-D case.

3.4.2 ROTATION BY 0 TO 90 DEGREES

Rotation by an arbitrary angle requires additional calculations. The algorithm below rotates a 2-D object by any angle between 0 deg. and 90 deg. An algorithm to rotate an N-Dimensional object by an angle in some specified plane is a straightforward extension but is not presented here for brevity. An algorithm to rotate an object by all possible angles requires a duplication and extension of the basic algorithm.

The overlay scheme is extended for use in rotation. The dimensions cannot be analyzed independently, however because each angle influences two dimensions. Thus, a 2-D overlay structure is required for each angle.

Theorem. Set up a 3 by 3 overlay of 2-D obels labeled A through I as shown in Figure 12. Locate a target obel of the same size as the overlay obels within the overlay at an angle of TH relative to axis 1 where 0<=TH<=pi/2. Restrict the target obel location such that the origin is within overlay obel B. See Figure 13. The target obel is covered by the overlay.

Proof. With no loss of generality, set the edge size of the target obel to 1. Starting with an angle of 0 deg., rotate the obel about its local origin through 90 deg. Let dimension 1 be X and dimension 2 be Y. Relative to the local origin, the X value of all points will be X*COS(TH)+(-Y*SIN(TH)). Within the angle limits, both terms are at a maximum at an angle of 0 deg. Thus, the X value of no point can be greater than it is at 0 deg. In a similar argument, the X value of a point can never be less than

it is at pi/2. This sets a limit of -1<=X<=1 for all points in a rotated square.

This can be applied to y to show that no Y value will be below the origin. The point most distant from the origin is at the opposite corner and is at a distance of SQRT(2). Thus, under rotation, the y value cannot exceed SQRT(2) relative to the local origin. The value of Y is thus limited to 0<=Y<=SQRT(2) or, within this limit, 0<=y<2.

Place the target obel in the overlay with its origin in overlay B. Arbitrarily set the origin of B to be the overlay origin. The overlay thus covers -1<=X<=2 and 0<=Y<=3. The target obel's local origin is thus within 0<=X<=1, 0<=Y<=1. All points within the target obel are thus restricted to -1<=X<=2 and 0<=Y<3 or, within the overlay.

The algorithm proceeds in a manner similar to translation in that the overlay obels which intersect the target obel are retained while disjoint obels are marked and ignored. The determination of case is more complicated, however. For all overlay obels except B, the determination of an intersection or ignore case is a function of rotation angle and the location of the origin of the target obel within B. It is essentially a linear programming problem of deciding which side of a line (edge of target obel) a point (corner of overlay obel) lies in. It is greatly simplified because all target obel edges are parallel to the corresponding edges in the new universe and the points are

regular and well defined. The decisions are based on the location of the control points shown in Figure 14.

The points V1, V2 V3 and V4 are the vertices of the target obel. The point P1 is the point of intersection of the line defined by points V1 and V2 and the Y axis. The origin of overlay obel B is the overlay origin. The points P2 through P4 are defined in like manner. Note that the intersection does not necessarily fall within the segment of the line between vertices.

The determination of the status of the three left overlay obels (A, D and G) proceeds as follows (assume edge size of 1):

```
if X(V2)>=0 then mark A=D=G='IGNORE'
else, if Y(P1)>1 then A='IGNORE'
      if Y(P2)<2 then G='IGNORE'
      if Y(P2)<1 then D='IGNORE'
      if Y(P1)<2 then D='IGNORE'
```

The status of the remaining overlay obels are calculated from the remaining control points in like manner.

There are numerous sets of control points and methods of calculating them. The control points used in the algorithm below are calculated from two sets of numbers, universal parameters which are calculated once at the beginning of the algorithm and local parameters which are passed from parent to child via queue or stack.

The universal parameters are shown in Figure 15. They are calculated at level 0 as follows:

```
A=E/TAN(TH)
B=E/SIN(TH)
```

C=E/TAN(pi/2-TH)

D=E/SIN(pi/2-TH)

P1=E*COS(TH)

P2=E*SIN(TH)

where E is the size of an edge of the universe and TH is the rotation angle.

The corresponding values at lower levels are generated by shifting right (division by 2) a number of places equal to the level number. These values remain fixed during algorithm operation.

The local rotation parameters are shown in Figure 16. The origin of the target obel is (X,Y) relative to the origin of overlay obel B. They are calculated at level 0 as follows:

(X,Y)=origin of new universe

R=Y-(E-X)/TAN(TH)

S=Y-X*TAN(TH)

The control points can be calculated as follows:

V1=(X,Y)

V2=(X+P2,Y+P1)

V3=(X-P2+P1,Y+P1+P2)

V4=(X+P1,Y+P2)

P1=(0,A+R)

P2=(0,D+S)

P3=(0,A+B+R)

P4=(E,C+D+S)

P5=(E,B+R)

P6=(E,C+S)

New data formats are defined to handle rotation parameters:

rot_univ_parms - a record containing 6 integers as follows:

1) A
2) B
3) C
4) D
5) P1
6) P2

rot_loc_parms - a four integer record containing:

1) X value of V1
2) Y value of V1
3) R
4) S

Procedure OVERLAY_INIT is embellished, becoming OVERLAY_INIT2 which is defined as follows:

```
OVERLAY_INIT2(ROOT,ROT_POINT,ANG,OVERLAY,NEW_ROOT,LOC,
                        U_PARMS,L_PARMS);
value obel ROOT;
vector ROT_POINT;
value integer ANG;
overlay OVERLAY;
obel NEW_ROOT;
integer LOC;
rot_univ_parms U_PARMS;
rot_loc_parms L_PARMS;
```

The function of OVERLAY_INIT2 is similar to that of

OVERLAY_INIT. The selection of the overlay obel in which to put the old universe is slightly more complex as shown in Figure 17. The origin of the new universe must be located in overlay obel B. The new origin is calculated by rotating the old origin about the point of rotation (ROT_POINT). The old universe is located in one of the overlay obels and the origin is reset as follows:

X	Y	Overlay_Obel	New_X	New_Y
<E	<0	E	X	Y+E
>=E	<0	D	X-E	Y+E
<E	>=0	B	X	Y
>=E	>=0	A	X-E	Y

The R and S values are calculated as above and placed in L_PARMS. The level 0 universal parameters are calculated and placed in U_PARMS. The 9 obel overlay is returned in OVERLAY. Other arguments are as previously defined.

Procedure SUB_OVERLAY is given the responsibility for case evaluation and redefined as follows:

```
NEW<-SUB_OVERLAY2(OLD,CHILD,U_PARMS,L_PARMS,L_PARMS_NEW);
    value overlay OLD;
    value integer CHILD;
    value rot_univ_parms U_PARMS;
    value rot_loc_parms L_PARMS;
    rot_loc_parms L_PARMS_NEW;
```

First the new origin of the target obel child is calculated based on the old origin of the parent from

L_PARMS and the child number in CHILD. This is placed in L_PARMS_NEW, the new set of local parameters for the child. The location of the new origin is used to set up the new sub-overlay in overlay NEW so that the new origin is in new sub-overlay obel B. This involves a simple selection table. The universal parameters in U_PARMS and the local parameters in L_PARMS are used to calculate the new R and S values for L_PARMS_NEW and then to calculate the control points. The control points are examined and the appropriate new overlay obels are marked as IGNORE cases.

A new procedure is defined as follows:

UNIV_NEW<-ROT_DIV(LEVEL,UNIV_OLD);

value rot_univ_parms UNIV_OLD;

rot_univ_parms UNIV_NEW;

integer LEVEL;

Each of the universal parameters in UNIV_OLD is divided by 2 and returned in UNIV_NEW. LEVEL is decremented.

3.4.3 FORMAL DESCRIPTION OF ROTATION ALGORITHM

The procedure to rotate a 2-D object follows. The augmented universe is initialized by OVERLAY_INIT2. The universal and local parameters for the target universe are computed and a new tree is established. The overlay and local parameters are inserted in the queue.

An overlay and local parameter set is next deleted from the queue and the four sons are processed. If the new overlay is at a lower level, the universal parameters are divided by 2 in procedure ROT_DIV. The new sub_overlay is calculated by SUB_OVERLAY2 and evaluated by EVAL. As in previous algorithms, obels with a value of PARTIAL are converted to FULL at MINLEV. The value is output to the new tree and, if a PARTIAL, the new overlay and local parameters are inserted in the queue. REDUCE removes any redundant nodes and the root of the new tree is returned as the function value of the procedure.

```
obel procedure ROTATE_2D_BREADTH(ROOT,ROT_POINT,ANG,MINLEV);
/* Rotate 2D object ROOT by angle ANG about the center of
   rotation ROT_POINT. Resolve to minimum  level of MINLEV.
   Perform in breadth-first operation. */
begin
    value obel ROOT;
    value vector ROT_POINT;
    value integer ANG,MINLEV;
    overlay NEW,OLD;
    integer LEV,CHILD,NEXT;
    rot_univ_parms U_PARMS;
    rot_loc_parms NEW_LOC,OLD_LOC;
    obel NEW_ROOT;
    string V;
    /* Initialize level and first overlay. Set up universal and
       local parameters at level 0 and new tree. Insert overlay
       and local parameters in queue. */
    LEV<-0;
    OVERLAY_INIT2(ROOT,ROT_POINT,ANG,NEW,NEW_ROOT,NEXT,U_PARMS,
                  NEW_LOC);
    Q_INSERT(NEW);
    Q_INSERT(NEW_LOC);
    while not Q_EMPTY do
        begin
        /* Delete overlay and local set of parameters from queue.
           Reduce universal parameters if change level. Process
           four children. */
```

```
Q_DELETE(OLD);
Q_DELETE(OLD_LOC);
if not LEV=LEVEL(OLD) then
   U_PARMS<-ROT_DIV(LEV,U_PARMS);
for CHILD<-0 step 1 until 4 do
    begin
    /* Generate new sub-overlay for this child and
       evaluate. If at MINLEV, convert PARTIALS to
      FULL and output. Reinsert if PARTIAL. */
NEW<-SUB_OVERLAY2(OLD,CHILD,U_PARMS,OLD_LOC,
    NEW_LOC);
V<-EVAL(NEW);
if(V='PARTIAL' and LEVEL(NEW)=MINLEV) then
    V<-'FULL';
OUTPUT(V,NEXT);
if V='PARTIAL' then
    begin
        Q_INSERT(NEW);
        Q_INSERT(NEW_LOC);
                end;
            end;
        end;
REDUCE(NEW_ROOT);
return(NEW_ROOT);
end;
```

This algorithm is easily extended into the rotation of a 3-D object about a single axis. Instead of 4 child obels being generated, two sets of 4 are processed corresponding to the two segments along the axis of rotation. All possible 3-D rotations can be performed by repetitive 90 deg. and 0 deg. to 90 deg. rotations in 3-D.

In an algorithm allowing a single pass rotation of a 3-D object along all 3 axes of rotation a 3 by 3 by 3 overlay is not sufficient. Rotation on one projection plane will change the projection in another plane into a shape more complex than a square. The distance between the origin and a point could now be SQRT(3) for a unit square rather than SQRT(2).

4 DISPLAY ALGORITHMS

The task of a display algorithm is to convert a number of coincidental 3-D universes containing 3-D objects into a 2-D image corresponding to a view of the object from an arbitrary point in space. The surface of an object hidden behind the object or other objects should not be visible.

The view can be either an orthographic projection or a perspective view. In general, a perspective view is more costly in computation.

The algorithm below allows anti-aliasing to be performed at additional computation cost. This generates gray scale pixels along the edge of an object, the intensity of which is proportional to the degree of overlap of the object and the pixel. This tends to eliminate the jagged edge of objects common on raster displays when pixels can be only the color of an object or the color of the background.

Four algorithms are possible, corresponding to the four combinations of orthographic projection or perspective and anti-aliasing or no anti-aliasing.

4.1 HIDDEN SURFACE REMOVAL

Because of the nature of the octree data structure used to store objects, they are spatially pre-sorted in 3 dimensions. Proper traversal of the tree will generate obels in a spatially sorted order. Thus, if the obels so generated are painted on the

screen of a raster display in which new obels cover old obels, the result after traversal is an image with all hidden surfaces removed. No additional sorting or processing is required.

The analysis of the hidden surface properties of the octree scheme can be analyzed by first segmenting all possible directions into sections of what will be called angle space. There are 8 regions of 3-D angle space corresponding to the 8 combinations of the signs of the three displacement values along the 3 axes for a vector. A displacement of 0 is assumed to be of either sign. Thus, a vector with a displacement of 0 in one or more dimensions is in more than one region.

A blocking obel is defined to be an obel which can obscure the view of another obel. Obel A is defined to be a blocking obel within a region of angle space to obel B if there exists a vector in that region of angle space whose origin is in obel B and which intersects A.

Theorem. Given a universe of obels at a common level, for obel A to be a blocking obel to obel B, within a region of angle space, the vector from the origin of B to the origin of A must be within that region of angle space.

Proof. All obels are at the same level and are, therefore, of the same size. The origins lie on intersections of a regular grid the spacing being the size of an edge of an obel at this level. Because of the regular spacing, if, in some dimension, the value of the origin of an obel is greater than or less than the origin of another obel, all points in the obel are greater than or less than, respectively, in that dimension.

Assume that A blocks B or that there exists a vector from B to A in region Q of angle space but that the vector from the origin of B to A is not in region Q. In at least one dimension the sign of the displacements of the two vectors are different. But, because of regular spacing, this is not possible, and, therefore, such a situation cannot exist.

Theorem. Given an octree obel, there is a sequence of traversal of the children such that for each region of angle space a child later in the sequence cannot block a child earlier in the sequence.

Proof. It is sufficient to show that for each transisition the vectors from the origin of child n to the origins of n+1,n+2,... are not within the angle region. Each vector must have a sign difference relative to the region in at least one dimension.

It is sufficient to simply enumerate the sequences. The regions and a sequence (they are not unique) are as follows:

Region	Z	Y	X	Sequence
0	+	+	+	7,6,5,4,3,2,1,0
1	+	+	-	6,7,4,5,2,3,0,1
2	+	-	+	5,4,7,6,1,0,3,2
3	+	-	-	4,5,6,7,0,1,2,3
4	-	+	+	3,2,1,0,7,6,5,4
5	-	+	-	2,3,0,1,6,7,4,5
6	-	-	+	1,0,3,2,5,4,7,6
7	-	-	-	0,1,2,3,4,5,6,7

The origin vectors for region 0 are as shown in Table 2. A negative sign for one dimension in each case is enclosed in parentheses indicating no combinations form a blocking situation.

The application to display becomes clear when children are displayed on a raster display in back to front order (relative to the observer). Objects later in the sequence cannot be obscured by earlier objects. In each dimension the section of the obel at a greater distance from an observer within the corresponding region is displayed before any part closer is displayed in that dimension.

Extension beyond a single obel to all obels in a unierse applies this sequence recursively. Thus, a depth-first traversal of the tree with a child sequence corresponding to the viewer's region generates a view with all hidden surfaces removed.

4.2 ORTHOGRAPHIC VIEW

The first step in generating an orthographic view is to set up a view coordinate system as shown in Figure 18. The viewer is assumed to be on the dimension 1 axis at an infinite distance from the origin in a positive direction. The origin of the universe in view coordinates is calculated. The three vectors between the origin and vertices 1,2 and 4 are calculated. The corners of the universe can be easily computed from the origin and the 3 vectors. At lower levels, the vertex offset vectors are divided by 2 for each level.

The image will be the projection of the object on a plane defined by a constant value in dimension 1. The projection of all objects at one level will be identical except for translation on the screen. It will be composed of three parallelograms.

The general strategy is to traverse the trees representing the objects in the appropriate order for the location of the viewer in angle space. All FULL obels are displayed on a raster display in a uniform direction in each dimension.

4.2.1 DEPTH-FIRST DISPLAY

When a linked allocation scheme is used, a simple depth first traversal can be performed. The order of child selection corresponds to the appropriate sequence for the viewers region of angle space. All trees are traversed simultaneously. Terminal nodes with a value of FULL are displayed immediately. When an

EMPTY terminal node is encountered in one tree, the others are simply traversed below this.

For the case of two or more FULL terminal nodes representing the same obel, an interference has been detected. It could cause special action or be ignored. For the case of a FULL node in one tree and one or more PARTIAL nodes in other trees, two possibilities exist. If the PARTIAL node is from a fully reduced tree (no redundant nodes) an interference has been detected but the magnitude of the overlap has not yet been determined. This requires traversal to lower levels.

For the case in which unreduced trees are in use, such as would be generated in a level by level breadth-first pipelined scheme, the interference is not certain. Lower levels must be examined.

The algorithm to perform a depth-first display is a simple depth-first traversal using a recursive procedure and will not be presented here for brevity.

4.2.2 BREADTH-FIRST DISPLAY

In a breadth-first traversal, obels are visited in a spatial ordering on each level, but terminal nodes are not necessarily encountered in an ordered sequence. A queue will be used to maintain the proper ordering.

In a sequential allocation scheme the node storage order determines the order of traversal. This limits the view to one region, typically region 0. For other views, the tree must be

resequenced or addresses from the level being processed to the next lower level must be provided.

The general idea in breadth-first display is to generate the image of FULL terminal obels on the screen as they are encountered if no PARTIAL obels earlier in the sequence are in the queue. If it cannot be displayed, it is inserted or reinserted in the queue. A flag is used to indicate when a partial has been inserted after a level change.

Several new data formats are defined:

disp_token - a 2 field record representing an obel and the status of the coincidental obels from the object to be displayed.

1) field: VIEW_ORIGIN
   format: vector
   value: origin of obel in view coordinates
2) field: VIEW_OBELS
   format: overlay
   value: the m coincidental obels of the m objects

disp_vector - record containing the 3 universal vertex offset vectors at a level.

1) field: VIEW_VECTOR_1
   format: vector
   value: vector in viewspace coordinates from origin to vertex point 1
2) field: VIEW_VECTOR_2
   format: vector
   value: vector in viewspace coordinates from origin to

vertex point 2

3) field: VIEW_VECTOR_3

format: vector

value: vector in viewspace coordinates from origin to vertex point 4

disp_vector_set - set of universal vertex offset vectors for all levels

1) field: DISP_VECTOR_LEV_0

format: disp_vector

value: the set of vertex offset vectors at level 0

2) field: DISP_VECTOR_LEV_1

format: disp_vector

value: the set of vertex offset vectors at level 1

.

.

.

n) field: DISP_VECTOR_LEV_n

format: disp_vector

value: the set of vertex vector offset vectors at level n

obel_set - a record containing a number of obels

1) field: OBEL_NO_0

format: obel

value: obel number 0

.

.

.

n) field: OBEL_NO_n

format: obel

value: obel number n

The following new procedures are defined:

NEW<-VIEW_INIT(ROOTS,NUM,ANG,ORIGIN,PROJ);

value obel_set ROOTS;

value integer NUM;

value vector ANG,ORIGIN;

disp_token NEW;

disp_vector_set PROJ;

Generates display token NEW for the universe. The VIEW_OBELS field of NEW is populated by the NUM objects from ROOTS. The set of vertex offset vectors PROJ is generated. ORIGIN is the origin of the view coordinate system and ANG specifies the view angles.

NEW<-SUB_DISPLAY(OLD,CHILD,PROJ);

value disp_token OLD;

value integer CHILD;

disp_token NEW;

disp_vector_set PROJ;

Returns display token NEW as the child CHILD of OLD. The level is decremented. The display vertex vector set PROJ is used to calculate the new obel origin in view coordinates. All obels with a terminal value of 'EMPTY' are marked 'IGNORE'.

DISPLAY(DISPLAY_TOKEN,PROJ);

value disp_token DISPLAY_TOKEN;

value disp_vector_set PROJ;

The three sides of the common obel are painted on the raster display. They form 3 parallelograms. The vertices are computed from the obel origin in DISPLAY_TOKEN and the vertex offset vectors in PROJ. The color of the obel can be controlled by the location of the full obel in the VIEW_OBELS field in DISPLAY_TOKEN or by external means. The relative illumination of the 3 sides can be controlled universally by the location of the uniform illumination source. A simple external table loaded at initialization time would be sufficient or it could be manipulated after display through a display color translation table.

If more than one full obel is found, several options are possible. If detection of interference is important, special action should be taken. A flashing display overlay plane could be used to indicate the situation, for example. If this is not necessary, the color of the first (or last) FULL could be used.

4.2.3 FORMAL DESCRIPTION OF DISPLAY ALGORITHM

The breadth-first display algorithm begins by initializing the current node level to 0 and the queue flag to FALSE indicating that no PARTIAL obels have been inserted in the queue at this level. The display token for the universe is generated and the vertex offset vectors are calculated. The display token is inserted in the queue.

The main loop begins by deleting display token OLD from the queue. If a level change is encountered, LEV is decremented and the flag is reset to FALSE.

The status of the object obels is then examined. Note that the IGNORE case is now used to eliminate EMPTY obels from consideration. This saves a new EVAL procedure. At MINLEV, all partials are converted to FULL. FULLs are then displayed if the flag is FALSE (no partials inserted) or reinserted if flag is TRUE. EMPTY obels are ignored. PARTIALs are subdivided into the 8 children.

Procedure SUB_DISPLAY generates the new display token for the child. It is then evaluated in a manner similar to above. Children with a value of PARTIAL are inserted and cause the flag to be set to TRUE, inhibiting further display at this level.

It should be noted that the operations calculate dimension 1 values in the view coordinate system. This is not really necessary for the orthographic view but is defined here to eliminate redefinition for the perspective display algorithm.

In an advanced system of display it might be desirable to have an image which would present increasing resolution with time. One way to do this would be to maintain 2 frame buffers. One would always be above MINLEV. That is, partials are never displayed. When the level was changed, this would be dumped into the second buffer. While the algorithm proceeds at the new level, PARTIAL obels as well as FULL obels would be displayed. The user would view the second frame buffer with perhaps a third buffer to hold the previous image until the level was completed.

```
procedure DISPLAY_BREADTH(ROOTS,NO_ROOTS,ANG,ORIGIN,MINLEV);
/* Display the NO_ROOTS objects  ROOTS from a viewpoint defined
   by defined by ANG and ORIGIN. Perform in a breadth first
   manner to a maximum depth of MINLEV. */
value obel_set ROOTS;
value integer NO_ROOTS;
value vector ANG,ORIGIN;
value integer MINLEV;
disp_vector_set PROJ;
disp_token NEW,OLD;
string V;
integer LEV,CHILD;
logical FLAG;
begin
    LEV<-0;
    FLAG<-.FALSE.;
    NEW<-VIEW_INIT(ROOTS,NO_ROOTS,ANG,ORIGIN,PROJ);
    Q_INSERT(NEW);
    while not Q_EMPTY do
begin
/* Delete display token from queue into OLD. If change
   level, reset flag and increment LEV. */
     Q_DELETE(OLD);
     if LEVEL(VIEW_OBELS(OLD))<-LEV then
         begin
             LEV<-LEV-1;
             FLAG<-.FALSE.;
```

```
    end;
/* Evaluate and set PARTIAL to FULL if at MINLEV.
   If FULL and haven't, display it.  If EMPTY
   ignore it. If PARTIAL, process 8 children. */
V<-EVAL(VIEW_OBELS(OLD));
IF(V='PARTIAL' and LEV=MINLEV) then V='FULL';
if(V='FULL' and FLAG=.TRUE.) then Q_INSERT(OLD);
if(V='FULL' and FLAG=.FALSE.) DISPLAY(OLD,PROJ);
if V='PARTIAL' then
    begin
        for CHILD<-0 step 1 until 8 do
            begin
            /* Generate new display token for this
               child. Evaluate as above except, if
               PARTIAL, set flag and insert. */
             NEW<-SUB_DISPLAY(OLD,CHILD,PROJ);
             V<-EVAL(VIEW_OBELS(NEW));
             if (V='PARTIAL' and LEV=MINLEV)
                 then V='FULL';
if (V='FULL' and FLAG=.TRUE.)
    then Q_INSERT(NEW);
if (V='FULL' and FLAG=.FALSE.)
    then DISPLAY(NEW,PROJ);
if V='PARTIAL' then
    begin
        FLAG<-.TRUE.;
        Q_INSERT(NEW);
```

```
                                    end;
                                end;
                            end;
                        end;
                end;
```

4.3 PERSPECTIVE VIEW

In a perspective view algorithm the viewer is no longer at an infinite distance from the origin. Figure 19 shows the viewer at a distance of X2 from the origin. The view plane is the plane at X=0. The Y value of point A is Y2 instead of simply Y1 as in an orthographic view.

By similar triangles,

$$Y1/Y2=(X2-X1)/X2$$

or

$$Y2=Y1*(X2/(X2-X1))$$

This conversion can be performed by procedure DISPLAY with the only additional piece of information required being X2, the distance to the observer. This value will not change and must simply be made available to the display procedure. Of course, the equivalent operation must be performed for the Z value.

In many cases, the value of X2 may be chosen to be a power of 2 so that shift operations are sufficient. The value of X1, however, cannot, in general, be restricted. Thus a divide for perspective display is necessary.

Two special cases can be considered to eliminate divides. The non-linear scaling operation outlined above could be used to distort the objects so as to appear as in a perspective view when using the orthographic display algorithm.

A second method can be used only if the view coordinate system is simply a translated (no rotation) version of the object coordinate system. In general, the object could be translated

and rotated so that no coordinate system rotation was necessary.

In Figure 20, it is clear that the center point of edge A-B is the average of the Y values of A and B. The projection of this point on the view plane is the average of Y1 and Y2. This is because segment A-B is parallel to the view plane. For segment B-C however, the projection of the midpoint is not, in general, the average of Y2 and Y3.

By keeping the value of Y4 and Y5 the Y value of the intersection of a line from the viewpoint to (Y2+Y3)/2 and a vertical line through the midpoint of segment B-C can be quickly determined. It is simply (Y4+Y5+Y(B)+Y(C))/4. This can be tested against Y(B). If it is greater, the projected Y value is too high and another iteration placing a new line between the line just calculated and segment O-Y3. When the error is reduced below a threshold value, the process terminates.

In an actual system it may be more desirable to simply provide a high speed integer divide capability. This usually involves a high speed inversion followed by a multiply.

The polygons painted on the screen for each side are no longer parallelograms but quadrilaterals. All four vertex points must be calculated using the perspective equations. The interior is then, of course, filled with the proper color.

One problem with a perspective view arises because of the fact that the viewing vectors are not parallel. If viewing vectors are used which are not within the region of angle space corresponding to the sequence in use, proper display operation is no longer guarranteed. A solution to the problem is to separate

the objects along the planes separating the vectors into regions. They can then be displayed separately with the proper sequence in each.

4.4 ANTI-ALIASING

The anti-aliasing display algorithms are almost identical to the above algorithms except that instead of filling in parallelograms or quadrilaterals on the final viewing screen, they are written into a higher resolution memory. Each 2**n by 2**n region is later averaged to a gray scale pixel value for final display as shown in Figure 21 for n=2. Since each section to be averaged contains 4**n elements the divide part of the average calculation can be performed by a shifting operation.

In practice, a memory smaller than 4**n times the number of pixels can be used by segmenting the display screen and performing the display algorithm once for each section.

BIBLIOGRAPHY

1. Baer, A., Eastman, C., and Henrion, M., "Geometric Modeling: A Survey", Computer-Aided Design, Vol. 11, No. 5 Sept. 1979

2. Khullar, P. and Wang, K., K., "Description and Interpretation of TIPS-1", Technical Report No. 9, NSF Injection Molding Project, Cornell University, Oct. 1976

3. Gomez, D., and Guzman, A., "Digital Model for Three-Dimensional Surface Representation", Comunicaciones Tecnicas, Vol. 9, No. 167, National University of Mexico, 1978

4. Freeman, H., "Computer Processing of Line-Drawing Images", Computing Surveys, 6, (1), March 1974

5. Ruttenberg, K., "Digital Computer Analysis of Arbitrary Three-Dimensional Geometric Configurations", Technical Report 400-69, Dept. of Electrical Engineering, New York University, Oct. 1962

6. Ruttenberg, K., "Algorithms for the Encoding of Three-Dimensionsl Geometric Figures", Technical Report

400-86, Dept. of Electrical Engineering, New York University, June 1963

7. Meagher, D., "Computer Analysis of Shape: A Literature Survey", IPL-TR-79-001, Image Processing Laboratory, Rensselaer Polytechnic Institute, May 1979

8. Williams, R., "A Survey of Data Structures for Computer Graphics Systems", Computing Surveys, Vol. 3, No. 1, March 1971

9. Clark, J., "Hierarchical Geometric Models for Visible Surface Algorithms", Communications of the ACM, Vol. 19, No. 10, Oct. 1976

10. Bentley, J. L., "Multidimensional Divide-and-Conquer", Communications of the ACM, Vol. 23, No. 4, April 1980

11. Bentley, J. L., "Multidimensional Binary Search Trees in Database Applications", IEEE Transactions on Software Engineering, Vol. SE-5, No. 4, July 1979

12. Franklin, W. R., "Locating a Point in Overlapping Regions of Hyperspace", Technical Report CLR-64, Rensselaer Polytechnic Institute, Dec. 1978

13. Sidhu, G. S., and Boute, R. T., "Property Encoding:

Applications in Binary Picture Encoding and Boundary Following", IEEE Transactions on Computers, Vol. C-21, No. 11, Nov. 1972

14. Tanimoto, S. L., "A Pyramid Model for Binary Picture Complexity", IEEE Computer Society Conference on Pattern Recognition and Image Processing, Rensselaer Polytechnic Institute, June 1977

15. Hunter, G. M., and Steiglitz, K., "Operations on Images Using Quad Trees", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 2, April 1979

16. Rosenfeld, A., "Tree Structures for Region Representation", Computer Vision Laboratory, University of Maryland, 1979

17. Samet, H., "Computing Perimeters of Images Represented by Quadtrees", TR-755, Computer Science Center, University of Maryland, College Park, April 1979

18. Samet, H., "Deletion in Two-Dimensional Quad Trees", Computer Science Dept., University of Maryland, March 1979

19. Samet, H., "Region Representation: Quadtrees from Boundary Codes", Communications of the ACM, Vol. 23, No. 3, March 1980

20. Dyer, C. R., Rosenfeld, A., and Samet, H., "Region Representation: Boundary Codes from Quadtrees", Communications of the ACM, Vol. 23, No. 3, March 1980

21. Samet, H., "Region Representation: Raster-to-Quadtree Conversion", TR-766, Computer Science Dept., University of Maryland, May 1979

22. Samet, H., "Region Representation: Quadtree-to-Raster Conversion", TR-768, Computer Science Dept., University of Maryland, June 1979

23. Samet, H., "Region Representation: Quadtree from Binary Arrays", TR-767, Computer Science Dept., University of Maryland, May 1979

24. Samet, H., "A Quadtree Medial Axis Transformation", TR-803, Computer Science Dept., University of Maryland, August 1979

25. Samet, H., "A Distance Transform for Images Represented by Quadtrees", TR-780, Computer Science Dept., University of Maryland, 1979

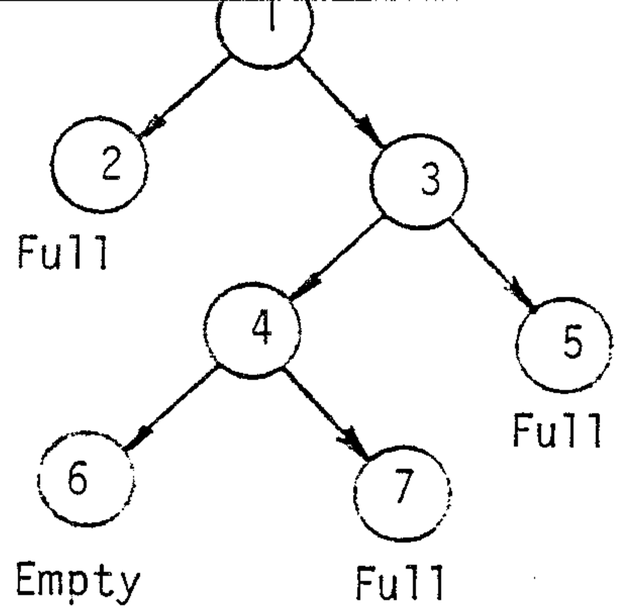
26. Cohen, J. and Hickey, T., "Two Algorithms for Determining Volumes of Convex Polyhedras", Journal of the Association of Computing Machinery, Vol. 26, No. 3, July 1979

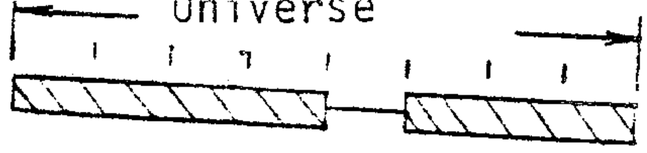
case	child	offset	overlay	type	obel-no	child-no
0	0	offset=0	0	IGNORE	-	-
0	0	offset=0	1	POINTER	1	0
0	1	offset+(h/2)=e/2	0	IGNORE	-	-
0	1	offset+(h/2)=e/2	1	POINTER	1	1
1	0	0<offset<h/2	0	POINTER	0	1
1	0	0<offset<h/2	1	POINTER	1	0
1	1	0<=offset<h-(e/2), and not case 0	0	POINTER	1	0
1	1	0<=offset<h-(e/2), and not case 0	0	POINTER	1	1
2	0	h/2<=offset<=e/2	0	POINTER	0	1
2	0	h/2<=offset<=e/2	1	IGNORE	-	-
2	1	h-e/2<=offset<=h/2	0	POINTER	1	0
2	1	h-e/2<=offset<=h/2	1	IGNORE	-	-
3	0	e/2<offset<e/2+h/2	0	POINTER	0	0
3	0	e/2<offset<e/2+h/2	1	POINTER	0	1
3	1	h/2<offset<h	0	POINTER	1	1
3	1	h/2<=offset<h	0	POINTER	1	0
4	0	e/2+h/2<=offset<=e	0	POINTER	0	0
4	0	e/2+h/2<=offset<=e	1	POINTER	-	-
4	1	H<=offset<=e/2+h/2	0	POINTER	0	1
4	1	h<offset<=e/2+h/2	0	POINTER	-	-
5	0	(none)	0	-	-	-
5	0	(none)	1	-	-	-
5	1	e/2+h/2<offset<=e, and not case 6	0	POINTER	0	0
5	1	e/2+h/2<offset<=e, and not case 6	1	POINTER	0	1
6	0	(none)	0	-	-	-
6	0	(none)	1	-	-	-
6	1	offset=e and h=e/2	0	POINTER	0	0
6	1	offset=e and h=e/2	1	IGNORE	-	-

Table 1 Algorithm SCALE Cases

7 to 6	0	0	(−)
7 to 5	0	(−)	0
7 to 4	0	(−)	−
7 to 3	(−)	0	0
7 to 2	(−)	0	−
7 to 1	(−)	−	0
7 to 0	(−)	−	−
6 to 5	0	(−)	+
6 to 4	0	(−)	0
6 to 3	(−)	0	+
6 to 2	(−)	0	0
6 to 1	(−)	−	+
6 to 0	(−)	−	0
5 to 4	0	0	(−)
5 to 3	(−)	+	0
5 to 2	(−)	+	−
5 to 1	(−)	0	0
5 to 0	(−)	0	−
4 to 3	(−)	+	+
4 to 2	(−)	+	0
4 to 1	(−)	0	+
4 to 0	(−)	0	0
3 to 2	0	0	(−)
3 to 1	0	(−)	0
3 to 0	0	(−)	−
2 to 1	0	(−)	+
2 to 0	0	(−)	0
1 to 0	0	0	(−)

Table 2 Origin Transition Vectors for Region #0 of Angle Space

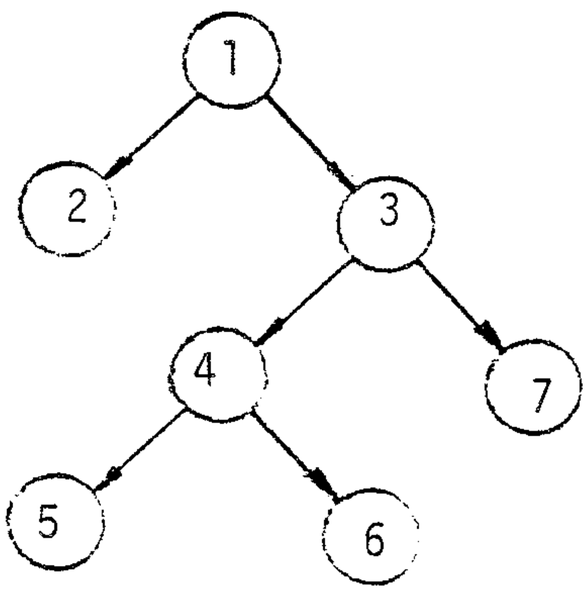




Location: 1 2 3 4 5 6 7
Value: 1 2 1 1 2 0 2

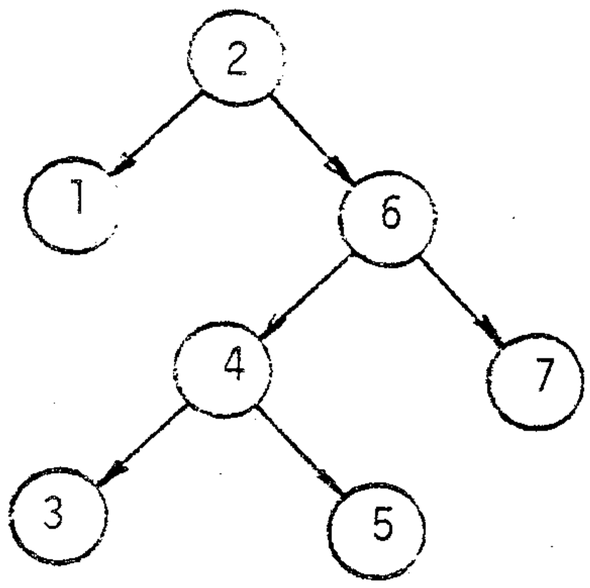
(Values: 0 = 'EMPTY', 1 = 'PARTIAL',
2 = 'FULL')

(a) Bredth-First Sequential Allocation



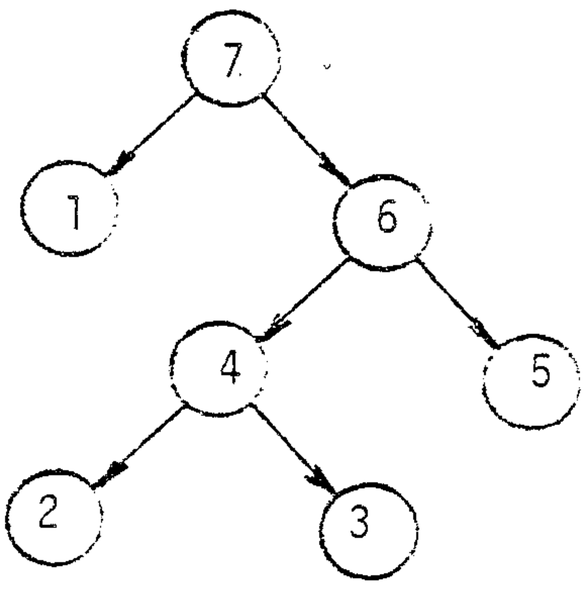
Location: 1 2 3 4 5 6 7
Value: 1 2 1 1 0 2 2

(b) Preorder Depth-First Sequential Allocation



Location: 1 2 3 4 5 6 7
Value: 2 1 0 1 2 1 2

(c) Inorder Depth-First Sequential Allocation



Location: 1 2 3 4 5 6 7
Value: 2 0 2 1 2 1 1

(d) Postorder Depth-First Sequential Allocation

Figure 1 Sequential Allocation Format

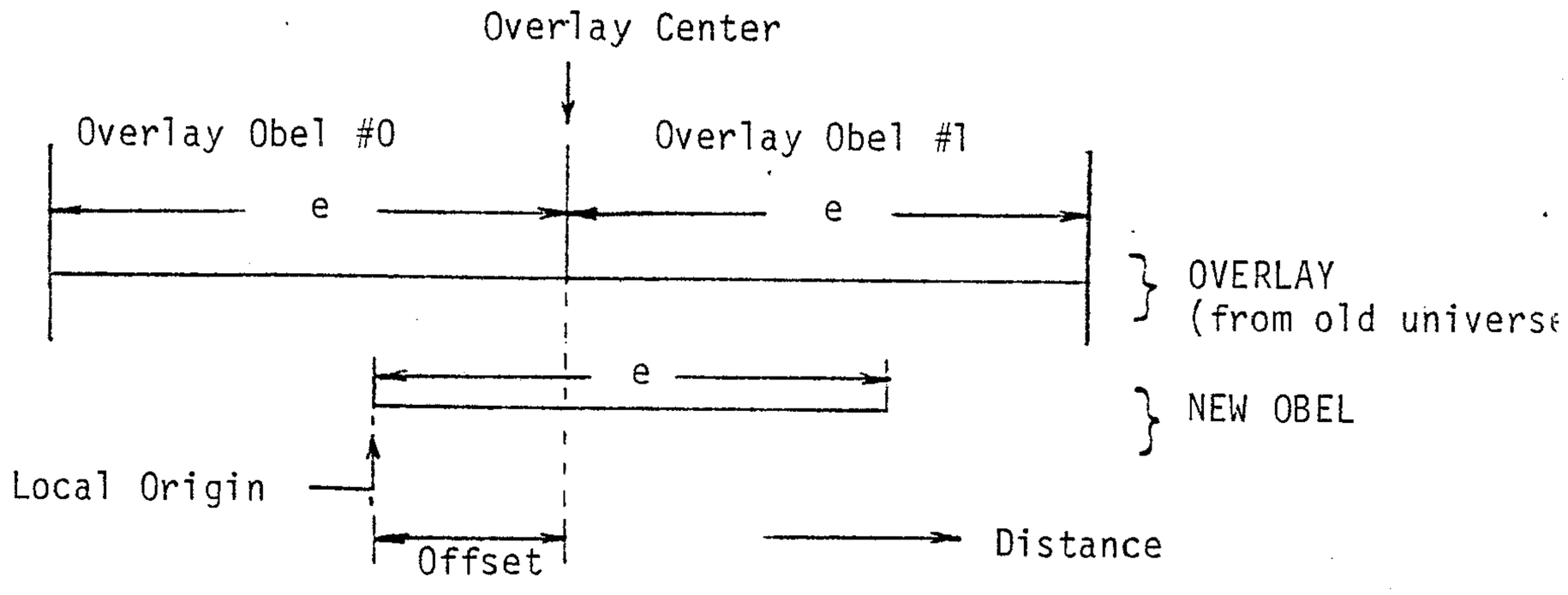


Figure 2 One-Dimensional Overlay

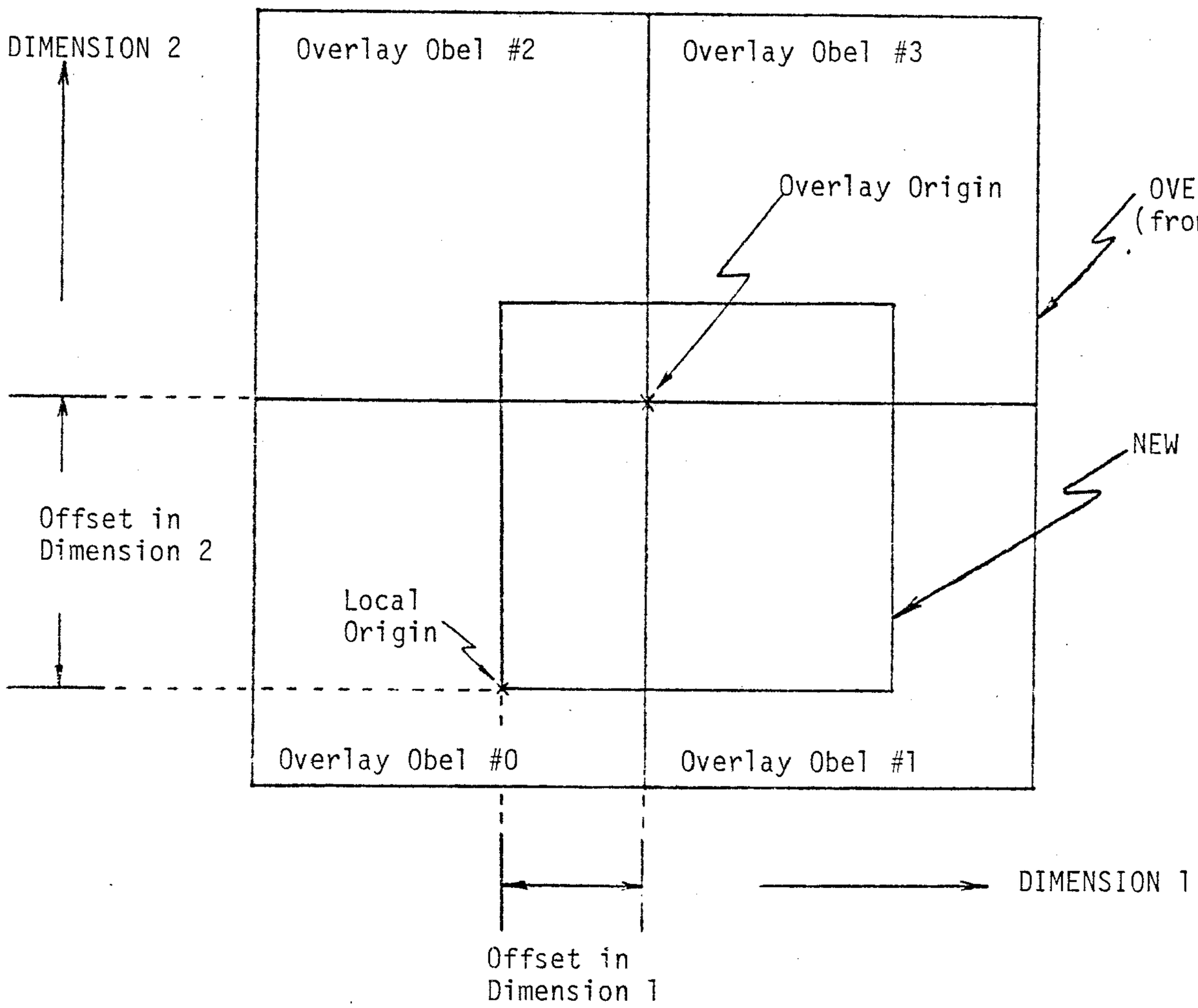
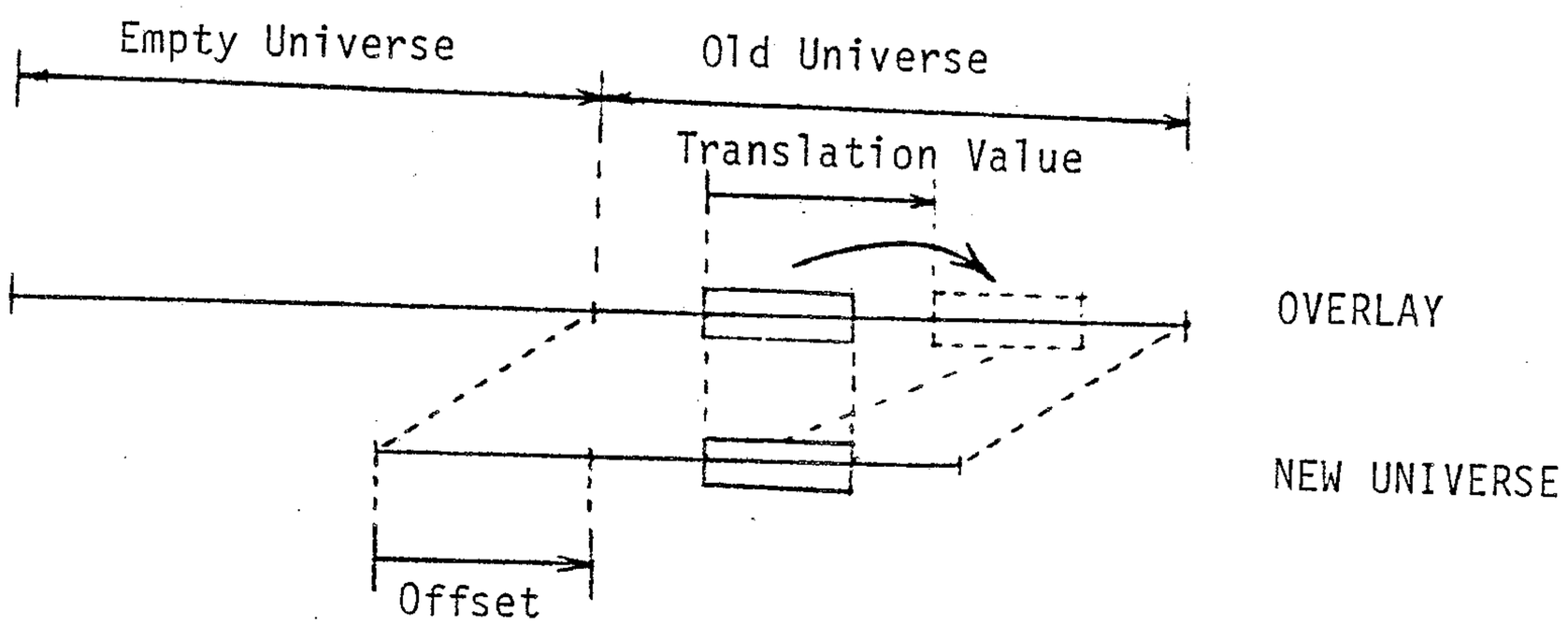
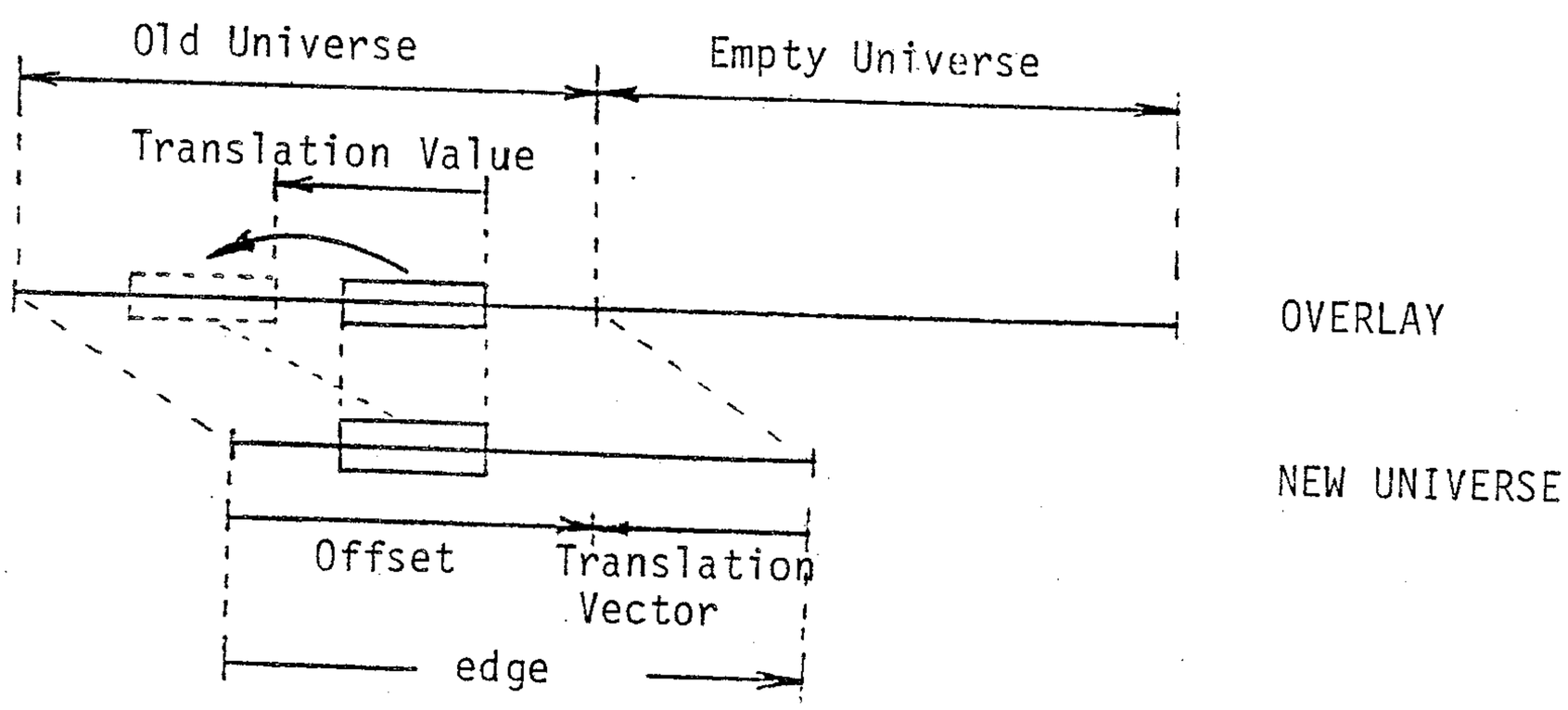


Figure 3 . Two-Dimensional Overlay



(a) Positive Translation Vector (offset = translation vector)



(b) Negative Translation Vector (offset = edge + translation value)

Figure 4 Placement of Object Universe in Augmented Universe for Initial Overlay

OVERLAY OBEL 0 OVERLAY OBEL 1

| Child 0 | Child 1 | Child 0 | Child 1 |
|<------- e ------->|

Target Obel:	Child 0	Child 1	Child	Overlay	Type	Obel	Child
sub-overlay: (Child 0)	new 0 | new 1 — New Overlay		0	0	IGNORE	-	-
			0	1	POINTER	1	0
sub-overlay: (Child 1)		new 0 | new 1 — New Overlay	1	0	IGNORE	-	-
			1	1	POINTER	1	1

(a) Case 0 (OFFSET=0)

| Child 0 | Child 1 | Child 0 | Child 1 |

Target Obel:	Child 0	Child 1	Child	Overlay	Type	Obel	Child
sub-overlay: (Child 0)	new 0 | new 1 — New Overlay		0	0	POINTER	0	1
			0	1	POINTER	1	0
sub-overlay: (Child 1)		new 0 | new 1 — New Overlay	1	0	POINTER	1	0
			1	1	POINTER	1	1

(b) Case 1 (0<OFFSET<e/2)

| Child 0 | Child 1 | Child 0 | Child 1 |

Target Obel:	Child 0	Child 1	Child	Overlay	Type	Obel	Child
new 0 | new 1 — New Overlay			0	0	IGNORE	-	-
			0	1	POINTER	0	1
sub-overlay: (Child 1)	new 0 | new 1 — New Overlay		1	0	IGNORE	-	-
			1	1	POINTER	1	1

(c) Case 2 (OFFSET=e/2)

| Child 0 | Child 1 | Child 0 | Child 1 |

Child 0 | Child 1	Child	Overlay	Type	Obel	Child
new 0 | new 1 — New Overlay	0	0	POINTER	0	0
	0	1	POINTER	0	1
sub-overlay: (Child 1) new 0 | new 1 — New Overlay	1	0	POINTER	0	1
	1	1	POINTER	1	0

(d) Case 3 (e/2<OFFSET<e)

Figure 5 Translation Cases

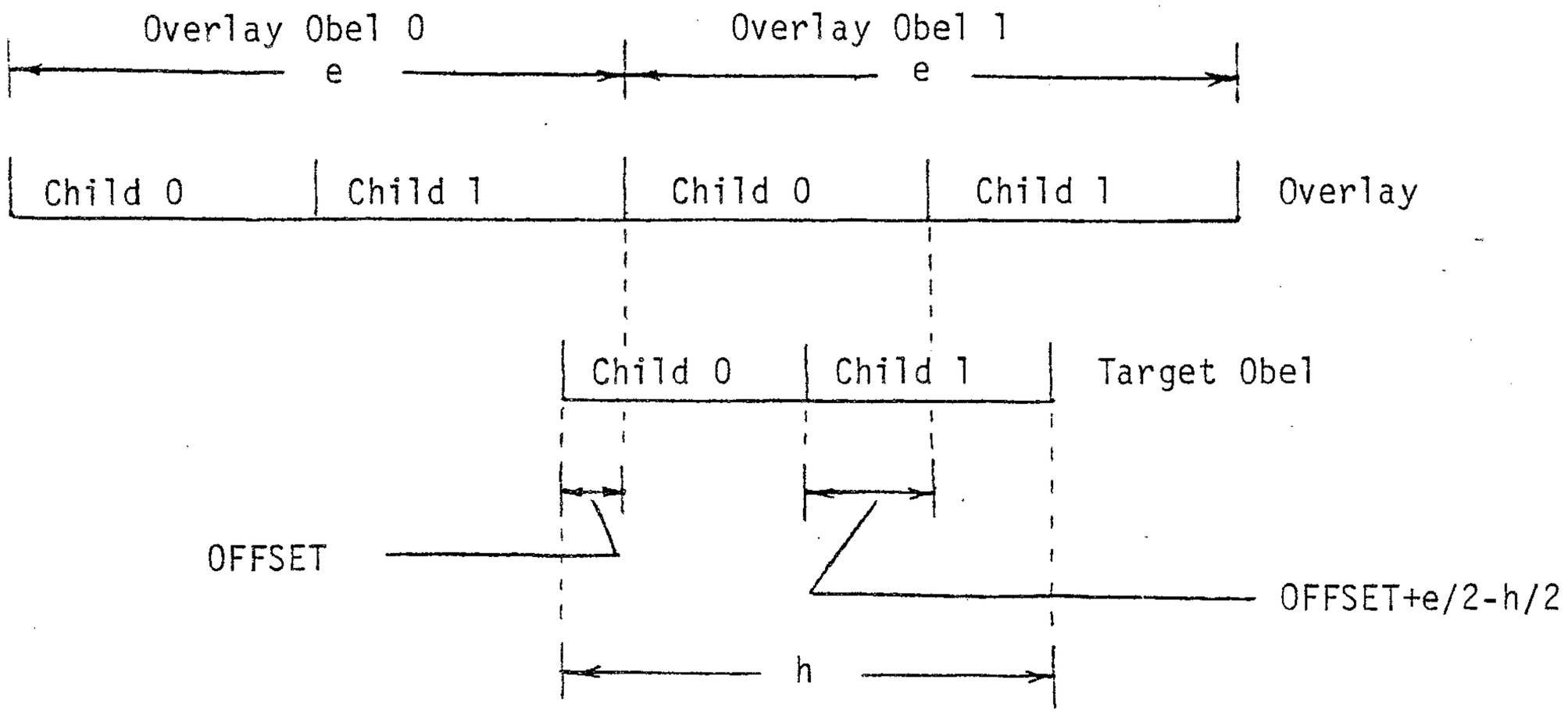


Figure 6 Algorithm SCALE One-Dimensional Overlay
(0.5<=h/e<=1.0)

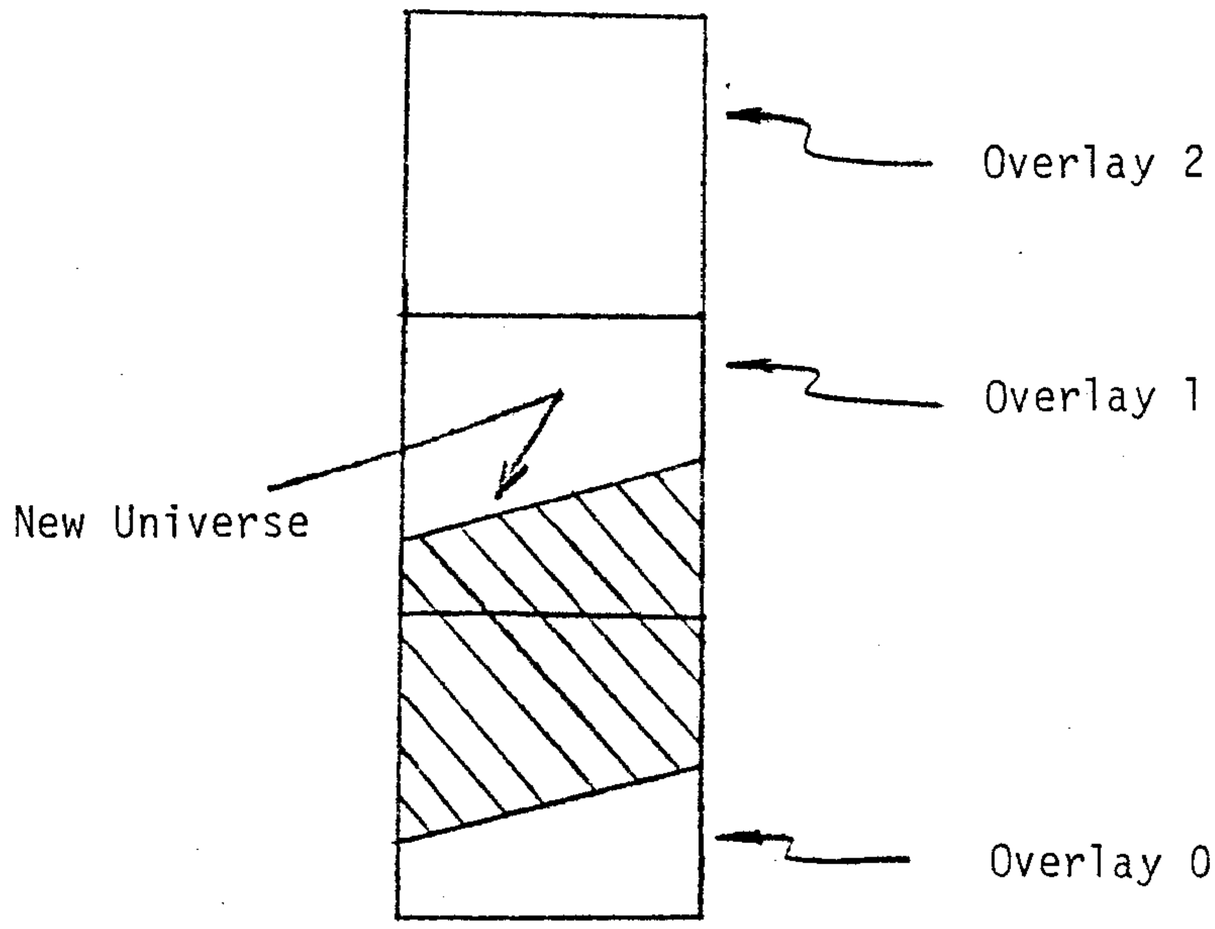


Figure 7 Overlay Scheme for Shear Transformation

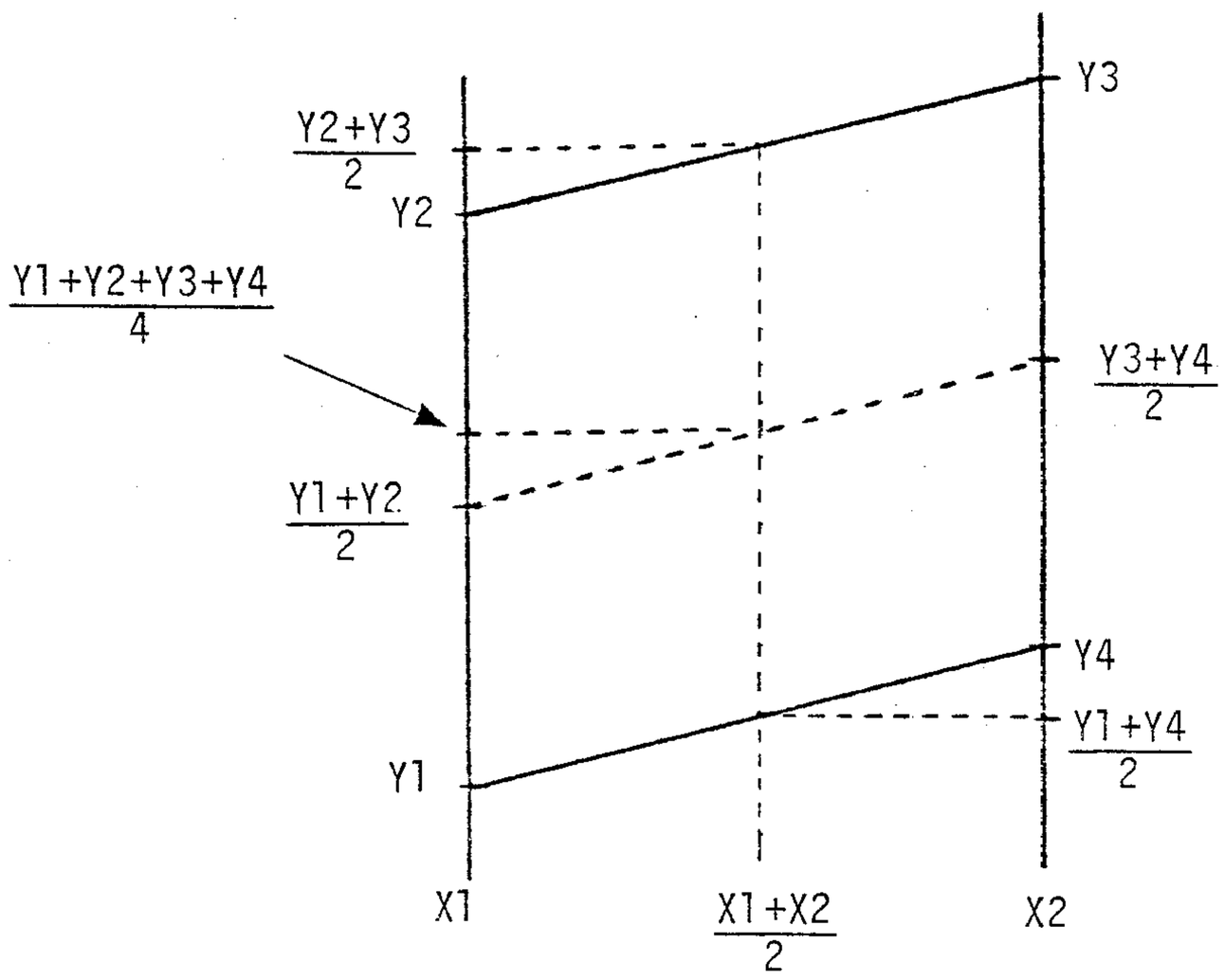


Figure 8 Children of Shear Target Obel

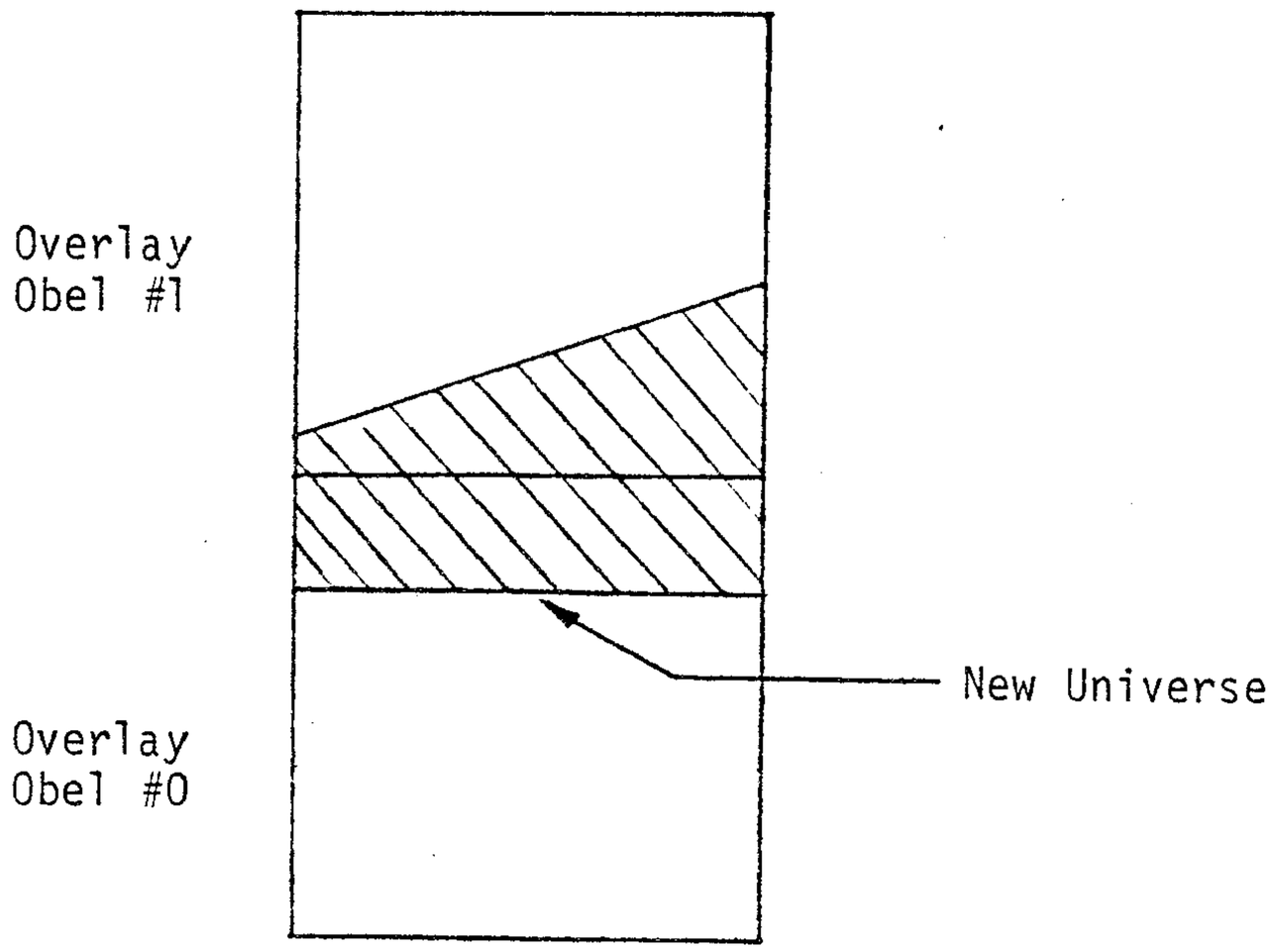


Figure 9 Nonlinear Scaling Operation

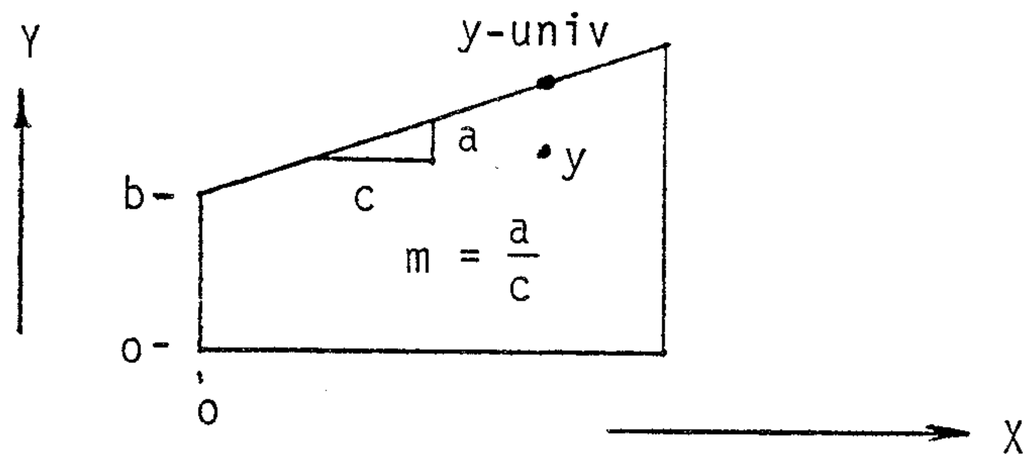


Figure 10 Nonlinear Scaling Target Obel

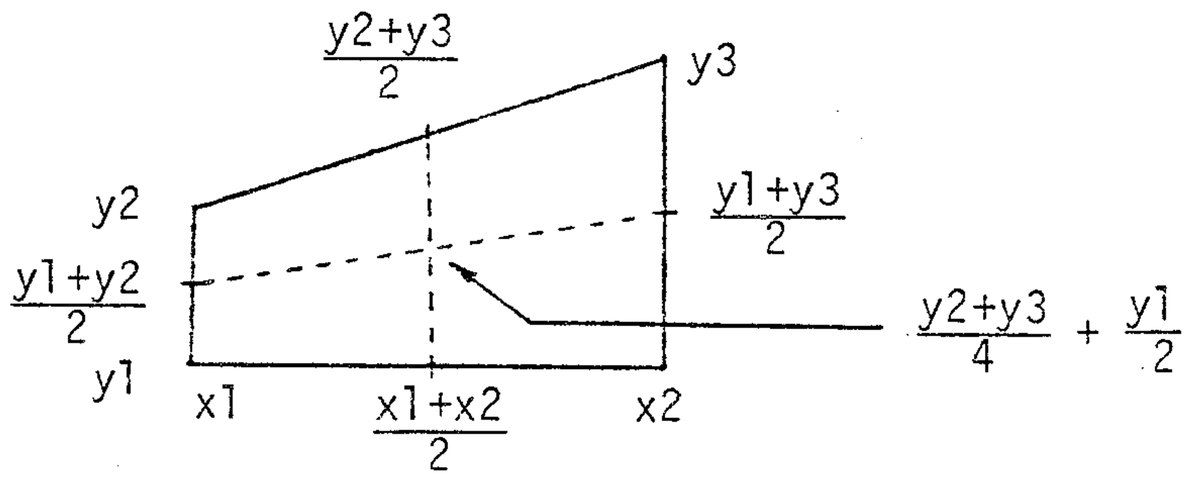


Figure 11 Nonlinear Scaling Target
Obel Child Generation

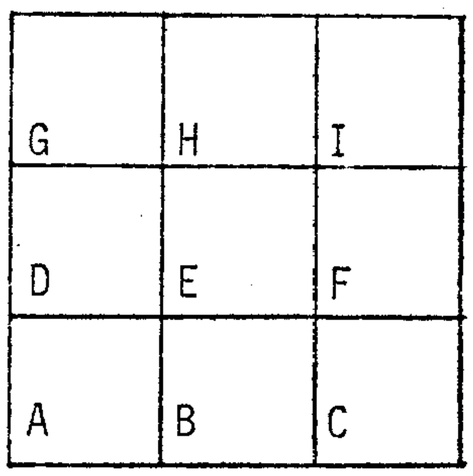


Figure 12 3 by 3 Overlay for Rotate

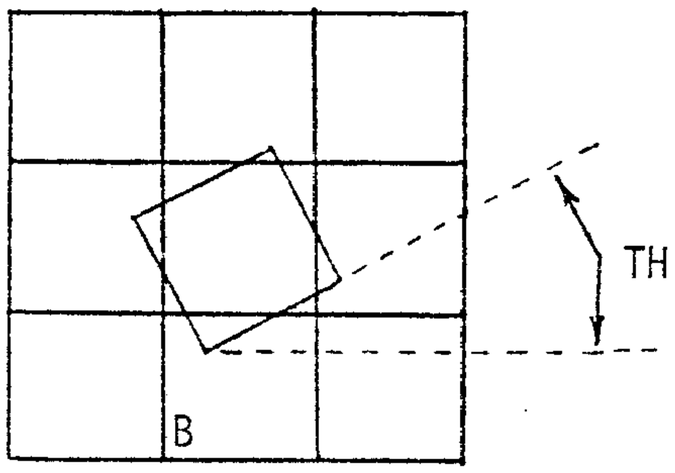


Figure 13 Target Obel in 3 by 3 Overlay

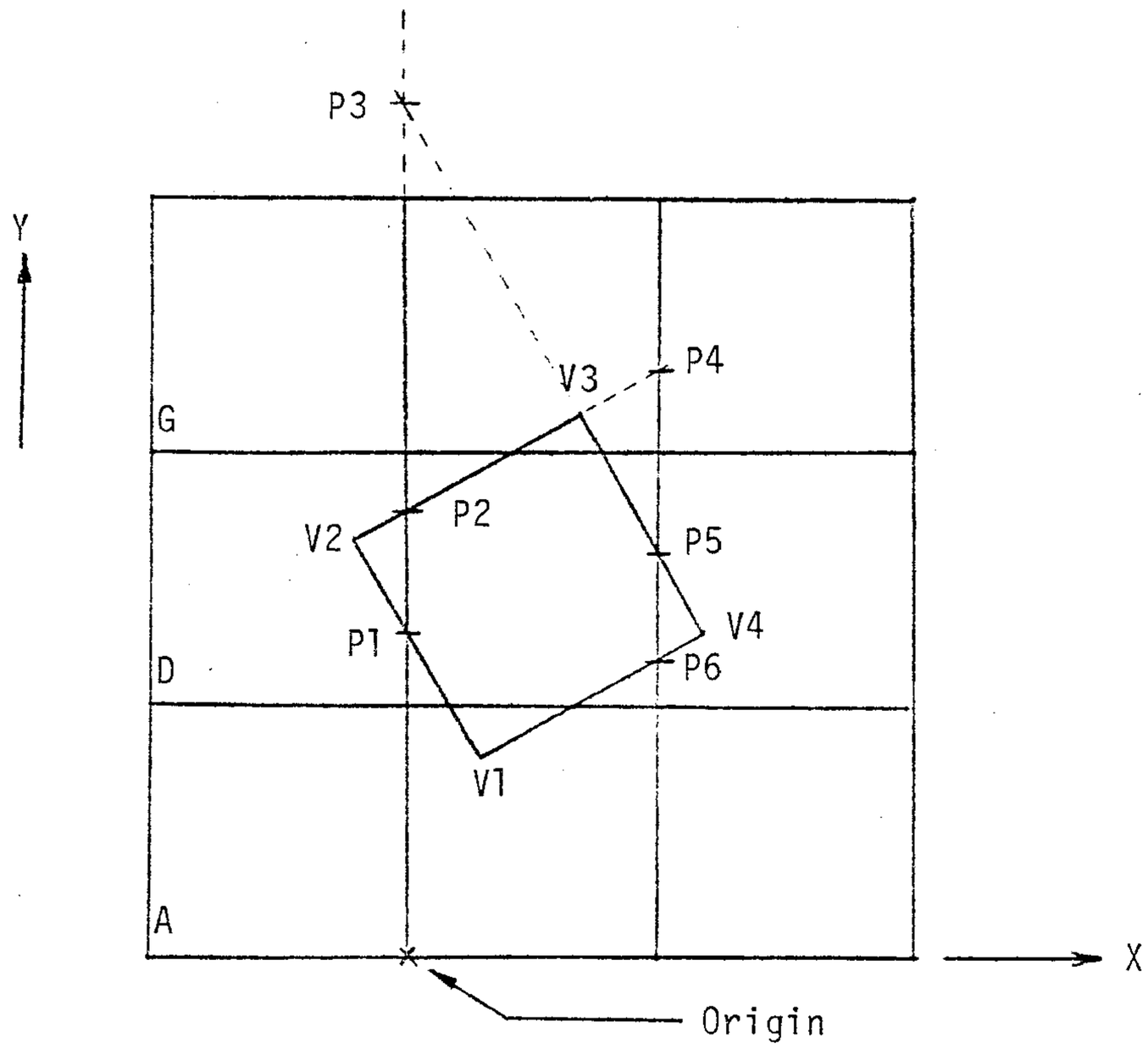


Figure 14 Control Points for Determination of Case in Rotation

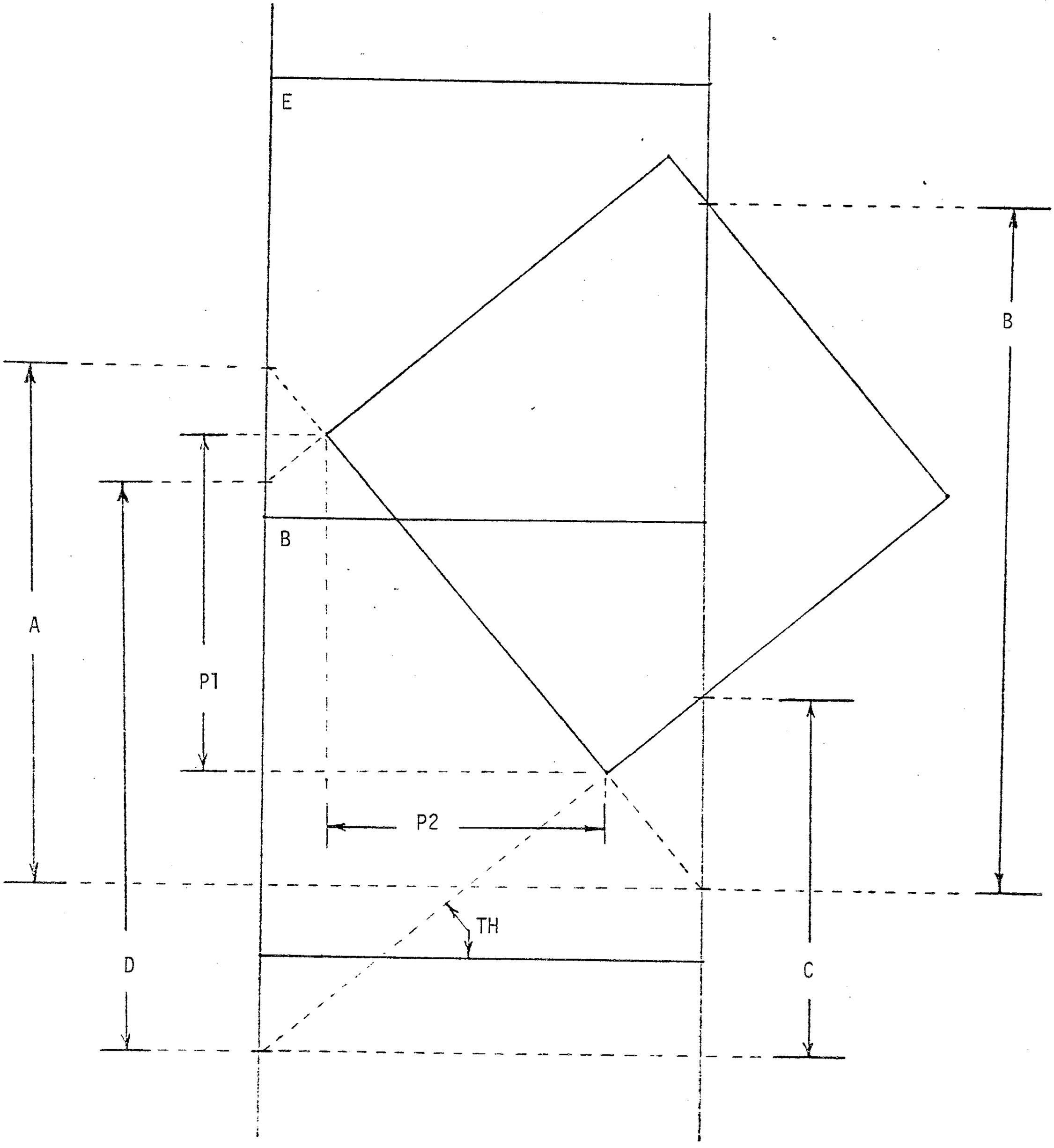


Figure 15 Universal Rotation Parameters

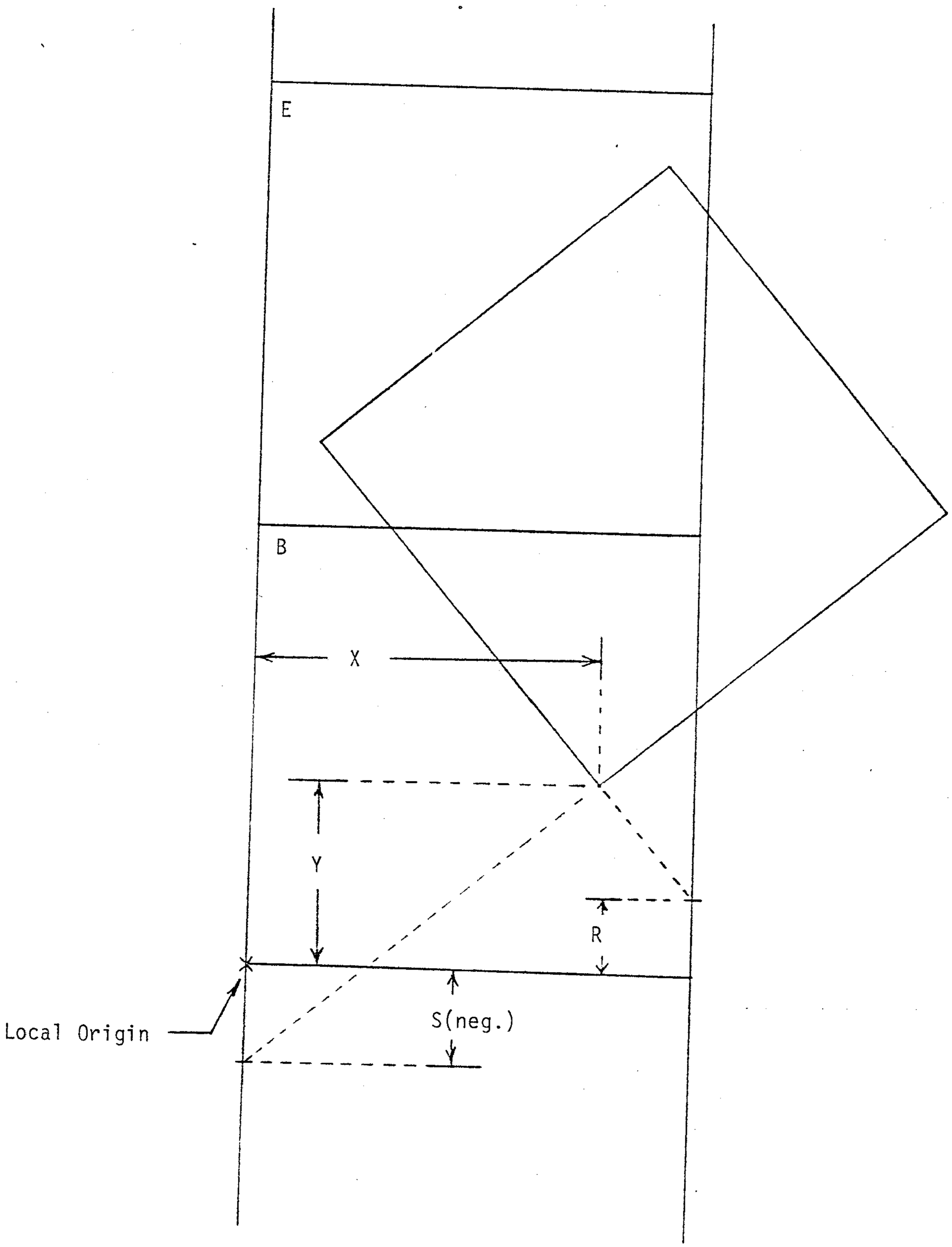


Figure 16 Local Rotation Parameters

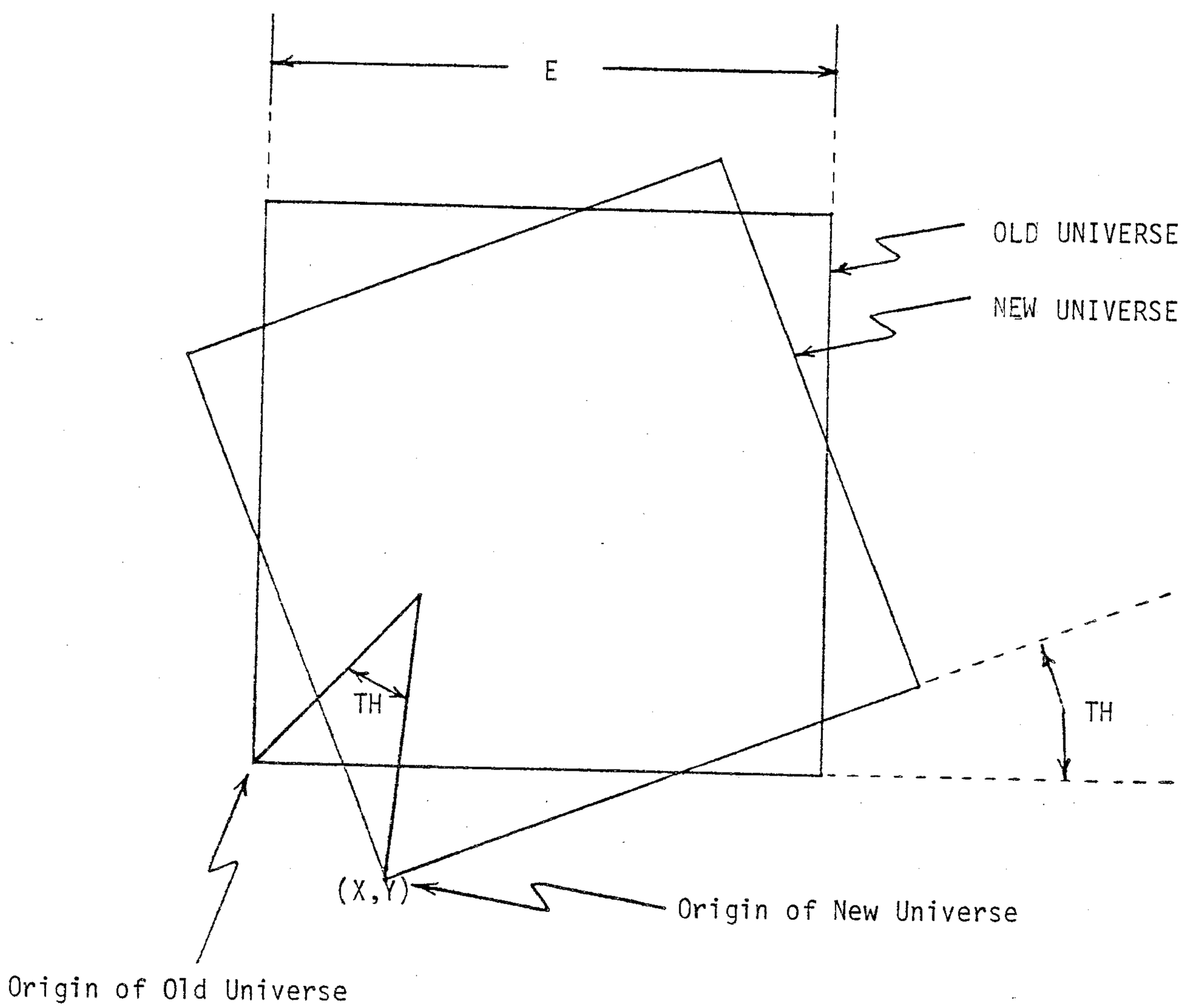


Figure 17 Origin of New Universe

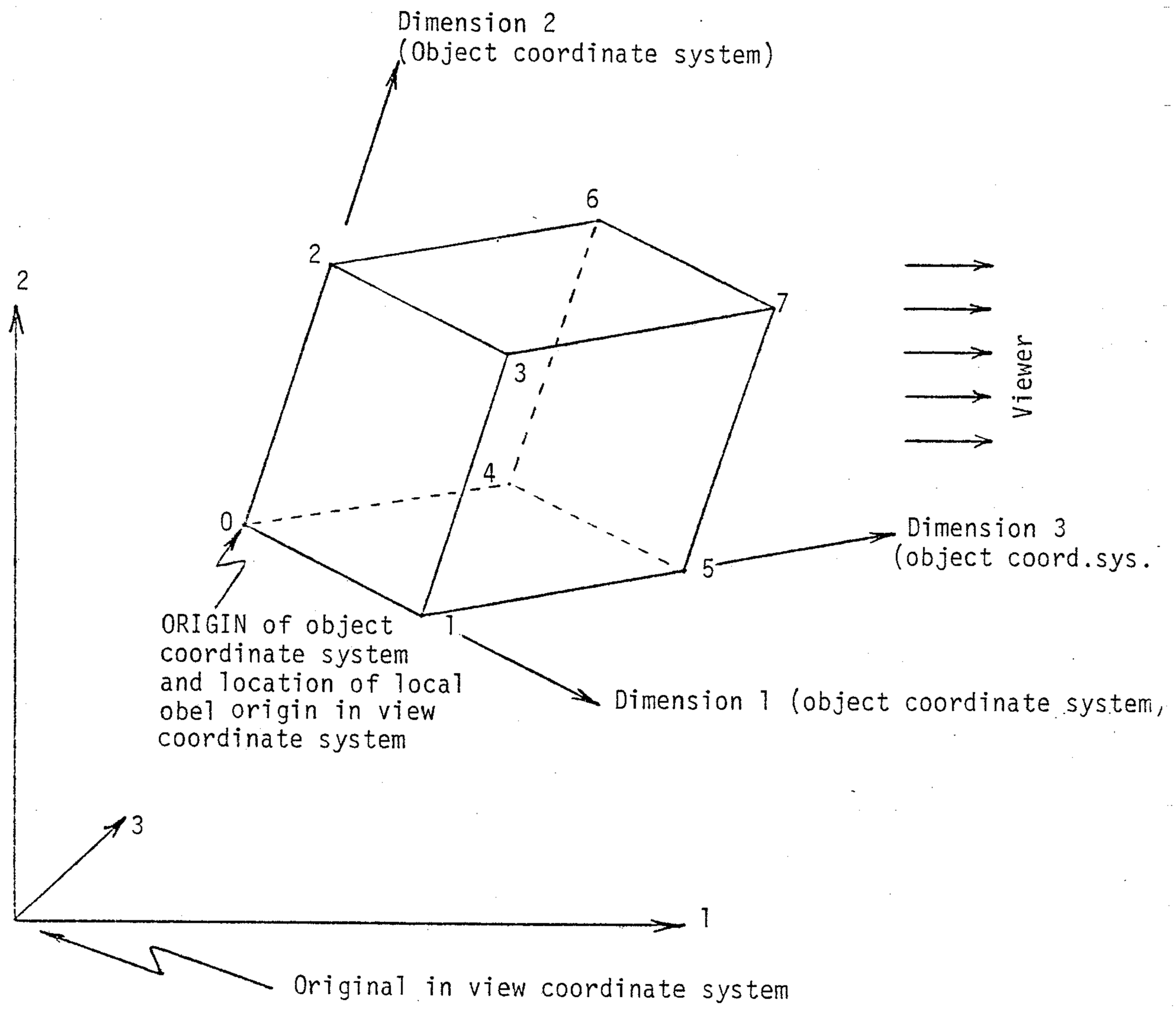


Figure 18 View Coordinate System and Vertex Points in View Coordinates.

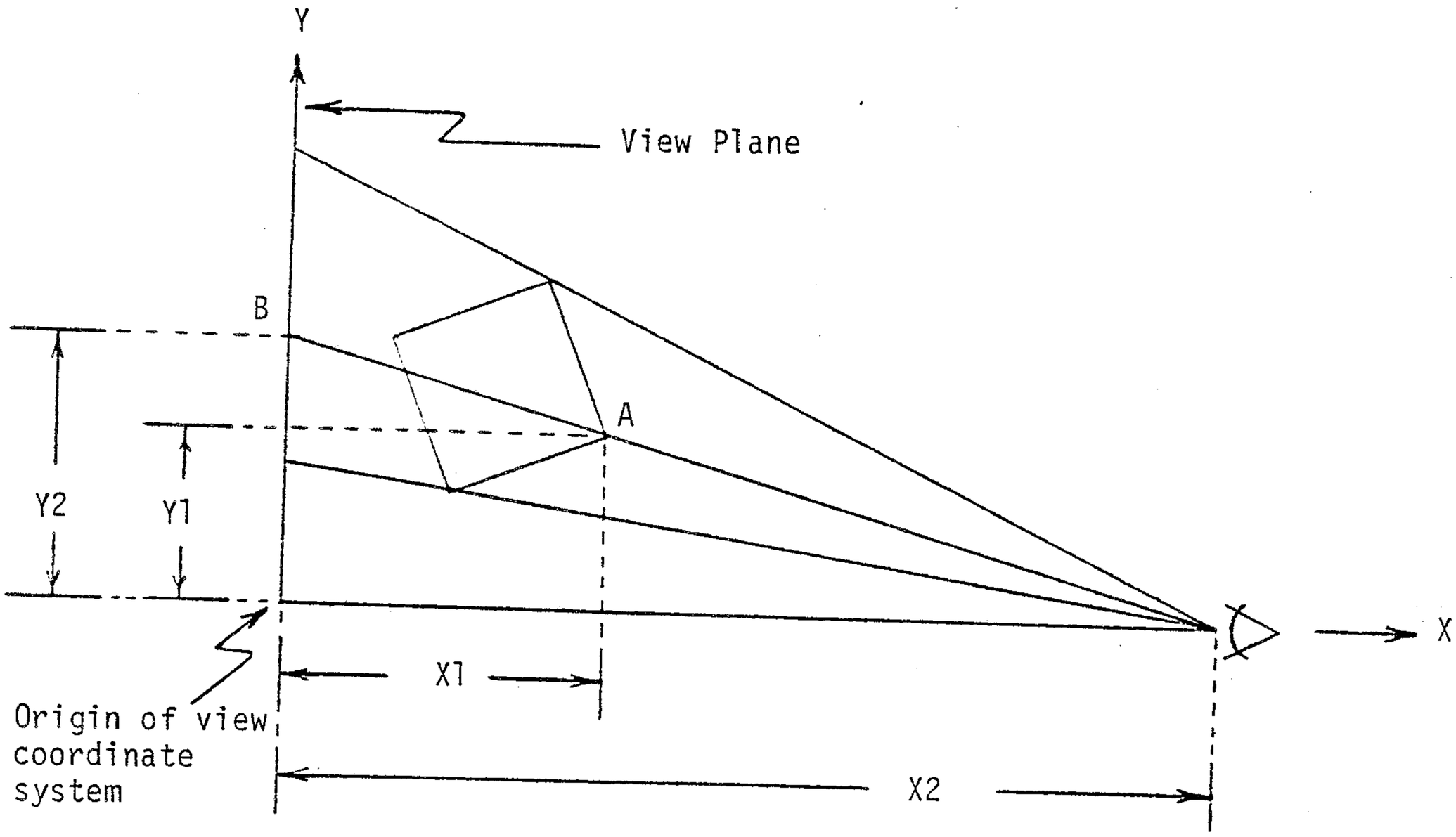


Figure 19 Perspective View

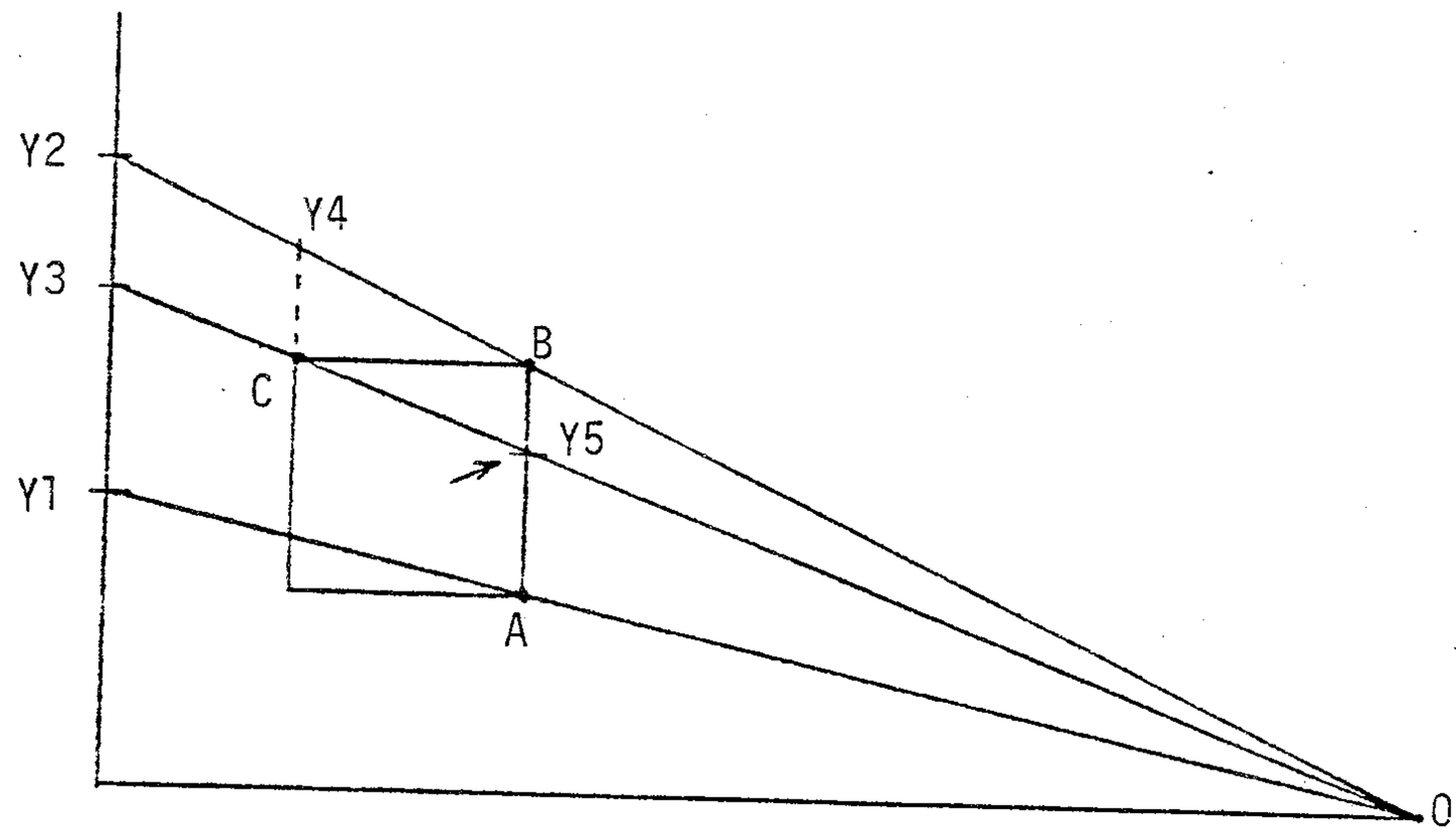
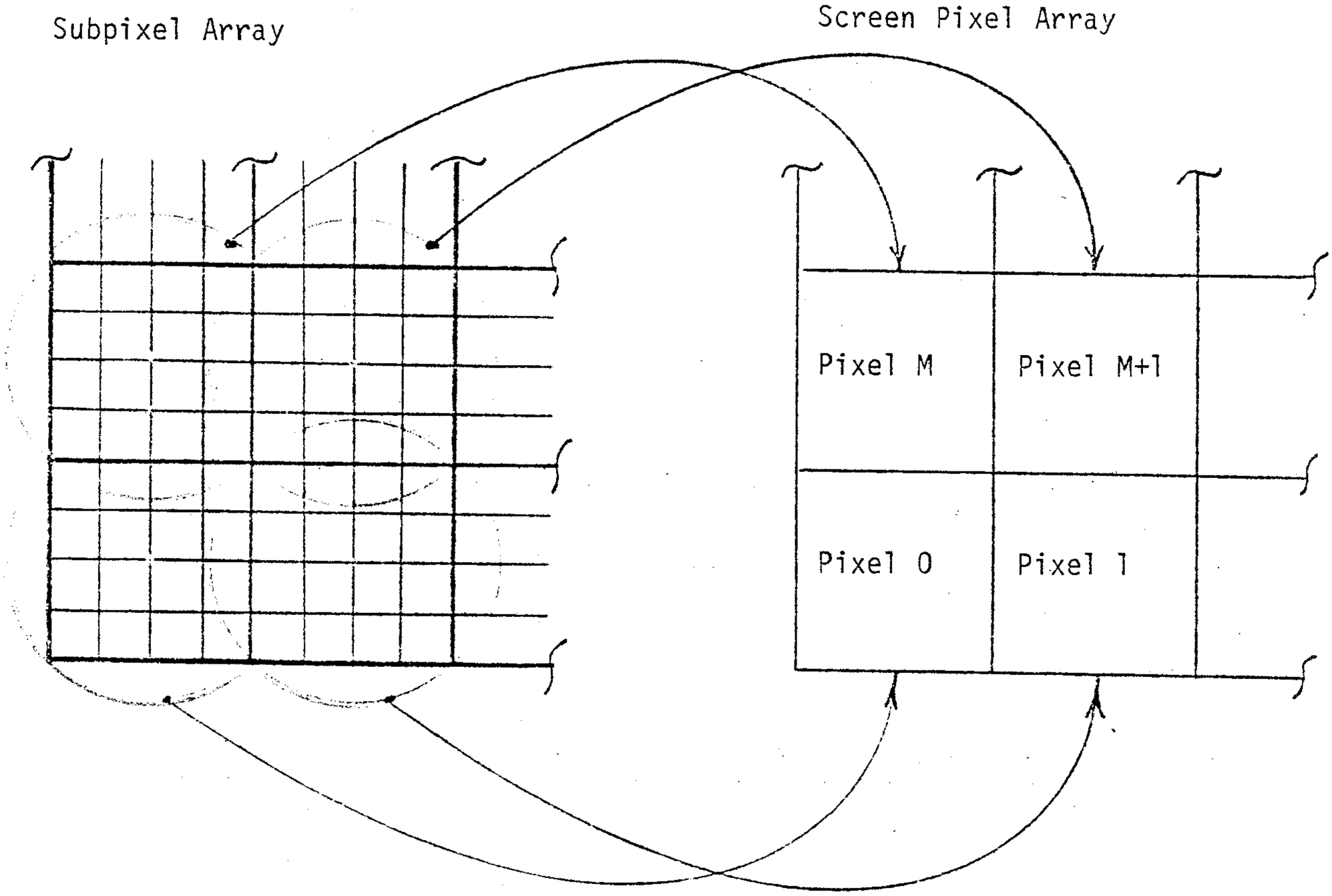


Figure 20 Approximation of Perspective View



Example: 16 subpixel array elements averaged into gray scale value for each pixel.

Figure 21 Anti-Aliasing Technique