A HIERARCHICAL REPRESENTATION FOR 3-D OBJECTS

O.D. FAUGERAS and J. PONCE

INRIA Domaine de Vol uceau, Rocquencourt
BP 105-78133 Je O sna> Cedex FRANCE

ABSTRACT

fn this paper we propose a new hierarchical structure for describing 5-0 objects, called the Prism Tree. This ternar> tree structure, inspired from the planar Strip Tree, is built from an initial trianguiation of the object by using a polyhedral approximation algorithm. Different properties of this representation are shown, as well as algorithms for intersecting objects and neighbor finding techniques.

INTRODUCTION

Multilevel representations have been found useful in Computer Vision for several years now [l,3,4,5] . By allowing the access and manipulation of regions and objects at various levels of details, they provide a natural data structure for thinking about objects in the context of a given task. They also are excellent tools for speeding up various algorithms by the fact that "divide and conquer" techniques are easily implemented on such data structures. In this paper we propose a representation for 3-D objects which is a generalization of Ballard's Strip Trees [1] • In this method space curves or surfaces are hierarchically approximated by using a prism as enclosing box, yielding a ternary tree structure called the Prism Tree. One advantage of this structure is that it is intrinsic to the object shape, and therefore invariant through rigid transformations. Moreover, this tree structure allows to implement efficiently neighbor finding operations and surfaces intersections as tree traversal algorithms. We first explain the general principle of our approach and show how to build a Prism Tree from a polyhedral representation of an object, then prove a theorem that allows to efficiently test intersections of surfaces, and describe algorithms for performing various operations on objects.

I THE ENCLOSING BOX SCHEME

There are two basic approaches to the hierarchical description of shapes in Computer Vision: The representation may be either attached to a fixed grid, or to the object itself. The former approach gave birth to Quadtrees and their 3-D generalization, the Oct-Trees [3,4,5]. Ballard [1] investigated the second approach. The idea is to use a recursive polygonal approximation algorithm [6] for curves and, at each level of the recursion, to associate to each line segment SEG the smallest rectangular box with sides parallel to SEG which encloses the part of the curve delimited by the extremities of SEG (fig. 1). Such a box (called a Strip by Ballard) represents a part of the object as a pixel would but it has the advantage of being attached to the curve itself. As the recursion proceeds, each line segment is split into two parts, and one can associate to the initial box the corresponding "children" boxes. So a binary tree structure is built quite naturally, that is invariant through rotations and translations, and on which a variety of algorithms (intersections..) can be efficiently implemented. A simple example of application is verifying that two objects do not intersect by traversing their Strip Trees and checking at each node that the Strips do not intersect. Other applications are in Graphics, with multilevel displays.

The idea of enclosing the object in a succession of boxes is readily extendable to the 3-D space. Using a polyhedron for describing 3-D objects is a way of structuring the space quite analog to representing a planar shape by a circular ordered list of points. In this context, we presented elsewhere [2] a generalization of the previously mentioned polygonal approximation, the object being split in triangles instead of segments. In the next Section, we briefly describe this algorithm and show how one can associate to each triangle a part of the surface and a prismatic box enclosing it.

II POLYHEDRAL APPROXIMATION AND PRISM TREE CONSTRUCTION

We assume that at the lowest level, the objects are represented b> polyhedra: Our data is the graph GO defined by the points in 3-D space, and the polyhedral edges joining them. There are several ways of obtaining such polyhedra from range data [7 ,8]-

i. *1* . Polyhedral approximation

We use a top-down approach. We consider a triangulation of the object which is refined at each step until some error threshold THR is reached. During the execution of the algorithm, the triangulation is considered as a "face"-graph GP. We associate to each triangle T a node with diverse attributes: the vertices, denoted P, 0, R, the "part ST of the surface associated to T" (we will see later what we mean by these words), and the corresponding error ERR (e.g. The maximal distance from the triangle to the surface). The edges of GP are defined by the neighboring relations of the triangles (two triangles sharing an edge are neighbors). The algorithm loops over the following steps

```
.SPLIT STEP (fig. 2)
Po1 each triangle T verifying ERR greater than THR do
 Find M, point of ST lying at maximum distance of T
  Let Tl, T2, T3 be the nodes defined b>:
   the vertices of Tl are P, Q, M
   the vertices of T2 are Q, R, M
   the vertices of T3 are R, P, M
   ST "has been split" into three parts corresponding to Tl, T2, T3, ERR1, ERR2, ERR3 are the corresponding errors, then replace the node T by Tl, T2, T3 in the graph (in particular update the neighbors of T).
 ADJUSTMENT STEP (fig. 3)
if we only used the SPLIT STEP , old edges would ever remain, even if they are very far from the surface.This is the reason why we add this step which removes the edge between two faces having just been split
Por each pair Tl, T2 of such triangles do
 find M, point of STl union ST2 which is the closest, from the bissector plane of Tl and T2 , and the furthest from their common edge
 split Tl in two triangles of vertices P1,Q,M and P1,R,M, split T2 in two triangles of vertices P2,Q,M and P2,R,M
 update the graph the same way as in the SPLIT STEP.
```

Let us add some details. Pirst the algorithm has to be initialized with a starting graph: it is usually obtained by taking three points which define a first triangle. This triangle is then split in a way quite analog to the SPLIT STEP, so the starting GP describes a 6 faces polyhedron made of two tetrahedra sharing a face that is the initial triangle (fig. 4). But notice that any other triangulation could be used as well. It is now time to define clearly the meaning of ST and the way to cut it in three (or two for the adjustments) parts. In fact there are two possible algorithms, denoted ALGRA and ALGLI) :

The first one, described in [2], works for objects defined by polyhedra without holes (genus zero). It uses the property that in this case GO is planar, so each cycle drawn on this graph cuts it into two connected components. Let us consider a pair of neighboring triangles Tl and T2 of GP, let Q and R be their common vertices, if we associate with each of these pairs a path (Q,R) in GO, we define for each node T of GP a cycle (P,Q) ,(Q,R),(R,P) of GO and, as it is a planar graph, a connected subset ST of GO. Thus we only have to give an algorithm for finding non intersecting paths to define a partition of the surface in ST's. We define them recursively by using the SPLIT STEP (the case of the ADJUSTMENT STEP is quite analog and will be omitted). Let T be a node to split (fig. 5), and M the splitting point. We suppose the paths (P,Q), (Q,R), and (R,P) defined. Let Tl, T2, and T3 be the triangles PQM, QRM, and RPM , and Pl, P2, and P3 the bissector planes of the pairs T2,T3 , T1,T3 , and T1,T2. We define (P,M) as the path in ST which is the closest to the plane P2. The path (Q,M) is defined similarly using the plane P3 and only considering points which do not belong to (P,M). These two paths and the path (P,Q) define a cycle which cuts ST in two components Cl and C2. Let Ci be the component which contains R, we now find a path (R,M) in Ci by using the plane Pl. The three paths do not intersect, and they split ST into three components STl, ST2, and ST3 which correspond to the cycles (P,M,Q), (Q,M,R), and (R,M,P) .

The second possible way of defining the ST's does not make explicit use of the graph GO. ST is no more a subgraph but. simply a list of points. We will also define it recursively by using the SPLIT STEP. Suppose ST is defined for a triangle T that we are going to split. We keep the same notations and still call Pl, P2, P3 the three bissector planes defined by the splitting. STl will simply be the subset of ST composed of the points of ST which lie between the planes P2 and P3, ST2 and ST3 will be defined similarly, and it is clear that we so obtain a partition of ST. We must though remark that there is in general no guaranty that

ST is connected (fig. 6), so ws have to be cautious when using this method (in the case of a convex shape, one can nevertheless easily prove that the ST's stay connec ted).

2) Prism Tree .construction

We now consider GF *at* some level of the recursion. Let T be one of its nodes, and ST the associated part of the surface (which can be either a list or a subgraph). We wish to define an enclosing box of ST. With its three neighboring triangles, T defines three bissector planes PI, P2, and P3 (fig. 7). The "Prism" associated to T will be the smallest 5-faces polyhedron of which three faces are parallel to PI, P2, and P3, and the two remaining ones are parallel to the PQR plane, that contains all the points of ST (fig. 8). These five planes will be denoted PP1, PP2, PP3, P31, and PB2 and the two triangles of the prism parallel to T will be denoted TBI and TB2. We can now build a ternary tree as the recursion proceeds. Although it is surely not the most concise description of a prism, we suppose for the sake of clarity and efficiency that one finds 3t each node of the tree the following informations: P, Q, R, PP1, PP2, PP3, TBI, TB2, three flag bits REG , MARK (we will explain their use in the sequel) and ADJ (=1 if the node has been obtained through an adjustment), and four pointers S0N1, SON2, S0N3, and FATHER. The root points on two nodes (first pointer at NIL) associated to the two half surfaces defined by the initial triangle. We define, the children nf :. . A triangle T split into three sub-triangles Ti,T2,T3 corresponds to a node the three children of which are defined by T1,T2,T3. An adjusted triangle has only two children, the third pointer being set at NIL. In turn, each child points towards its father. In the sequel, a pointer to a node will be denoted PT, and the corresponding node PT*. Fig. 9 show3 an example of Pri3m Tree for a simple objec t .

III SOME PROPERTIES OF THE PRISM TREES

In order to prove some important properties of Prism Trees, we need to define a regular prism: a prism will be said to be regular if the associated ST is connected, and if its three planes PP1, PP2, and PP3 are PI, P2, and P3 themselves (fig. 10). This notion is quite similar to the regular strips of Ballard [l] and prevents pathological configurations (fig. 8). This is the reason why at each node of the Prism Tree we keep the regularity bit REG which will be set if the corresponding prism is regular.

It. is clear that a node of a tree built by using ALGRA is regular iff PP1, PP2, and PP3 are P1, P2, and P3 themselves. Conversely, if the tree is built by using ALGLI the condition is the connexity of ST (which can easily be checked after each splitting by using GO). We now give two lemmas (only indications of their proofs are given for space savings) and prove a theorem.

3) Lemma 1: Surface-Prism Intersection

Let T be a regular node of a Prism Tree. Then the intersection of ST with PP1,PP2,PP3 is a closed curve (fig. 10). (easy to verify with ALGRA and ALGLI)

4) Lemma 2: Curve-Surface Intersection

Let T be a regular node of a Prism Tree. Then any continuous curve C starting from a point U inside TB1, ending at a point V inside TB2, and staying within the prism intersects the surface of the object an odd number of times (fig. 11). (use the previous lemma and the Jordan closed surface theorem)

5) Prism-Prism Intersection Theorem

Let T and T' be two regular nodes of Prism Trees, and TB1, TB2 and TBI', TB2' the associated basis triangles. Then if for any i and j (i,j=1,2), the intersection of the interiors of of the triangles TBi and TBj' is non empty (CLEAR intersection) then the underlying surfaces ST and ST' intersect (fig. 12).
(analog of the Clear Intersection Lemma for Strip Trees)

PROOP: From the hypothesis we deduce that the intersections of at least one PPk' with both TB1 and TB2 are non empty , and so consist of two segments A,B and C,D. From Lemma 1, the intersection of PPk' with ST' is a continuous curve, and so is the intersection of ST' with the polygon A,B,C,D. The extremities of this curve lie inside TB1 and TB2, so from Lemma 2 we deduce that it intersects ST. Q.E.D

By using this theorem and the obvious fact that if two prisms do not intersect (NULL intersection) then the underlying surfaces do not intersect either, the next two Sections will show how one can check whether two surfaces intersect, and find their intersection curve considered as a Prism Tree. Prom now on, we suppose that we dispose of two functions VOL(PT) and PRISMINT(PT,PT'). VOL(PT) returns the volume of the prism associated to PT*. PRISMINT(PT,PT•) returns CLEAR, POSSIBLE, or NULL according to the type of intersection of the prisms associated to PT* and PT'*. Moreover, if a node is not regular (REG=0) PRISMINT will never return CLEAR unless both nodes are leaves (in this case we assume that the intersection is either CLEAR or NULL). If one of the pointers is NIL, PRISMINT returns NULL. As they are highly recursive, we will express our intersection algorithms in a pseudo PASCAL form and, following Ballard, we take the heuristic to divide first the largest prism during the recursion (so the prisms compared are always approximatively equal sized).

IV TESTING SURFACE-SURFACE INTERSECTION

Our algorithm is a direct extension of the Ballard's one. The two trees pointed by PT and PT' are visited until a CLEAR intersection has been found or all the nodes have been checked.

```
Function CHECKI(PT,PT'):boolean;
begin
case PRISMINT(PT,PT') of
 NULL    : CHECKI:=false;
 CLEAR   : CHECKI:=true;
 POSSIBLE: if VOL(PT) > VOL(PT')
  then CHECKI:=(CHECKI(PT↑.SON1,PT') or
   CHECKI(PT↑.SON2,PT') or
   CHECKI(PT↑.SON3,PT')
  else CHECKI:=(CHECKI(PT,PT'↑.SON1) or
   CHECKI(PT,PT'↑.SON2) or
   CHECKI(PT,PT'↑.SON3)
 end
end;
```

V INTERSECTION OP TWO SURFACES

We now wish to find the intersection curve of two surfaces. This will be done by finding and marking (using the MARK flag) in the two trees all the nodes whose prism intersects the other surface. We use the obvious property that if a node intersects a surface, then all its ancestors intersect this surface. Conversely, all the descendants of a node that does not intersect a surface do not intersect it either. This gives the following algorithm, where all the flags MARK have previously been initialized 0.

```
Procedure SURF_INTER(PT,PT');
begin
(*PRISMINT = NULL is the stopping
condition. It corresponds to the case
where a node is at nil or has an empty
intersection with the surface*)
if PRISMINT(PT,PT') <> NULL then
begin
if PRISMINT(PT,PT') = CLEAR
 (*nodes intersect the other surface*)
 then begin PT↑.MARK:=1;PT'↑.MARK:=1 end;
(*the recursion proceeds*)
if VOL(PT) > VOL(PT')
 then begin
  SURF_INTER(PT↑.SON1,PT');
  SURF_INTER(PT↑.SON2,PT');
  SURF_INTER(PT↑.SON3,PT');
  (*if the mark of the children is 1*)
  (*then PT^ intersects the surface*)
  if(PT↑.SON1<>NIL)and(PT↑.SON1↑.MARK=1)
  or(PT↑.SON2<>NIL)and(PT↑.SON2↑.MARK=1)
  or(PT↑.SON3<>NIL)and(PT↑.SON3↑.MARK=1)
  thenPT↑.MARK:= 1
  end
  else begin
  SURF_INTER(PT,PT'↑.SON1);
  SURF_INTER(PT,PT'↑.SON2);
  SURF_INTER(PT,PT'↑.SON3);
  (*if the mark of the children is 1*)
  (*then PT'^ intersects the surface*)
  if(PT'↑.SON1<>NIL)and(PT'↑.SON1↑.MARK=1)
  or(PT'↑.SON2<>NIL)and(PT'↑.SON2↑.MARK=1)
  or(PT'↑.SON3<>NIL)and(PT'↑.SON3↑.MARK=1)
  then PT'↑.MARK:=1
  end
end;
```

When the execution of the algorithm is complete, all nodes having a "CLEAR" descendant have been marked, and no descendant of a "NULL" node has been marked. To obtain the intersection curve, one has only to visit one of the trees and to prune it from all the nodes marked 0 (in that case, the regularity flag of ail the ancestors of these nodes must be set to 0). Notice that we have two representations of the curve, one for each tree.

VI A NEIGHBOR FINDING ALGORITHM

Algorithms which operate on hierarchical structures often need to explore the "neighbors" of the visited nodes [5], In our case, the neighbors of a node are clearly defined : During the construction of the tree, a graph structure has been maintained. So at each level of the tree, the neighbors of a node T can be defined as the nodes corresponding to the neighbors in GP of the triangle associated to T. By storing GP at each level of the tree, one would have an easy, but very memory consuming, way of finding the neighbors. We will show in this Section how one can find the neighbors of each node without, storing the

GF's. Notice first that the only tree transformation we have defined is the SURP_INTER procedure, which prunes the tree without modifying it3 structure, so that each node of a tree has always three neighbors, some of them being eventual!) empty (NIL pointer).

We first define the "eldest" son of a node. When splitting a node by using the approximation algorithm, three new triangles, and three new edges are created, so that each triangle has always two "new" and one "old" edge. The child of a node which corresponds to this "old" edge will be called the eldest son of the node (e.g. in fig. 9, the triangle 353 is an eldest son). In the case of an adjustment, it is clear that the eldest son will be the missing one. During the construction the pointer SQN1 will always represent the eldest son.

We first consider the case where no adjustment has occured. Then we find the three neighbors of a node using the following algorithm:

```
Let T be the node visited
  .two of its neighbors are its two
  brothers
   let E be the remaining edge (it can be
  found by examining the vertices of T)
   if T is an eldest son
     then ascend the tree until the current
     node is no more an eldest son. let Tl
     be this son
     else T1:=T
  .let T2 be the father of Tl
  .descend the tree by visiting at each
  level the only son that shares E with T
  until the node is a leaf or it is at the
  same level as T
  .this node is the third (exterior)
  neighbor of T
```

It is easy to see that the algorithm "works": if a node is an eldest son, its father shares E with a neighbor which is not its brother (obvious), so we have to ascend the tree. Moreover the edge E of a node is always the "old" one, suppose that the node is not an eldest son, then E is an edge of its father, but cannot be its "oldest" one, so the father shares it with one of its brothers.

We can now extend this method to the case of the adjusted nodes (ADJ=l). The difference is that an adjusted node T ha3 not two neighboring brothers, but only one, the third one is its adjustment neighbor(e.g. triangles 753 and 736 in Pig. 9). Let E be the corresponding edge. We find the two usual neighbors by the previous algorithm. For the adjustment neighbor, we look at the father Tl. Tl has been adjusted with one of its neighbors T2, which is its exterior neighbor. We simply have to find T2 by the previous algorithm, and the neighbor of T is the son of T2 which shares E with T.

VII INTERSECTION OP TWO VOLUMES

In [l]» Ballard gives an algorithm for computing the intersection of two areas defined by the Strip Trees of their boundaries. The method consists in finding in each of the trees all the nodes which intersect or are inside the other area. The intersection area is then obtained b> considering all these nodes. Let us denote A and A' the areas, and S and S' the corresponding trees. Ballard uses a function STRIP INSIDE to check wether a node of S which has a NULL intersection with S' lies or not inside A'. This is verified by traversing the whole tree S', and counting the number of CLEAR intersections of a semi-infinite Strip issued from the node with S'. If this number is odd the node lies inside A' (analog to the usual method for testing wether a point lies within a polygon). The disadvantage of this algorithm is that it computes the function STRIP_INS1DE for each NULL intersection node, implying at each time a full traversal of S'.

Such a method could be used for Prism Trees. A 5-D solid is defined oy its closed surface, and the associated Prism Tree PT. Any prism can be classified in one (and one only) of the three following types:

1: the intersection of the prism with the surface is non empty
2: prism fully inside the surface
3: prismfully outside the surface

Let us consider two objects represented by two Prism Trees PT and PT', it is possible to mark all their nodes of type 1 by using the algorithm of section V. Testing the type of a NULL intersection node can be made by the following 3-D equivalent (P_INSIDE) of the function STRIP_INSIDE: Let us consider a node T of PT. We can associate to it a semi-infinite straight line L normal to T, issued from the barycenter of this triangle on the TBI side (Pig. 13). The notion of CLEAR intersection can be extended by saying that L has a CLEAR intersection with a regular prism T' if it intersects the corresponding TBI' and TB2'. Prom Lemma 2 we deduce that in this case L intersects ST^1 an odd number of times. The function P_INSIDE is then computed by traversing PT' and counting the number of these CLEAR intersections, it returns the type (2 or 3) of the node. The intersection of the two objects is described by the two trees pruned of their nodes of type 3. Remark that a node of type 2 or 3 has all its descendants of same type, so they do not need to be examined.

However, one his still to traverse PT• at each NULL node. The situation is even worse than in the 2-0 case as we deal with a ternary tree and a more complex INSIDE function. He propose a Rightly different algorithm for avoidinq too many computations of P INSIDE It is Dosed on the fact that if Fwo neighboring Prisms of PT have a NULL intersection with PT , then they lie both inside or outside the surface associated to PT' (proof- suppose that one lies inside and the other outside, then their common edge lies it the same time inside and outside the surface, which is impossible). So one has only to compute P IN3IDE for a few nodes' and to explore and mark for each of these modes its connected component of NULL intersection nodes by using the neighbor endmg algorithm of the previous section this gives the following alagorithm.

```
Procedure VOL_INTER(PT,PT');
(*marks the nodes of PT.          *)
(*SURF_INTER has already been applied*)
begin
if PT <> NIL then
  begin
  (*look at the mark of the node*)
  case PT↑.MARK of
  0: begin (*NULL intersection. Inside?*)
      M:=P_INSIDE(PT,PT');
      (*we mark the node,and its neighbors
      using the algorithm of section V*)
      MARK(PT↑,M);
      EXPLORE_AND_MARK(PT,M);
      end;
  1: begin (*inters. Look at children*)
      VOL_INTER(PT↑.SON1,PT');
      VOL_INTER(PT↑.SON2,PT');
      VOL_INTER(PT↑.SON3,PT');
      end;
  otherwise:STOP(*already marked 2 or 3*)
  end;
end;
```

Notice that in the descendant phase of the neighbor finding algorithm, one can mark all the ancestors of the desired node that have a NULL intersection with PT' (if a node is inside a surface, all its ancestors having an empty intersection with the surface are inside it). When the execution of this algorithm is complete for the two trees, one has only to prune them from their nodes marked 3. In spite of the complexity of the neighbor finding procedure, it is clear that this algorithm is more efficient than the previous one, as finding a neighbor only needs to simply follow two branches of a tree, which is faster than traversing a full tree.

CONCLUSION

We have presented a new method for representing 3-D objects. The representation is object-centered and is therefore invariant to rigid motions. It is hierarchical and allows us to think about and describe objects at various levels of resolution. Last but not least it is an efficient tool for performing various boolean operations on objects. W- are pursuing the analysis of the corresponding algorithms. It is clear that they will be more efficient for trees composed essentiall) of regular nodes'. This implies that the polyhedral approximation, described in Section II be adapted to the object shape. It. turns out that this is true for convex (or almost convex) objects and therefore we plan to use the prism tree representation on such subparts when necessarv. We are also exploring applications in CAD (combining primitive solids by boolean operations) and Robotics (obstacle collision avoidance b) interference checking).

REFERENCES

[1] D.H. Ballard, "Strip Trees: A Hierarchical Representation For Curves", Comm. of the ACM, Vol 24, No 5, 1981

[2] O.D. Faugeras, M. Hebert, P. Mussi and J.D. Boissonnat, "Polyhedral Approximation of 3-D Objects Without Holes", Computer Graphics and Image Processing, in press

[3] S. Tanimoto and T. Pavlidis, "A Hierarchical Data Structure for Picture Processing", Computer Graphics and Image Processing, Vol. 4 No 2, 1975

[4] C.L. Jackins and S.L. Tanimoto, "Oct-Trees and their Use in Representing 3-D Objects", Techn. Report No 79-07-06, Dept. of Comptr. Science, Univ. of Washington, 1980

[5] H. Samet, "Neighbor Finding Techniques for Images Represented by Quadtrees", Computer Graphics and Image Processing, Vol. 18 , 1982

[6] R.O. Duda and P.E. Hart, "Pattern Classification and Scene Analysis", Wiley-Interscience, New York, 1973

[7] O.D. Faugeras and E. Pauchon, "Measuring the Shape of 3-D Objects," CVPR-83

[8] J.D. Boissonnat, "Representation of Objects by Triangulating Points in 3-D Space," Proc. Of the 6th Int. Conf. On Pattern Recognition, pp.830-832 , Munich, 1982

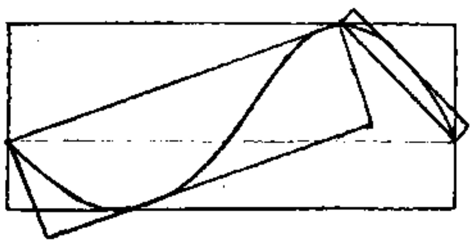

Fig.1. A curve and its Strip representation.

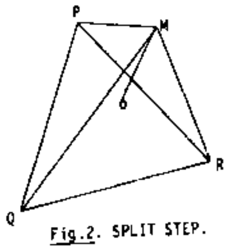

Fig.2. SPLIT STEP.

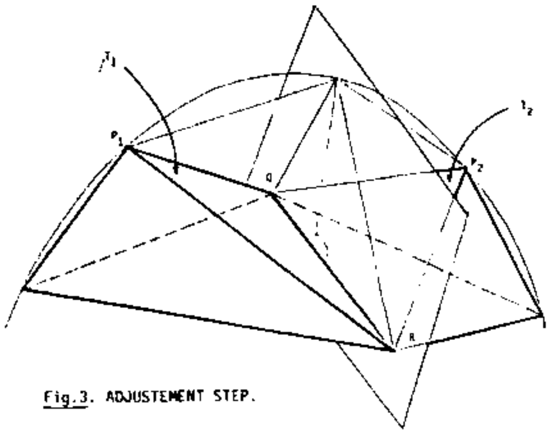

Fig.3. ADJUSTEMENT STEP.

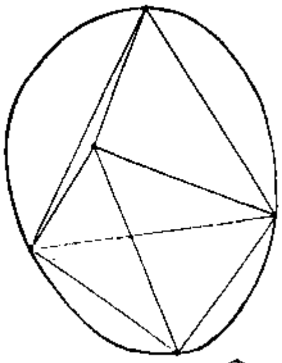
Fig.4. The initial GF.

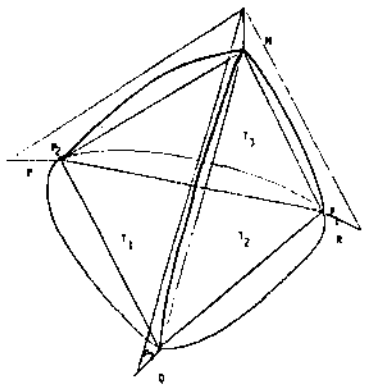

Fig.5. Finding the paths in the ALGRA algorithm.

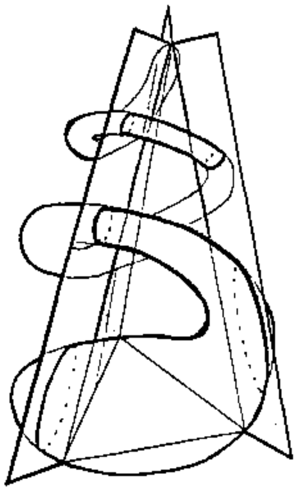
Fig.6. Disconnected ST's obtained by ALGLI.

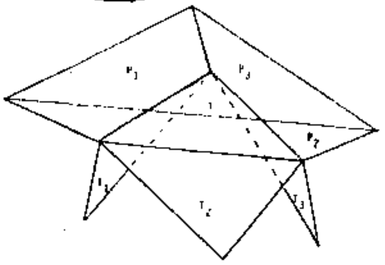

Fig.7. A triangle and the associated bissector planes.

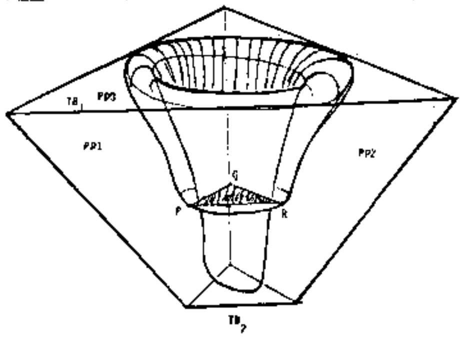

Fig.8. The prism associated to a complex ST.

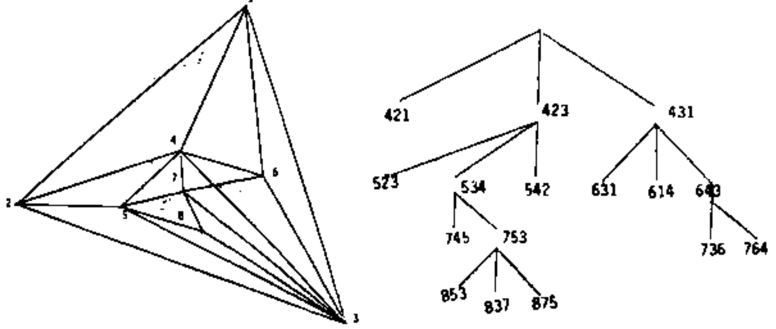

Fig.9. A node, its descendants, and the associated tree.

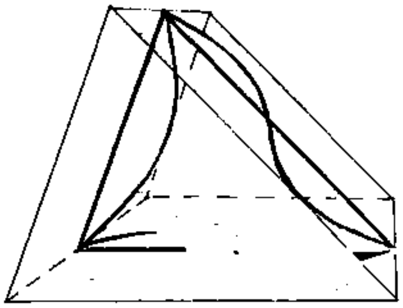
Fig.10. A regular node, the intersection of ST and P1,P2,P3 is a closed curve.

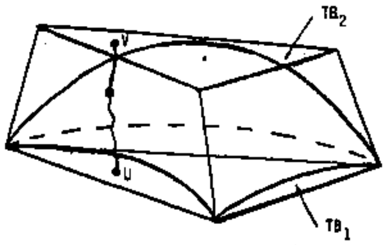

Fig.11. Curve-Surface intersection.

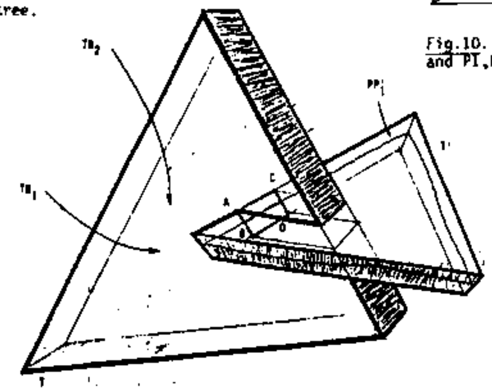

Fig.12. The CLEAR intersection of two prisms.

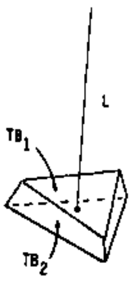

Fig.13. The infinite half line associated to a prism.