INPUT AND USAGE OF SPATIAL INFORMATION BASED ON THE OCTREE REPRESENTATION

Hiroshi NOBORIO

Department of Mechanical Engineering
Faculty of Engineering Science
Osaka University

1987

INPUT AND USAGE OF SPATIAL INFORMATION BASED ON THE OCTREE REPRESENTATION

A DISSERTATION
SUBMITTED TO OSAKA UNIVERSITY
FOR THE DEGREE OF
DOCTOR OF ENGINEERING

by
Hiroshi NOBORIO

January 1987

ABSTRACT

This dissertation is concerned with solid models focusing its attention on input and retrieval of geographic and spatial information in a real space. Among such solid models, the octree proposed in computer graphics is the most adequate for this purpose. It has a hierarchical structure in positioning and an inherent feature that spatial information resources on a selected portion of the space can be efficiently enrolled and retrieved. Hence, the goal of this research is to construct the octree representation for solid objects in computer memory and to use the representation in robotic applications.

Construction of the octree is treated in the following two cases; 1) shape of the object is already defined by one solid model, 2) the shape is not known at all. Then, in each case, an efficient algorithm that constructs the octree for the object is proposed.

Next, as an application of the octree representation to robotics, a fast interference checking algorithm between the octree and the B-reps. is considered. This algorithm is useful in graphics simulator for off-line robot teaching.

In Chapter 2, some solid models investigated in computer graphics are explained and discussed. In Chapter 3, a conversion algorithm from the B-reps. to the octree is described. Since the B-reps. (polyhedron) is the most general solid model, this algorithm can be applied to many application fields. In Chapter 4, a construction algorithm of the octree approximating a real solid object by using multiple views is discussed. The algorithm is useful in case that the object is unknown. This is regarded as one application of the above conversion algorithm for computer vision. Chapter 5 presents a fast and general interference checking algorithm. This algorithm is also regarded as one application of the conversion algorithm to robotics.

ACKNOWLEDGMENT

I am very grateful to the supervisor Prof. S.Arimoto for his persistent encouragement, and to Prof. S.Tsuji and Prof. Y.Seguchi for their invaluable advice to improve the thesis.

I would also like to express my thanks to Associate Prof. F.Miyazaki, Dr.S.Kawamura, Dr.S.Takegaki, and Mr.T.Ohi for their invaluable contribution to my studying the robotics.

I also wish to thank Associate Prof. T.Hashimoto and Associate Prof. N.Ishimura for valuable discussions on tree searching.

This work presented here is an outcome of collaboration with graduate students; Mr. A.Noda and Mr. S.Fukuda. Without their efforts made in experiments, the development would not have been accomplished.

Finally, I would like to thank my family for their unconditional support, without which I could not have accomplished this work.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 The beginning

In many fields such as computer graphics, computer vision, and computer-aided design, much attention has been focused on the representation problem of three dimensional objects, which is called the computer-based three-dimensional geometrical modeling. The main proposed models are Simple Sweep, Boundary Representation, and CSG [1]-[3]. Whereas these models are mainly investigated for retrieving the topological information of an solid object, they are not suitable for preserving the geographical information of one whole space including many objects.

The octree representation proposed by Jackins and Tanimoto [4] has such a structure as underpinning for Warnock's algorithm [5], planning [6], and pyramid data structure [7]. This tree is one of the hierarchical representations whose structures are described in various ways [8]-[14]. It has been already used for investigating the ray-tracing [15]-[17] in computer graphics and the pattern recognition [18]-[20] in computer vision. This is based on the property that spatial information resources on a selected portion of a three-dimensional object can be efficiently retrieved because of the convenient hierarchical structure for top-down, bottom-up, and split-and-merge operations. In addition to this property, since the octree is a solid model that integrates all objects in one whole space, it can deal with some objects in the space simultaneously. These properties meet several requirements in robotics and contribute to the decisive reason for adopting the octree as a model of robot's environment.

Then, in this dissertation, some effective algorithms concerning the octree are devised by using those properties and applied to some important problem in robotics.

The first one is a conversion algorithm from the B-reps., which is the most general solid model, to the octree. Similar conversion algorithms from other models, for example, CSG, and Simple Sweep, etc., to the octree have been proposed in several researches [21],[22]. Thus, if shape of a solid object has been defined by a set of solid models including the B-reps., the corresponding octree can be surely constructed.

The second one is to construct an approximating region for some real object with the octree using multiple two-dimensional images. This algorithm is useful in case that shape of the object is unknown. Several algorithms for this purpose have been studied in many fields (Computer Vision, Computer Graphics, and Robotics) and many important technical methods (Stereo vision, Shape from shading, etc.) have been presented [23]-[27]. However, some methods can not build an accurate solid model for the object if degraded images are only obtained, and others require enormous calculation time. To solve those problems, we propose a fast and general algorithm based upon a so-called volume intersection method and a hierarchical property of the octree.

The proposed two algorithms are considered to be fundamental in a sense that they construct an environment model necessary to accomplish various requirements for decision making in robot motion planning. For that reason, an interference check algorithm between a robot model and its environment model in graphics simulator (off-line teaching) is treated here as one of such requirements.

On-line robot teaching which is mostly used in the field of robot teaching is useful if a robot is of a simple shape and its environment is a mere space. However, this teaching needs much time if the robot structure is complicated and its environment is cluttered, and thereby a production line in factory must be stopped for a long period. In that case, off-line teaching with graphics simulator would be used. Then, performance of the simulator depends on the efficiency of the interference check and therefore the fast interference check algorithm between the robot model with the B-reps. and the environment model with the octree is proposed. Owing to the hierarchical structure of the octree, the algorithm can select regions such that the robot model probably intersects an obstacle in its environment model by adaptively changing the resolution level of the octree, and therefore discuss the interference for only some parts of obstacles around the robot model. Consequently, this algorithm does not depend on the complexity of shape of the robot and its environment and therefore is even fast in the complex robot and cluttered environment.

Not only the robot simulator but also a so-called intelligent robot requires an environment model. For example, automatic decision of the robot motion avoiding obstacles is investigated in this research field and is called "mover's problem" or "findpath problem". As compared with some data structures which have been proposed for representing an environment model in this problem, the octree is promising enough for such a data structure in the following reason: Due to indexing a three-dimensional space hierarchically, the octree is capable of determining whether a part of obstacles exists or not in a region easily. Hence, it can fast search for a free region in path-planning of the robot. In addition to this, the octree is easily modified with respect to a movement of the obstacle [28] [69]. Thus, the octree with capability of representing the spatial information is suitable for a model of robot's environment.

1.2 Dissertation outline

The rest of the dissertation is organized into six chapters:

Chapter 2 discusses some solid models that have been mainly proposed in computer graphics. Especially in that chapter, the octree representation that is adopted as an environment model is explained in detail.

Chapter 3 proposes a conversion algorithm from the B-reps. to the octree. In contraposition to similar other algorithms, the proposed algorithm is fast because it is a one-pass method in the sense that it can deal with all nodes (cubes) of the octree simultaneously, whenever they are inside, outside, or intersecting for the B-reps. Furthermore, since it has the property that building a subtree of the octree in some region is independent of building subtrees of it in other regions, it can be used in many application fields.

Chapter 4 is concerned with an algorithm for constructing a real object in memory of the computer by using the octree as one application of the above conversion algorithm in computer vision field. This algorithm should be used for such a case that shape of the real object is not known at all.

Chapter 5 is concerned with an interference check algorithm in graphics simulator as one application of the conversion algorithm in robotics. The efficiency of off-line robot teaching with the simulator depend on the efficiency of the algorithm. This algorithm can run fast even in a cluttered environment and a complicated robot since it can fast select only regions that intersect both the robot model and obstacles in the environment with the aid of the hierarchical structure of the octree.

Some concluding remarks and further developments of the research are to be given in Chapter 6.

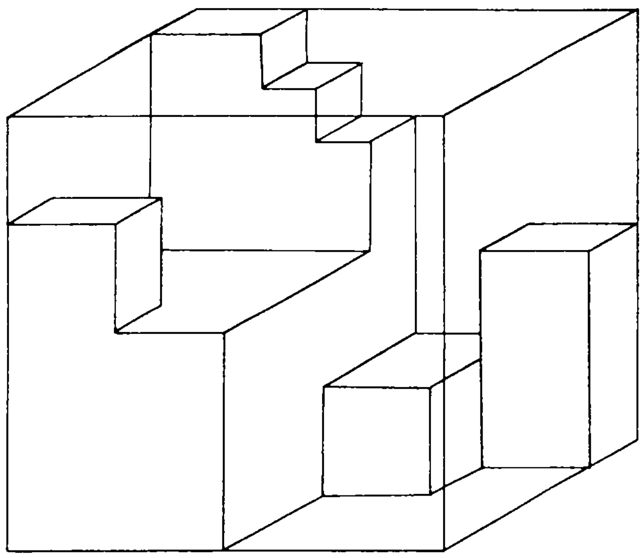
CHAPTER 2

MODEL DEFINITIONS

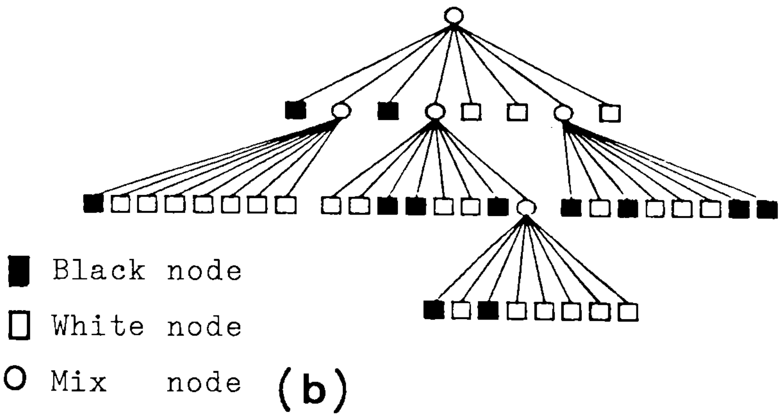
2.1 Introduction

According to the literature on solid modeling, some methods for representing solid objects have been proposed for many purposes [1]-[3]. For constructing a solid model, Simple Sweep is mostly used. To evaluate integral properties of a solid, CSG (Constructive Solid Geometry) is a natural representation. For the purpose of display of a model, B-reps. (Boundary representation) is extensively used.

Octree representation [4],[14],[28],[29] proposed recently is a natural hierarchical solid model that can change the resolution in positioning adaptively in reference to a region in the world space and retrieve efficiently the geometric information. This adaptability is useful in ray-tracing and pattern recognition, and many application algorithms in the fields have been proposed [15]-[20]. Also, the greater part [30]-[35] of valuable algorithms based on the quadtree whose hierarchical structure in two dimension is equivalent to the structure of the octree in three dimension is applicable to the octree. Further, the view-transformation algorithm [36],[37] for displaying on a screen 3-D pictures represented by the octree and other application



(a)



(b)

(1(10000000)1(0011001(10100000))00(10100011)0)

(c)

Fig.1. An object (a) and its corresponding octree (b) and DF-representation (c).

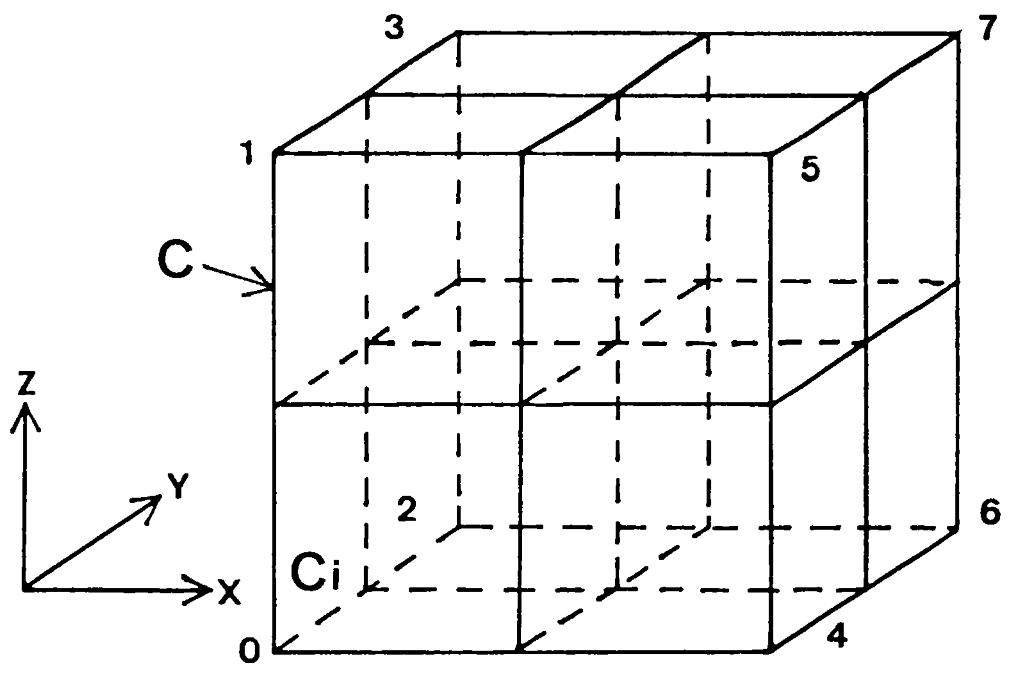


Fig.2. The order of octants.

algorithms [38]-[40] using the octree are presented.

2.2 Octree representation

Octree is a solid model that hierarchically represents a three dimensional space as shown in Fig.1. In octree, a region (cube C) that intersects a given object is represented by a mix node and a region (cube C) inside [outside] the object is represented by a black [white] node. A region (cube C) corresponds to a mix node is decomposed into eight subregions (subcubes C_i) and its subregions are represented by eight child nodes of the mix node. The order of octants is shown in Fig.2.

The decomposition is recursively processed by the depth-first search until the finest resolution level (n) of the octree. At the finest resolution level, all mix nodes are converted to black nodes. And if these are complete eight black child nodes who have the same parent node, then the set of these child nodes is merged into the parent black node.

Several advantages of this octree representation are summarized below:

(1) Calculation of mass properties (volume, weight, center of gravity, and moment, etc.) is easy.

(2) Due to the hierarchical property, the search of

geometric information can be carried out rapidly (ray tracing, etc.).

(3) Since most algorithms that use the octree possess generally a recursive property, hardware implementation of those is simple and easy [41].

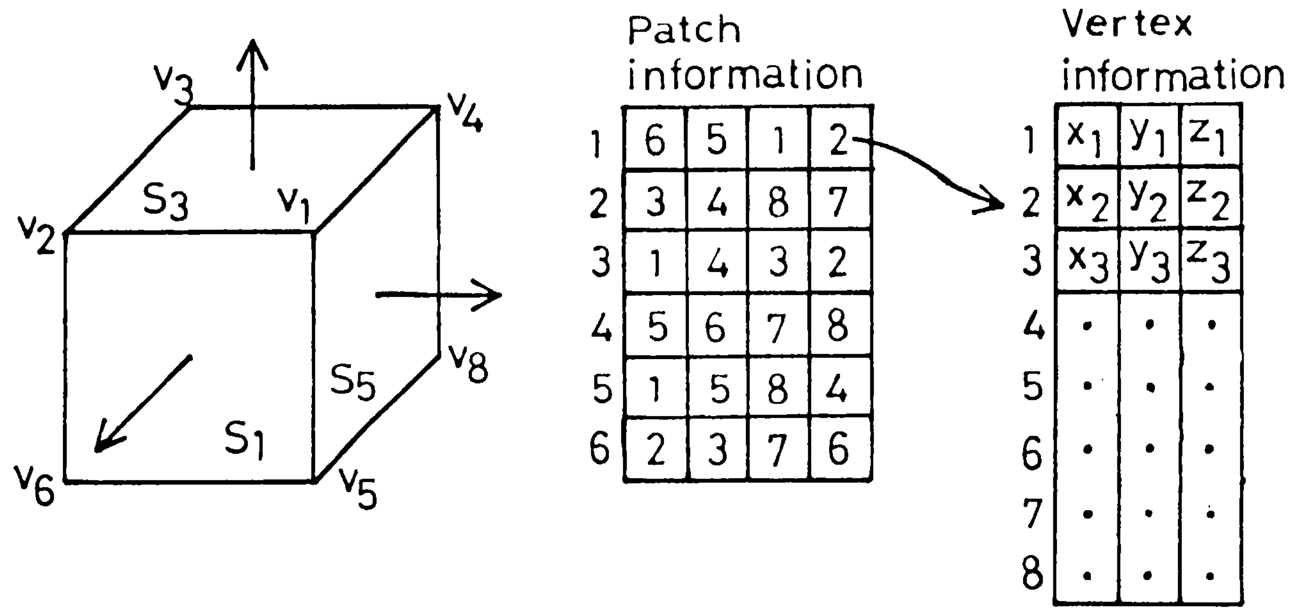
(4) Set operations (intersection, complement, and union etc.) can be processed fast.

2.3 DF-representation

A DF (Depth-First) representation [42],[43] is a linear encoding of the octree as shown in Fig.1(c) and is made by pre-order scanning on the octree. Here, symbol '(' indicates a mix node and yields an increment of one tree level, and symbol ')' yields a decrement of one tree level. Symbols '0' and '1' imply a black and white node, respectively. The DF-representation is useful in condensing the storage of the octree.

2.4 Boundary representation (B-reps.)

B-reps. is the most general solid model that is represented by a set of patches. In this dissertation, a patch of the B-reps. is represented by a sequence of vertexes which are clockwise ordered



Patch information

1	6	5	1	2
2	3	4	8	7
3	1	4	3	2
4	5	6	7	8
5	1	5	8	4
6	2	3	7	6

Vertex information

1	x_1	y_1	z_1
2	x_2	y_2	z_2
3	x_3	y_3	z_3
4	•	•	•
5	•	•	•
6	•	•	•
7	•	•	•
8	•	•	•

Fig.3. Boundary representation (B-reps.).

for the normal vector of the plane that includes the patch as shown in Fig.3.

Not only some algorithms to display the solid object but also several algorithms [44],[45] for the set operation using this model are presented.

2.5 Simple sweeps

Sweep representation [2],[21],[46] is represented as an occupying volume of an region guided by a given motion. In general, the motion is defined as a combination of translation and rotation. Contrary to this, the sweep representation is called "Simple sweep" if and only if the motion is defined as translation or rotation illustrated in Fig.4. Using this simple sweep, one solid object is easily constructed in computer because only decision of both shape of the region and one motion is necessary and sufficient for the construction. Further, the integral properties of some simple sweep may be computed by exploiting dimensional separability to convert a triple (volume) integral into a double (surface) integral over a planar set [21]. Needless to say, this model is not capable of the construction of one complicated object.

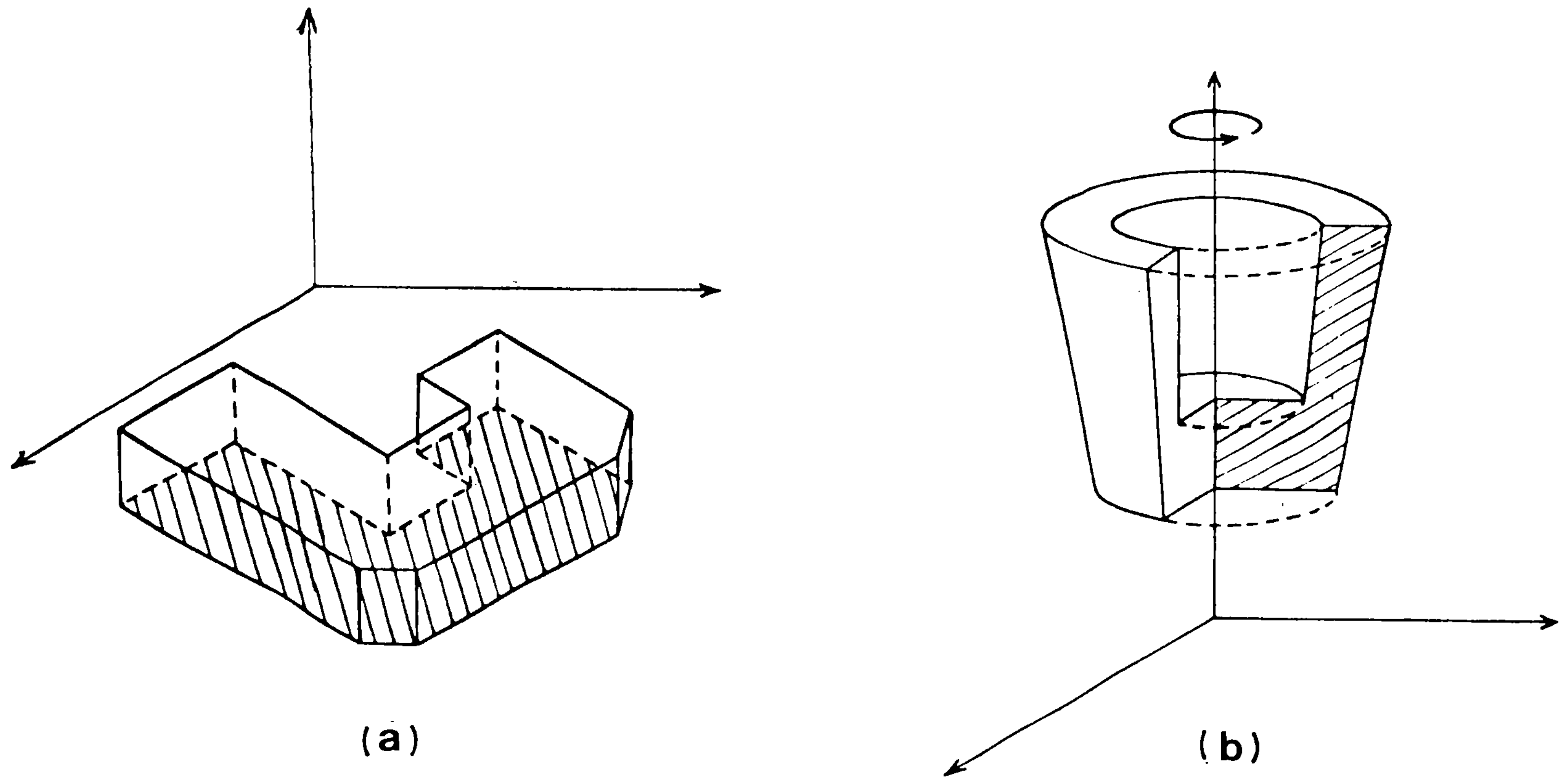


Fig.4. Translation (a) and rotation (b) sweeps.

2.6 Constructive solid geometry (CSG)

Constructive solid geometry (CSG) [2],[3],[47] is based on the notion of "addition" and "subtraction" of several primitive solid objects illustrated in Fig.5. The natural method for evaluating integral properties of solids with CSG is to apply a "divide-and-conquer" strategy to each primitive solid object [21]. Furthermore, many algorithms for reporting and counting geometric intersection between two different CSG models are proposed [48]-[50]. This CSG model is also unfit for the representing a complicated object because the computational load for constructing the representation is enormous with increasing the number of primitive objects.

The first two models are suitable for retrieving the geographical information for a set of objects in one whole space. On the contrary, the other models are suitable for doing the topological information of a three-dimensional object. In robotics, a data structure for retrieving effectively the geographical information of a Cartesian space including many obstacles is especially needed in an environment model that represents the whole space. As a result, the former models are useful for many applications to

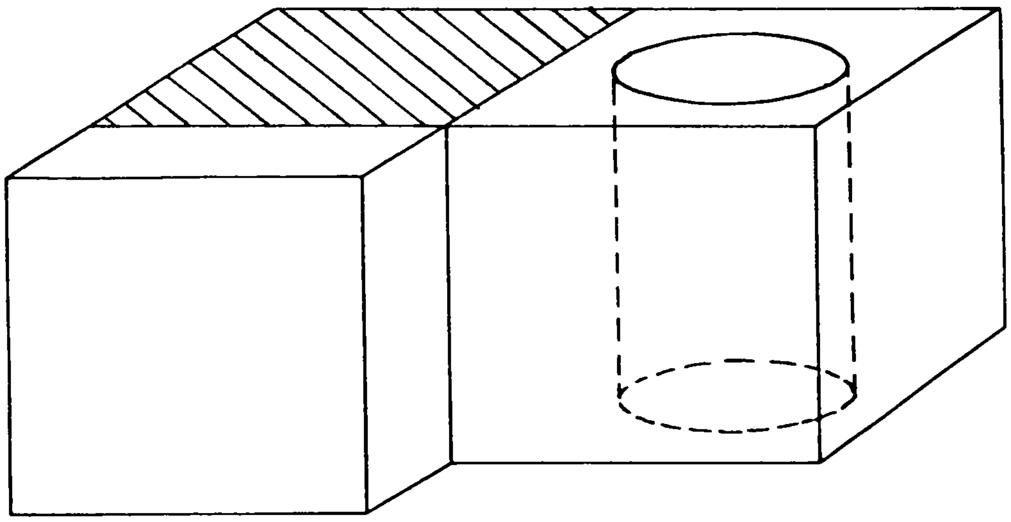


Fig.5. Constructive solid geometry (CSG).

robotics. Therefore, we claim that the octree is adequate as robot's environment model in order to develop important application algorithms based on the octree.

CHAPTER 3
CONVERSION ALGORITHM FROM THE BOUNDARY REPRESENTATION TO THE OCTREE AND ITS COMPLEXITY

3.1 Introduction

To use the octree, a conversion from other solid models proposed in computer graphics (Simple Sweep, CSG, and B-reps., etc.) to the octree is necessary. For this purpose, Lee and Requicha [22] presented an algorithm that converts efficiently from the CSG scheme into the octree. Franklin and Akman [51] described a conversion algorithm from a set of parallelepipeds to the octree. However, the algorithm is not general because of the necessity of elaborate calculation of a set of parallelepipeds, which is an approximating region for a given object. Yau and Shihari [52] gave an algorithm for constructing a tree of d-dimensional binary images from trees of its (d-1)-dimensional cross section. Since trees of a two-dimensional cross section for a given object are obtained easily, this algorithm is general. On the contrary, it has a large computational complexity that is proportional to 8^n for parameter n (the finest resolution level of the octree). Tamminen and Samet [53],[54] proposed a conversion algorithm from the B-reps. to a linear representation of a binary voxel tree (bintree), which is like an octree. Since B-reps. is a primary scheme used for representing a solid object, the algorithm is general. However, there are some defective points in the algorithm. First, the algorithm is inefficient in calculation time because it is based on a two-pass approach: It makes nodes in the bintree, which correspond to the region intersecting patches of the B-reps, and then it makes other nodes in the bintree, which correspond to the internal and external regions for the B-reps., using the connected components labeling technique. In this connection, the construction of nodes that represent the region intersecting patches of the B-reps. is not complete. Second, due to the two-pass approach, the algorithm can not execute any partial conversion from a part of the B-reps. to the bintree.

Thus, we propose a conversion algorithm from the B-reps. to the DF-representation that is a linear representation of the octree. First, capability of building nodes of the octree, which correspond to the region intersecting patches of the B-reps., is assured in Theorem 1 and Theorem 2 in this chapter. Second, by restricting the B-reps. to be of convex shape, the proposed algorithm can deal with all nodes of the octree simultaneously, whenever they are inside, outside, or intersecting for the B-reps. Hence, the algorithm is composed by an one-pass approach. In other words, it follows naturally from this property that building a subtree of the octree in some region is independent of building subtrees of it in other regions. Using this fact, the algorithm can convert a part of the B-reps. to the octree and therefore it can be effectively used in many applications in the fields of robotics and computer vision. Finally, the computational complexity of the algorithm and the storage of the octree with respect to S (surface area of the B-reps.), N (number of patches of the B-reps.) and n (the finest resolution level of the octree) are analyzed and evaluated, which is neglected in the literature on previous research works.

The effectiveness of the algorithm is demonstrated by implementing it in C language on the VAX 11/750 under UNIX. In an illustrative example for a set of B-reps. data given by approximating a solid sphere, it is shown that the actual conversion time from the B-reps. to the DF-representation and the number of nodes of the octree are coincident with the estimated time-complexity and storage of the octree, respectively.

Listing 1. Notations and basic functions

```
r ≡ cube corresponding to a root node (the world space)
     of an octree.
C, Ci ≡ cube corresponding to a node, and cube corresponding
          to the i-th child node of that node.
n    ≡ finest resolution level of the octree.
n'    ≡ current resolution level of the octree.
CHILD(C,i)    ≡i-th subcube Ci of cube C.
CENTER(C ,i) ≡ X, Y, or Z-axis coordinate of the center
                point of cube C. (i = 1, 2 or 3)
UPPER(obj,i)  ≡ upper coordinate for the parallelepiped
                enclosing an object (B-reps.) in X, Y, or
                Z-axis (i = 1, 2 or 3).
LOWER(obj,i)  ≡ lower coordinate for the parallelepiped
                enclosing an object (B-reps.) in X, Y, or
                Z-axis (i = 1, 2 or 3).
MARKED(map,i)≡ this function is 'True' if the state of the
                i-th subcube Ci in map is'intersection', else
                is 'False'.
MARK(i)    ≡Divide a cube C into two volumes by the plane
             which is vertical to the X, Y, or Z-axis (i= ±1,
             ±2, or ±3) and proceed with one of the following
             two processes:
(i > 0) This function gives information that states of four
         cubes in the volume of positive direction with
         respect to the axis are'intersection', and other
         states are 'non-intersection' .
(i < 0) This function gives information that states of four
         cubes in the volume of negative direction are
         'intersection' and other states are 'non-intersection'.
Boundary map (map)   ≡ This map keeps current all states
                        ( 'intersection' or 'non-intersection' ) of
                        eight subcubes.
```

3.2 Conversion algorithm

In this section, a conversion algorithm from a given B-reps. to a DF-representation is proposed and explained in detail. The basic process in the algorithm is that eight subcubes of an 'intersection' cube (a region including a part of the B-reps.) are classified into 'inside', 'outside' or 'intersection' for the B-reps. Here, we should note that the 'inside', 'outside', and 'intersection' subcubes are considered to be the 'black', 'white', and 'mix' nodes, respectively, and therefore are converted into the symbol '0', '1', a pair of the symbol '(' and ')', successively. This classification is managed by the following three procedures (MESH, PLURAL and SINGLE procedures). In accordance with a degree of intersection when the B-reps. intersects a cube, one of the three procedures is selected to deal with the cube. The classification is carried out using only patch information of the B-reps. in the cube. Thus, by adopting this basic process recursively, a subtree of the octree (DF-representation), which corresponds to one cube (region), can be completely built using only partial information of the B-reps. in the cube.

The basic process is recursively used for each

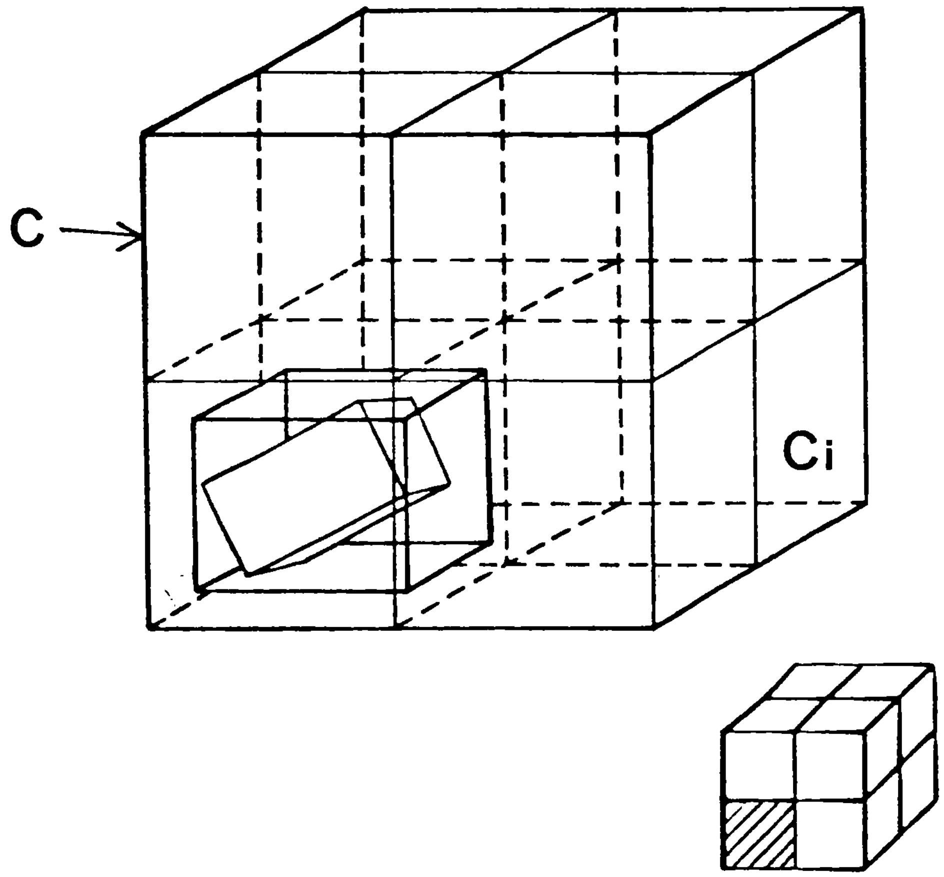


Fig.6. MESH procedure

'intersection' cube by the depth-first search on the octree and therefore its DF-representation is naturally made. Here, if the initial 'intersection' cube r is the world space, then proposed algorithm constructs the octree for the whole B-reps.

3.2.1 MESH Procedure

This procedure deals with a cube (region) that encloses completely a given B-reps (See Fig.6). The procedure determines only one subcube enclosing the B-reps. at each level of the octree and considers it the 'intersection' subcube for the given B-reps. Other seven subcubes at the level are 'outside' for it. If the subcube enclosing the B-reps. can be uniquely determined, this procedure deals with the subcube as an input cube anew. Otherwise, the procedure is stopped and some subcubes intersecting a part of the B-reps. are processed by the following PLURAL procedure.

This procedure concretely consists of the following two steps: The first step is to define the minimum parallelepiped enclosing the B-reps. The second step is to divide the 'intersection' cube into eight subcubes and to search a subcube intersecting the parallelepiped. The subcube is specified as the

Listing 2. MESH procedure.

```
procedure MESH(C,n')
begin
   map  ← intersection;
   for i  ←1 until 3 do
      if UPPER(obj, i) < CENTER(C, i) then
         map   ← map ∩ MARK(-i);
      else if LOWER(obj, i) > CENTER(C, i) then
         map  ← map ∩ MARK(+i);
      if COUNTER_MARK(map) > 1 then
         PLURAL(C̄, list, n');
      else if n'+1 ≥ n then

      WRITE('(');
      begin
         for i  ←0 until 7 do
            if MARKED(map, i) then
               WRITE('1');
            else
               WRITE('0');
         end;
      else
      begin
         for i  ←0 until 7 do
           if MARKED(map,i) then
              MESH(CHILD(C, i), n'+1);
           else
               WRITE('0');
      end;
      WRITE(')');
end;

COUNT_MARK(map)  ≡count the number of states of 'intersection'
                   in map.
```

'intersection' one and other subcubes are specified as the 'outside' one.

3.2.2 PLURAL Procedure

This procedure deals with a cube (region) that intersects some patches of the B-reps. First, by assigning each patch in the cube to some of eight subcubes, the following three routines (BBOX, PLANE, and PROJECTION routines) classify each subcube into 'intersection' (a subregion including a part of the B-reps.) or 'non-intersection' (a subregion which does not include any part of the B-reps.) (Fig.7). Next, the JUDGE routine classifies each 'non-intersection' subcube into being 'inside' or 'outside' for the B-reps. The first two routines check necessary conditions so that a patch intersects a subcube, and the third routine checks a sufficient condition.

1) Bounding box routine (BBOX): This routine selects subcubes that the minimum parallelepiped enclosing a patch P intersects. Such subcubes are considered to be a candidate for the 'intersection' subcube.

Now, this routine is concretely described as

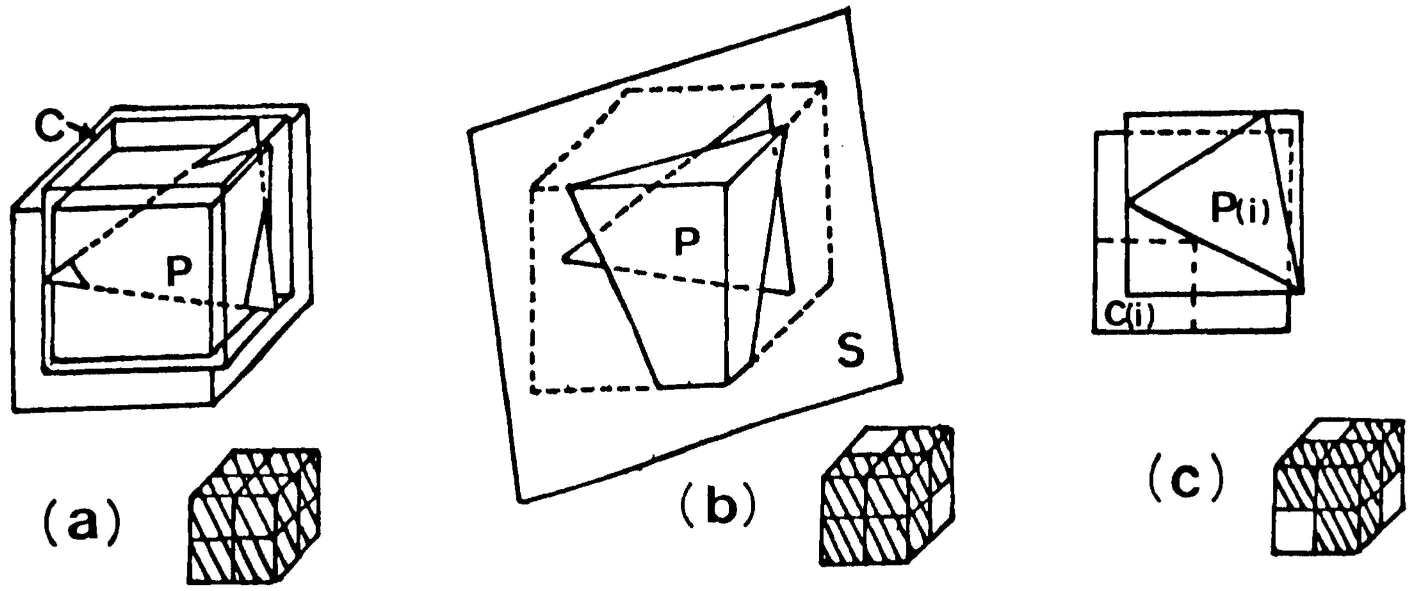


Fig.7. PLURAL procedure (a) BBOX routine (b) PLANE routine
(c) PROCEDURE routine.

Listing 3. PLURAL procedure.

```
procedure PLURAL(C , list, n')
begin
    for i  ← 1 until LENGTH(list) do
       begin
          P  ← list[i];
          map   ← BBOX(P,C);
          map ←  map ∩ PLANE(P,C);
          PROJECT(P,C,map);
          ENTRY(P,map);
       end;

    WRITE('(');
    if n'+1  ≥ n then
      begin
        for i ← 0 until 7 do
            if LENGTH(lis[i]) ≠ 0 then
            WRITE('1');
            JUDGE(map);
      end;
    else
      begin
         for i  ← 0 until 7 do
            if LENGTH(lis[i]) = 1 then
               SINGLE(CHILD(C,i),lis[i],n'+1);
            else if LENGTH(lis[i]) > 0 then
               PLURAL(CHILD(C,i),lis[i],n'+1);
             JUDGE (map);
      end;
      WRITE(')');
end;
```

list, lis[i]	≡	patch lists correspond to cube C and subcube C_i, respectively.
LENGTH(list)	≡	length of the list (number of patches).
ENTRY(P,map)	≡	append a patch P to the lis[i] of subcube C_i whose state is 'intersection' in map.

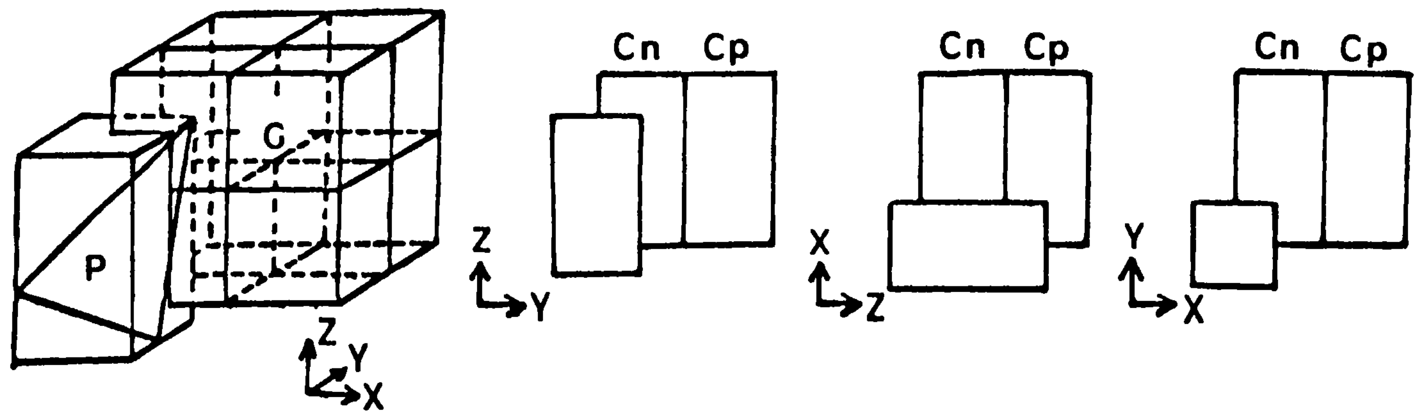


Fig.8. Process of BBOX routine.

Table 1. Conjunction process in BBOX routine

Plane	Region									
	Cn	Cp	C0	C1	C2	C3	C4	C5	C6	C7
ZY-Plane	1	0	1	1	0	0	1	1	0	0
ZX-Plane	1	1	1	1	1	1	1	1	1	1
XY-Plane	1	0	1	1	1	1	0	0	0	0
Conjunction	-	-	1	1	0	0	0	0	0	0

1: Intersection
0: Non-intersection

Listing 4. BBOX routine.

```
procedure BBOX(P,C)
begin
   map  ← intersection;
   for i  ←1 until 3 do
      if UPPER(P, i) < CENTER(C, i) then
         map ←  map  ∩ MARK(-i);
      else if LOWER(P, i) > CENTER(C, i) then
         map ←  map  ∩ MARK(+i);
      return map;
end;
```

follows: It divides a cube C into two regions (Cn, Cp) - each region includes four subcubes C_i - by the plane that is vertical to each axis (X, Y, or Z axis) and intersects the center point G of the cube, and checks whether the above parallelepiped intersects each region or not (See Fig.8). By processing these three results for the X, Y and Z axes as shown in Table 1, the routine can determine some subcubes intersecting the parallelepiped.

2) PLANE routine: This routine selects subcubes that intersect the plane including a patch P.

Denote the normal vector of the plane S including a patch P by $\vec{n}=(n_x,n_y,n_z)$. For some given point $P_0(\vec{p_0})$ on the plane S and any point P: $\vec{p}=(p_x,p_y,p_z)$, define

$$d(S,P) = \vec{n} \cdot (\vec{p} - \vec{p_0})$$
$$= n_x*p_x + n_y*p_y + n_z*p_z - d. \quad (d = \vec{n} \cdot \vec{p_0})$$

This function implies a distance from the plane S to a point P. When the world space is divided into two volumes by the plane, one volume that includes the given B-reps is defined to be inside and another volume that does not include it is defined to be outside. Then, a point P is inside [outside] the plane if the function defined above is negative [positive].

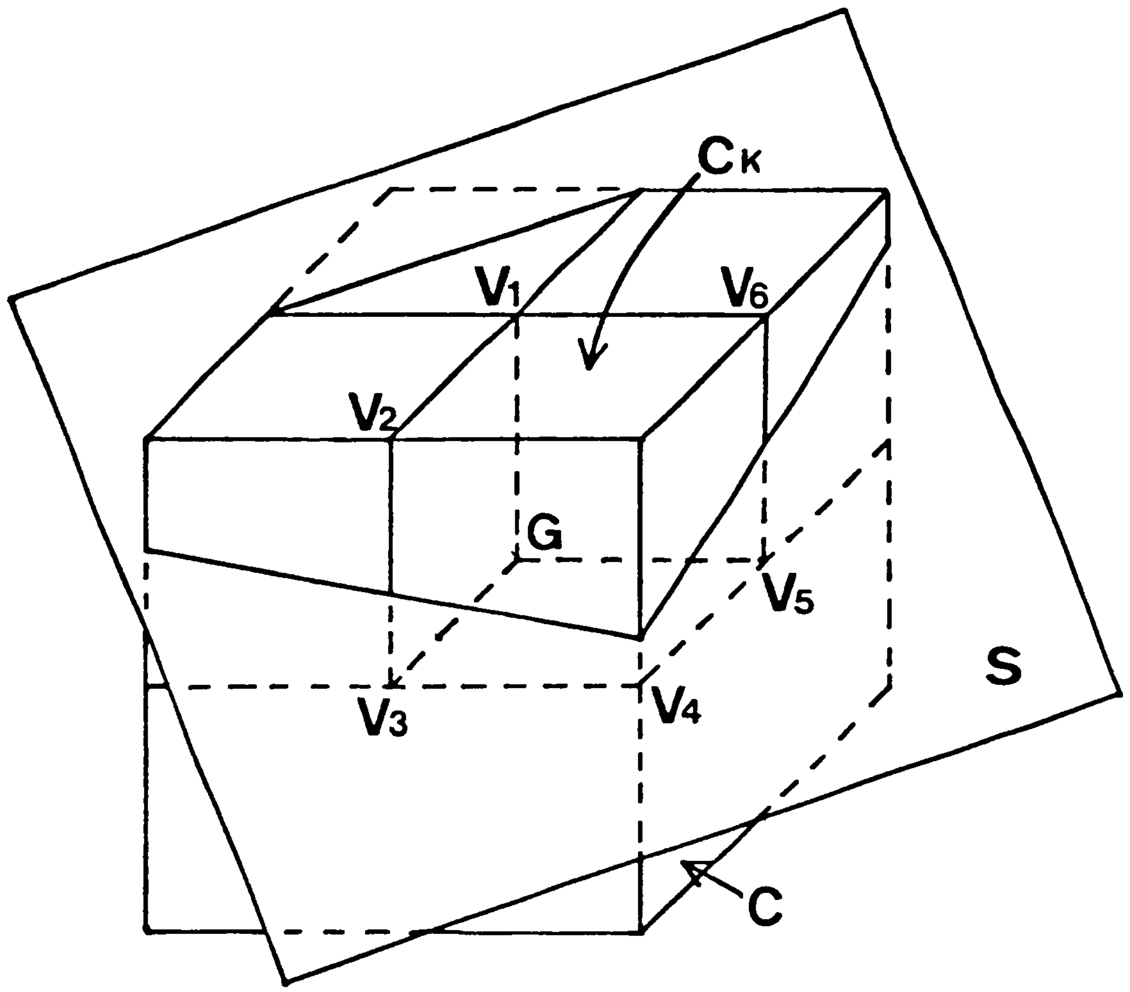


Fig.9. Intersection between plane S and subcube.

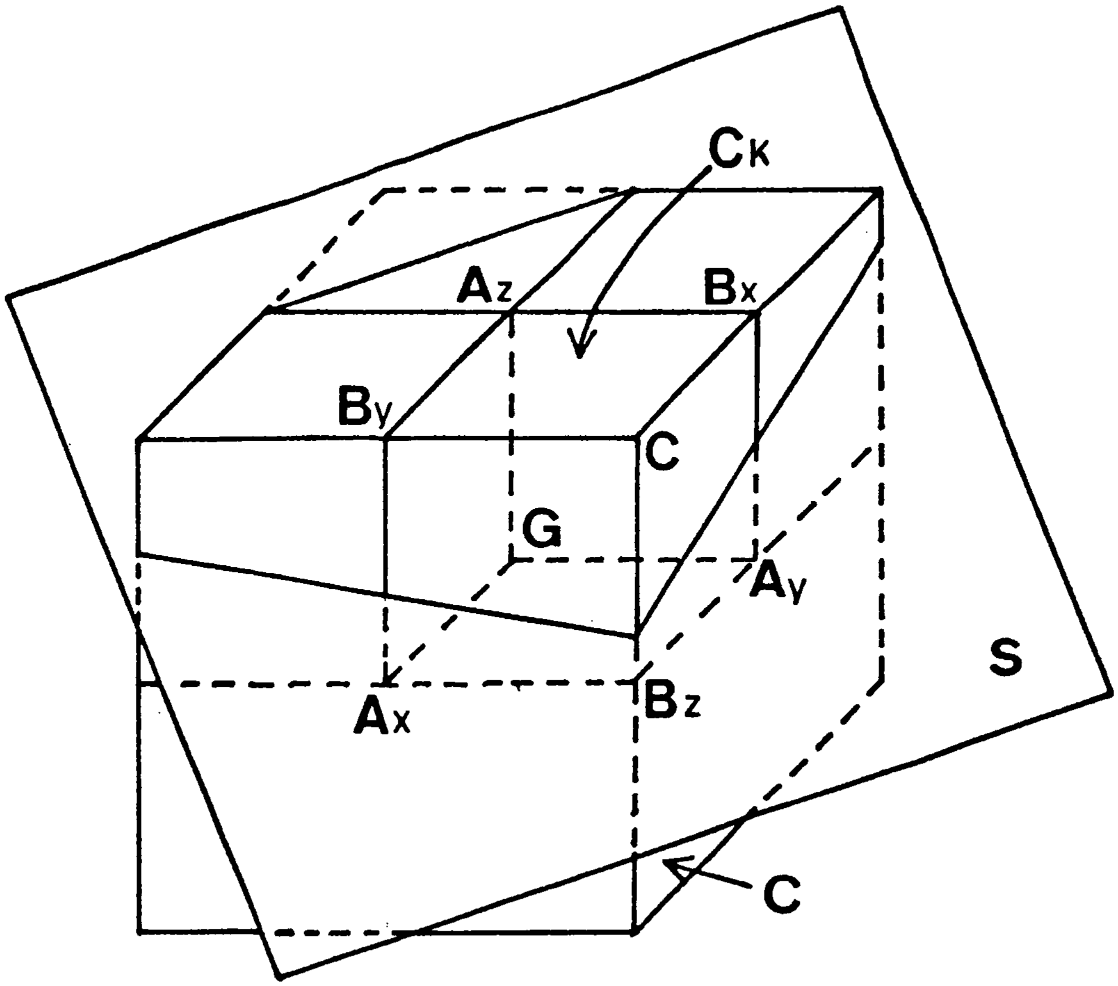


Fig.10. Classification of seven vertexes into three groups A, B, and C, and relation between the groups and plane S.

The vector $\vec{i} = (r_0, r_1, r_2)$ is defined such that $(r_0 r_1 r_2)_2$ is the binary representation of a decimal number i. The number i represents the position of a subcube as shown in Fig.2.

We now explain an efficient way for finding some subcubes that intersect the plane S. First, calculate $d(S, G) = d_g$ for the center point G of cube C and proceed with one of the following processes:

(i) Function d(S, G) is negative [positive]

The point G is inside [outside] the plane S. In this case, the plane S intersects some subcubes which are adjacent to the subcube C_k maximizing [minimizing] $\vec{n} \cdot \vec{i}$. Now, if the function $d(S,v_j)$ is positive [negative] for some vertex v_j of the subcube C_k, then v_j is outside [inside] the plane S and therefore the plane S intersects a line Gv_j (Fig.9). Consequently, adjacent subcubes to the line are considered to be a candidate for the 'intersection' subcube. Here, an efficient method is used in calculation of the function d(S,vj). Seven vertexes that are all the vertexes of subcube C_k except for the vertex G are firstly classified into three groups A (vertexes Ax, Ay, and Az), B (vertexes Bx, By, and Bz) and C as shown in Fig.10. The value of d_g is already calculated. Therefore, once calculating $\Delta di = h * |n_i|$ (i=x,y and z), we obtain the value of each function $d(S,v_j)$ easily (given in Appendix A) by the following equations:

$$d(S, A_i) = d(S, G) \pm h * |n_i|$$
$$= d_g \pm \Delta d_i \ (i = x,\ y \text{ or } z).$$
$$d(S, B_i) = d_g \pm (\Delta d_j + \Delta d_k).$$
$$(i,j,k = x,y,z,\ i \neq j,\ i \neq k)$$
$$d(S, C) = d_g \pm (\Delta d_x + \Delta d_y + \Delta d_z).$$
(h: a side length of a subcube.)

Moreover, since the plane S intersects four subcubes adjacent to the line GA_z if it intersects the line while the functions for vertexes in the A group are calculated (See Fig.10), calculation of the function for only a vertex Bz is necessary and sufficient in the B group. This geometric property makes the algorithm run faster.

(ii) Function d(S,G) equals zero

A subcube minimizing [maximizing] $\vec{n} \cdot \vec{i}$ is inside [outside] the B-reps. Then, since the plane S intersects other six subcubes exactly, these six subcubes are considered to be a candidate for the 'intersection' subcube.

Listing 5. PLANE routine.

```
procedure PLANE(P,C)
begin
   $d_g$ ← d(P,CENTER(C));

  map ← outside;

   for i  ←1 until 3 do
     $\Delta$d[i]  ←h* |nv[i]|;

   if $d_g$ < 0 then
      for i ← 1 until 7 do
         if dg + SUM(i,$\Delta$d) > 0 then
            map ←  map $\cup$ MARK_ROUND(+i);
   else if $d_g$ > 0 then
      for i ← 1 until 7 do
         if $d_g$ - SUM(i,$\Delta$d) < 0 then
            map ← map $\cup$ MARK_ROUND(-i);
   else
      begin
         black   ← intersection;
         white   ← intersection;
         for i  ←1 until 3 do
            if nv[i] > 0 then
               begin
                  black ←  black $\cap$ MARK(+i);
                  white ←  white $\cap$ MARK(-i);
               end;
            else if nv[i] < 0 then
               begin
                  black ←  black $\cap$ MARK(-i);
                  white ←  white $\cap$ MARK(+i);
               end;
            map←~ (black $\cup$ white);
     end;
     return map;
end;
```

nv[i] ≡normal vector of a patch P where (i = X, Y, Z).
SUM(i,Δd) ≡ summation of all terms but the first term in the equations (See in Paragraph 3.2.2).
MARK_ROUND(i) ≡ information that states of adjacent subcubes to line Gvi are 'intersection'.

3) PROJECTION routine: The above two routines still leave some subcubes which do not intersect a patch P. This reason is that only a necessary condition for a subcube to intersect the patch is taken into consideration in the above two routines.

Thus, a sufficient condition is required, which selects a subcube exactly intersecting a patch P. This condition is obtained by the following property: The shadows P(i) and C(i) (i=X, Y, and Z) are defined as projections of the patch P and a subcube Ci respectively onto the planes D(i) which are vertical to the X, Y, and Z axes, respectively. The subcube Ci intersects exactly the patch P if and only if the shadow C(i) intersects the shadow P(i) in each of the three planes D(i) and it is regarded as the actual 'intersection' subcube. This property is obtained by the following Th.1 and Th.2.:

Defining notations (Fig.11)

R: intersecting region between a subcube Ci and a plane S. ($R = Ci \cap S$)

$R(i) \cup R$: the region on S, whose projection onto the plane D(i) is coincident with the shadow C(i) (i=X, Y or Z).

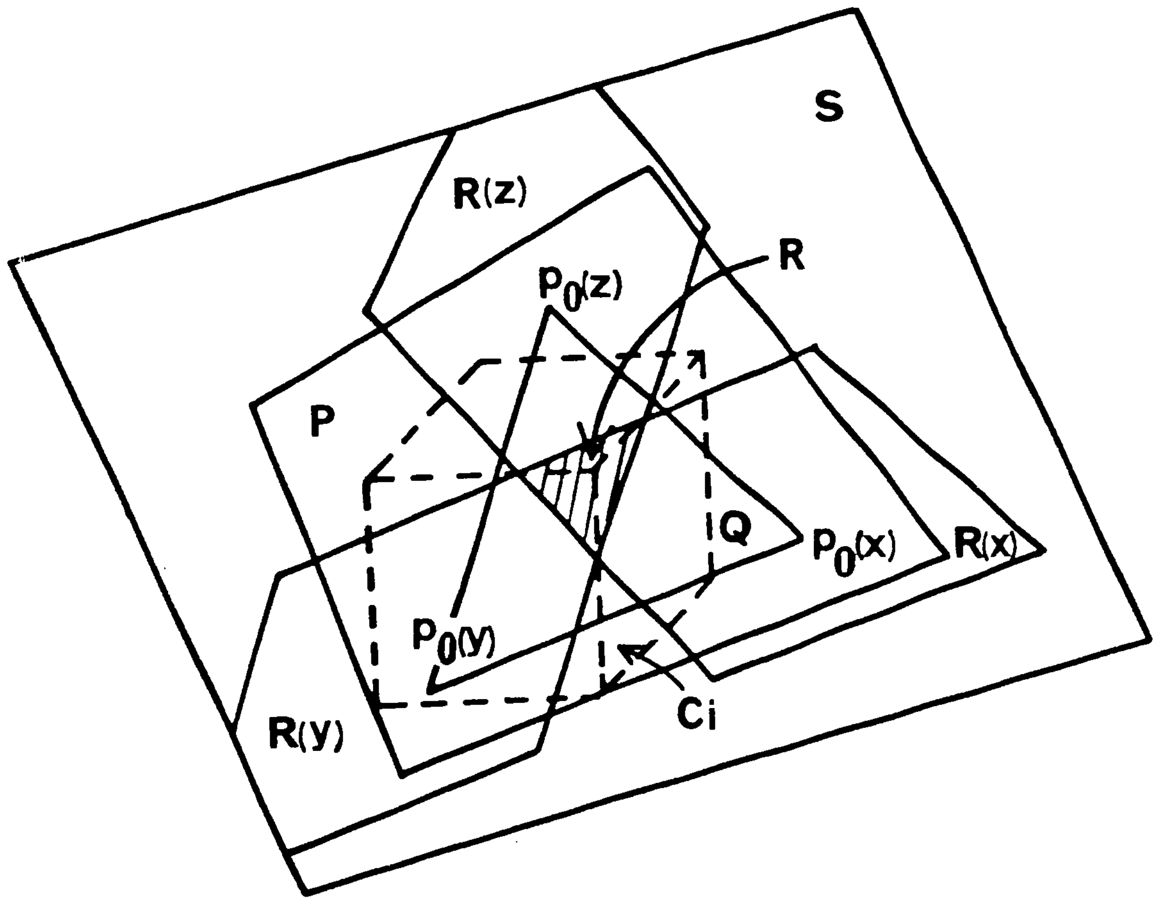


Fig.11. Projecting region of cube onto the plane S.

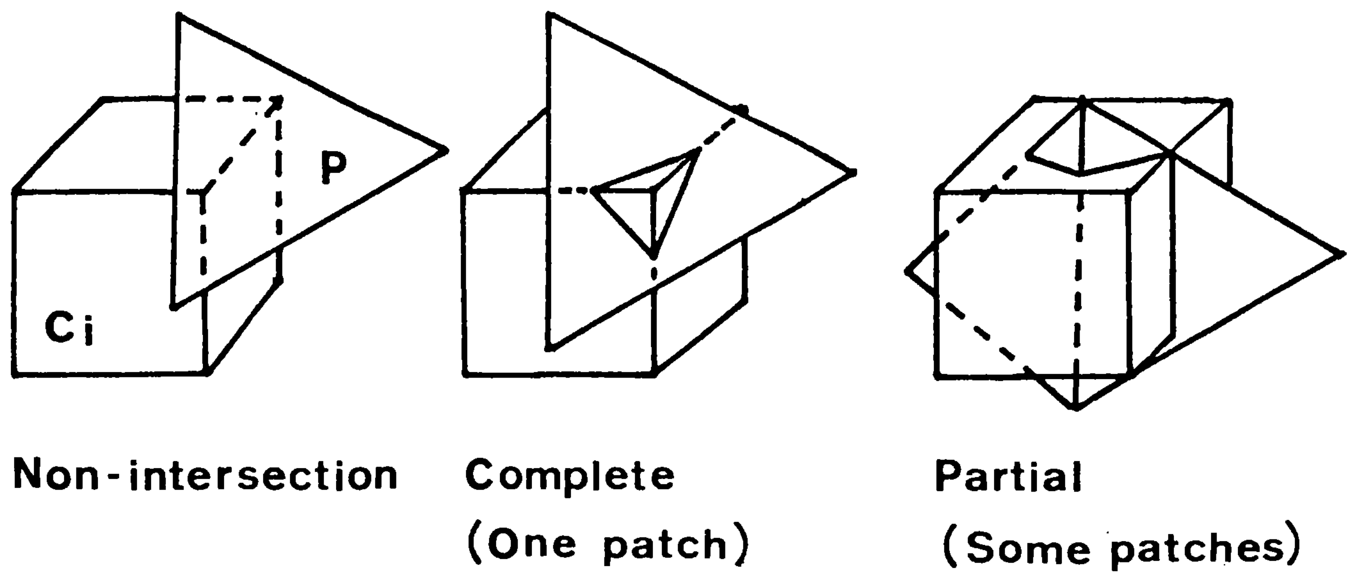


Fig.12. Three dimensional intersecting pattern between patch and cube.

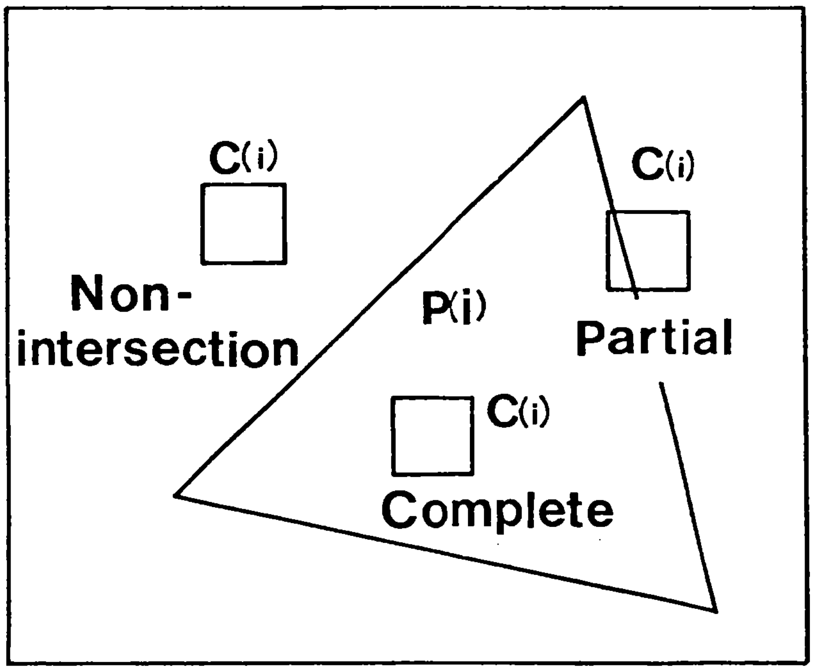


Fig.13. Two dimensional intersecting pattern between patch and cube.

[3D information] (Fig.12)

[non-intersection] $Ci \cap P = \phi$.

[partial] $Ci \cap S \supsetneq Ci \cap P \neq \phi$.

[complete] $Ci \cap S = Ci \cap P \neq \phi$.

In three-dimensional space, a state "non-intersection" implies that a subcube Ci does not intersect any patch , a state "partial" does that a subcube Ci intersects some patches, and a state "complete" does that a subcube Ci intersects only one patch. The subcube having one of the last two states corresponds to the 'intersection' subcube.

[2D information] (Fig.13)

[non-intersection] $C(i) \cap P(i) = \phi$.

[partial] $C(i) \supsetneq C(i) \cap P(i) \neq \phi$.

[complete] $C(i) = C(i) \cap P(i) \neq \phi$.

In two-dimensional space, a state "non-intersection" implies that the shadow C(i) is outside the shadow P(i), a state "partial" does that the shadow C(i) intersects some edges of the shadow P(i), and a state "complete" does that the shadow C(i) is inside the shadow P(i). The last two states indicate that the shadow C(i) intersects the shadow P(i).

[Theorem 1] For any plane D(i), $C(i) \cap P(i) \neq \Phi$
$\Leftrightarrow Ci \cap P \neq \Phi$.

Proof: Direct part) For any plane D(i), $C(i) \cap P(i) \neq \Phi$ implies that each point $p_0(i) \in R(i) \cup R$ (i=X, Y, and Z) exists on a patch P. A triangle Q that consists of those points $p_0(i)$ intersects the region R as shown in Fig.11. The triangle Q is inner for the patch P because any patch P is assumed to be a convex region in this dissertation. Therefore, $Ci \cap P \neq \Phi$.

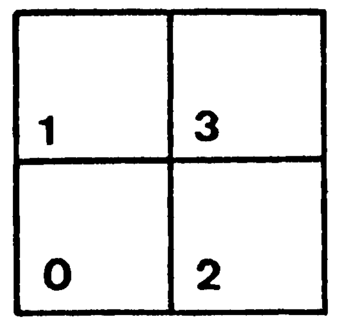
Converse part) A point $p_1(i)$ represents the projection of a point $p \in Ci \cap P \neq \Phi$ onto the D(i) plane. Hence for any plane D(i), $p_1(i) \in C(i) \cap P(i) \neq \Phi$. The assertion follows. □

[Theorem 2] For some plane D(i), $C(i) \cap P(i) = \Phi$
$\Leftrightarrow Ci \cap P = \Phi$.

Proof: Direct part) This part is given by the contraposition of the converse part in Th. 1.

Converse part) This part is given by the contraposition of the direct part in Th. 1. Hence, the assertion follows. □

When the projected plane is divided into two regions by the line that includes an edge constructing a shadow P(i), the region that includes the shadow



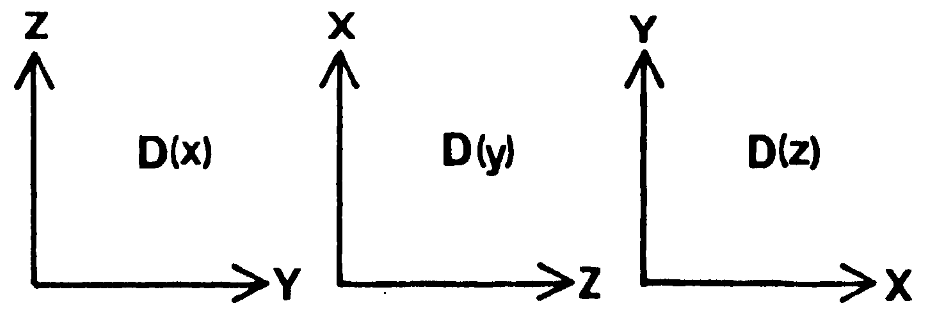


Fig.14. (a) The order of quadrant
(b) Selection of a projecting plane.

P(i) is defined to be inside for the edge and the region that does not include it is defined to be outside for it.

Since the subcube Ci intersects the minimum parallelepiped B that encloses a patch P, shadow C(i) intersects the region B(i) that is a projected region of the parallelepiped B. Here, the shadow P(i) is a convex region, and all angles of P(i) within the region B(i) are obtuse. From those properties, it follows that the shadow C(i) is "non-intersection" for the shadow P(i) if it is outside for some edge of P(i), and the shadow C(i) is "complete" for the shadow P(i) if it is inside for all edges of P(i). Otherwise, the shadow C(i) is "partial" for it. The PROJECTION routine uses this classification and the Th.1 and Th.2.

A position OUT is defined as the position j that maximizes $\vec{k}\cdot\vec{j}$ ($\vec{k}$: the outer vector that is vertical to an edge). The vector $\vec{j}=(r_0,r_1)$ is also defined such that $(r_0r_1)_2$ denotes the binary representation of a decimal number j. The number j represents the position of the shadow C(i) in a projected region of the cube illustrated in Fig.14.

4) JUDGE routine: This routine classifies each 'non-intersection' subcubes into being 'inside' or

Listing 6. PROJECTION routine.

```
procedure PROJECTION(P, C, map)
begin
   for i ← 1 until 3 do
   begin
     while (EDGE(PROJ(P, i)) ≠ NULL)
     begin
       case JUDGE(EDGE(PROJ(P, i)), CENTER[i]) of
       inside: break;
       outside:
       on:
          MAP(i, OUT, map);
          break;
     end;
   end;
end;
```

```
PROJ(P, i) ≡ projected region of a patch P onto the plane
             D(i).
EDGE(P) ≡ edge of a patch P
JUDGE(e, p) ≡ judging whether a point p is inside,
              ,intersecting, or outside.
CENTER[i] ≡ coordinates of the center point of cube C
            except the i-axis component.
MAP(i, j, map) ≡ changing states of two subcubes into
                 'non-intersection', whose projected region
                 onto the plane D(i) corresponds to the
                 position j as shown in Fig. 14.
```

'outside' for the B-reps. By checking the position of the point G for each plane including a patch in the present 'intersection' cube, i.e., inside, intersect, or outside, which has been already obtained through the PLANE routine, we proceed with one of the following processes.

1) If the point G is outside some plane, then the 'non-intersection' subcubes are 'outside' for the B-reps.

2) If the point G is inside all planes, then the 'non-intersection' subcubes are 'inside' for the B-reps.

3) If neither of case 1) nor 2) occur, it is possible to determine by the information obtained through PLANE routine (ii) whether each 'non-intersection' subcube is 'inside' or 'outside' for the B-reps.

These processes are based on the following properties that are acquired if and only if the B-reps. is convex.

[Property 1] When the B-reps. does not intersect the point G, all 'non-intersection' subcubes are of the same state ('inside', or 'outside') and they are 'inside' ('outside') if the point G is inside (outside) the B-reps. When the B-reps. intersects

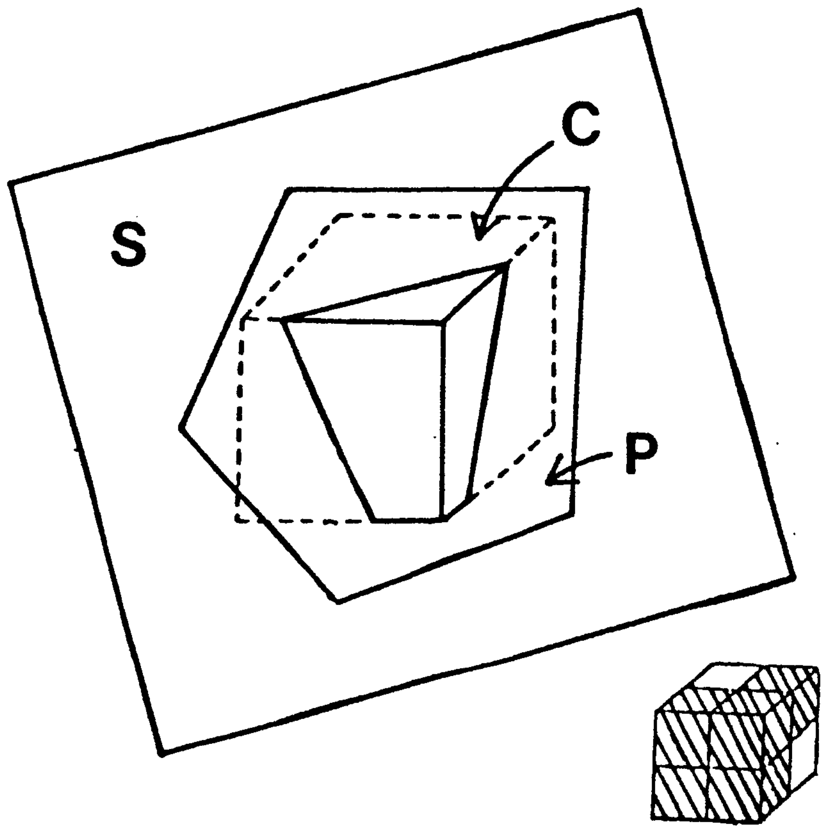


Fig.15 SINGLE procedure

the point G, whether each subcube is 'inside' or 'outside' for the B-reps. can be determined by the information obtained through the PLANE routine (ii).□

[Property 2] The PLANE routine judges whether the point G is inside, intersect, or outside for the plane including a patch in the present 'intersection' cube. Then, the point G is outside the B-reps. when the function d(G,S) is positive for some plane S. The point G intersects the B-reps. when the function d(G,S) is zero for one plane S and d(G,S) is negative for other planes. Otherwise the point is inside the B-reps. □

3.2.3 SINGLE Procedure

An intersecting region between a patch P and a cube C corresponds to only an intersecting region between the plane S including the patch P and the cube C if the cube C intersects only the patch P (See Fig.15). Therefore in this procedure, the 'intersection' subcubes intersecting a patch P can be easily obtained using only the PLANE routine. In addition, the JUDGE routine also deals with 'non-intersection' subcubes.

Listing 7. SINGLE procedure.

```
procedure SINGLE(C, P, n')
begin
  map ← PLANE(P, C);
  WRITE('(');
  if n'+1 ≥ n then
     begin
       for i ← 0 until 7 do
          if MARKED(map, i) then
          WRITE('1');
          JUDGE (map);
     end;
  else
     begin
       for i ← 0 until 7 do
          if MARKED(map, i) then
          SINGLE(CHILD(C, i), P, n'+1);
          JUDGE (map);
     end;
  WRITE(')');
end;
```

3.3 Computational complexity of the algorithm and the storage of the resultant octree

The computational complexity of the algorithm can be evaluated by a total number of cubes that intersect at least one patch of the B-reps., and the storage of the resultant octree can be also evaluated by a total number of cubes that are built by the algorithm.

For each patch P, consider a projection of the patch P onto the plane D(i) for the i direction maximizing $|\vec{n}\cdot\vec{i}|$ ($\vec{n}$: normal vector of the plane S including a patch P, $\vec{i}$: unit vector of X, Y and Z axes, respectively) and denote the projected region by P'. When the area and periphery length of some patch P are denoted by S_p and S_c, respectively, the area S_p' and periphery length L_p' in the projected region P' are obtained by $S_p' = S_p * \cos\theta$, $L_p' = L_p * \cos\theta$ ($\cos\theta = |\vec{n}\cdot\vec{i}| / |\vec{n}| * |\vec{i}|$:constant). Also, a cube C at the level i is projected onto the plane D(i) and the projecting region is called a BLOCK region. Then, an area (Sc) and periphery length (Lc) for the BLOCK region are expressed by $Sc=4^{nmax-i}$ and $Lc=2^{nmax-i}$:

nmax = $\log_2$((an edge length of the world space)/ (an edge length of a cube at the finest resolution level)}.

In each level, a number of "complete" regions is bounded by S_p'/Sc, and a number of "partial" regions is also bounded by $\sqrt{2}L_p'/Lc + 2\alpha$ (α: number of edges that construct the patch) (Appendix B). A number of cubes is bounded by three, which intersects a plane S and is on the BLOCK region (Appendix C).

Finally, the computational complexity of the algorithm until PLANE routine is denoted by C_1' and that of PROJECTION routine is denoted by C_2'. Until the resolution level n, the computational complexity of the algorithm for the patch P_k is denoted by C_k and a number of nodes for it is also denoted by N_k. A total calculation time (Cost) of the algorithm and a total number (Storage) of nodes can be expressed as:

$$C_k \leqq 3*\sum_{i=0}^{n-1}\{C_1'*S_p'/4^{(nmax-i)}+\sqrt{2}*C_2'*L_p'/2^{(nmax-i)}+2C_2'*\alpha)$$

$$= C_1S_p*4^n/4^{nmax} + 3\sqrt{2}C_2L_p2^n/2^{nmax} + 6C_2'*n*\alpha.$$

$$N_k \simeq 24*\sum_{i=0}^{n-1}\{S_p'/4^{(nmax-i)}\}$$

$$= 8S_p*4^n/4^{nmax}.$$

$$\mathrm{Cost} \leqq \sum_{P} C_k$$

$$= C_1 S * 4^n / 4^{nmax} + 3\sqrt{2} C_2 L * 2^n / 2^{nmax} + 6C_2' * N * n * \alpha.$$

$$\text{Storage} \approx \sum_{P} N_k$$

$$= 8S * 4^n / 4^{nmax}.$$

Consequently, the computational complexity of the algorithm and the storage of the resultant octree are proportional to S or 4^n for total surface area S of the given B-reps., the resolution level n. The computational complexity is also proportional to number N of patches. However, in case that parameter N is not large, the complexity is proportional to $\sqrt{N}$ because the length L is proportional to $\sqrt{N}$ if the area S is constant.

3.4 Experimental results

In this section, we report an actual calculating time of our conversion algorithm and a node number in the resultant octree for the area S, number N of patches of the B-reps., and resolution level n of the octree. We should note that the area is represented as the unit pixel that is the projection of a cube at the finest resolution level onto one plane D(i).

The conversion algorithm is implemented in C language on the VAX 11/750 (without a floating point accelerator) under Unix. An approximated sphere by 100 or 400 patches is used for the investigation.

A sphere is said to be 'unit' if the diameter of it is a side length of the world space, and also a sphere is said to be '1/8 unit' if the diameter of it is a half times as long as the diameter of the 'unit' sphere. The finest level n represents the resolution such that a world space is divided by 8^n.

Figures 16, 17 and 18 show that evaluated equations explained in the previous section are reasonable.

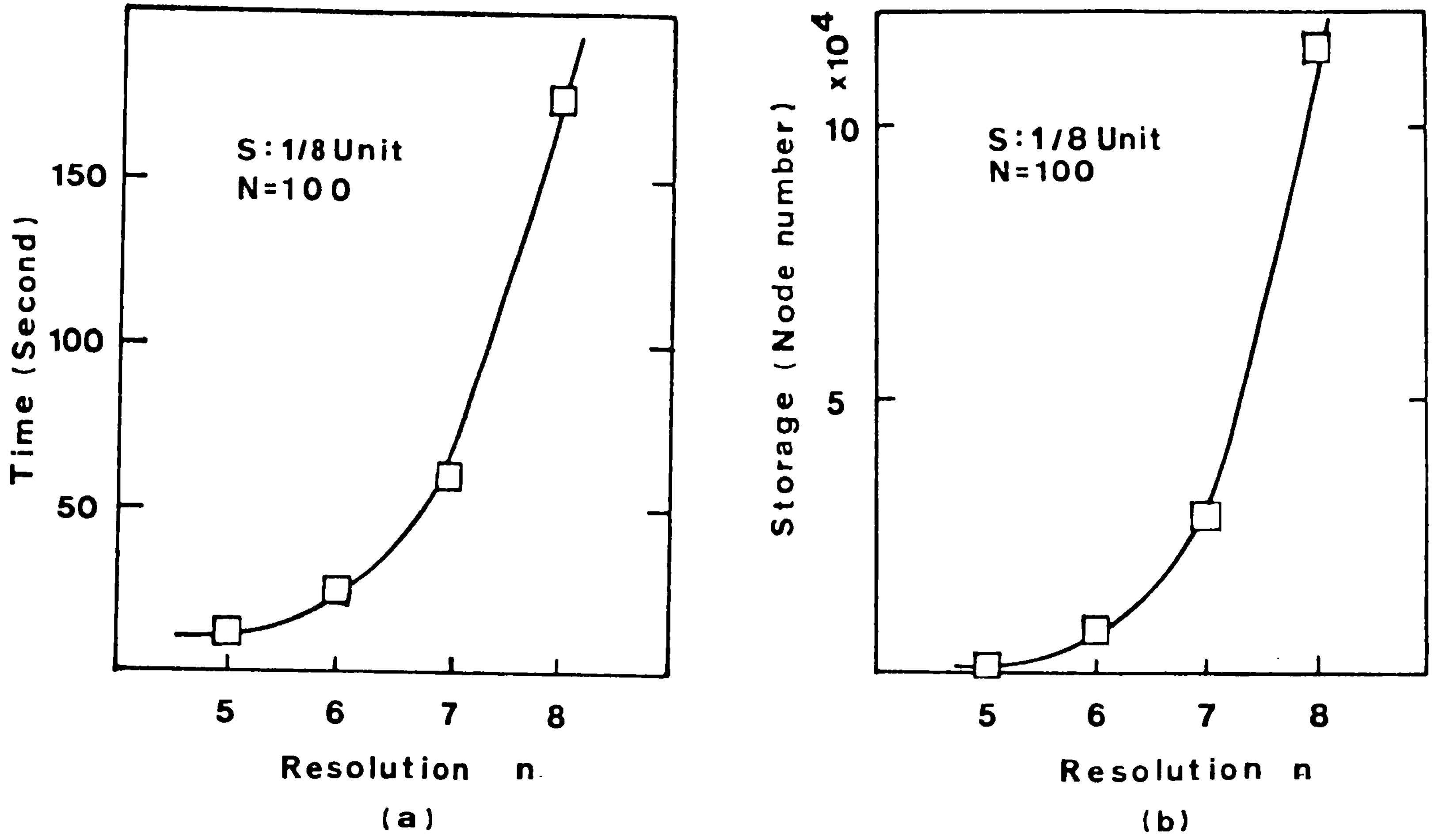


Fig.16. (a) Computational complexity of the algorithm
(b) Number of nodes in the octree as a function of the resolution (n).

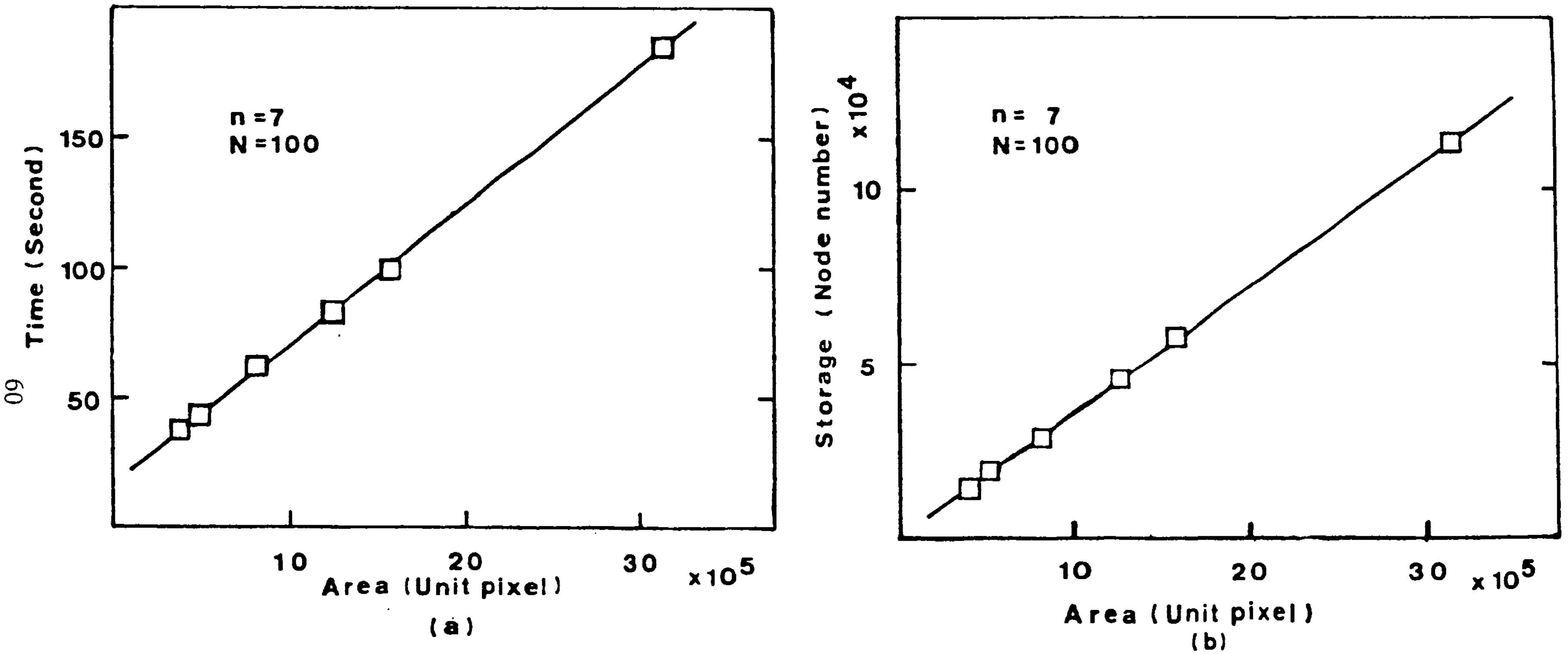


Fig.17. (a) Computational complexity of the algorithm
(b) Number of nodes in the octree as a function of the area (S).

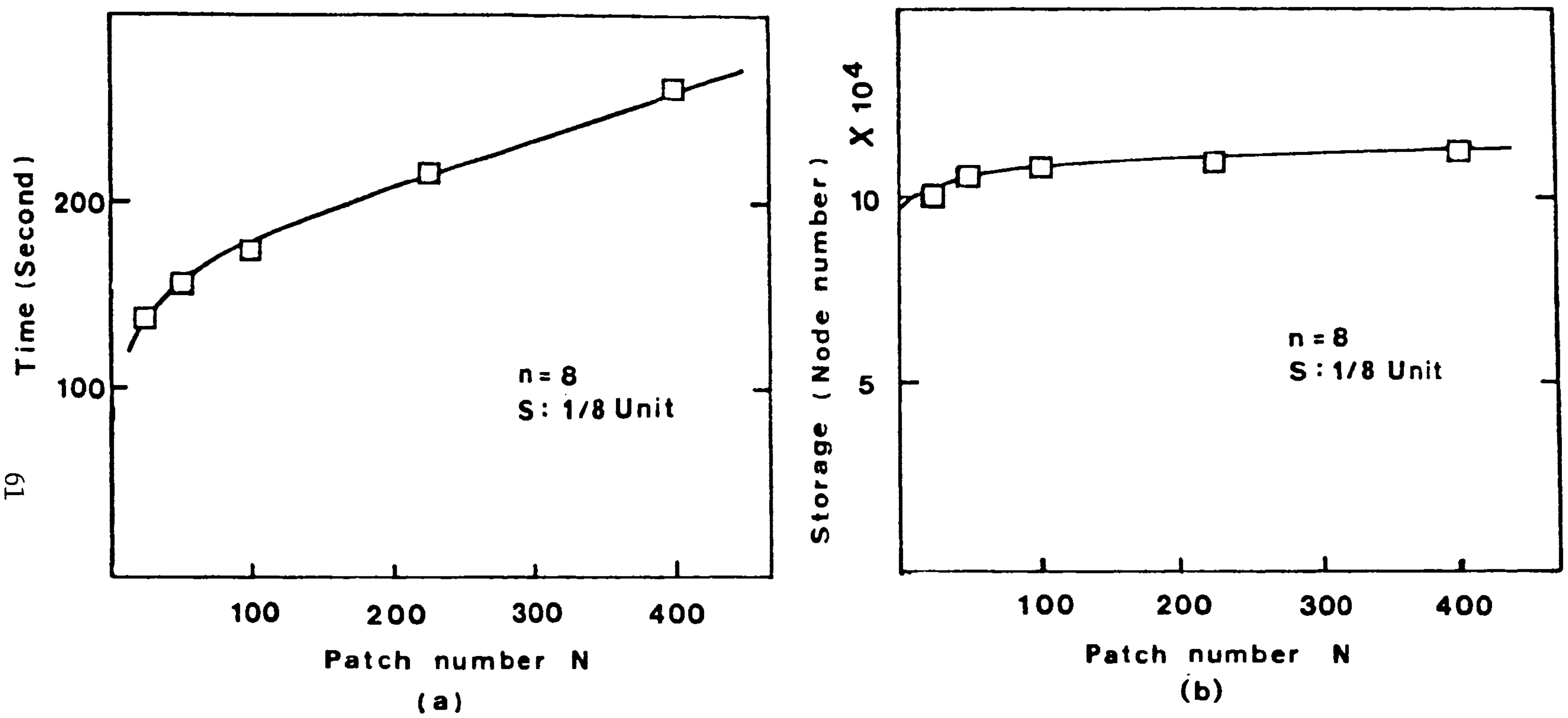


Fig.18. (a) Computational complexity of the algorithm
(b) Number of nodes in the octree as a function of the patch number (N).

3.5 Conclusion

Using the one-pass approach, a fast conversion algorithm from the B-reps. to the DF-representation is proposed. Since selection of a region (cube) in which its corresponding octree (DF-representation) should be built is adaptable, the conversion algorithm can be easily applied to construction (See in Chapter 4) of a description of an real object by projecting cones (B-reps.) of multiple two-dimensional views in the field of computer vision and to interference detection (See in Chapter 5) in the field of robotics.

CHAPTER 4
CONSTRUCTION OF THE OCTREE APPROXIMATING A THREE-DIMENSIONAL OBJECT BY USING MULTIPLE VIEWS

4.1 Introduction

Reconstruction of a solid model that approximates a three-dimensional object by using multiple two-dimensional images has been investigated in many fields (Computer Vision, Computer Graphics, and Robotics). In relation to this, many important technical methods (Stereo vision, Shape from shading, etc.) have been proposed. However, some methods can not build an accurate solid model for the object if degraded images are only obtained, and others require enormous calculation time.

To solve those problems, we propose a new algorithm based upon a so-called volume intersection method [55], which can be explained in the following way: First, a cone that is defined by both a viewpoint and a polygon approximating the contour of an image of the object from the viewpoint is built. The cone contains the object and is represented by a polyhedron. Second, by constructing a common region of the cones for multiple views, we obtain an approximating region for the object. Since contours in an image are the clearest information within the whole image, the image degradation does not affect the approximating precision of the octree for the object.

To make this method run faster, we should first construct the approximating polygon effectively. Fortunately, this problem has been already investigated in detail and fast algorithms have been proposed in the literature [57]. Hence, we assume in this chapter that a set of polygons for all views is already obtained. Next, we should construct the common region effectively. In case that a polyhedron, which is the most general representation for a solid object, is adopted as the solid model [59], the model generated in the middle of the construction is represented by a polyhedron with k*N patches (k: number of views that have been already used, N: average number of segments of the approximating polygon). Then, the computational complexity of the construction is proportional to at least n^2, or N^2 because the intersection between one cone and the model generated should be checked in all views successively. Instead of this, we adopt the octree representation as a solid model. The octree represents a three-dimensional world space by successive refinements that increase the resolution of the object's details and keeps regions in the world space as a set of nodes with a hierarchical description in positioning. The computational complexity of the construction is proportional to at most n, or N, since the hierarchical description can be effectively used in checking the intersection between the cone and the model (See in Section 4.5).

Some algorithms have been proposed to construct the octree from two dimensional images on the basis of the volume intersection method. Kim and Aggarwal [60] proposed a fast algorithm. However, there are two difficulties in their algorithm: 1) a coordinate system for the octree depends on the posture of the object. That is, the system of the world coordinate can not be flexibly selected, and 2) although an actual contour photographed with a camera should be defined by the perspective projection, their algorithm deals with the given contour under the parallel projection. Moreover, the number of views used in the algorithm is limited to three. As a result, the octree that approximates the object with arbitrary accuracy can not be built.

In [61],[62], those difficulties have been solved. Namely, 1) the octree can be constructed in an arbitrary coordinate system, 2) since the cone is defined in the perspective transformation and the number of views is not limited at all, the resultant octree can approximate the object with arbitrary precision. However, their algorithm is slow. In their paper, the approach that enrolls patch information concerning a cone in a set of regions in the octree is managed on an image plane. To put it concretely, intersection between a cone and a region is judged by that between the approximating polygon for the cone on the image plane and the hexagon that is the projection of the region onto it. Construction of the projection is need much time because of perspective transformation. The intersection checking on the image plane also requires a considerable work [56]-[58]. Besides, since the projection becomes an unequal hexagon, the algorithm can not smoothly assign patch information of cone in a region to eight subregions in division of the region, which is the basic process of the algorithm. Thus, the number of patches of cones in one region does not decrease in progression of the algorithm and therefore the calculation cost of data-processing in it does not decrease in the progression.

In addition, the algorithm builds an octree for a cone and next does a final octree by means of cutting successively the initial octree by other cones. Consequently, it is inefficient in a sense that they must deal with irrelevant regions which do not lie in a neighborhood of the object.

The algorithm proposed in this chapter is capable of constructing the same octree faster than Hong's algorithm. The reason is that an essential part of the algorithm is not processed in an image plane but is processed directly in the three-dimensional world space. Though this approach is suggested in Section 2 of [61], its thorough investigation did not be carried out. Major reason of this abandonment is considered that a conversion algorithm from arbitrary part of the cone to the octree did not yet be presented. However, this problem is solved by the conversion algorithm in Chapter 3. Using this approach, the perspective transformation is not necessary and a fast intersection checking under the three-dimensional space is also attained. Moreover, this algorithm can smoothly assign patch information of cone in one region to eight subregions and therefore the calculation cost of data-processing in it also decreases in the progression of the algorithm (See in Section 4.4 and 4.5).

Finally, since our algorithm deals with all cones simultaneously using the hierarchical description of the octree in positioning, it need not deal with regions which do not lie in a neighborhood of the object. In other words, it manages only regions at the present resolution level of the octree, which intersect the object.

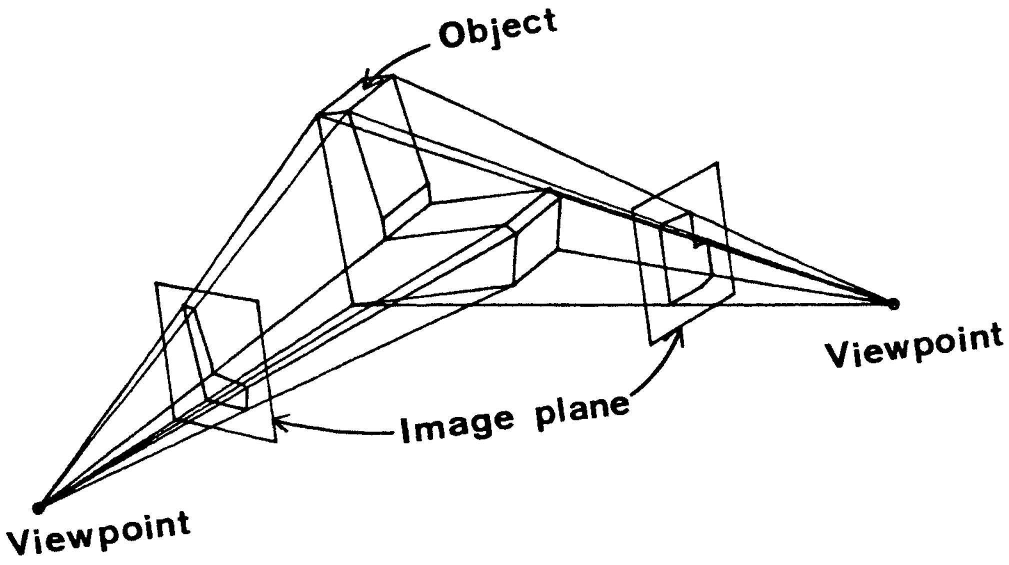


Fig.19. Volume intersection method.

4.2 Volume intersection method

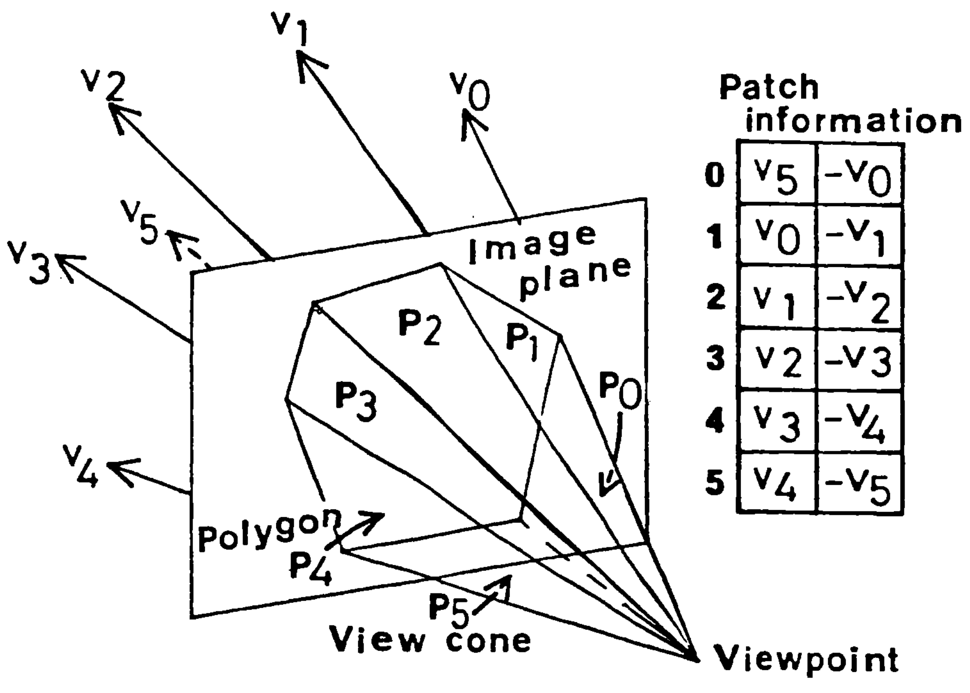
When a three-dimensional object is seen from one view, the object always exist within a cone that is determined by an approximating polygon of an image of the object and the center of the lens. Thus, a common region of cones for multiple views represents an approximating solid model for the original object as shown in Fig.19.

First, certain advantages of this method, which is compared with other methods (Stereo vision, CT scanner, and Shape from shading), are described as follows:

1) The algorithm can run fast because approximation of a contour is simpler than solution of a matching problem in Stereo vision [23]-[25].

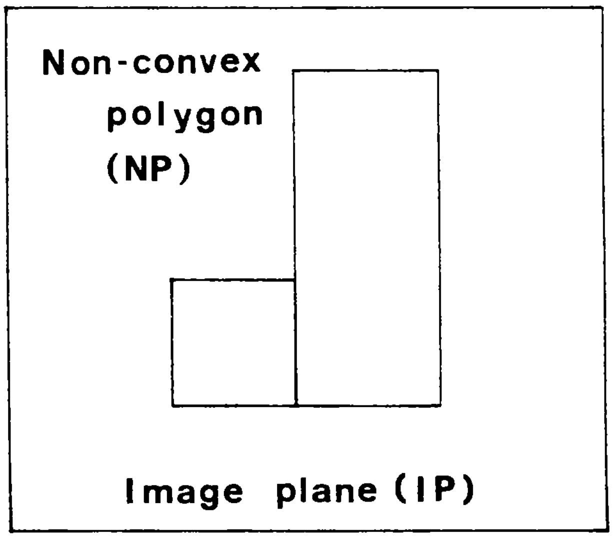
2) Since a contour yields the clearest information in an image, image degradation does not affect the precision of approximation of the octree for the object, in comparison with the Shape from shading [26].

The disadvantage of the method is that a non-convex part of the three-dimensional object, which does not appear in any two-dimensional image in principle, can not be constructed in the model.

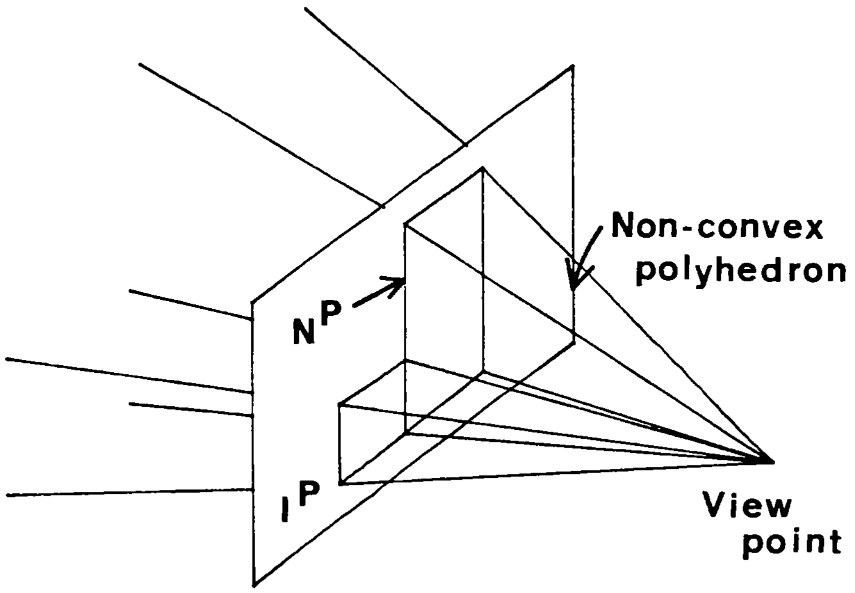


0	v_5	$-v_0$
1	v_0	$-v_1$
2	v_1	$-v_2$
3	v_2	$-v_3$
4	v_3	$-v_4$
5	v_4	$-v_5$

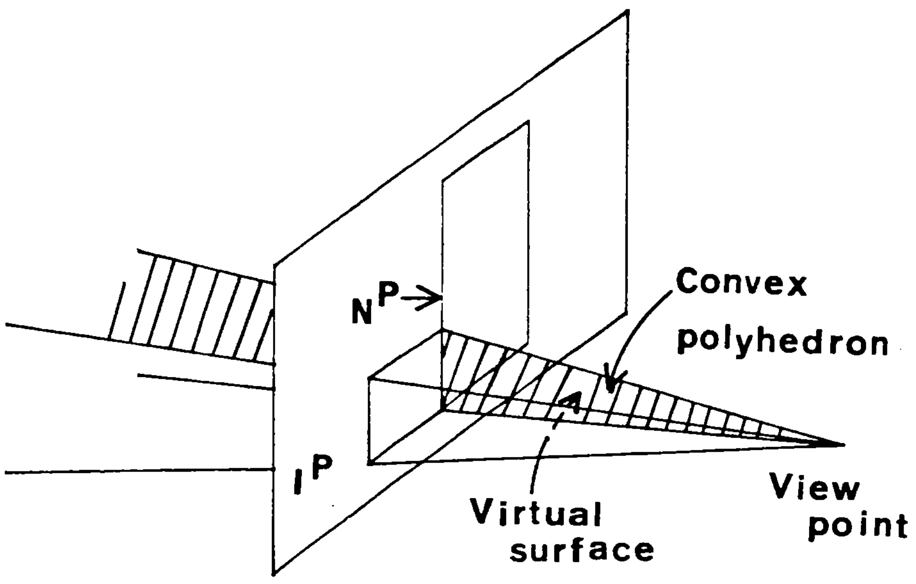
Fig.20. Cone model.


Non-convex polygon (NP)
Image plane (IP)

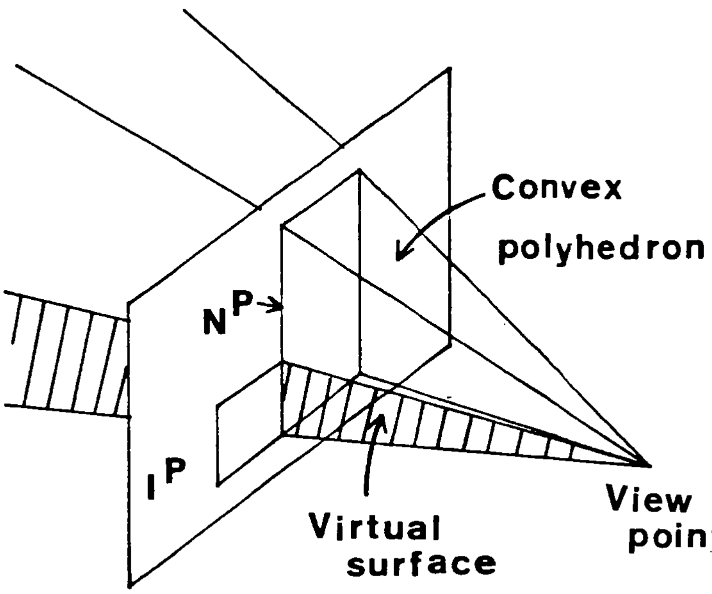
(a)


Non-convex polyhedron
NP
IP
View point

(b)



(c)



(d)

Fig.21. Convex partition of the cone. (a) Partition of non-convex polygon, (b) view cone, (c) and (d) convex cones for the view cone.

4.3 Definition

A cone, which is defined by an approximating polygon of an image of a three-dimensional object and the center of the lens (viewpoint), is an infinite cone whose patch P is represented by two vectors. A vector is defined by one vertex of the polygon and the viewpoint. It has an end point which is coincident with the viewpoint. The direction of the vector is defined from the viewpoint to one vertex. Every two vectors representing a patch are arranged in clockwise order for a normal vector of the plane that includes the patch (See Fig.20).

By these definitions, this cone can be considered to be the B-reps. defined in Chapter 2. Therefore, the algorithm in Chapter 3 is also applicable to a part (Cone making procedure) of the algorithm in this chapter if the cone is of convex shape. Then with the help of a partition of a non-convex approximating polygon into some convex polygons, the non-convex cone is parted into some convex cones as shown in Fig.21. The computational complexity of this partition is $O(N_1+N_2^3)$ [56]-[58] (N_1: number of vertexes in the approximating polygon, N_2: number of obtuse angles in the polygon).

4.4 Algorithm for constructing the octree

A basic_process of this algorithm consists of the following two substeps.

4.4.1 Classification of eight subcubes

Using three procedures (Cone making, Cone combining, Cone regulating procedures) successively, this substep classifies eight subcubes Ci (subregions) of an input cube C (region) into being of 'inside', 'outside', or 'intersection' state for the three-dimensional object. The produced states of eight subcubes are kept as a classification. Here, the first input cube is the world space.

1) Cone making procedure: The procedure makes the classification for a convex cone that is a component of a cone for one view. One of the following two procedures (PLURAL, or SINGLE procedure) takes part in the above classification. If the number of patches of the convex cone in the input cube equals one, then the SINGLE procedure is selected. Otherwise, the PLURAL procedure is selected.

PLURAL procedure: This procedure deals with an input cube C that intersects some patches of the convex cone. By considering the cone as the B-reps. defined in Chapter 2, the way proposed in the PLURAL procedure in Chapter 3 is useful in processing this procedure.

SINGLE procedure: This procedure deals with an input cube C that intersects only a patch P of the convex cone. Since the cone is considered to be the B-reps. defined in Chapter 2, the SINGLE procedure in Chapter 3 can be adopted as this procedure.

Using one of these procedures, eight subcubes is classified into begin of 'inside', 'outside', and 'intersection' state for the convex cone.

2) Cone combining procedure: This procedure makes the classification for a cone, named a view cone, which is defined for one view and is not convex. The view cone consists of some convex cones (See in Section 4.3). A subcube is 'outside' for the view cone if it is outside all the convex cones. A subcube is 'inside' for the view cone if it is inside one convex cone. The other subcubes are 'intersection' for it. Thus, applying the rules in Table 2 to the classification for each convex cone successively, we

Table 2. Cone combining rules.

Cone combining rules		Previous state		
		outside	intersection	inside
Present state	outside	outside	intersection	inside
	intersection	intersection	intersection	inside
	inside	inside	inside	inside

Table 3. Cone regulating rules.

Cone regulating rules		Previous state		
		outside	intersection	inside
Present state	outside	outside	outside	outside
	intersection	outside	intersection	intersection
	inside	outside	intersection	inside

obtain the classification for the view cone. Then, the algorithm is capable of maintaining calculative efficiency by only overlooking the virtual patches generated by the convex partition. Namely, a 'intersection' subcube that intersects only some of the virtual patches is converted to an 'inside' one there and then.

3) Cone regulating procedure: This procedure makes the classification for a common region of all the view cones (an approximating region for the three-dimensional object). A subcube is 'outside' if it is outside some view cone. A subcube is 'inside' if it is inside all the view cones. The other subcubes are 'intersection'. Thus, applying the rules in Table 3 to the classification for each view cone successively, we obtain the classification for the common region.

4.4.2 Making the DF-representation

Using the classification for the common region obtained by the above substep, this substep makes the DF-representation for eight subcubes. The substep assigns firstly the symbol '(' to the input cube C as the DF-representation because the basic_process deals with a cube corresponding to a mix node. Then, one of the following three processes deals with a subcube in the order of octant (See Fig.2).

1) The subcube is 'inside': This process assigns the symbol '1' to it as the DF-representation of the subcube.

2) The subcube is 'outside': This process assigns the symbol '0' to it as the DF-representation of the subcube.

3) The subcube is 'intersection': It corresponds to a region at the present resolution level, which intersects the common region. Then, a geometric structure for the object within this subcube must be examined further and be expressed by the DF-representation. Thus in this process, the basic_process deals with this subcube as an input cube anew. We should note that only the patch information of cones, which the above substep assigns to the subcube, is turned over to the new processing. Due to this recursive processing, the algorithm is accomplished by the depth-first search on the octree and therefore its DF-representation is naturally made. Exceptionally, if the level of the subcube that is presently managed by this process agrees with the finest resolution level, this process assigns the symbol '0' to the subcube as the DF-representation

instead of using the basic_process.

When eight subcubes have been dealt with by the above processes, this substep assigns lastly the symbol ')' to the input subcube C as the DF-representation and then the substep and the present basic_process are completed.

Since the algorithm deals with only regions ('intersection' cubes) at the present resolution level, which are neither inside nor outside the common region, the processing of irrelevant regions at the level, which do not intersect the object, can be prevented in the algorithm.

4.5 Computational complexity of the algorithm

Computational complexities of several algorithms for constructing a common region that approximates an object are evaluated in regard to parameters N (average number of segments of the approximating polygon), L (length of the polygon), n (number of views), or r (finest resolution level of the octree). The evaluation is neglected in the literature on previous research works.

4.5.1 Comparison among expressions of the model

a) Polyhedron model: The model generated in the middle of constructing the common region is represented by the polyhedron with k*N patches (k: number of views that have been already used). For constructing the common region with a polyhedron, the intersection between one view cone (polyhedron) with N patches and the model generated must be checked in all viewpoints successively. Hence, the computational complexity for this construction is obtained by

$$\sum_{k=0}^{n-1} C_0 * kN * N = C_0 * (n-1)(n-2) * N^2/2 \quad (C_0\text{: constant}).$$

The equation indicates that the complexity is proportional to n^2 or N^2.

b) Octree model: Due to the hierarchical structure in positioning of the octree, the algorithm for constructing the region with the octree can deal with only cubes that intersect the cone exactly. In other words, it deals with a set S_c of cubes that intersect the contour in a section of the object. Here, we should note that this contour in the world space corresponds to the previous contour in the image plane. The number of the cubes is denoted by $S_n = C*2^jL/2^{nmax}$ (Appendix D) (nmax=$\log_2${(an edge length of the world space)/(an edge length of cube at the finest resolution level: unit length)}, j: level of the cube in the octree).

When we assume that calculation cost of a data-processing in one cube is uniformly proportional to N, which is "pessimistic" assumption for the complexity, the complexity for construction of the common region with the octree is given by

$$\sum_{k=0}^{n-1} C_1 * NS_n = C_1 * nNS_n. \quad (C_1\text{: constant})$$

The equation also indicates that the complexity is proportional to n or N.

If an intersection between two patches happens in the former, then the algorithm must calculate lines, or a point where two patches intersect each other and arrange them in company with adjacent connection of the patches. On the other hand, if an intersection between a cube and a patch happens in the latter, then the algorithm only registers the patch number to the cube. Thus in general, the cost C_1 is smaller than the cost C_0. Moreover, the better the approximating precision of the common region for the object, the larger the parameters n and N. To obtain the region that approximates the object exactly, the octree model is advantageous in point of the calculation time.

4.5.2 Comparison among methods for checking the intersection between a cube and a cone

In general, the computational complexity of algorithms for constructing the common region with the octree can be evaluated as the summation of [unit processing cost] * [calculation cost for a data-processing in one cube] * [number of cubes intersecting the cone] for each view and each level of the octree.

a) The check in the image plane [61],[62]: A perspective projection of a cube onto the image plane is represented by an unequal hexagon. Then, the intersection between one cube and a view cone corresponds to the intersection between the hexagon and the approximating polygon in the image plane, and therefore the calculation cost of data-processing in one cube is assumed to be N+6 [56]-[58], which is "optimistic" assumption for the computational complexity. (This cost is assumed to be 6N in "pessimistic" assumption.) Thus in this case, the computational complexity for construction of the common region with the octree is evaluated by

$$C_0 \sum_{i=1}^{n} \sum_{j=1}^{r} S_n(N+6) \quad : \quad C_0'\{C*n*2^{r+1}*L(N+6)/2^{nmax}\}.$$

(C_0': unit processing cost)

The unit processing cost is defined as checking the intersection between two segments.

b) The check within the three-dimensional space: In this case, it is possible to assign patches of a view cone to a set S_c of cubes easily. Then, our algorithm assigns N patches of the cone to the set of the S_n cubes at each level of the octree, and therefore the calculation cost of data-processing in one cube is evaluated to be $1+N/S_n$, which is estimated as the average number of patches in one cube. Accordingly, the computational complexity for construction of the region is also evaluated by

$$C_1' \sum_{i=1}^{n} \sum_{j=0}^{r-1} S_n(1+N/S_n) \quad : \quad C_1'(C*n*2^r*L/2^{nmax}+nrN).$$

(C_1': unit processing cost)

In general, since the finest resolution level is fixed at not a little value, the unit processing cost is defined as checking the intersection between one cube and a plane and thereby is equivalent to the calculation cost of the SINGLE procedure. Then, this cost C_1' is nearly equal to the cost C_0'.

In addition, the former algorithm deal with cubes until the finest resolution level r, on the contrary, the latter one deal with cubes until r-1 level because of the assignment of patches.

Now, we make a comparison between the two cases.

1) The perspective projection of the cube is necessary in the former, on the other hand, it is unnecessary in the latter. The perspective projection of all cubes needs much time.

2) In division of a cube into eight subcubes, the eight subcubes whose shape are the same do not overlap at all and the position of each subcube is

recursively calculated. Then the latter algorithm can assign the patch information into the subcubes smoothly. In contrast to this, since eight hexagons that are produced from one hexagon in the former algorithm overlap one another and are not of the same shape, it is difficult to assign segments of the approximating polygon to the unequal hexagons, which is not considered in [61]. As a result, the latter is faster than the former in a sense that the calculation cost of data-processing in one cube decreases in progression of the algorithm.

Consequently, the check within the three-dimensional space is advantageous in point of the calculation time.

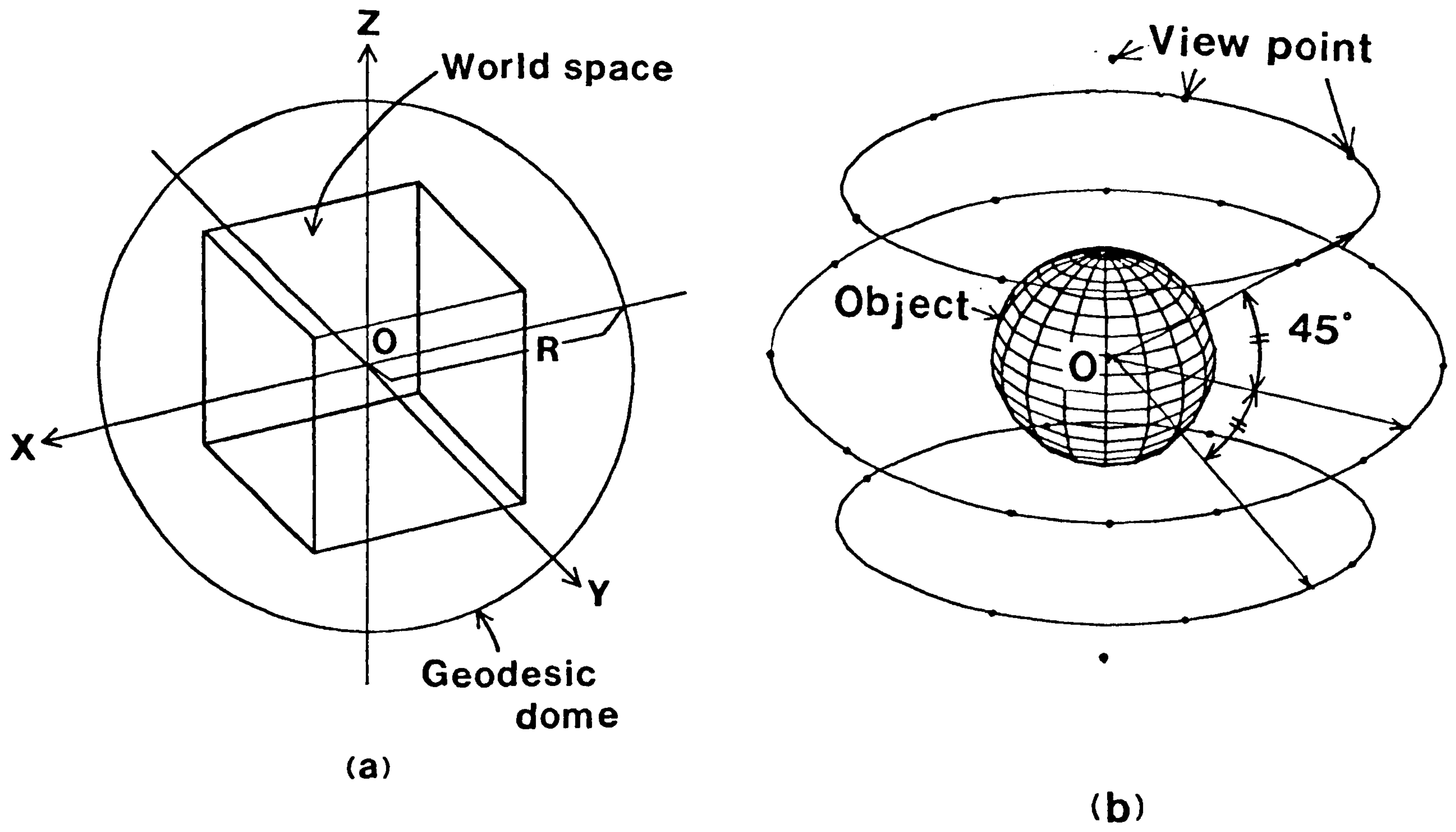


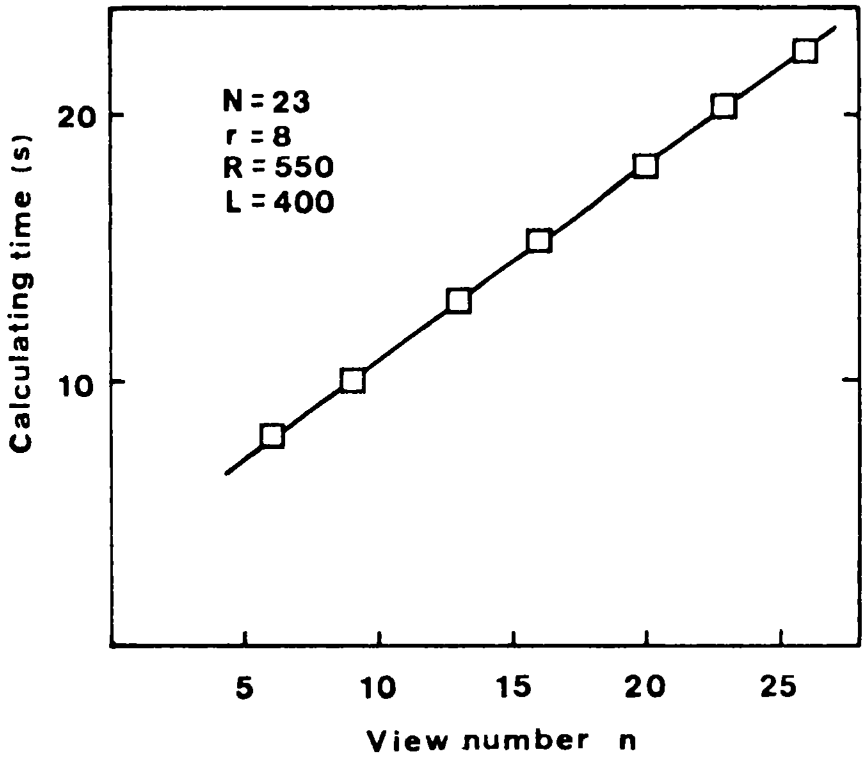
Fig.22. Geodesic dome (a) and view points (b).

4.6 Experimental results

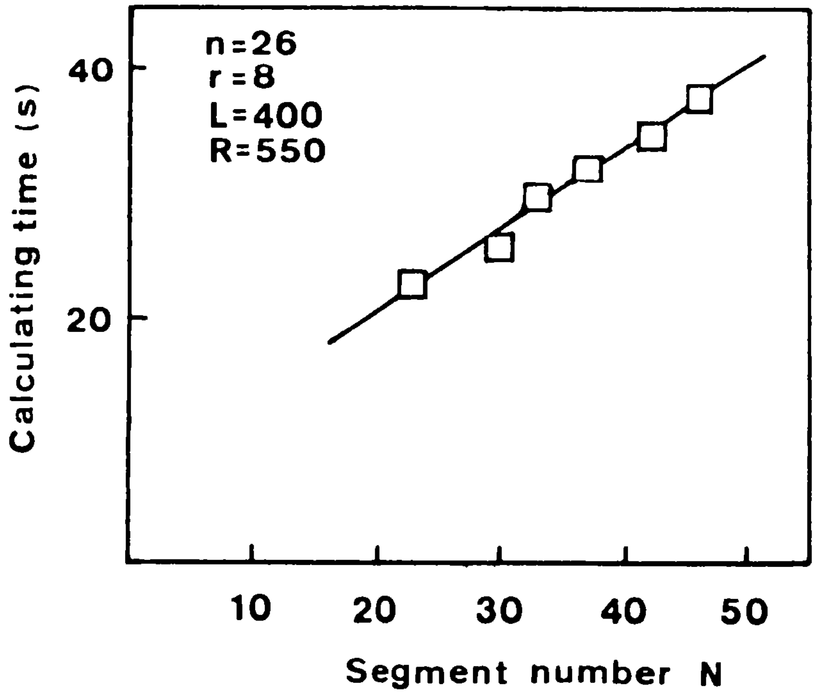
A sphere (R: radius; O: the center point), named geodesic dome, and all viewpoints on the geodesic dome are defined. Both a given object and the world space are placed in such a way that the center of gravity G for them are coincident with the point O (See Fig.22). The length of an edge of the world space is 1024 times as long as the unit length, i.e., an edge length of cube at the finest resolution level. In the following A and C paragraphs, some spheres are adopted as a given object in order to investigate actual tendencies of the algorithm.

4.6.1 Evaluation for properties of the algorithm

To ascertain the evaluation of the computational complexity of the algorithm in Section 4.5, we investigate the calculation time of the algorithm for some parameters n, N, r, or L as shown in Fig.23. Here, the radius of the geodesic dome is defined by 550, i.e., R=550, and the twenty-six viewpoints on the geodesic dome are prepared.


N = 23
r = 8
R = 550
L = 400
Calculating time (s)
View number n
20
10
5
10
15
20
25

(a)


n=26
r=8
L=400
R=550
Calculating time (s)
Segment number N
40
20
10
20
30
40
50

(b)

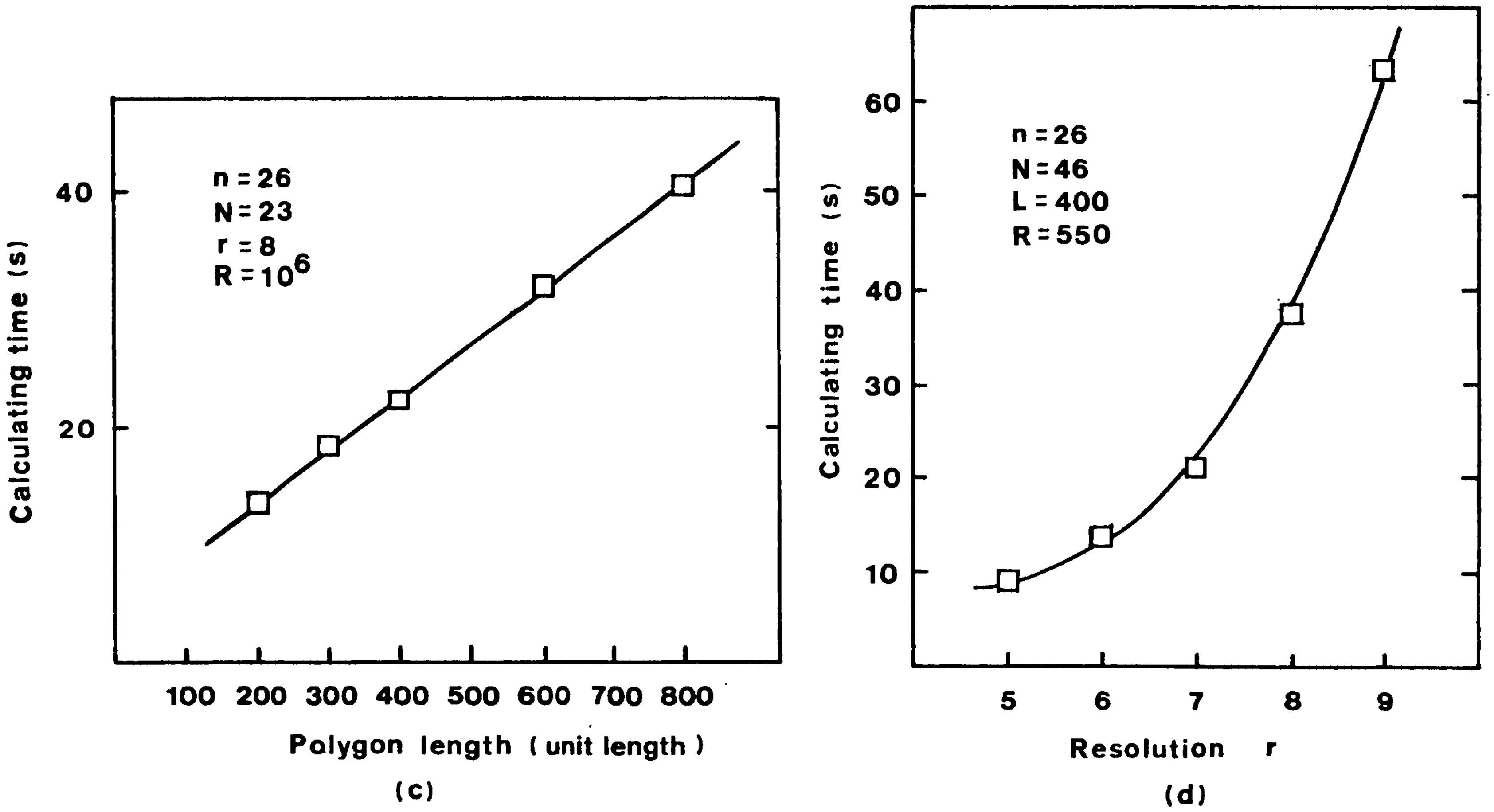
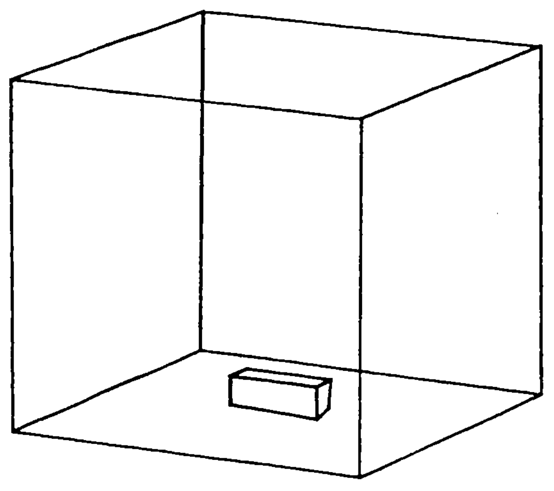
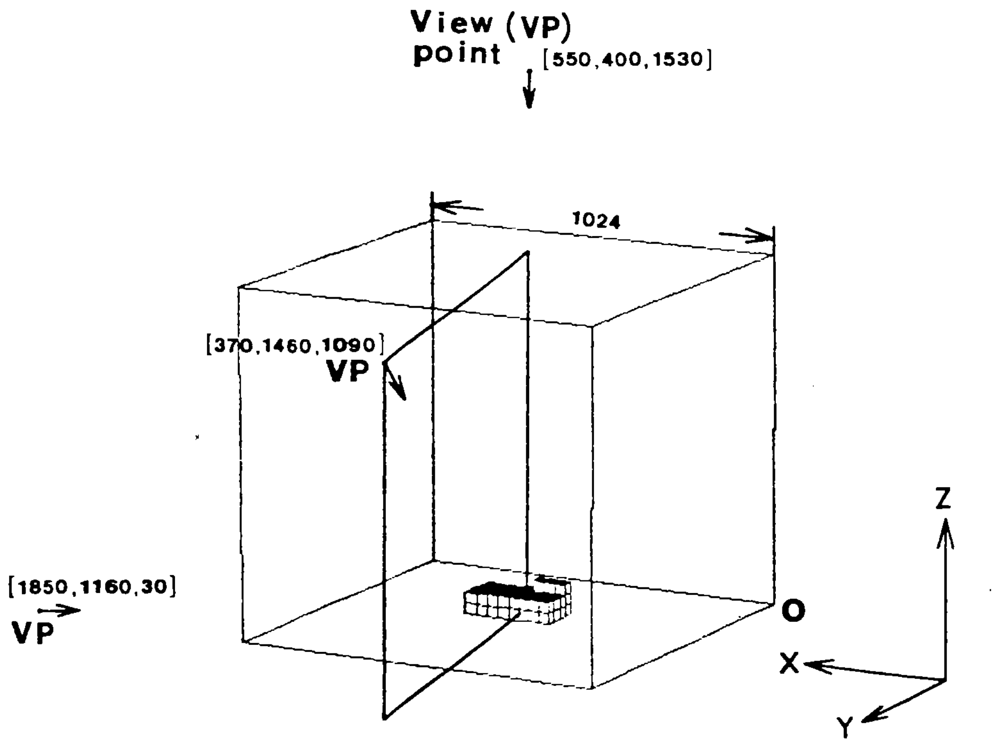


Fig.23. Calculating time of the algorithm, view number n (a), number N of segments for the polygon (b), length L of the polygon (c), the finest resolution level r of the octree (d).



(a)



(b)

Fig.24. An parallelepiped (a) and its corresponding octree (b).

4.6.2 Efficiency of the algorithm

We compare experimental results of our algorithm with those of Hong's algorithm, which is similar to our proposed algorithm in a sense that the cone is defined as the perspective transformation and the octree can be constructed in an arbitrary coordinate system. It is shown that the calculation time of the proposed algorithm is smaller than that of their algorithm. The algorithm proposed in this chapter needs only one hundred milli-second (Melcom 350-60: 3.7MIPS) for constructing the octree (n=5) for the parallelepiped [220,75,60] with three viewpoints (R=1500) (See Fig.24). On the other hand, their algorithm needs three minutes (Vax 11/780: 1.8MIPS) to construct a similar octree, though it uses a custom hardware being constructed to perform matrix manipulations for the perspective projection.

4.6.3 Approximating precision of the octree for the object

With regard to the parameter n or N, the approximating precision of the octree Ov, which the proposed algorithm constructs, for the octree Oo that approximates directly the original object is evaluated as shown in Fig.25. We use the following notations in this paragraph:

Vo: Volume of the octree Oo.

D: Volume of the difference between the octree Oo and the octree Ov.

$\varepsilon 1$: Approximating precision of the octree Ov for the octree Oo ($\varepsilon 1$= 100*D/Vo).

* Comparative experimentation with Kim's method [60].

(n=3; three image planes which are vertical together, $R=10^6$)

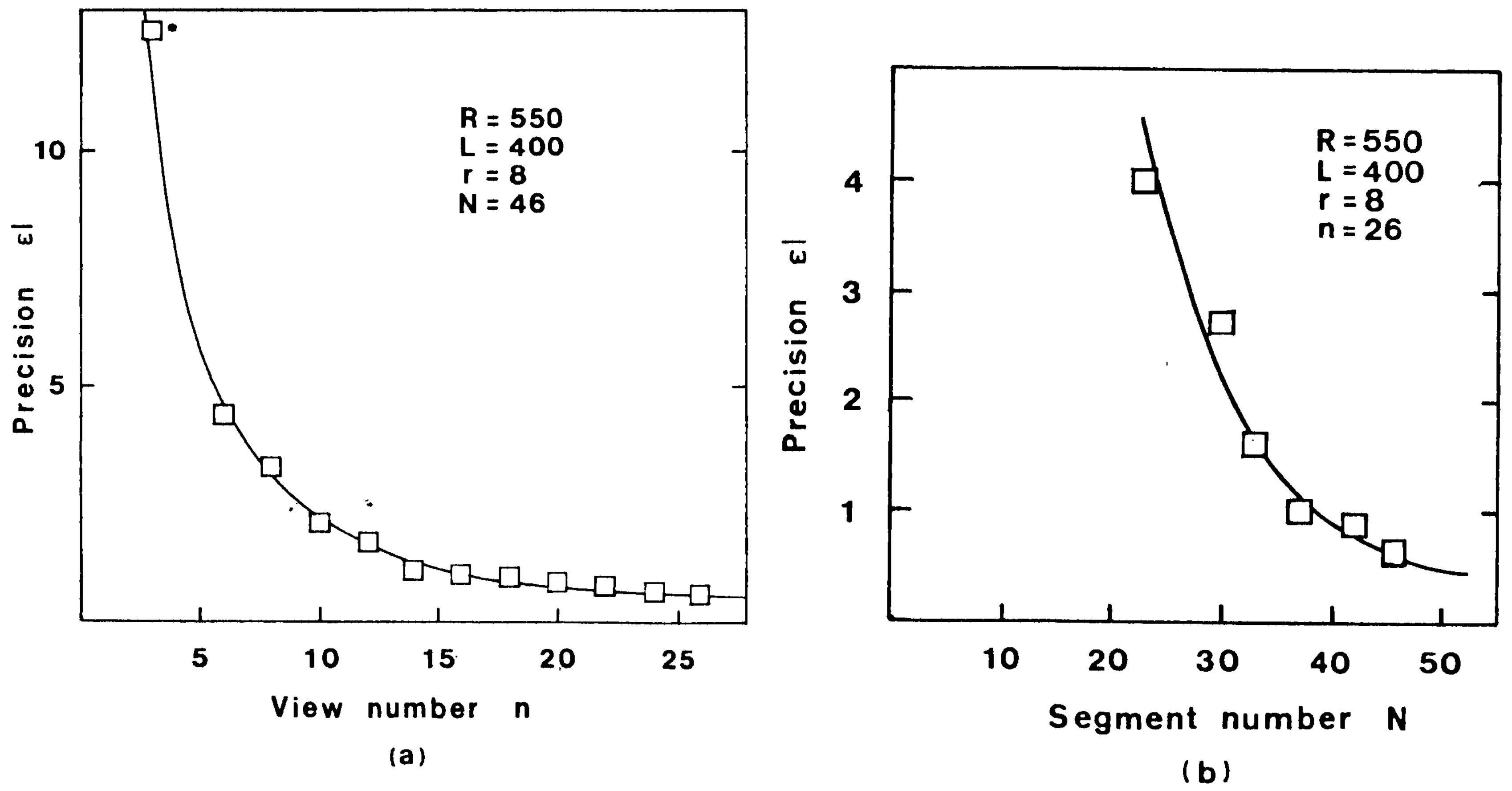


Fig.25. Approximating precision of the octree for the object, view Number n (a), number N of segments for the polygon (b).

4.7 Discussions

In data processing for one view, the volume intersection method approximates the model in only the contour part of a section for the object. The area of the part is smaller than that of the surface for the object. To obtain the octree that approximates the object with high precision, we should prepare many pairs of a viewpoint and its approximating polygon with high precision. This indicates that the parameters n and N must be kept large. Thus, to make the method run faster, the processing for one view should be efficient. In addition to this, we should pay attention to selection for a position of a viewpoint such that many shapes of the object can be received from one image. That is, if the object is a polyhedron, we must select a position of a viewpoint such that the projection of a patch of the polyhedron onto one image plane becomes a line. This problem must be investigated further.

4.8 Conclusion

A fast algorithm that constructs an approximating octree for a given object in an arbitrary coordinate system is proposed. On the basis of simple pre-processing (convex partition of a polygon), the algorithm is also useful in construction of an octree model for an non-convex object. Moreover, the computational complexity of the algorithm with respect to n, N, r, or L are evaluated and analyzed.

Even if several real objects exist in one space, this algorithm can construct the octree that keeps all approximating regions of the objects. That is, when there are some contours of these objects in an image, the corresponding cones are managed by the Cone combining procedure. Also, when there is a contour that is represented by overlapping some objects, the contour is considered to be a contour of an object and is managed by the Cone making procedure. Further, if an image of each object is distinguished from one another in some image planes, then the algorithm becomes run faster using this distinction of the object.

CHAPTER 5

A NEW INTERFERENCE CHECK ALGORITHM USING OCTREE

5.1 Introduction

Current researches on interference check are divided generally into two categories; one [63]-[65] is to search an intersection region occupied commonly by a robot model and obstacles in an environment model, and another [66]-[68] is to calculate the minimum distance between them. In those researches, components of the robot and obstacle are approximately represented by some simple shaped solid objects that are like a set of spheres and cylinders. Then, since the number of components can be decreased by simplifying the shape and each component can be further described in terms of a simple shape function, it is possible to reduce the computational cost for searching the common region and calculating the distance between two components.

However, those methods pass over possible intricate movements for a complicated robot in a cluttered environment because of excessive interference check with rough approximation. To solve this problem, a more faithful approximation for a cluttered environment and a complicated robot is required, which needs an enormous number of solid objects, and hence those methods need large calculation time. The reason is that the computational complexity of the methods depends on the complexity of shape of the robot and environment model.

To overcome this drawback, we point out that selection of only parts of obstacles is desirable, which lie around the robot model, and only interference check for those parts is necessary. In other words, the longer the distance between the robot model and its nearest obstacle, the less the calculation time of the algorithm. This property is desirable in the sense that it makes the algorithm run faster since actual interference between the robot and the obstacles happens rarely and therefore non-interference between them must be preferably ascertained in robot teaching.

The desired property is obtained by using the octree as a robot model and its environment model, which is a hierarchical solid model in positioning. In general, robot motion includes rotation when the robot must move flexibly. However, it is hard to represent the rotation for some object within the octree [69],[70]. Moreover, to represent each component of the robot model by the octree, much memory space is required in computer (See in Chapter 3, [28],[52]). Hayward [71] adopted the octree as an environment model. Furthermore, he suggested adoption of the B-reps. as a robot model but did not pursue it further, because the adoption is considered to lead to a complex interference checking algorithm. Then, Boyes's idea [63] is used in his paper when boundaries intersect. However, he did not give any obvious and concrete way to check the boundaries intersection of the robot model for the environment model and to check exactly the interfere.

In view of these points, we adopt the B-reps, which is not a hierarchical representation but preserves the topological information of an object, in representation of each component of a robot, and the octree, which preserves the geographic information in a world space, as an environment model. By assigning each patch of B-reps. to a set of cubes of the octree fastly, we show that this combination of different solid models leads to a relatively simple algorithm. From this reason, the proposed algorithm can fulfil the exact and fast interference checking.

To show how computation time of the algorithm change with the distance between the robot model and its nearest obstacle within the environment model, an illustrative model set is given, which consists of both a robot model represented by one component approximating a sphere and an environment model that includes a single parallelepiped. Finally, to show how the algorithm work in real-time in even a cluttered environment and a complicated robot, a model set is devised, which consists of both a robot model represented by twelve components and an environment model including twenty-four polyhedra. As a result of the use of this set, it is shown that the execution time of interference check is around 250 ms.

5.2 Definitions

5.2.1 Environment Model

Octree, which is adopted as an environment model in this chapter, is a solid model that hierarchically represents a three dimensional space including several obstacles for a given robot as shown in Fig.1 and Fig.32(a).

5.2.2 Robot Model

B-reps. is adopted as a representation of each component of a robot in this chapter as shown in Fig.3 and Fig.32(b).

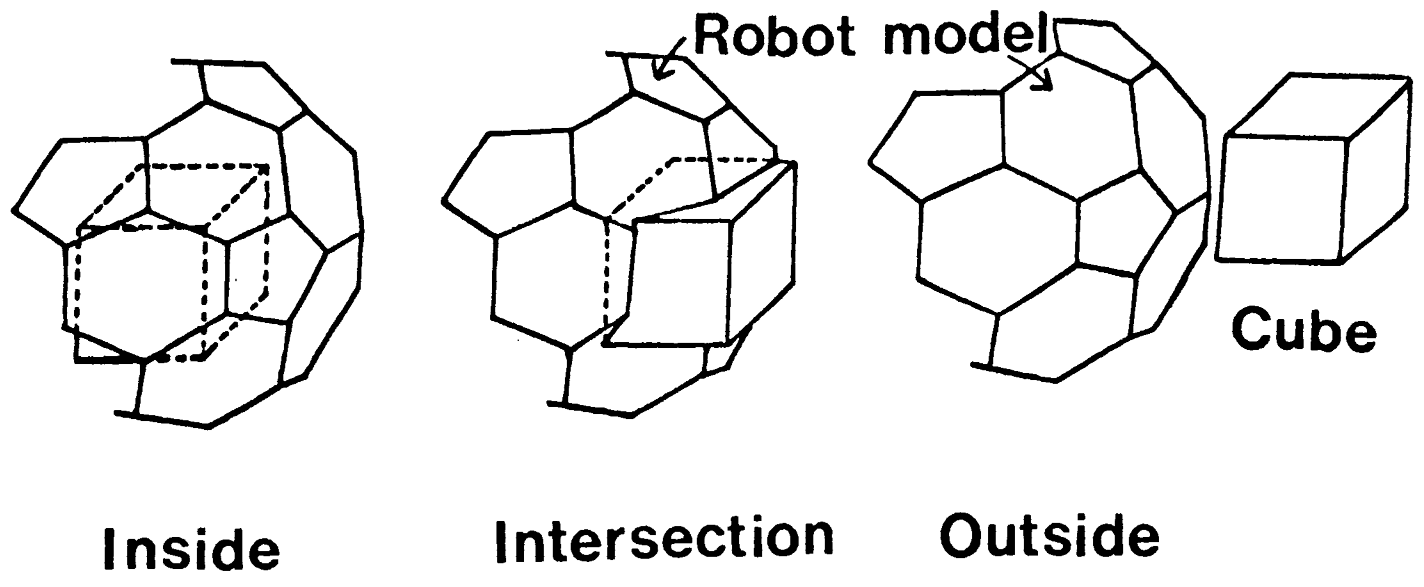


Fig.26. Relation between robot and cube.

Table 4. Relation between color of cube in the octree and state of the cube for the robot model.

Color	State		
	inside	intersetion	outside
black	interfere	interfere	non-interfere
mix	interfere	undecidedness	non-interfere
white	non-interfere	non-interfere	non-interfere

5.3 Interference check algorithm

5.3.1 Judgment of interference and non-interference

In connection with the environment model, a cube is classified into three color; 'black' (obstacles), 'white' (free space), and 'mix' (obstacle and free space). On the other hand, in connection with the robot model, the cube is also classified into three states; 'outside' (outside the robot model), 'inside' (inside the robot model) and 'intersection' (intersecting the robot model) as shown in Fig.26. With the aid of these classifications, the cube is again classified into three states 'interference', 'non-interference', or 'undecidedness' as shown in Table 4. The 'interference' state indicates that the robot model interferes some obstacle in the environment model within the cube. The 'non-interference' state indicates that the robot model does not interfere any obstacle within it. The 'undecidedness' state does that interference or non-interference between them can not be determined within it. For example, 1 row 1 column in Table 4 indicates that the robot model interferes obstacles, for a black cube that is a volume of obstacles is inside the robot model.

Since a cube having the state 'undecidedness' can not be determined to be in 'interference' or in 'non-interference' at the present resolution level, the cube is divided into eight subcubes and then the state of each subcube is also classified into one of the three states. Using recursively this process that is the basic process in the interference check algorithm, only cubes (regions) that intersect both the robot model and obstacles are checked at each resolution level. Consequently, the algorithm deals with only parts of obstacles which the robot model approaches.

Finally, the initial 'undecidedness' cube in the interference check algorithm is the cube (region) corresponding to the root node of the octree.

5.3.2 Classification of 'intersection' cube into three scenes

The basic process described above deals with a 'undecidedness' cube having the color 'mix' and the state 'intersection'. Here, color of a cube is originally defined in the octree. On the contrary, a state of it is changed in accordance with motion of the robot. Then, it is necessary to classify efficiently the state into 'outside', 'inside', or

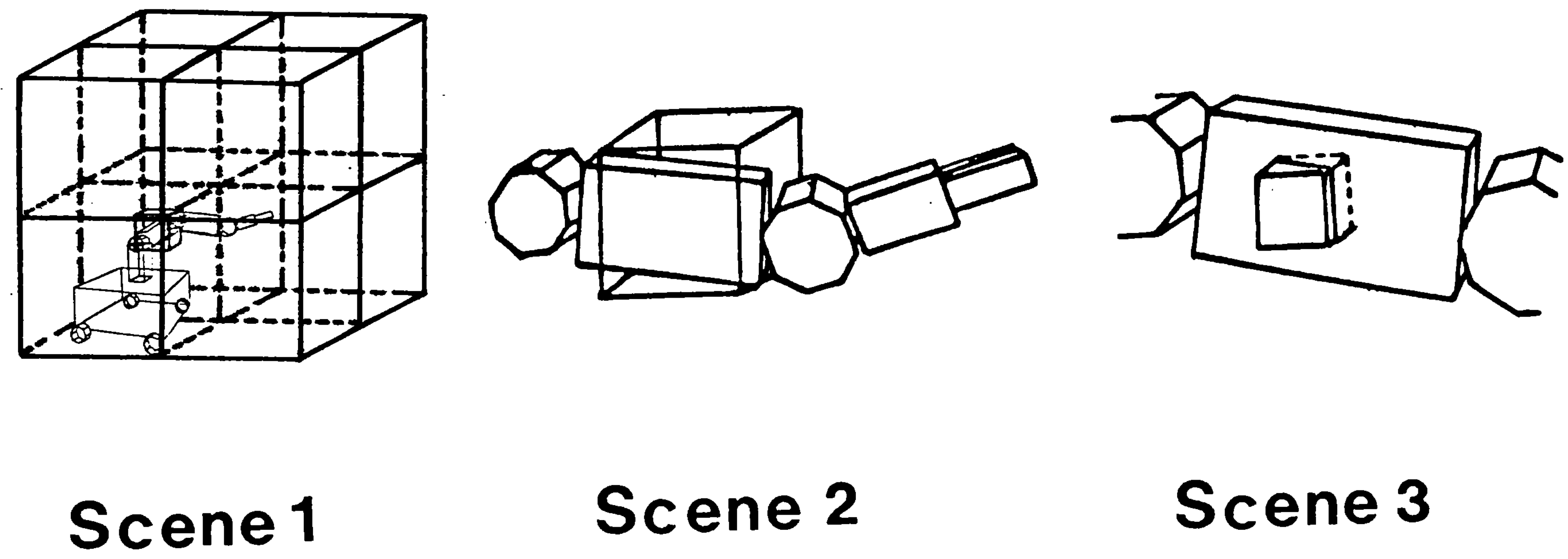


Fig.27. Three types of 'intersection' cube.

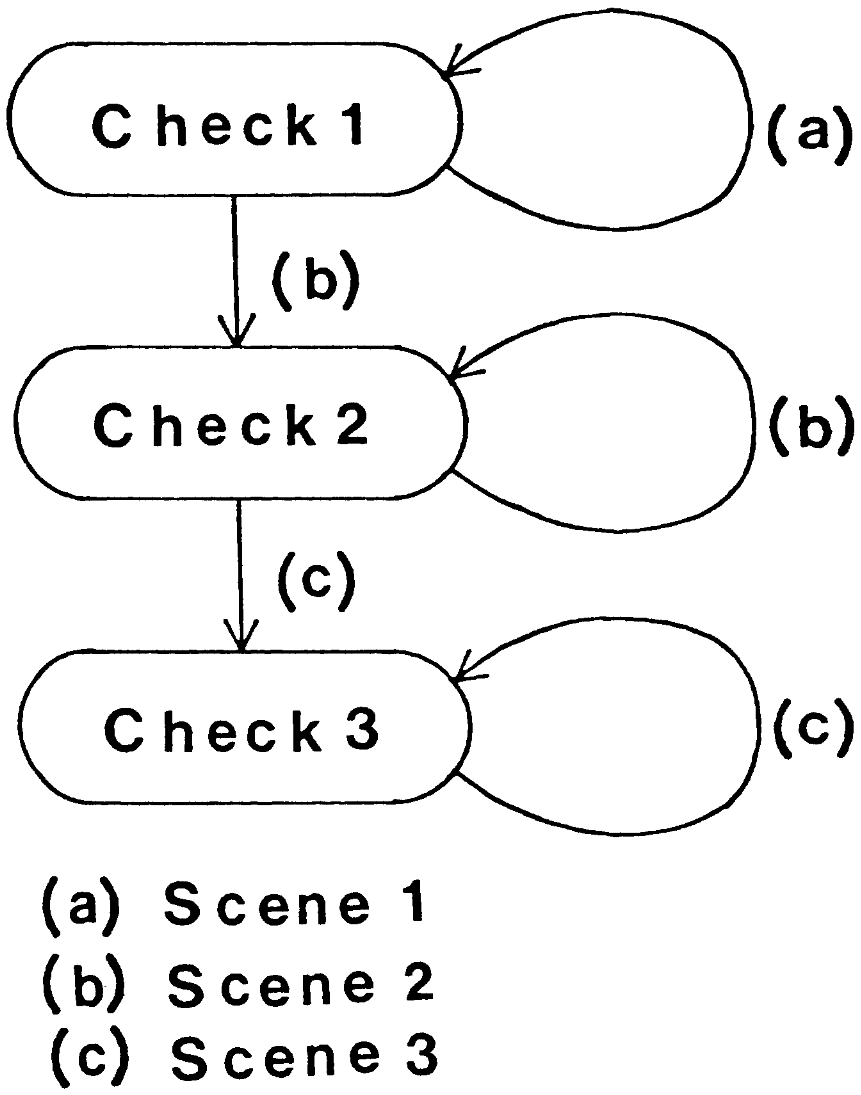


Fig.28. Process of 'undecidedness' cube in the algorithm.

'intersection'.

To make this classification run faster, an 'intersection' cube is further classified into three scenes: Scene 1 , 2, and 3 (Fig.27), and in accordance with the scene one of three procedures (Procedure 1, 2 and 3) is selected for fulfilling the basic process.

Scene 1 : The 'intersection' cube encloses a component of the robot model completely. The procedure CHECK1 deals with this cube.

Scene 2 : The cube intersects some patches of the component. The procedure CHECK2 deals with it.

Scene 3 : The cube intersects only one patch of the component. The procedure CHECK3 deals with it.

Since size of some 'intersection' cube is decreased with increasing the resolution level of the octree in progression of the algorithm, the type of the 'intersection' cube changes from Scene 1 to Scene 3. In accordance with a block diagram as shown in Fig.28, these three procedures properly dispose of 'undecidedness' cubes in the interference check algorithm.

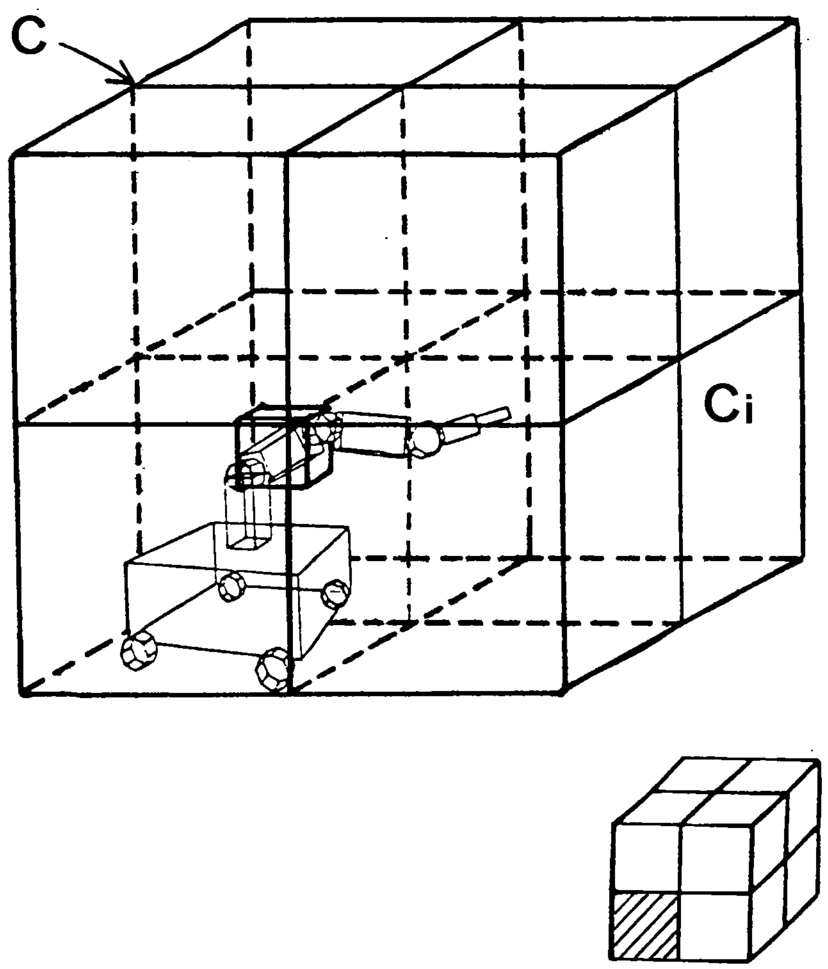


Fig.29. CHECK1 procedure.

5.3.3 Three CHECK procedures

In this paragraph, the three CHECK procedures composing of the interference check algorithm are explained in detail.

1) Procedure CHECK1: This procedure deals with an 'intersection' cube enclosing a component of the robot model (See Scene 1).

The method proposed in the MESH procedure in Chapter 3 is used for processing in this procedure (See Fig.29). Then, using this method for the component, some subcubes specified as the 'intersection' subcube and the others specified as the 'outside' subcube are produced. Therefore, the eight subcubes can be classified into 'interference', 'non-interference' or 'undecidedness' subcube with the aid of the color of eight subcubes in connection with the environment model (octree).

If the number of the 'intersection' subcube is more than one, the procedure CHECK2 deals with 'undecidedness' subcubes because they correspond to the 'intersection' cube of Scene 2. If it is one, the procedure CHECK1 deals with the 'undecidedness' subcube again because it corresponds to the 'intersection' cube of Scene 1.

2) Procedure CHECK2: This procedure deals with an 'intersection' cube intersecting some patches of a component of the robot model. By considering the component as the B-reps. defined in Chapter 2, the PLURAL procedure in Chapter 3 classifies the eight subcubes of the cube into 'inside', 'outside' or 'intersection' subcube (Fig.7). Therefore, the subcubes are also classified into 'interference', 'non-interference' or 'undecidedness'. Finally, the 'undecidedness' subcube that corresponds to Scene 3 is processed by the procedure CHECK3, and the subcube corresponds to Scene 2 is processed by the procedure CHECK2.

3) Procedure CHECK3: This procedure deals with an 'intersection' cube that intersects only a patch of a component of the robot model (Scene 3). By considering the component as the B-reps. defined in Chapter 2, the SINGLE procedure in Chapter 3 is used for processing this procedure and eight subcubes of the cube can be easily determined into 'outside', 'inside' or 'intersection' subcube (Fig.15). Then, the eight subcubes are naturally classified into 'interference', 'non-interference' or 'undecidedness' with the aid of the color of eight subcubes. Finally, the 'undecidedness' subcubes are processed by the procedure CHECK3.

5.4 Computational complexity of the algorithm

To evaluate the computational complexity of the algorithm, it is necessary to know the number of cubes which intersect the robot model at all levels of the octree. The number N_i of cubes that the algorithm deals with at some level i can be evaluated by the number N'_{i-1} of mix cubes at the level i-1, which intersect the robot model. That is, the number N_i is defined by $8*N'_{i-1}$. Then, it is shown that a number N'_i at the level i is given as surface area Si on the robot model within mix cubes (regions) corresponding to the level i.

First, each patch P of the robot model is projected onto the plane maximizing $|\vec{i}\cdot\vec{n}|$ (XY plane $[\vec{i}=(0,0,1)]$, YZ plane $[\vec{i}=(1,0,0)]$, ZX plane $[\vec{i}=(0,1,0)]$). The projecting area S'_P is evaluated as the area S_P of the patch P; The area S'_P is defined by $Sp'=Sp*\cos\theta$ ($\cos\theta = |\vec{i}\cdot\vec{n}| \ / \ |\vec{i}|*|\vec{n}|$: constant). Thus, the area S_i' of the projection for the area S_i is obtained by $S_i'=C'*S_i$ ($\sqrt{3}/3 \leq C' \leq 1$, C': constant). The area of projection (BLOCK) for a cube at some level i is defined by $S_{Bi}(=4^{nmax-i})$. As a result, a number of BLOCK regions intersecting the area S'_i is represented by S'_i/S_{Bi}. Furthermore, a maximum number of cubes that intersect a plane on the

BLOCK region is three (Appendix C).

Accordingly, the total number of cubes which are managed by the algorithm can be expressed as

$$\begin{aligned} N_{i+1} &= 8 * N'_i \\ &\leq 8 * 3(C' * S_i/S_{Bi}) \\ &= C * 4^i * S_i/4^{nmax} \quad \text{(C: constant).} \end{aligned}$$

$$nmax = \log_2[\text{(the edge length of the world space)/(the edge length of a cube at the finest resolution level)}]$$

$$T = \sum_{i=1}^{n} N_i \leq C \sum_{i=0}^{n-1} 4^i * S_i/4^{nmax}.$$

Consequently, several properties that are concerned with the computational complexity of this algorithm are as follows:

(1) In general, the area S_i is increased by letting a robot approach obstacles. The complexity depends on the area S_i and increases with decreasing the distance between the robot and its nearest obstacle.

(2) Since the algorithm can deal with only a surface area S_i on the robot model, which is within cubes (regions) that intersect both the robot model and the obstacles, a change in total calculation time is relatively little even if the resolution level of the

environment model is increased.

5.5 Experimental results

To show how calculation time of the algorithm change with the distance between a robot model and its nearest obstacle in the environment model, we present firstly an illustrative model set (Experiment A). This consists of an environment model that includes an obstacle representing a parallelepiped and a robot model with a B-reps. approximating a sphere. Secondly, to show how the algorithm work in real-time in even a cluttered environment and a complicated robot, we present a model set (Experiment B). This consists of an environment model that includes twenty-four polyhedra and a robot model with components of twelve polyhedra.

From definition of the octree, the edge length of the world space is 2^n times as long as that of the finest resolution cubes. In this example, the edge length of finest resolution cube at the level n=7 is about 2.3 cm (300cm/2^7) when the length of edge for the world space is 3m.

The proposed algorithm is implemented in C language on the Melcom 350-60 (3.7MIPS, 8MB) under Unix.

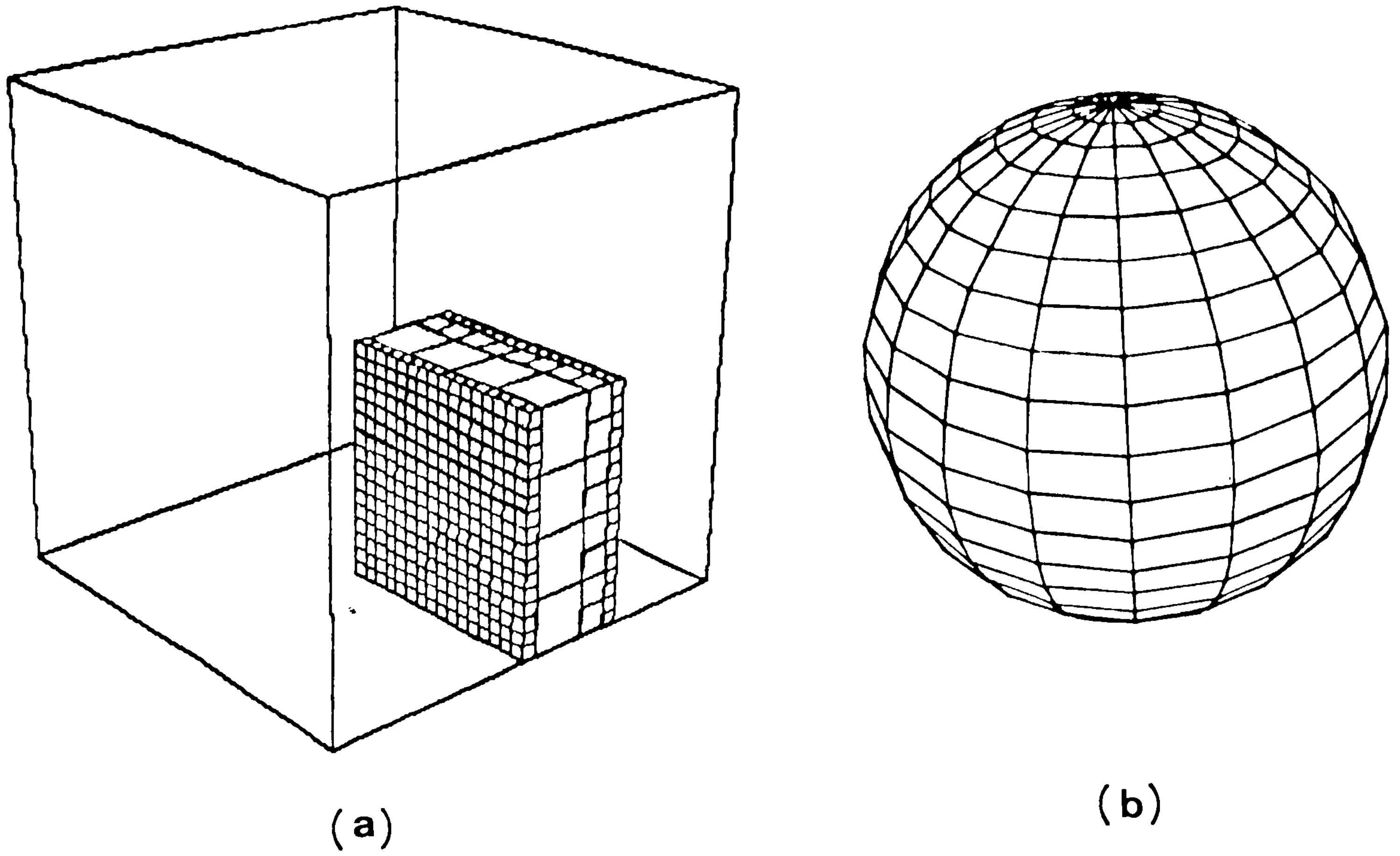


Fig.30. Experiment A. (a) Environment model.
(b) Robot model.

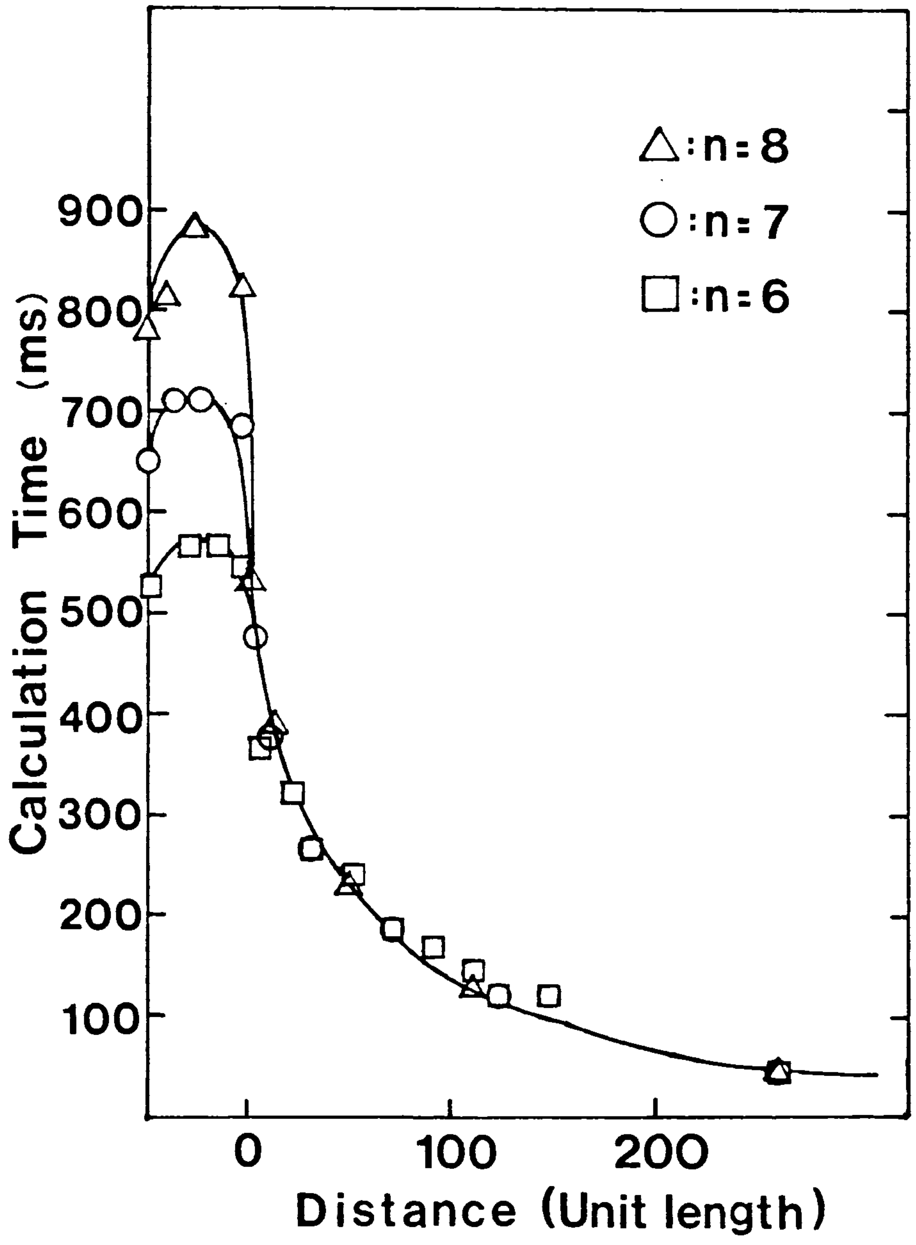


Fig.31. Calculation time of the algorithm.

5.5.1 Experiment A

a) Environment Model: This model is the octree that represents the world space [0-1024, 0-1024, 0-1024] including a parallelepiped [511-711, 0-512, 0-512] at the resolution level n=7 (See Fig.30(a)).

b) Robot Model: This model is a B-reps. with one hundred patches, which approximates a sphere whose radius is fifty (See Fig.30(b)).

Figure 31 shows a transition in calculation time when the robot model approaches the parallelepiped in the environment model. It is seen from Fig. 31 that the properties of the algorithm derived in Section 5.4 are reasonable.

5.5.2 Experiment B

a) Environment Model: This model with the octree at the resolution level n=7 includes twenty-four polyhedra having one hundred thirty-eight patches (See Fig.32(a)).

b) Robot Model: This model represents a manipulator and is composed of twelve components that are a set of polyhedra. (See Fig.32(b)).

The calculation time of the algorithm is 258 ms in the experiment of Fig.33. As a result, it is shown that this algorithm is still fast in a cluttered

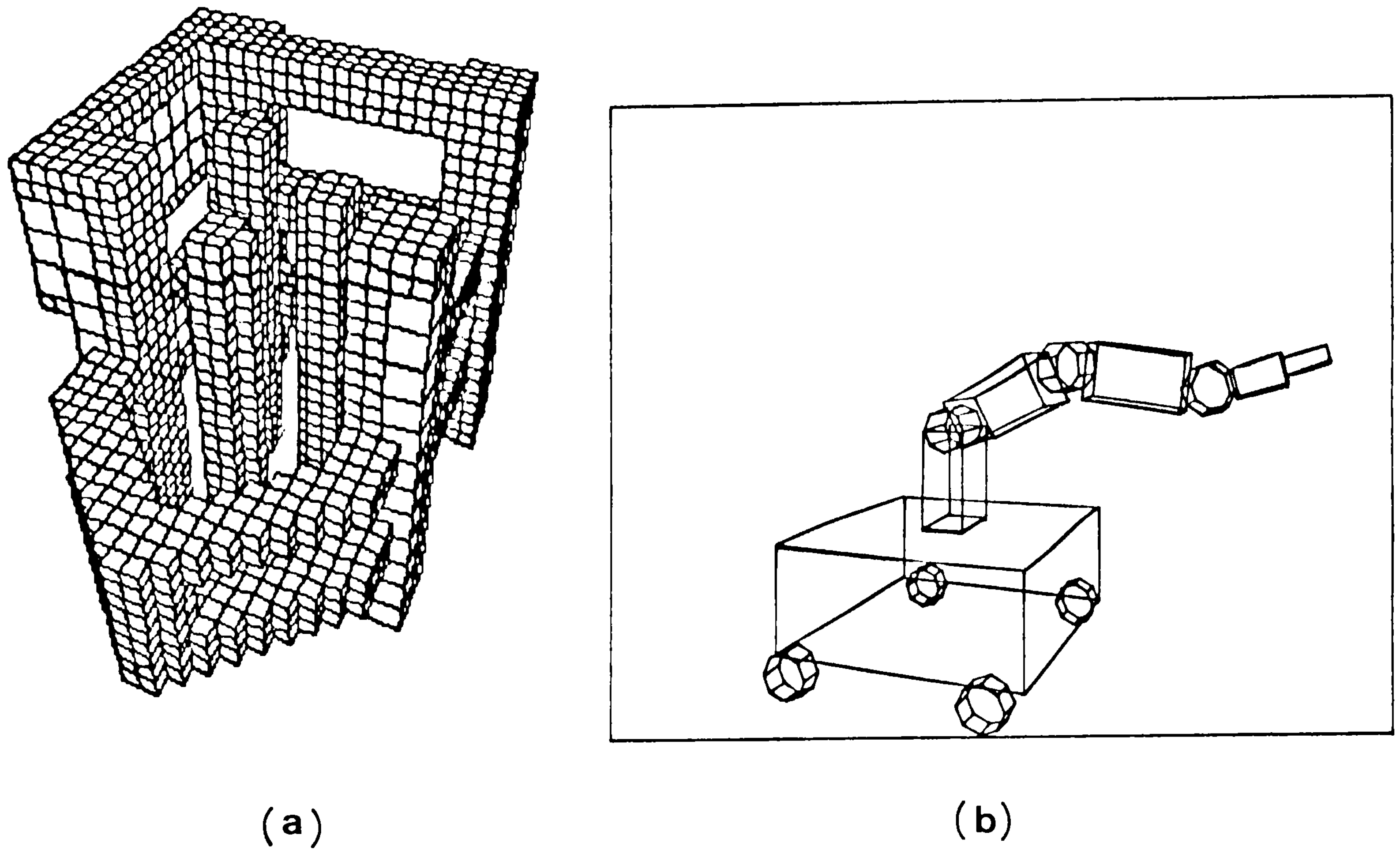


Fig.32. Experiment B. (a) Environment model.
(b) Robot model.

environment and a complicated robot.

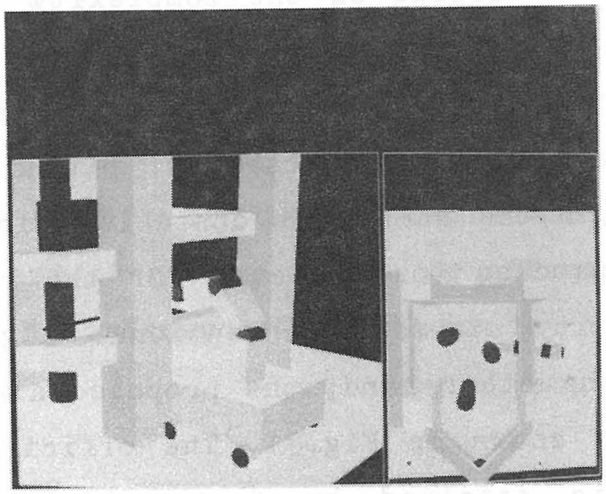


Fig.33. Manipulator in the experiment B.

5.6 Conclusion

We have proposed an interference check algorithm which does not depend on the complexity of shape of the robot and environment but on the distance between the robot and its nearest obstacle in the environment. If the number of patches for the obstacles (or the number of obstacles) is N and the number of patches (or objects) for the robot is M, the complexity of most conventional algorithms is O(NM). On the other hand, the proposed algorithm can run fast as shown in Fig.31. The efficiency of the algorithm is attained by fast selection of only regions that intersect both the robot model and obstacles with the aid of the hierarchical structure of the octree representation. Though the proposed interference checking algorithm is not a dynamic one but a static one, the algorithm seldom pass over an actual interference because the algorithm can run fast and hence is applied dynamically to the interference check every time updating coordinates table in the B-reps. that follows the movement of the robot.

By restricting each component of the robot model to be of convex shape, the basic process of the proposed algorithm classifies eight subcubes of an 'intersection' cube simultaneously, whenever they are inside, outside, or intersecting for the component (See in Chapter 3). Hence, the algorithm can be composed by a one-pass approach. In other words, it follows naturally from this property that processing in some cube (region) is independent of that in other cubes (regions). Using this fact, the algorithm can deal with a part of the component within some cube flexibly and therefore it can be effectively applied to this interference detection in the field of robotics.

CHAPTER 6
DISCUSSION AND SUMMARY

In this dissertation, three basic octree algorithms in relation to robotics and computer vision are proposed. In addition, the computational complexities of these algorithms are evaluated and reasonableness of each evaluation is ascertained by several experiments.

In the present work, the depth-first search on the octree is adopted. By virtue of the inherent characteristics of the algorithm that a processing in some cube (region) is independent of that in other cubes, the breath-first search and parallel processing can be used in implementation of the algorithm, which makes it run faster.

In Chapter 3 and 5, it should be noted that if a component is not convex then it must be divided into some convex components. This may induce artifically virtual patches. Even in that case, the algorithm is capable of maintenance of calculative efficiency by only overlooking those virtual patches generated by the division. Namely, an 'intersection' cube that intersects only some of the virtual patches is converted to the 'inside' one there and then.

In development of intelligent robots, it is a major theme for a robot manipulator and a mobile robot to avoid their obstacles in the whole space automatically [72]-[83]. This theme is called "mover's problem" or "path-planning problem". Principal methods investigated in path-planning are distinguished into the following two ways:

1) The data structure in which a path-planning algorithm searches is a mere model representing real shape of obstacles in the world space. For example, in the potential method proposed by Khatib [79], the structure is defined as simple shaped objects, which are like a parallelepiped or a cube. In this case, though the data structure is easily constructed and modified, this method can not be applied to some complicated robot or environment because the potential function corresponding to a complex object can not be defined.

2) The data structure is defined by an elaborate representation for generating efficiently some robot motions. In this case, shape of a robot and obstacles and position of an obstacle can not be modified flexibly because the structure depends on the shape and position. Furthermore, for example, the configuration approach proposed by Lozano-Perez and Brooks [72]-[77] can not be applied to a manipulator in accordance with increasing the number

of freedom of the manipulator because the storage of the data structure (configulation space) becomes large.

Consequently, we should point out the importance of a problem what path-planning must be a compromise between these two ways [84]-[86]. In this respect, roughly speaking, the level of a white node of the octree expresses the distance from the node to the nearest obstacle and therefore it is possible to derive the minimum distance between two models. As a matter of fact, this derivation makes use of a fast algorithm that calculates a minimum distance from some point to a set of black nodes in the octree and presents the black node deriving the distance. This result can be applied to the potential based path-planning for a complex robot and a cluttered environment. Under this direction, we have been exploring a new method for automatic path planning of the robot manipulator.

In Chapter 2, several solid models are explained and properties of them are discussed in relation to applicable research problems.

In Chapter 3, a fast and general conversion algorithm from the B-reps. to the octree is proposed. Summarizing and reviewing previous research works, we show the possibility of converting any solid model including the B-reps. to the octree.

In Chapter 4, a fast algorithm that reconstructs shape of a real object in computer with the aid of the octree is proposed. The computational complexity of this algorithm is proportional to some parameters concerning the precision of the approximating octree for the object. From the point of view of the computational complexity, this property of the algorithm is better than that of similar algorithms with the aid of the B-reps.

In Chapter 5, a new interference check algorithm between the robot model and the environment model in graphics simulator is explained. Due to the hierarchical structure of the environment model, this algorithm can run faster in a cluttered environment model and a complicated robot model than any other known algorithms.

APPENDIX A

EFFICIENT METHOD FOR CALCULATING FUNCTIONS OF SEVEN VERTEXES

We give the proof of the equation for the vertex in the group C when the value d_g is negative. The vector that is defined from the center point G to the vertex is represented by

$\vec{c}' = h * (n_x/|n_x|,\ n_y/|n_y|,\ n_z/|n_z|)$.

Also, the vector that is defined from the origin O of the world coordinate system to the center point G is denoted by $\vec{g}$. Thus, the vector that is defined from the origin O to the vertex is defined by

$\vec{c} = \vec{g} + \vec{c}' = g + h * (n_x/|n_x|,\ n_y/|n_y|,\ n_z/|n_z|)$.

Hence,

$$\begin{aligned}
d(S,C) &= \vec{n} \cdot \vec{c} - d \\
&= \vec{n} \cdot (\vec{g} + \vec{c}') - d \\
&= \vec{n} \cdot \vec{g} + \vec{n} \cdot \vec{c}' - d \\
&= d_g \\
&\quad + h * (n_x,\ n_y,\ n_z) \cdot (n_x/|n_x|,\ n_y/|n_y|,\ n_z/|n_z|) \\
&= d_g \\
&\quad + h * (|n_x|^2/|n_x| + |n_y|^2/|n_y| + |n_z|^2/|n_z|) \\
&= d_g + h * (|n_x| + |n_y| + |n_z|).
\end{aligned}$$

Similarly, the equations for vertexes in the groups A and B are also given by selecting each term of the vector $\vec{c}$. Moreover, by changing positive signs of vector $\vec{c}$ into negative signs, the proof of all equations when the value d_g is positive is easily given. $\square$

APPENDIX B

NUMBER OF BLOCK REGIONS INTERSECTING A PROJECTION

Lc is an edge length of the BLOCK region.

(number of BLOCK regions which intersect some edge [length li] of a projected region P')

$\leq \lceil (1+\tan\rho) * li\cos\rho/Lc \rceil + 1$

$\leq li * (\cos\rho + \sin\rho)/Lc + 2 = li * \sin(\rho + \pi/4)/Lc + 2$

$\leq \sqrt{2} * li/Lc + 2$ (ρ : the smaller angle of two angles that are constructed when the line including the edge crosses the two axes on the projecting plane).

Therefore,

(number of BLOCK regions which intersect all edges of a projecting region P')

$\leq \sum_{P'} (\sqrt{2}li/Lc + 2) = \sqrt{2}L_{P'}/Lc + 2\alpha$

(α is a number of edges) $\Box$

APPENDIX C

NUMBER OF CUBES INTERSECTING A PLANE ON THE BLOCK REGION

Without loss of generality, we define a plane equation
$f = n_x*x + n_y*y + n_z*z - d$, and two vertexes (x_0,y_0) and (x_1,y_1) of the opposite position in the BLOCK region ($n_z \geqq n_x, n_y$).

$0 = n_x*x_0 + n_y*y_0 + n_z*z_0 - d$

$z_0 = (d-n_x*x_0 - n_y*y_0)/n_z$

$0 = n_x*x_1 + n_y*y_1 + n_z*z_1 - d$

$z_1 = (d-n_x*x_1 - n_y*y_1)/n_z$

$0 \leqq | z_1 - z_0 |$

$= | (x_0 - x_1)*n_x + (y_0 - y_1)*n_y | / |n_z|$

$\leqq (|x_0 - x_1|*|n_x| + |y_0 - y_1|*|n_y|) / |n_z|$

$\leqq Lc* (|n_x| + |n_y|)/|n_z|$.

$0 \leqq | z_1 - z_0 | / Lc \leqq (|n_x| + |n_y|)/|n_z|$.

$1 \leqq$ (average number of cubes intersecting a plane S)

$= | z_1 - z_0 | / Lc + 1 \leqq (|n_x| + |n_y|) / |n_z| +1 \leqq 3$

$\square$

APPENDIX D

NUMBER OF CUBES INTERSECTING A POLYGON UNDER THREE-DIMENSIONAL SPACE

A polygon within the world space and its length are denoted by P and L, respectively. Some segments of the polygon and their lengths are also denoted by Pi and Li (i=1,⋯,n), respectively. Without loss of generality, the straight line m including the segment Pi is given by $m = p_1 + \vec{n}*t$ ($\vec{n} = (n_x, n_y, n_z)$, $(n_z \leqq n_y \leqq n_x)$; $\vec{n}$: vector on the straight line, t: one parameter).

First, we assume that the line m intersects a cube with one vertex $p_1=(x_0, y_0, z_0)$ and another point $p_2=(x_1, y_1, z_1)$ and then the parameter t in the point p_2 is denoted by t_1. An edge length of the cube can be represented by Lc ($Lc = 2^{nmax-j}$; j: level of the cube in the octree). Then, the following equations are obtained:

$Lc = |x_0 - x_1| = |n_x|*|t_1|$, $|t_1| = Lc / |n_x|$.

$0 \leqq |z_1 - z_0| = |n_z|*|t_1| = |n_z| * Lc / |n_x|$.

Here, the projection of the cube onto XY-plane is called BLOCK region. The number of cubes intersecting the segment Pi, whose projections agree with the BLOCK region, can be represented by

$|z_1-z_0|/Lc+1$. As a result, maximum of this number is two.

Next, when a segment Pi is also projected onto the XY-plane, the length L' of the projecting segment Pi' is denoted by $Li'=Li*\cos\theta$, $(\cos\theta=(n_x{}^2+n_y{}^2)/(n_x{}^2+n_y{}^2+n_z{}^2))$.

Therefore,

(Number of BLOCK regions intersecting the projection Pi')

$\leqq \lceil(1+\tan\rho)*Li'\cos\rho/Lc\rceil + 1 \leqq Li'*(\cos\rho + \sin\rho)/Lc+2$
$= Li'*\sin(\rho+\pi/4)/Lc+2 \leqq \sqrt{2}Li'/Lc+2 \leqq C'*Li/Lc$

(ρ: the smaller angle of two angles that are constructed when the line including the projection Pi' crosses the X and Y axes).

As a result, number of cubes which intersect the segment Pi is bounded by $2(C'*Li/Lc)$ and then number of cubes which intersect the polygon P is also bounded by $C*2^jL/2^{nmax}$ (C, C': constant)$\square$.

BIBLIOGRAPHY

[1] A. A. G. Requicha and H. B. Voelcker, "Solid modeling: Current states and research directions," IEEE Computer Graphics and Applications, vol.3, no.7, pp.25-37, 1983.

[2] A. A. A. Requicha, "Representation for rigid solids: Theory, methods and systems," Comput. Surveys, vol.12, no.4, pp. 437-464, Dec. 1980.

[3] H. Voelcker and A. A. G. Requicha, "Geometric modeling of mechanical parts and processes," IEEE Computer, vol.C-10, no.12, pp.48-57, Dec. 1977.

[4] C. L. Jackins and S. L. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," Computer Vision, Graphics, Image Process., vol.14, no.3, pp.249-270, Nov. 1980.

[5] J. K. Warnock, "A hidden surface algorithm for computer generated half tone pictures," TR4-15, Computer Science Dept., University of Utah, Salt Lake City, June 1969.

[6] M. D. Kelly, "Edge detection in pictures by computer using planning," Mach. Intell., vol.6, pp.397-409, 1971.

[7] S. L. Tanimoto and T. Pavilidis, "A hierarchical data structure for picture processing," Computer Vision,

Graphics, Image Process., vol.4, no.2, pp.104-119, June 1975.

[8] J. L. Bentley, "Multidimensional divided-and--conqure," Commun. ACM, vol. 23, no.4, April 1980.

[9] S. L. Tanimoto and A. Klinger, Eds., Structured Computer Vision, Academic Press, New York 1980.

[10] M. M. Yau and S. N. Srihari, "Recursive generation of hierarchical data structures for multidimensional digital images," Technical Report no.170, Dept. of Computer Science, State University of New York at Buffalo, January 1981.

[11] S. N. Srihari, "Representation of three--dimensional digital images," ACM Comput. Surv., vol.13, no.4, pp.400-424, 1981.

[12] N. Ahuja, "On approaches to polygonal decomposition for hierarchical image representation," Computer Vision, Graphics, Image Process., vol.24, pp.200-214, 1983.

[13] C. L. Jackins and S. L. Tanimoto, "Quadtrees, Octrees, and K-trees: A generalized approach to recursive decomposition of euclidian space," IEEE Trans. Pattern Analysis Machine Intelligence, vol. PAMI-5, no.5, pp.533-539, 1983.

[14] H. Samet, "The quadtree and related hierarchical data-structures," ACM Computing Surveys, vol.16, no.2, pp.187-260, June 1984.

[15] L. J. Doctor and J. G. Torborg, "Display

techniques for octree-encoded objects," IEEE Computer Graphics and Application, vol.1, no.1, pp.39-46, July 1981.

[16] J. T. Kajiya, "New techniques for ray-tracing procedurally defined object," Computer Graphics, vol.17, pp.91-99, 1983.

[17] A. S. Glassner, "Space subdivision for fast ray tracing," IEEE Computer Graphics and Application, vol.4, pp.15-22, 1984.

[18] A. Rosenfeld, Tree structures for region representation, Computer Vision Laboratory, University of Maryland, 1979.

[19] A. Rosenfeld, "Quadtrees and pyramids for pattern recognition and image processing," Proceedings of the 5th International Conference on Pattern Recognition, Dec. 1980.

[20] T. Pavlidis, Structural Pattern Recognition. Berlin: Springer-Verlag, 1977.

[21] Y. T. Lee and A. A. G. Requicha, "Algorithms for computing the volume and other integral properties of solids. I. Known method and open issues," Commun. ACM, vol.25, no.9, pp.635-641, Sept. 1982.

[22] Y. T. Lee and A. A. G. Requicha, "Algorithms for computing the volume and other integral properties of solids. II. A family of algorithms based on representation conversion and cellular approximation,"

Commun. ACM, vol.25, no.9, pp.642-650, Sept. 1982.

[23] Y. Ohta and T. Kanade, "Stereo by two-level dynamic programming considering inter-scanning consistency," J. Information Processing Society of Japan, vol.2, pp.1356-1363, 1985. (in Japanese)

[24] D.Marr, "Early processing of visual information," Phil. Trans. R. Soc. Lond., vol.B275, pp.483-524 1976.

[25] M.M.Thompson ed., Manual of photogrammetry (third edition), American Society of Photogrammetry 1966.

[26] B. Horn, "Obtaining shape from shading information," in The Psychology of Computer Vision, P. H. Winston (ed.), McGraw-Hill, New York. 1975.

[27] A. Rosenfeld and A. C, Kak, Digital Picture Processing, Academic Press, New York, 1982.

[28] D. Mergher, "Octree Encoding: A new technique for the representation, manipulation and display of arbitrary 3-D objects by computer," Technical Report IPL-TR-80-111, Image Processing Laboratory, Rensselaer Polytechnic Institute, October 1980.

[29] D. Mergher, "Geometric modeling using octree encoding," Computer Vision, Graphics, Image Process., vol.19, no.2, pp.129-147, June 1982.

[30] G. M. Hunter and K. Steiglitz, "Operations on images using quadtrees," IEEE Trans. Pattern Analysis and Machine Intelligence, vol.PAMI-1. no.2, April

1979.

[31] G. M. Hunter, "Efficient computation and data structure for graphics," Ph.D. dissertation, Dept. of Electrical Engineering and Computer Science, Prinston University, 1978.

[32] H. Samet, "A top-down quadtree traversal algorithm," IEEE Trans. Pattern Analysis, Machine Intelligence, vol.7, no.1, Jan. 1985.

[33] H. Samet, "Neighbor finding techniques for images represented by quadtrees," Computer Vision, Graphics, Image Process., vol.18, no.1, pp.37-57, Jan. 1982.

[34] H. Samet, "Connected component labeling using quadtrees," J. ACM, vol.28, no.3, pp.487-501, July 1981.

[35] H. Samet, "Distance transform from images represented by quadtrees," IEEE Trans. Pattern Anal. Mach. Intell., vol.4, no.3, pp. 298-303, May 1982.

[36] K. Yamaguchi, T. L. Kunii, and K. Fujimura, "Octree-related data structure and algorithms," IEEE Computer Graphics and Applications, vol.4, no.1, pp.53-59, Jan. 1984.

[37] I.Gargantini, T.R.Walsh, and O.L.Wu, "Viewing transformations of voxel-based objects via linear octrees," IEEE Comput. Graph. Appl., vol.6, no.10, pp.12-21, Oct. 1986.

[38] T. L. Kunii and T. Satoh, "Generation of

topological boundary representations from octree encoding," IEEE Computer Graphics and Application, vol.5, no.5, pp.29-38, March 1985.

[39] M. M. Yau, "Generating quadtrees of cross sections from octrees," Comput. Vision, Graphics, Image Process., vol.27, pp.211-238, Aug. 1984.

[40] M. Yau, "Hierarchical representation of three--dimensional digital objects," Ph.D. diss., Dept. of Computer Science, State University of New York at Buffalo, Jan. 1983.

[41] D. Meagher, "The solids engine : a processor for interactive solid modeling," Proc. NICOGRAFH '84 Conf, 1984.

[42] E. Kawaguchi and T. Endo, "On a method of binary picture representation and its application to data compression," IEEE Trans. Pattern Analysis, Machine Intelligence, vol. PAMI-2, no.1, pp.27-35, Jan. 1980.

[43] E. Kawaguchi, T.Endo, and J. Matsunaga, "Depth--first expression viewed from digital picture precessing," IEEE Trans. Pattern Analysis, Machine Intelligence, vol. PAMI-5, no.4, pp.27-35, July 1983.

[44] M. Mäntylä and M. Tamminen, "Localized set operations for solid modeling," Computer Graphics, vol.17, no.3, 1983.

[45] M. Mäntylä, "Inversion algorithm for geometric

models," Computer Graphics, vol.16, no.3, July 1982.

[46] H.B.Wilson and D.S.Farrior, "Computation of geometrical and inertial properties for general areas and volumes of revolution," Computer Aided Design, vol.8, no.4, pp.257-263., 1976.

[47] N. Okino, Y. Kakazu, and H. Kubo, "TIPS-1: Technical information processing system for computer-aided design, drawing and manufacturing," In Computer Languages for Numerical Control, J. Hatvany, Ed., North-Holland, Amsterdam, pp.141-150, 1973.

[48] J.L.Bentley and T.A.Ottman, "Algorithms for reporting and counting geometric intersections," IEEE Trans. Computer, vol.C-28, no.9, pp.643-647, Sept. 1979.

[49] J.L.Bentley and D.Wood, " An optimal worst case algorithm for reporting intersections of rectangles," IEEE Trans. Computer, vol.C-29, no.7, pp.571-576, July 1980.

[50] R.B.Tilove, "A null-object detection algorithm for constructive solid geometry," Commun. ACM, vol.27, no.7, July 1984.

[51] W. R. Franklin and V. Akman, "Building an octree from a set of parallelepiped," IEEE Computer Graphics and Application, vol. 5, no. 10, pp. 58-64, 1985.

[52] M. Yau and S. N. Srihari, "A hierarchical data

structure for multidimensional images," Comm. ACM, vol.26, no.7, pp.504-515, July 1983.

[53] H. Samet and M. Tamminen, "Bintree, CSG trees and time," Computer Graphics (Proc. SIGGRAPH '85 Conf.) vol.19, no.3, pp. 121-130, 1985.

[54] M. Tamminen and H. Samet, "Efficient octree conversion by connectivity labeling," Computer Graphics (Proc. SIGGRAPH '84 Conf.), vol.18, no.3, pp.43-51, July 1984.

[55] W. N. Martin and J. K. Aggarwal, "Volumetic descriptions of objects from multiple views," IEEE Trans. Pattern Analysis Machine Intelligence, vol.PAMI-5, pp.150-158, Mar. 1983.

[56] M. I. Shamos, " Computational Geometry", Ph.D Thesis, Yale University 1978.

[57] T. Pavlidis, Algorithms for Graphics and Image Processing. Rockville, MD: Computer Science Press, Ch.15 1982.

[58] T. Asano, "Computational geometry and its applications," Journal of Information Processing Society, vol.25, no.3, pp.208-211, Mar. 1984. (in Japanese)

[59] T. Miyake and J. Doi, "Three-dimensional shape approximation system with a polyhedron," Information Processing Society of Japan, vol.25, pp.745-754, Sept. 1984. (in Japanese)

[60] Y. C. Kim and J. k. Aggarwal, "Rectangular parallelepiped coding : A volumetric representation of three-dimensional objects," IEEE Journal of Robotics and Automation, vol.RA-2, pp.127-134, 1986.
[61] Tsai-Hong Hong and M.O. Shneier, "Describing a robot's work space using a sequence of views from a moving camera," IEEE Trans. Pattern Analysis, Machine Intelligence, vol. PAMI-7, no. 6, pp.721-726, 1985.
[62] E.W.Kent, M.O.Shneier and T-H Hong, "Building representations from fusions of multiple views," IEEE International Conf. on Robotics and Automation, San Francisco, CA, pp.1634-1639, April 1986.
[63] J. W. Boyes, "Interference detection among solids and patches," Commun. ACM, vol.22, no.1, pp.3-9., Jan. 1979.
[64] Y. Mizugaki, et al., "A simplified method for collision detection based on geometric modeling", Journal of the Japan Society of Precision Engineering, vol. 50, no. 6, pp. 944-950, 1984. (in Japanese)
[65] Y. Shigematsu et. al., "Interference detection algorithm by simplex method," Journal of the Japan Society of Precision Engineering, vol.49, no.11, pp.1561-1566, 1983. (in Japanese)
[66] H. Ozaki, et. al., "Planning of collision free movement of manipulator considering its body space," Journal of the Society of Instrument and Control

Engineers, vol.21, no.9, pp. 942-949, Sept. 1982. (in Japanese)

[67] H. Ozaki, "Collision free movement of manipulators," Journal of the Robotics Society of Japan, vol.2, no.6, pp.580-586, 1983. (in Japanese)

[68] K. Ozawa, et al., "A fast interference check method in off-line robot teaching", Journal of the Robotics Society of Japan, vol.4, no.2, pp.5-14, April 1986. (in Japanese)

[69] N. Ahuja and C. Nash, "Octree representation of moving objects," Computer Vision, Graphics, Image Process., vol.26, no.2, May 1984.

[70] J. Weng and N. Ahuja, "Octree representation of objects in arbitrary motion," Proc. IEEE Conf. on Computer Vision and Pattern Recognition, San Francisco, CA, pp.524-529, June 1985.

[71] V. Hayward, "Fast collision detection scheme by recursive decomposition of a manipulator workspace" IEEE International Conf. on Robotics and Automation, San Francisco, CA, vol. 2, pp.1044-1049, April 1986.

[72] T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," IEEE trans. on Systems, Man and Cybernetics, vol. SMC-11, pp.681-698, 1981.

[73] T. Lozano-Pérez, "Spatial-planning: a configuration space approach," IEEE Trans. on Computers, vol.C-32, no.2, Feb. 1983.

[74] T.Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral objects," Commun. ACM, vol.22, no.10, pp.560-570, Oct. 1979.

[75] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for finding with rotation," M.I.T. Artificial Intell. Lab., Rep. AIM-684, Dec. 1982.

[76] R. A. Brooks, "Solving the find-path problem by good representation of free space," IEEE Trans. on Systems, Man amd Cybernetics, vol.SMC-13, no.3, April 1983.

[77] R. A. Brooks, "Planning collision-free motions for pick-and place operations", International Journal of Robotics Research, vol.2, no.4, Winter 1983.

[78] S. M. Udupa, "Collision detection and collision avoidance in computer controlled manipulators," Proceedings of the Fifth IJCAI, MIT, Cambridge Massachusetts, August 1977.

[79] O. Khatib and J.-F.Le Maitre, "Dynamic control of Manipulators operating in a complex environment," On Theory and Practice of Robots and Manipulators, 3rd CISM-IFToMM Symp., pp.267-282, 1978.

[80] J. T. Schwartz and M. Sharir, "On the piano mover's problem: I. The special case of a rigid polygonal body moving amidst polygonal borriers,"

Commun. Pure Appl. Math., vol.36, pp.345-398, 1983.

[81] J. T. Schwartz and M. Sharir, "On the piano mover's problem: II. General techniques for computing topological properties of real algebraic maifolds," Adv. Appl. Math., in press, 1983.

[82] H.P.Moravec, "Obstacle avoidance and navigation in the real world by a seeding robot rover.," Ph.D. dissertation, Stanford University Rept. no. AIM-340. Stanford, Calif.: Stanford University Artificial Intelligence Laboratory.

[83] J. Reif, "Complexity of the mover's problem and generalizations," Proc. 20th IEEE Symp. Found. Comput. Sci. New York: IEEE, pp.241-247, 1979.

[84] S.K.Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," IEEE Journal of Robotics and Automation, vol.RA-2, no.3, Sept. 1986.

[85] M. Herman, "Fast path planning in unstructured dynamic, 3-D worlds," Procs. SPIE Applications of Artificial Intelligence III, April. pp.505-512, 1986.

[86] M. Herman, "Fast three-dimensional collision-free motion planning," IEEE International conf. on Robotics and Automation, San Francisco, CA, pp.1056-1063, April 1986.

[87] N. J. Nillsson, Problem-solving methods in artificial intelligence, New York: McGraw-Hill, 1971.

[88] B. Faverjon, "Obstacle avoidance using an octree

in the configuration space of a manipulator," IEEE First Symposium on Robotics, Atlanta, June 1984.

PAPERS RELATED TO THIS DISSERTATION

[1] H. Noborio, S. Fukuda, and S. Arimoto, "A new interference check algorithm using octree," in the 1987 IEEE Int. Conf. Robotics and Automation, to be appeared.

[2] H.Noborio, S.Fukuda, and S.Arimoto, "A new interference check algorithm using octree," submitted to the Journal of the Robotics Society of Japan. (in Japanese)

[3] H. Noborio, S. Fukuda, and S. Arimoto, "A new interference check algorithm using octree," submitted to the IEEE Journal of Robotics and Automation.

[4] H.Noborio, S.Fukuda, and S.Arimoto, "Conversion algorithm from the boundary representation to the octree and its complexity," submitted to the CG International '87.

[5] H.Noborio, S.Fukuda, and S.Arimoto, "Conversion algorithm from the boundary representation to the octree and its complexity," submitted to the IEEE Trans. on Pattern Analysis and Machine Intelligence.

[6] H.Noborio, S.Fukuda, and S.Arimoto, "Conversion algorithm from the boundary representation to the octree and its complexity," submitted to the Information Process. Society of Japan. (in Japanese)

[7] H.Noborio, S.Fukuda, and S. Arimoto, "Construction of the octree approximating a three-dimensional object by using multiple views," submitted to the Special Issue on IEEE Trans. on Pattern Analysis and Machine Intelligence on Industrial Machine Vision and Computer Vision Technology.

[8] H.Noborio, S.Fukuda, and S. Arimoto, "Construction of the octree approximating a three-dimensional object by using multiple views," submitted to the Information Process. Society of Japan. (in Japanese)