Fall 1-31-1997

Application of geometric hashing techniques to retrieval of high dimensional objects in scientific databases

Joyce Ye Lu
New Jersey Institute of Technology

APPLICATION OF GEOMETRIC HASHING TECHNIQUES TO RETRIEVAL OF HIGH DIMENSIONAL OBJECTS IN SCIENTIFIC DATABASES

by
Joyce Ye Lu

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

January 1997

APPROVAL PAGE

APPLICATION OF GEOMETRIC HASHING TECHNIQUES TO RETRIEVAL OF HIGH DIMENSIONAL OBJECTS IN SCIENTIFIC DATABASES

Joyce Ye Lu

Dr. Jason T.L. Wang, Thesis Advisor — Date
Associate Professor of Computer and Information Science,
NJIT, Newark, NJ

Dr. Peter Ng, Committee Member — Date
Chairman and Professor of Computer and Information Science,
NJIT, Newark, NJ

Dr. Marvin Nakayama, Committee Member — Date
Assistant Professor of Computer and Information Science,
NJIT, Newark, NJ

BIOGRAPHICAL SKETCH

Author:	Joyce Ye Lu
Degree:	Master of Science
Date:	January 1997

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology,
  Newark, New Jersey, 1997
- Bachelor of Science in Computer Science,
  Shanghai Jiao Tong University,
  Shanghai, P.R.China, 1994

Major: Computer Science

ABSTRACT

APPLICATION OF GEOMETRIC HASHING TECHNIQUES TO RETRIEVAL OF HIGH DIMENSIONAL OBJECTS IN SCIENTIFIC DATABASES

by
Joyce Ye Lu

An approach to designing very fast algorithms for tackling the problem of approximate object matching in very large databases of high-dimensional objects is proposed. Given are a target object C and a database D containing information about a set of high-dimensional objects each of which is represented as a set of points. Our algorithms have an off-line object preprocessing (shape representation) phase and a recognition phase. The described algorithms determine those objects from D which are the closest to object C, according to delete or insert some points, move and rotation. All of these can be achieved very efficiently with the help of geometric hashing techniques. This scheme has been successfully applied to a real scientific database.

This thesis is dedicated to
my parents

ACKNOWLEDGMENT

The author wishes to express her sincere gratitude to her supervisor, Professor Jason T.L. Wang, for his guidance, encouragement and support throughout this research. Special thanks to Professors Peter Ng and Marvin Nakayama for serving as members of the committee. And Finally, a thank you to Michael Luo for his suggestions and expert assistance throughout this project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

As existing molecular information repositories need to be processed more rapidly, and a greater variety of tools become available, the computer plays an increasingly important role in directing and streamlining the drug discovery and design process.

Computer can help researchers to quickly eliminate a priori unlikely candidates, thus avoiding long and expensive activity screenings. More important, they can allow researchers to identify new promising compounds based only on the available information on the receptor site, or on other lead compounds (see [10]).

To this date, hundreds of protein structures have been determined via X-ray crystallography and nuclear magnetic resonance (NMR) methods. These data are readily available as public resource of molecular structure data and allow pharmacologists and biologists to investigate various aspects of protein structures and their complex behaviors. In addition to these public databases, a number of other (public and proprietary) databases of small organic molecules have been assembled through the efforts of numerous pharmaceutical and biotechnology companies, and research organizations.

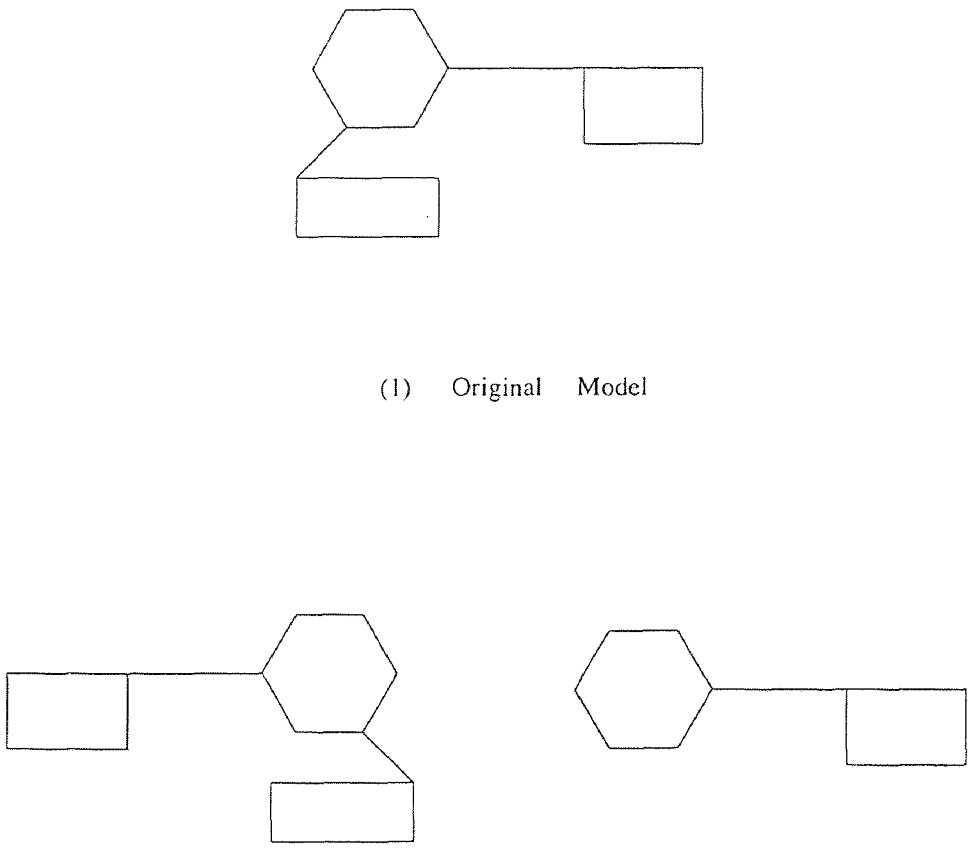
In many cases, the critical information that enables researchers to develop hypotheses, concerning potentially new molecular candidates for synthesis and testing, must be recovered through a search in a potentially very large database of relevant information. Indeed, the underlying common element to several stages of medicinal chemistry investigation requires searching of chemical information databases.

Given an object C and a database D containing information about the high-dimensional structures of a possibly large set of objects, the following operations need to be defined and carried out:

1. "structure insertion": the ability to incorporate all available structural knowledge about object C in the database D;
2. "structure membership": determination of whether the object C is already included in the database D;
3. "substructure search": identify and report all the member objects from D that contain a particular substructure of object C;
4. "similarity search": identify and report all the member objects from D that are similar to object C.

From all above, the most frequently used are substructure search and similarity search. For a substructure search the bit screen of the hit must contain exactly all of the query bits. Thus, the substructure search may not identify structures with minor deviations from the query structure. It is for this reason that similarity searches have been developed. Figure 1.1 illustrates the difference in results of a substructure search and a similarity search.

Before concluding this chapter a final distinction should be noted. This is the distinction between *identification* and *recognition* of those objects from the database D which are similar to the given target object C. *Identification* restricts itself to reporting only the identities of the objects from database D that match the target object C. On the other hand, *recognition* entails not only the reporting of the identities of the matching objects but also the determination and reporting of the necessary transformations that will bring each of the identified matching objects to best registration with the target object. Recognition is arguably a much more difficult problem than identification. This is particularly evident in the case of very large databases. In this thesis, we will deal with recognition.



(1) Original Model

(2) Result of Similarity Search

(3) Result of Substructure Search

Figure 1.1 Substructure Search vs Similarity Search

CHAPTER 2
HASHING CONCEPTS

Webster's dictionary defines the word "hash" as a verb *"to chop (as meat and potatoes) into small pieces"*. Strange as it may sound, this is correct. Basically, hashing allows us to chop up a big table into several small subtables so that we can quickly find the information once we have determined the subtable to search. This determination is made using a mathematical function, which maps the given key to hash cell i as shown in Figure 2.1. The cell i could then point us to the subtable of size n_i. Given a trace of R frames with N distinct addresses and a hash table of M cells, the goal is to minimize the average number of lookups required per frame (see [2]).

If we perform a regular binary search through all N addresses, we need to perform $1+\log_2(N)$ or $\log_2(2N)$ lookups per frame. Given an address that hashes to the ith cell, we have to search through a subtable of n_i entries, which requires only $\log(2n_i)$ lookups. The total number of lookups saved is

$$\sum_i r_i[log_2(2N) - log_2(2n_i)]$$

where r_i is the number of frames that hash to the $i$the cell, $\sum r_i = R$. The net saving per frame is

$$\sum_i(-r_i/R)log_2(n_i/N) = \sum_i -q_i log_2 p_i$$

Here, $q_i = r_i/R$ denotes the fraction of frames that hash to the ith cell, and $p_i = n_i/N$ is the fraction of addresses that hash to the ith cell. The goal of a hashing function is to maximize the quantity $\sum -q_i log_2 p_i$. Notice that p_i and q_i are not related. In the special case of all addresses being equally likely to be referenced, q_i is equal to p_i and the expression $\sum -p_i log_2 p_i$ would be called the *entropy* of the hashing function. It is because of this similarity that we will call the quantity $\sum -q_i log_2 p_i$ the entropy or *information* content of the hashing function. It is measured in units of "bits".

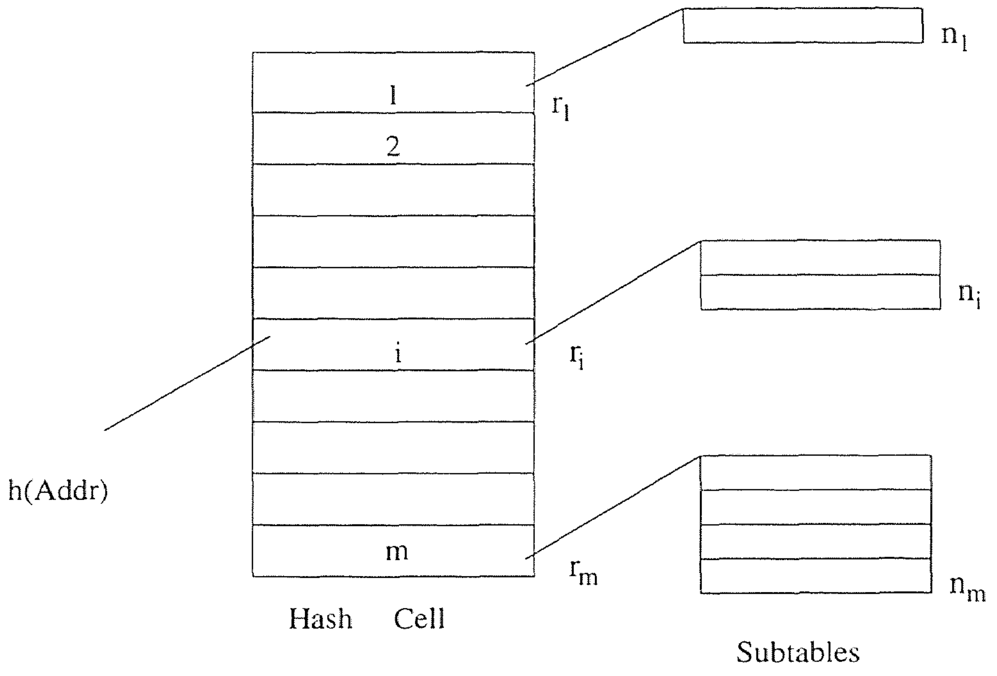


Figure 2.1 Hashing Concepts

CHAPTER 3

DESCRIPTION OF GEOMETRIC HASHING TECHNIQUES

3.1 General Introduction

A general object recognition scheme, geometric hashing, is used in this thesis. It is general in the sense that it applies to all important object recognition tasks. It creates a uniform approach for dealing with various types of objects. Another important merit of the method is its efficiency (see [1]).

The objects are represented as sets of local features, such as points or lines. The only important requirement from these features is invariance under the object transformation. The transformation invariant geometric relations among the object features are encoded, using minimal feature subsets as reference. This is achieved by standard methods of analytic geometry, invoking coordinate frames based on these minimal feature subsets. The minimal feature subsets constitute coordinate frames, in which other features are represented by their transformation invariant coordinates.

The matching procedure has two major steps. The first one precompiles the representations of the database objects, resulting in a hash-table based on these representations. This step is executed off-line on the database objects and is independent of the next phase of the algorithm. The second step, recognition proper, is executed on the target object using the previously prepared hash-table for fast on-line recognition. The hash-table serves as an associative memory, allowing for fast retrieval of 'similar' object feature subsets, and hence effectively prunes the space of 'candidate' object feature subsets.

The proposed recognition method has some major merits. First, it suggests a unified framework for coping with the object recognition problem under all feasible object transformation types. Second, it stands out for its efficiency in performing on-line recognition. Thus, it creates a good base for a general and practical object recognition system.

3.2 Features for Object Representation

Object representation plays an important role in every recognition scheme. Since object-based recognition system match stored objects against the target, the representation should include enough information to drive this matching process. It should be rich enough to enable discrimination between similar objects, which suggests the *completeness* of the representation. On the other hand, the representation should be succinct to allow efficient recognition. One wants to be able to match the objects against the target using some minimal characteristic information, so that the inherent complexity of the matching algorithm will be small.

One way to represent objects is by selected points. These points can be derived by any desired operator, provided that this operator is transformation invariant. We term these points as *interesting points* (see [3]). The method of *interesting points* extraction might vary with the type of objects in the database (see [4]).

In the implementation of geometric hashing, points were one of the main features used for matching. In the two-dimensional affine matching case, the object representation included a set of *interesting points*, which are invariant under this transformation. They were derived as points on the boundary curves of the objects, having sharp convexities and deep concavities. For recognition of nonconvex bodies, points induced by *concavity entrances* are used for object format.

When the objects under consideration are polyhedral, or can be well approximated by polyhedra, a representation by lines is effective. There are some important characteristics for the representation by lines. First, lines are invariant under any transformation type. Second, lines are, usually, more stable than points, and hence line matching can yield better recognition results.

Many objects can be described by sets of the so called *characteristic curves* (see [5]). These curves can be, for example, boundary curves of planar rigid bodies, or even artificial curves depicted on outer surfaces of objects. Such curves can be efficiently used for recognition under the similarity transformation, even in cluttered images. This is because under such a transformation angles and distance ratios are preserved (i.e. the shape changes up to a scale factor) (see [6]).

One can often find several types of suitable features in the objects. These can be a combination of points, lines and curve segments. If these features are all likely to appear in the target, one should include in the object representation the union of all the features.

In the above we described various suitable features for shape representation. Naturally, one might choose different features (or combination of features), depending on the nature of the objects in the database. However, both issues of completeness and efficiency of the representation have to be addressed.

When considering representation by the above described local features (points, lines, curve segments, etc.) one achieves two main goals. First, such a representation is invariant under the object transformation. Second, it creates a compact representation of the objects, which will later allow for efficient matching. The main drawback of representation by local features is its incompleteness. Usually, objects cannot be fully described by a set of local features. To overcome this problem, the object representation in geometric hashing utilizes two forms of data. One form is local object features, which are used in the matching procedure. In addition, a full description of the object is also stored. For example, the boundary curves of planar rigid bodies completely describe their shapes, while a wire frame model fully describes polyhedral objects. The local features are used for matching properly. The additional complete representation is used after the matching step for verification and further refinement of the objects' localization. This allows geometric hashing to achieve both efficient matching and reliable verification.

3.3 General Framework

The general geometric hashing algorithm can be summarized in the following steps (see Figure 3.1) (see [1]):

Preprocessing of model objects

For each database object :

1. Extract a set of *interesting features* (points, lines or other suitable features) from the database object. Denote their number by m.

2. Determine the minimal number of features which can serve as a basis for a coordinate frame, allowing expression of all other features by *transformation invariant* coordinates. The number of these features depends on the dimension of the object space and on the specific transformation. Denote their number by k. Now, for each k-tuple of object points compute the coordinates of all the other object points according to this basis k-tuple and hash these coordinates into a table which stores all the pairs (model, basis k-tuple) for every coordinate.

Recognition of similar objects to a given target

Given a target object:

1. Extract its *interesting features*. (Let their number be n.)

2. Choose a basis k-tuple and compute the coordinates of the other features in this basis.

3. For each such coordinate check the appropriate entrance in the hash- table and vote for the pairs(object, basis k-tuple) appearing there (see [7,8]).

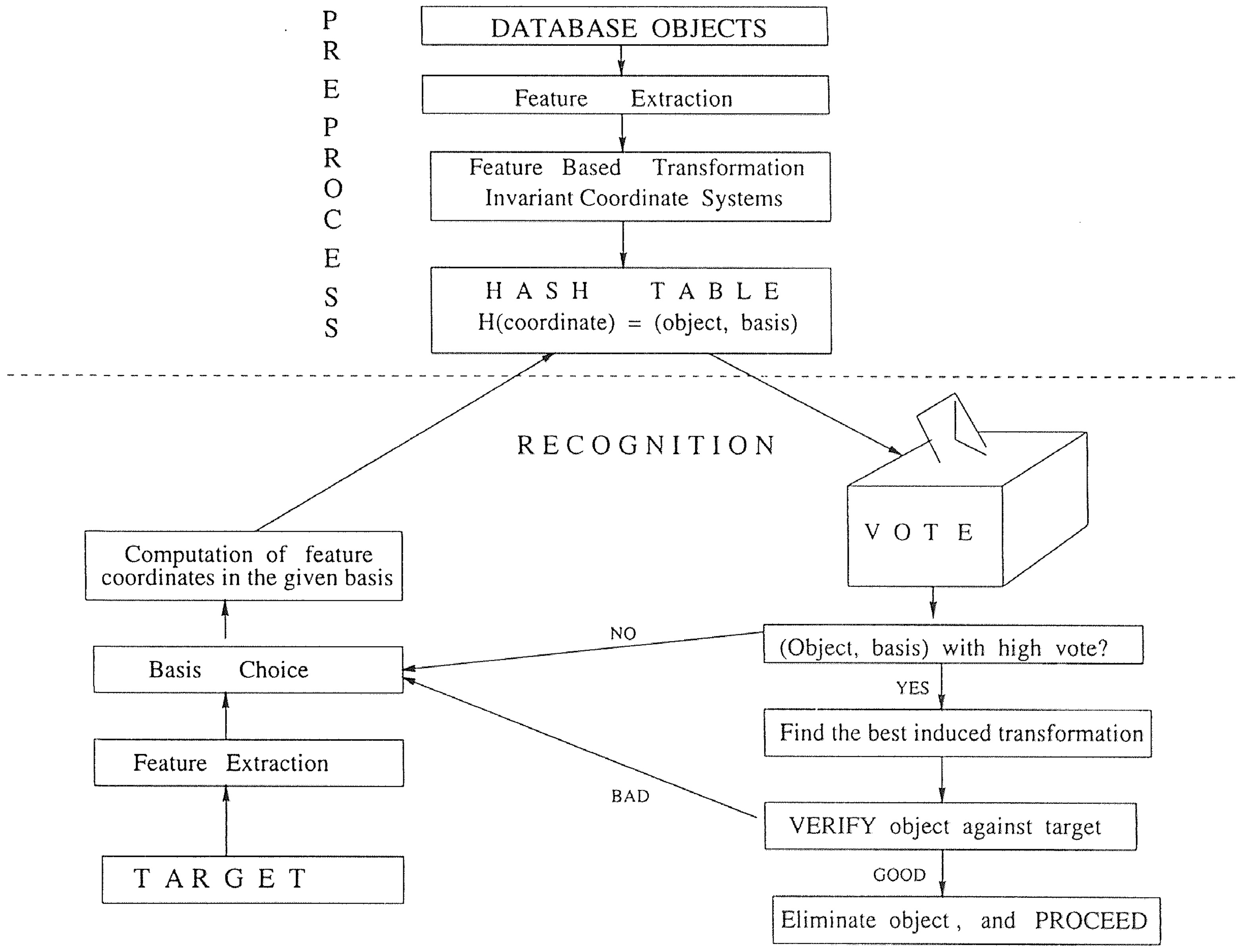

PREPROCESS
DATABASE OBJECTS
Feature Extraction
Feature Based Transformation Invariant Coordinate Systems
HASH TABLE
H(coordinate) = (object, basis)
RECOGNITION
VOTE
Computation of feature coordinates in the given basis
Basis Choice
Feature Extraction
TARGET
NO
(Object, basis) with high vote?
YES
Find the best induced transformation
BAD
VERIFY object against target
GOOD
Eliminate object, and PROCEED

Figure 3.1 The General Scheme of The Geometric Hashing Algorithm

4. Find the pairs which obtained the highest votes. If a certain pair scored a large number of votes, decide that its object and basis k-tuple correspond to the one chosen in the target.

5. For a candidate k-tuple compute the transformation giving the correspondence between it and the target's k-tuple, compute correspondences of additional *interesting features* that this transformation induces, and find the best transformation (say, in least squares sense) which produces all these correspondences.

6. Verify the transformed model against the target object. The verification can be done not only using the *interesting points*, but also using any other available information, relating to the object representation. If the verification fails, go back to Step 2 and choose another basis k-tuple in the target object.

CHAPTER 4

BASIC MEMORY IMPLEMENTATION

In this chapter we will start from the two-dimensional objects and describe the suggested method in detail. We begin the presentation with the definition of the problem.

4.1 Problem Definition

Consider the problem of information retrieval in a two-dimensional object database. Given are a target object C in the form of a set of coordinates of the object's *interesting points*, and also a database D which is a collection of the sets of *interesting points'* coordinates for each one of the member objects.

The task at hand is to determine, for the given object C and database D, all the objects of D that are closest to C, according to some kind of rotation, move, along with delete or insert some interesting points.

4.2 Algorithm

We have developed a hash based algorithm as follows:

Off-line preprocessing of database objects

For each database object :

1. Represent the database object by a set of points.
2. For each noncollinear triplet of the object points :
   (a) Compute the Theta angle of the triplet.
   (b) Compute the coordinates of all the other points according to the basis triplet.

(c) Compute the hash value and hash the coordinates into a table which stores all the pairs (basis triplet, angle, model number) of every coordinate. (Figure 4.1 shows the structure of the hash-table)

On-line recognition of target object

Given a target object:

1. Extract its *interesting points.*

2. For each noncollinear triplet of the target points :

   (a) Compute the Theta angle of the triplet.

   (b) Compute the coordinates of all the other points according to the basis triplet.

   (c) Compute the hash value and search in the hash-table for the same hash value, and then give each matching object a vote.

3. Determine answer by thresholding the voting records.

Figure 4.2 shows the flowchart of the off-line algorithm and Figure 4.3 shows the flowchart of the on-line algorithm.

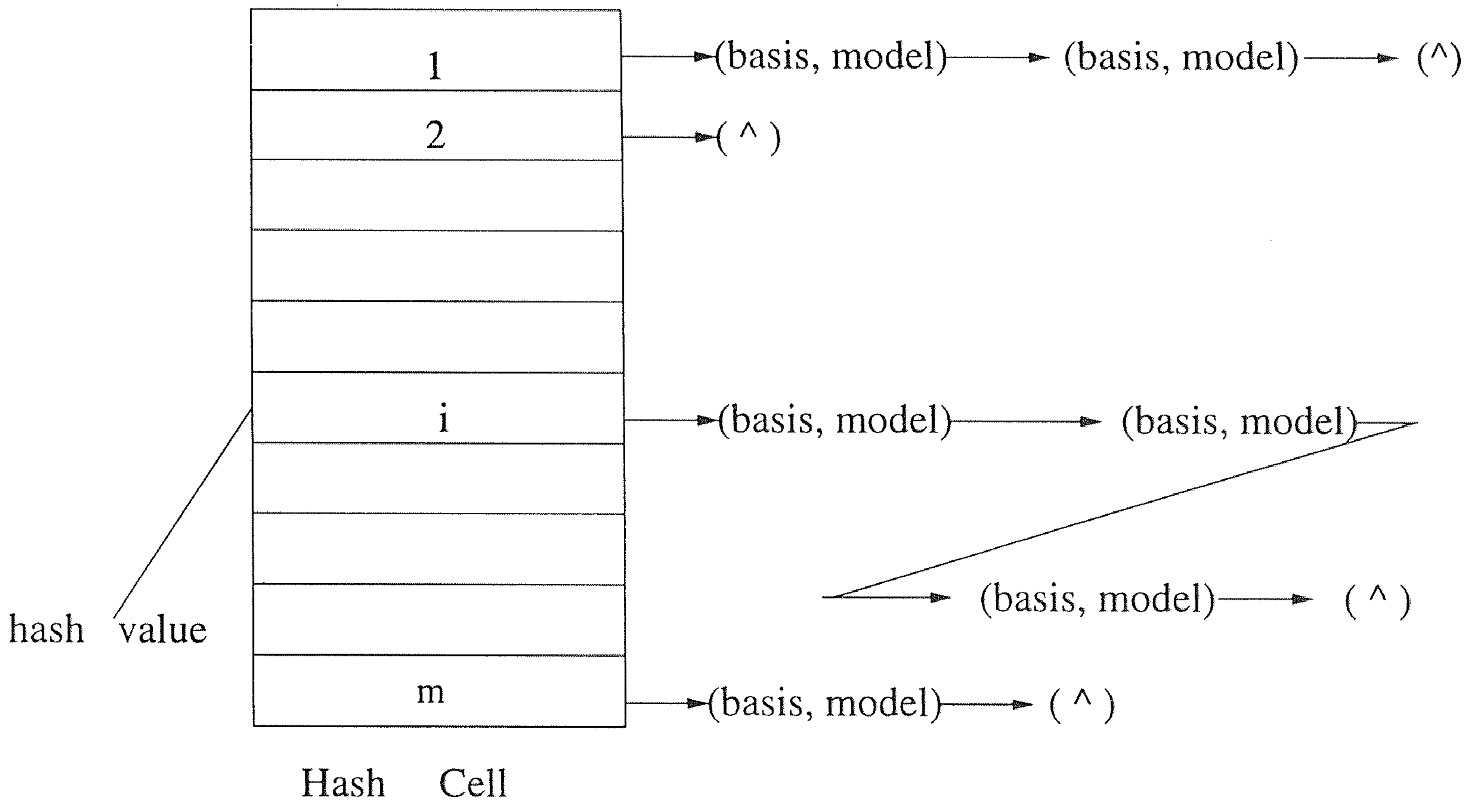
We have implemented the algorithm and developed a memory-based version (see Appendix I) and also a disk-based version (see Appendix II).

When we get a triplet, (x_0, y_0), (x_1, y_1), (x_3, y_3), the formula used to find the new X, Y values for an *interesting point* is as follows:

$$k_1 = \frac{y_1 - y_0}{x_1 - x_0}$$

$$k_2 = \frac{y_2 - y_0}{x_2 - x_0}$$

$$newX = \left(\frac{k_1 * (X - x_0) - (Y - y_0)}{k_1 - k_2}\right) * \sqrt{k_2^2 + 1}$$



The data structure used here to store the subtables are linked list.

Figure 4.1 Hash-table Used in the Memory Implementation

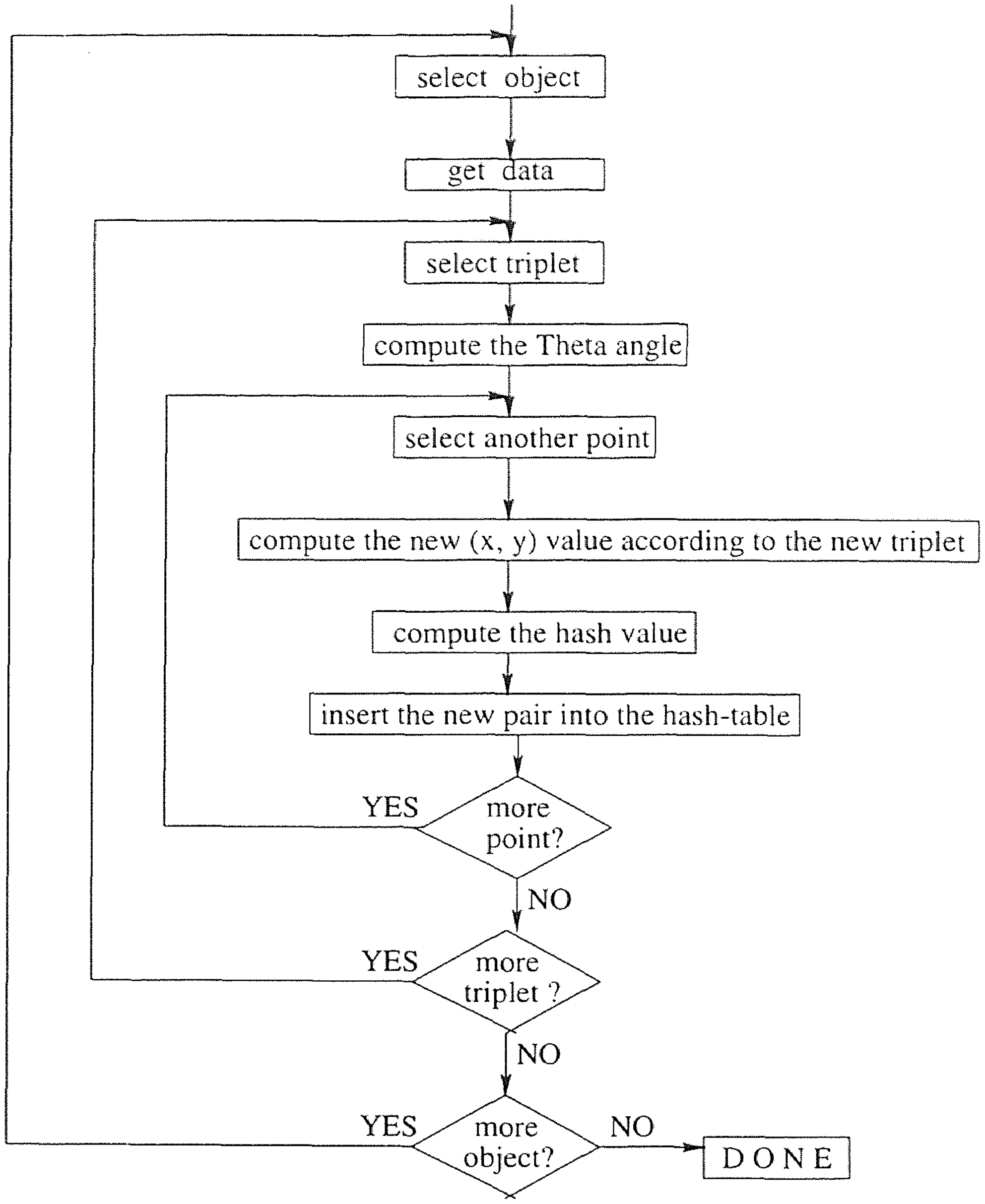


Figure 4.2 Flowchart of the Off-line Algorithm

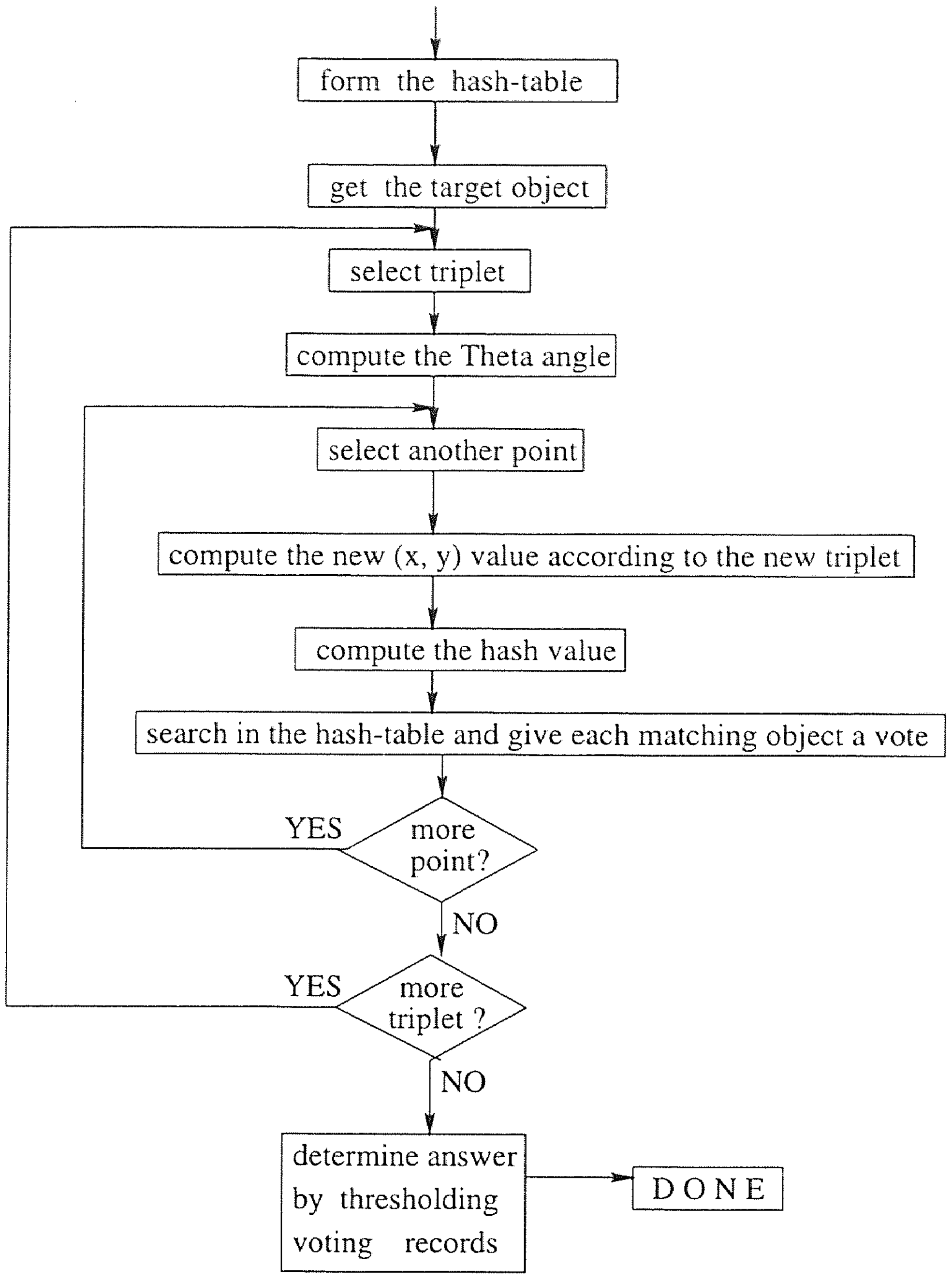

form the hash-table
get the target object
select triplet
compute the Theta angle
select another point
compute the new (x, y) value according to the new triplet
compute the hash value
search in the hash-table and give each matching object a vote
YES
more point?
NO
YES
more triplet ?
NO
determine answer by thresholding voting records
DONE

Figure 4.3 Flowchart of the On-line Algorithm

$$newY = (\frac{k_2 * (X - x_0) - (Y - y_0)}{k_1 - k_2}) * \sqrt{k_2^2 + 1}$$

if$(x_1 = x_0)$

$$newX = (X - x_0) * \sqrt{k_2^2 + 1}$$

$$newY = (Y - y_0) - (X - x_0) * k_2$$

if$(x_2 = x_0)$

$$newX = (X - x_0) * \sqrt{k_1^2 + 1}$$

$$newY = (Y - y_0) - (X - x_0) * k_1$$

4.3 An Example of the Algorithm

In this section, we will present a simple example. Given are a target object C containing only four *interesting points* and a database D which is a collection of three objects. (see Figure 4.4)

In the off-line preprocessing phase, for each database object (from 1 to 3), for each noncollinear triplet, compute the Theta angle, compute the coordinates of all the other *interesting points* according to the triplet and then compute the hash value and store them into the hash-table. Table 4.1 shows part of the hash-table after the off-line preprocess is done.

In the on-line recognition phase, for each noncollinear triplet, compute the Theta angle and for each interesting point compute the new coordinate, hash value, then search in the hash-table for the same index. If every item (Base, X, Y, Theta) is matched, give that object a vote. After search is completed, the one with the highest vote is the answer. For this example, we got the answer like this:

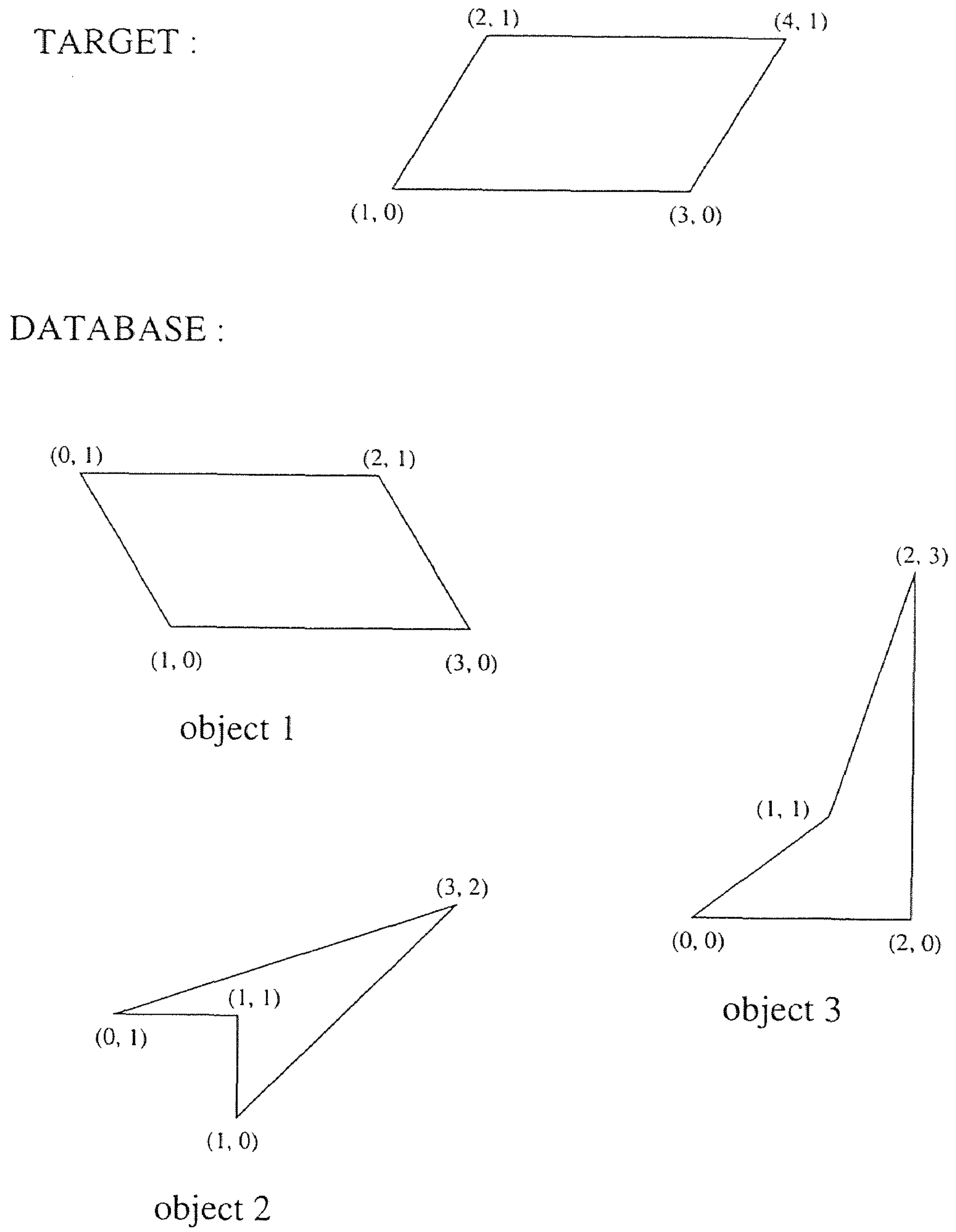


Figure 4.4 Example of the Algorithm

Table 4.1 Contents of the Hash-table

Index	Base	X	Y	Theta	Object
1	3	2	0	0.141897	2
3	1	0	2	1.5708	3
4	0	2	3	0.321751	1
8	3	1	2	0.321751	2
9	0	1	2	0.785398	2
9	2	0	3	2.67795	2
10	0	2	1	0.785398	3
12	0	2	3	0.982794	3
15	3	0	1	0.124355	3
19	3	2	1	0.463648	3
19	3	1	2	0.463648	3
19	0	1	2	0.785398	3
19	0	3	2	0.321751	2
20	0	3	1	1.10715	2
23	0	1	3	1.10715	2
27	1	3	0	2.81984	3
27	2	3	1	1.5708	1
27	2	1	3	1.5708	1
27	1	2	0	1.5708	1
27	1	0	2	1.5708	1
30	1	2	0	1.5708	3
32	0	1	3	0.197396	3
33	2	3	1	0.785398	3
33	2	1	3	0.785398	3
33	3	2	0	0.463648	1
33	0	1	3	0.463648	1
33	1	0	3	1.5708	2
33	1	3	0	1.5708	2
33	3	2	1	0.321751	2
36	2	0	3	2.35619	1
36	2	0	1	0.785398	1
36	1	3	2	0.785398	1
36	1	3	0	2.35619	1
37	3	0	2	0.141897	2
38	3	2	0	0.588003	3
38	3	0	2	0.588003	3
39	2	1	0	0.785398	3
.					
.					
.					

- object 1: 24
- object 2: 6
- object 3: 2
- the correct answer is object 1.

CHAPTER 5

3-DIMENSIONAL DISK-BASED IMPLEMENTATION

5.1 3D Implementation

Most objects in the real world are three-dimensional objects, like molecular, protein structures, etc. So, in order to improve the application of the project, we decide to extend the project to deal with 3D objects.

In three-dimensional spaces, triplets are not enough; we need four nonplanar points to create a new basis. The algorithm remains almost the same; only the formula used to find the new X, Y, Z values need some change.

$$\begin{pmatrix} newX \\ newY \\ newZ \end{pmatrix} = \begin{pmatrix} x_1 - x_0 & x_2 - x_0 & x_3 - x_0 \\ y_1 - y_0 & y_2 - y_0 & y_3 - y_0 \\ z_1 - z_0 & z_2 - z_0 & z_3 - z_0 \end{pmatrix}^{-1} \begin{pmatrix} X - x_0 \\ Y - y_0 \\ Z - z_0 \end{pmatrix}$$

Matrix A^{-1} is the inverse of matrix A. In order to get the inverse of the matrix, we need some knowledge about the Linear Algebra (see [9]).

Minors and cofactors are determinants of submatrices associated with particular entries in the original square matrix. The minor of entry a_{ij} is the determinant of a submatrix resulting from the elimination of the single row i and the single column j. For example, the minor corresponding to entry a_{11} in a 3x3 matrix A is the determinant of the matrix created by eliminating row 1 and column 1.

$$minorofa_{11} = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix}$$

The cofactor of entry a_{ij} is the minor of a_{ij} multiplied by $(-1)^{i+j}$.

The transpose, A^t, of an $m \times n$ matrix A is an $m \times n$ matrix constructed by taking the ith row and making it the ith column. The diagonal is unchanged.

The classical adjoint is the transpose of the cofactor matrix. For example,

$$A = \begin{bmatrix} 2 & 3 & -4 \\ 0 & -4 & 2 \\ 1 & -1 & 5 \end{bmatrix}$$

matrix of cofactors :

$$\begin{bmatrix} -18 & 2 & 4 \\ -11 & 14 & 5 \\ -10 & -4 & -8 \end{bmatrix}$$

$$A_{adj} = \begin{bmatrix} -18 & -11 & -10 \\ 2 & 14 & -4 \\ 4 & 5 & -8 \end{bmatrix}$$

For a 3x3 matrix, the inverse is determined by dividing every entry in the classical adjoint by the determinant of the original matrix,

$$A^{-1} = fracA_{adj}|A|$$

5.2 Disk-Based Implementation

Although geometric hashing naturally deals with many objects, the scheme might pose a problem when memory is concerned. As the algorithm encodes each database object's representation in a hash-table, the size of the table grows with the number of objects. When no classification of features is available, and no assumptions can be a-priori made about the appearance of the features in the target, all possible feature bases should be considered. This might result in big memory requirements, which grow with the complexity of the object transformation. So we extend the project to store the whole huge hash-table in the disk, not in the memory.

In the off-lime preprocess, for each database object, for each noncollinear triplet, for each other *interesting point*, compute the hash value and then write the pair to the right place (the same hash values are stored in the same block) in the hash-table in the disk.

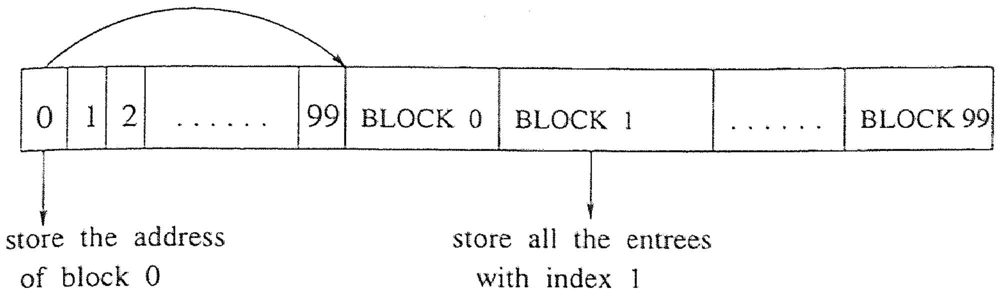

0
1
2
......
99
BLOCK 0
BLOCK 1
......
BLOCK 99
store the address of block 0
store all the entrees with index 1

Figure 5.1 Hash-table Used in the Disk-Based Implementation

In the on-line recognition phase, whenever we get a hash value from the target, search in the hash-table in the disk for the right index (the address of the block of data), read the related disk block into the memory and then do the matching. Figure 5.1 shows the hash-table we use in the disk-based implementation.

CHAPTER 6

CONCLUSIONS

In this thesis a general and efficient scheme for object recognition is presented. The scheme is suitable for a variety of important recognition tasks. The method is general in many aspects. The acquired input data can be either two-dimensional or three-dimensional data. Furthermore, the scheme can deal with the simplest case of exact match or some complex cases of move and rotation.

The emphasis in this thesis is on efficient recognition. Objects are represented as a set of *interesting points*. Matching is achieved by finding the object in all the database objects which has the highest vote for the target object. The matching procedure has two major steps. The first step precompiles the representations of the database objects, resulting in a hash-table based on these representations. The second step, recognition proper, is executed on the target object using the previously prepared hash-table. Since the first step is executed off-line in the database objects and is independent of the next phase of the algorithm, the on-line recognition stage is significantly sped-up.

In our algorithms, the hash-table serves as an associative memory, allowing for fast retrieval of 'similar' object feature subsets. As the algorithm encodes each database object's representation in a hash-table, the size of the table grows with the number of objects. When no classification of the size of objects is available, and no assumptions can be a-priori made about the appearance of the target object, all possible object bases should be considered. This might result in big memory requirements, which grow with the complexity of the object's representation. So we extend the project to deal with disk, storing the whole huge hash-table in the disk. Whenever the system gets a hash value, it reads the related disk block into the memory. According to this technique, the memory problem is solved.

The algorithm was successfully tested in 'real-life' situations. The implementations were made for the important cases of the two-dimensional artificial data and three-dimensional scientific data. These experiments demonstrated the ability of the scheme to deal with all major difficult issues in recognition.

6.1 Future Directions

Since this thesis addresses the object-based recognition problem from a general point of view, it is natural that future research can concentrate on different aspects of the scheme. For example, more research can be done in the area of implementation of the method for the perspective transformation.

Aside from a more elaborated investigation of different target objects, there are some other topics, which can be addressed. One important such topic is the use of the geometric hashing scheme for recognition of parameterized objects. The extension seems quite natural, and it is expected that at least some efficient algorithms can be realized.

Another important direction for research is in the area of complexity reduction. Some methods can be examined here, including perceptual grouping, feature classification, etc. This will make the algorithm more efficient, especially when dealing with very complex objects.

APPENDIX A

Memory-Based Implementation

```
/*********************************************************************/
/*Program: Hash.C                                                    */
/*Author : Joyce Ye Lu                                               */
/*                                                                   */
/*This program builds the hash-table for one object and store it in */
/*a file named "hashtable.dat" in the memory for use in the match.c */
/*program.                                                           */
/*********************************************************************/



#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

#define Maxsize 100
#define model_size 3

struct Item
       {
           int Base,X,Y;
           float Theta;
           int num;
           Item *next;
       };

typedef Item *Itempointer;
Itempointer hash[100];

int model_num;
int base,x,y;
int I, J, K, size, key;
float X0,Y0,X1,Y1,X2,Y2,U,V,NewU,NewV;
float theta;
float A[Maxsize];

void getdata();
void find_U_V();
void find_I_J_K(int n);
void find_X_Y();
void find_New_UV();
void find_key();
void find_angel();
void form_hashtable();
void insert(int);
void display();


main()
{
  form_hashtable();

  cout<<"please enter your model number:\n";
  cin>>model_num;

  getdata();
  find_I_J_K(size/2);
  display();
}


void getdata()
{
  char Inputfilename[25];
  cout<<"Please enter your input file name:";
  cin>> Inputfilename;

  fstream Instream;
  Instream.open(Inputfilename,ios::in);

  int i=0;
  while(!Instream.eof())
  {
     Instream>>A[i];
     i++;
  }

  Instream.close();
  size = i-1;
}
```

```
 void find_I_J_K(int n)
 {
       for(int i=0;i<n;i++)
           for(int j=0;j<n ;j++)
           {
            if(j != i)
              for( int k=0;k<n ;k++)
              {
                if ( k != i && k != j)
       {
I=i;J=j;K=k;
        find_X_Y();
find_angel();
        find_U_V();
       }
     }
   }
}


void find_X_Y()
{
X0 = A[2*I];
Y0 = A[I*2+1];
X1 = A[J*2];
Y1 = A[J*2+1];
X2 = A[K*2];
Y2 = A[K*2+1];
}


void find_U_V()
{
for(int i=0; i<size/2;i++)
 if(i != I && i != J && i != K)
   {  U = A[2*i];
      V = A[i*2+1];
      find_New_UV();
      find_key();
      base=I;
      x=J;
      y=K;
      insert(key);
   }
}


void find_angel()
{
   float a,b,c;

   a=sqrt((X1-X0)*(X1-X0) + (Y1-Y0)*(Y1-Y0));
   b=sqrt((X2-X0)*(X2-X0) + (Y2-Y0)*(Y2-Y0));
   c=sqrt((X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1));

   theta=acos((a*a+b*b-c*c)/(2*a*b));

}


void find_New_UV()
{
   float k1, k2;

   if( X1 != X0 && X2 != X0)
   {
      k1=(Y1-Y0) / (X1-X0);
      k2=(Y2-Y0) / (X2-X0);
      NewU = ((k1*(U-X0)-(V-Y0))/(k1-k2))*sqrt(k2*k2+1);
      NewV = ((k2*(U-X0)-(V-Y0))/(k1-k2))*sqrt(k1*k1+1);
   }

   if (X1 == X0)
   {
      k2=(Y2-Y0) / (X2-X0);
      NewU=(U-X0)*sqrt(k2*k2+1);
      NewV=(V-Y0)-(U-X0)*k2;
   }

   if (X2 == X0)
   {
      k1=(Y1-Y0) / (X1-X0);
      NewU=(U-X0)*sqrt(k1*k1+1);
      NewV=(V-Y0)-(U-X0)*k1;
   }
}
```

```
void find_key()
{
   key= (int) (NewU*210 + NewV*120);
}


void form_hashtable()
{
  Item *tmp;
  char i[5],base[15],x[15],y[15],theta[15],n[15];
  char line[100];

  for (int j=0;j<100;j++)
     hash[j]=NULL;

  FILE *fp;
  fp = fopen("hashtable.dat","r");

  while ( (fgets(line,100,fp))!=NULL)
  {
        sscanf(line,"%s%s%s%s%s%s",i,base,x,y,theta,n);
        tmp = new Item;
        tmp->Base = atoi(base);
        tmp->X = atoi(x);
        tmp->Y = atoi(y);
        tmp->Theta = atof(theta);
        tmp->num = atoi(n);
        int index=atoi(i);
        tmp->next = hash[index];
        hash[index]=tmp;

  }
  fclose(fp);
}


void insert(int n)
{
int index =abs( n) % 100;
Item *tmp;

tmp=new Item;
tmp->X = x;
tmp->Y = y;
tmp->Base = base;
tmp->Theta = theta;
tmp->num=model_num;

tmp->next=hash[index];
hash[index]=tmp;
}




void display()
{
Item *tmp;

char Outputfilename[] = "hashtable.dat";

fstream Outstream;
Outstream.open(Outputfilename, ios::out);

for(int i=0;i<100;i++)
  {
    tmp=hash[i];

    while(tmp != NULL)
    {
        Outstream<<i<<" ";
        Outstream<<tmp->Base<<" "<<tmp->X<<" "<<tmp->Y<<" "<<tmp->Theta
                 <<" "<<tmp->num<<endl;
        tmp=tmp->next;
    }
  }
}
```

```
/*******************************************************************/
/*Program: match.C                                                 */
/*Author : Joyce Ye Lu                                             */
/*                                                                 */
/*This program compares the target object with all the database    */
/*objects and finds out the closest one.                           */
/*******************************************************************/


#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

#define Maxsize 100
#define model_num 6

struct Item
        {
    int Base,X,Y;
    float Theta;
            int num;
            Item *next;
        };

typedef Item *Itempointer;
Itempointer hash[100];

float A[Maxsize];
int I, J, K, size, key;
float X0,Y0,X1,Y1,X2,Y2,U,V,NewU, NewV;
float Theta1;
int model[model_num];

void getdata();
void find_U_V();
void find_I_J_K(int n);
void find_X_Y();
void find_New_UV();
void find_angel();
void find_key();
void form_hashtable();
void find_model();
void display();


main()
{
  form_hashtable();
  getdata();
  find_I_J_K(size/2);
  display();
}


void getdata()
{
  char Inputfilename[25] ;

  cout<<"Please enter your input file name:";
  cin>> Inputfilename;

  fstream Instream;
  Instream.open(Inputfilename,ios::in);

  int i=0;
  while(!Instream.eof())
  {
     Instream>>A[i];
     i++;
  }

  Instream.close();
  size = i-1;
}


void find_U_V()
{
for(int i=0; i<size/2;i++)
 if(i != I && i != J && i != K)
   {  U = A[2*i];
      V = A[i*2+1];
```

```
       find_New_UV();
       find_key();
       find_model();
    }
}


void find_I_J_K(int n)
{
      for(int i=0;i<n;i++)
          for(int j=0;j<n ;j++)
          {
           if(j != i)
             for( int k=0;k<n ;k++)
             {
               if ( k != i && k != j)
       {
I=i;J=j;K=k;
        find_X_Y();
find_angel();
        find_U_V();
       }
     }
   }
}


void find_X_Y()
{
X0 = A[2*I];
Y0 = A[I*2+1];
X1 = A[J*2];
Y1 = A[J*2+1];
X2 = A[K*2];
Y2 = A[K*2+1];
}


void find_New_UV()
{
   float k1, k2;

   if( X1 != X0 && X2 != X0)
   {
      k1=(Y1-Y0) / (X1-X0);
      k2=(Y2-Y0) / (X2-X0);
      NewU = ((k1*(U-X0)-(V-Y0))/(k1-k2))*sqrt(k2*k2+1);
      NewV = ((k2*(U-X0)-(V-Y0))/(k1-k2))*sqrt(k1*k1+1);
   }

   if (X1 == X0)
   {
      k2=(Y2-Y0) / (X2-X0);
      NewU=(U-X0)*sqrt(k2*k2+1);
      NewV=(V-Y0)-(U-X0)*k2;
   }

   if (X2 == X0)
   {
      k1=(Y1-Y0) / (X1-X0);
      NewU=(U-X0)*sqrt(k1*k1+1);
      NewV=(V-Y0)-(U-X0)*k1;
   }
}


void find_angel()
{
   float a,b,c;

   a=sqrt((X1-X0)*(X1-X0) + (Y1-Y0)*(Y1-Y0));
   b=sqrt((X2-X0)*(X2-X0) + (Y2-Y0)*(Y2-Y0));
   c=sqrt((X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1));

   Theta1=acos((a*a+b*b-c*c)/(2*a*b));

}


void find_key()
{
   key= (int) (NewU*210 + NewV*120);
}


void form_hashtable()
{
  Item *tmp;
```

```
   char i[5],base[15],x[15],y[15],theta[15],n[5];
   char line[100];

   for (int j=0;j<100;j++)
      hash[j]=NULL;

   FILE *fp;
   fp = fopen("hashtable.dat","r");

   while ( (fgets(line,100,fp))!=NULL)
   {
sscanf(line,"%s%s%s%s%s%s",i,base,x,y,theta,n);
tmp = new Item;
tmp->Base = atoi(base);
tmp->X = atoi(x);
tmp->Y = atoi(y);
tmp->Theta = atof(theta);
        tmp->num = atoi(n);
int index=atoi(i);

tmp->next = hash[index];
hash[index]=tmp;

  }
  fclose(fp);
}


void find_model()
{
 Item *current;
 int i;
 int index = abs(key) % 100;

 current = hash[index];
 while(current != NULL)
 {
   if ((current->Theta - Theta1) < 0.00001 ||
        (current->Base == I && current->X == J && current->Y == K))
       model[current->num]++;
   current = current->next;
  }
}
```

```
void display()
{
  int largest=model[1];
  int largest_model=1;

  for (int i=1;i<model_num;i++)
    if(model[i]>largest)
    {
      largest = model[i];
      largest_model=i;
    }

  cout<<"The closest model is model "<<largest_model<<endl;

  for(int j=1;j<model_num;j++)
    cout<<model[j]<<endl;
}
```

APPENDIX B

Disk-Based Implementation

```
/*********************************************************************/
/*Program: diskhash.C                                                */
/*Author : Joyce Ye Lu                                               */
/*                                                                   */
/*This program builds the hash-table for all the database objects   */
/*and store it in a file named "newfile" on the disk for use in the */
/*match program.                                                     */
/*********************************************************************/




#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

#define Maxsize 100
#define model_size 5

struct Item
       {
           int Base,X,Y;
           float Theta;
           int num;
           long next;
       };

long hash;
long hash_array[100];
FILE *OutPutFile;
long total_item_in_file=0;

int model_num;
int base,x,y;
int I, J, K, size, key;
float X0,Y0,X1,Y1,X2,Y2,U,V,NewU, NewV;
float theta;
float A[Maxsize];

void getdata();
void find_U_V();
void find_I_J_K(int n);
void find_X_Y();
void find_New_UV();
void find_key();
void find_angel();
void form_hashtable();
void insert(int);
void make_final_datafile();
void process();
void display();


main()
{
  OutPutFile = fopen("outfile","w+b");

  if (OutPutFile == NULL)
     {fprintf (stderr,"open output data file error, exit from program!\n");
      exit (1);
     }

  form_hashtable();

  for(int i=1;i<=model_size;i++)
     process();

  fseek(OutPutFile,0,SEEK_SET);
  for(i=0;i<100;i++)
  {
     hash = hash_array[i];
     fwrite (&hash, (size_t)sizeof(long),1,OutPutFile);
  }

  fclose (OutPutFile);

  make_final_datafile();

  display();
}
```

```
void process()
{
  cout<<"please enter your model number:\n";
  cin>>model_num;

  getdata();
  find_I_J_K(size/2);
}


void getdata()
{
  char Inputfilename[25];
  cout<<"Please enter your input file name:";
  cin>> Inputfilename;

  fstream Instream;
  Instream.open(Inputfilename,ios::in);

  int i=0;
  while(!Instream.eof())
  {
     Instream>>A[i];
     i++;
  }

  Instream.close();
  size = i-1;
}



void find_I_J_K(int n)
{
      for(int i=0;i<n;i++)
          for(int j=0;j<n ;j++)
          {
           if(j != i)
             for( int k=0;k<n ;k++)
             {
               if ( k != i && k != j)
       {
```

```
I=i;J=j;K=k;
        find_X_Y();
find_angel();
        find_U_V();
       }
     }
   }
}


void find_X_Y()
{
X0 = A[2*I];
Y0 = A[I*2+1];
X1 = A[J*2];
Y1 = A[J*2+1];
X2 = A[K*2];
Y2 = A[K*2+1];
}



void find_U_V()
{
for(int i=0; i<size/2;i++)
 if(i != I && i != J && i != K)
   {  U = A[2*i];
      V = A[i*2+1];
      find_New_UV();
      find_key();
      base=I;
      x=J;
      y=K;
      insert(key);
   }
}


void find_angel()
{
   float a,b,c;

   a=sqrt((X1-X0)*(X1-X0) + (Y1-Y0)*(Y1-Y0));
   b=sqrt((X2-X0)*(X2-X0) + (Y2-Y0)*(Y2-Y0));
```

```
    c=sqrt((X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1));

    theta=acos((a*a+b*b-c*c)/(2*a*b));

}


void find_New_UV()
{
   float k1, k2;

   if( X1 != X0 && X2 != X0)
   {
      k1=(Y1-Y0) / (X1-X0);
      k2=(Y2-Y0) / (X2-X0);
      NewU = ((k1*(U-X0)-(V-Y0))/(k1-k2))*sqrt(k2*k2+1);
      NewV = ((k2*(U-X0)-(V-Y0))/(k1-k2))*sqrt(k1*k1+1);
   }

   if (X1 == X0)
   {
      k2=(Y2-Y0) / (X2-X0);
      NewU=(U-X0)*sqrt(k2*k2+1);
      NewV=(V-Y0)-(U-X0)*k2;
   }

   if (X2 == X0)
   {
      k1=(Y1-Y0) / (X1-X0);
      NewU=(U-X0)*sqrt(k1*k1+1);
      NewV=(V-Y0)-(U-X0)*k1;
   }
}


void find_key()
{
   key= (int) (NewU*210 + NewV*120);
}



void form_hashtable()
{
  int i;

  hash = 0;
  for (i = 0; i < 100;i++)
   {   fwrite ( &hash,(size_t)sizeof(long),1,OutPutFile);
                // write 100 time zero hash to new file
       hash_array[i] =0;
   }

}


void insert(int n)
{
int index =abs( n) % 100;
Item tmp;

tmp.X = x;
tmp.Y = y;
tmp.Base = base;
tmp.Theta = theta;
tmp.num=model_num;

tmp.next = hash_array[index];

fseek(OutPutFile,0,SEEK_END); //position to end of file
hash_array[index] = ftell(OutPutFile);
fwrite(&tmp,sizeof(Item),1,OutPutFile);

}


void make_final_datafile ()
{
  FILE *newfile;
  int i,i2;
  long nexttmp,oldnext;
  Item itemtmp;
  int numofitem;

  OutPutFile = fopen ("outfile","rb");
  newfile = fopen ("newfile", "w+b");
  fwrite (hash_array,sizeof(long),100,newfile);
```

```
 for (i = 0; i< 100;i++)
 {
    nexttmp = hash_array[i];
    if (nexttmp)
       hash_array[i] = ftell(newfile);

    numofitem = 0;
    while (nexttmp)
    {
      fseek(OutPutFile,nexttmp,SEEK_SET);
      fread (&itemtmp,sizeof(Item),1,OutPutFile);
      oldnext = itemtmp.next;

      if (oldnext)
       itemtmp.next = ftell(newfile)+sizeof(Item);
      fwrite (&itemtmp,sizeof(Item),1,newfile);
      nexttmp = oldnext;
      numofitem++;
    }

    numofitem = (numofitem*200)/100;
    itemtmp.next = 0;
    for (i2 = 0 ; i2< numofitem;i2++)
       fwrite (&itemtmp,sizeof(Item),1,newfile);
 }

 fseek(newfile,0,SEEK_SET);
 fwrite (hash_array,sizeof(long),100,newfile);
 fclose (OutPutFile);
 fclose (newfile);
}


void display()
{
 Item tmp;
 FILE *newfile;

 char Outputfilename[25];
 cout<<"Please enter your new output file name: ";
 cin>> Outputfilename;

 fstream Outstream;
 Outstream.open(Outputfilename, ios::out);
 newfile = fopen ("newfile","rb");
 for(int i=0;i<100;i++)
  {
   fseek(newfile,sizeof(long)*i,SEEK_SET);
   fread(&hash,sizeof(long),1,newfile);

   while(hash != 0)
    {
      fseek(newfile,hash,SEEK_SET);
      fread(&tmp,sizeof(Item),1,newfile);
        Outstream<<i<<" ";
        Outstream<<tmp.Base<<" "<<tmp.X<<" "<<tmp.Y<<" "
        <<tmp.Theta<<" "<<tmp.num<<"\n";
       hash = tmp.next;

   }
 }
 fclose(newfile);
}
```

```
/*****************************************************************/
/*Program : d_match.C                                            */
/*Author  : Joyce Ye Lu                                          */
/*                                                               */
/*This program compares the target object with all the database  */
/*objects and finds out the closest one.                         */
/*****************************************************************/



#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

#define Maxsize 100
#define model_id 5

struct Item
        {
            int Base,X,Y;
            float Theta;
            int num;
            long next;
        };

long hash;
long hash_array[100];
FILE *OutPutFile;
long total_item_in_file=0;

int model_num;
int base,x,y;
int I, J, K, size, key;
float X0,Y0,X1,Y1,X2,Y2,U,V,NewU, NewV;
float theta;
float A[Maxsize];
int model[model_id];

void getdata();
void find_U_V();
void find_I_J_K(int n);
void find_X_Y();
void find_New_UV();
void find_key();
void find_angel();
void form_hashtable();
void insert(int);
void make_final_datafile();
void find_model();
void display();


main()
{
  form_hashtable();

  getdata();
  find_I_J_K(size/2);
  make_final_datafile();
  display();
}


void getdata()
{
  char Inputfilename[25];
  cout<<"Please enter your input file name:";
  cin>> Inputfilename;

  fstream Instream;
  Instream.open(Inputfilename,ios::in);

  int i=0;
  while(!Instream.eof())
  {
     Instream>>A[i];
     i++;
  }

  Instream.close();
  size = i-1;
}
```

```
void find_I_J_K(int n)
{
      for(int i=0;i<n;i++)
          for(int j=0;j<n ;j++)
          {
           if(j != i)
             for( int k=0;k<n ;k++)
             {
               if ( k != i && k != j)
       {
I=i;J=j;K=k;
        find_X_Y();
find_angel();
        find_U_V();
       }
     }
   }
}


void find_X_Y()
{
X0 = A[2*I];
Y0 = A[I*2+1];
X1 = A[J*2];
Y1 = A[J*2+1];
X2 = A[K*2];
Y2 = A[K*2+1];
}



void find_U_V()
{
for(int i=0; i<size/2;i++)
 if(i != I && i != J && i != K)
   {  U = A[2*i];
      V = A[i*2+1];
      find_New_UV();
      find_key();
      base=I;
      x=J;
      y=K;
      find_model();
    }
}


void find_angel()
{
   float a,b,c;

   a=sqrt((X1-X0)*(X1-X0) + (Y1-Y0)*(Y1-Y0));
   b=sqrt((X2-X0)*(X2-X0) + (Y2-Y0)*(Y2-Y0));
   c=sqrt((X2-X1)*(X2-X1) + (Y2-Y1)*(Y2-Y1));

   theta=acos((a*a+b*b-c*c)/(2*a*b));

}


void find_New_UV()
{
   float k1, k2;

   if( X1 != X0 && X2 != X0)
   {
      k1=(Y1-Y0) / (X1-X0);
      k2=(Y2-Y0) / (X2-X0);
      NewU = ((k1*(U-X0)-(V-Y0))/(k1-k2))*sqrt(k2*k2+1);
      NewV = ((k2*(U-X0)-(V-Y0))/(k1-k2))*sqrt(k1*k1+1);
   }

   if (X1 == X0)
   {
      k2=(Y2-Y0) / (X2-X0);
      NewU=(U-X0)*sqrt(k2*k2+1);
      NewV=(V-Y0)-(U-X0)*k2;
   }

   if (X2 == X0)
   {
      k1=(Y1-Y0) / (X1-X0);
      NewU=(U-X0)*sqrt(k1*k1+1);
      NewV=(V-Y0)-(U-X0)*k1;
   }
```

```
}


void find_key()
{
   key= (int) (NewU*210 + NewV*120);
}


void form_hashtable()
{
  int i;

  hash = 0;
  for (i = 0; i < 100;i++)
   {   fwrite ( &hash,(size_t)sizeof(long),1,OutPutFile);
       hash_array[i] =0;
   }

}


void find_model()
{
 Item current;
 int i;
 int index = abs(key) % 100;

 current = hash_array[index];
 while(current != 0)
 {
   if ((current.Theta - Theta1) < 0.00001 ||
        (current.Base == I && current.X == J && current.Y == K))
       model[current.num]++;
   current = current.next;
  }
}


void make_final_datafile ()
{
  FILE *newfile;
  int i,i2;
  long nexttmp,oldnext;
```

```
    Item itemtmp;
    int numofitem;

    OutPutFile = fopen ("outfile","rb");
    newfile = fopen ("newfile", "w+b");
    fwrite (hash_array,sizeof(long),100,newfile);

   for (i = 0; i< 100;i++)
   {
      nexttmp = hash_array[i];
      if (nexttmp)
         hash_array[i] = ftell(newfile);

      numofitem = 0;
      while (nexttmp)
      {
        fseek(OutPutFile,nexttmp,SEEK_SET);
        fread (&itemtmp,sizeof(Item),1,OutPutFile);
        oldnext = itemtmp.next;

        if (oldnext)
         itemtmp.next = ftell(newfile)+sizeof(Item);
        fwrite (&itemtmp,sizeof(Item),1,newfile);
        nexttmp = oldnext;
        numofitem++;
      }

      numofitem = (numofitem*200)/100;
      itemtmp.next = 0;
      for (i2 = 0 ; i2< numofitem;i2++)
         fwrite (&itemtmp,sizeof(Item),1,newfile);
  }

  fseek(newfile,0,SEEK_SET);
  fwrite (hash_array,sizeof(long),100,newfile);
  fclose (OutPutFile);
  fclose (newfile);
}


void display()
{
  int largest=model[1];
  int largest_model=1;
  for (int i=1;i<model_id;i++)
    if(model[i]>largest)
    {
      largest = model[i];
      largest_model=i;
    }

  cout<<"The closest model is model "<<largest_model<<endl;

  for(int j=1;j<model_id;j++)
    cout<<model[j]<<endl;
}
```

APPENDIX C

3-Dimensional Implementation

```
/************************************************************************/
/*Program: 3d_hash.C                                                   */
/*Author : Joyce Ye Lu                                                 */
/*                                                                     */
/*This program builds the hash-table for each 3-dimensional object and*/
/*store it in a file named "hashtable.dat" in the memory for use in   */
/*the match program.                                                   */
/************************************************************************/



#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

#define Maxsize 100
#define model_size 3

struct Item
       {
           int Base,X,Y,Z;
           int num;
           Item *next;
       };

typedef Item *Itempointer;
Itempointer hash[100];

int model_num;
int base,x,y,z;
int I, J, K,L, size, key;
float X0,Y0,Z0,X1,Y1,Z1,X2,Y2,Z2, X3, Y3, Z3,U,V,W, NewU, NewV, NewW;
float A[Maxsize];

void getdata();
void find_U_V_W();
void find_I_J_K_L(int n);
void find_X_Y_Z();
void find_New_UVW();
void find_key();
void form_hashtable();
void insert(int);
void display();
double cal2(double[][2]);
double cal3(double[][3]);
void rev_matrix3(double[][3]);


main()
{
  form_hashtable();

  cout<<"please enter your model number:\n";
  cin>>model_num;

  getdata();
  find_I_J_K_L(size/3);
  display();
}


void getdata()
{
  char Inputfilename[25];
  cout<<"Please enter your input file name:";
  cin>> Inputfilename;

  fstream Instream;
  Instream.open(Inputfilename,ios::in);

  int i=0;
  while(!Instream.eof())
  {
     Instream>>A[i];
     i++;
  }

  Instream.close();
  size = i;
```

```
void find_I_J_K_L(int n)
{
      for(int i=0;i<n;i++)
          for(int j=0;j<n ;j++)
          {
           if(j != i)
             for( int k=0;k<n ;k++)
             {
               if ( k != i && k != j)
          for( int l=0;l<n; l++)
                  {
     if ( l!=i && l!=j && l!=k)
     {
        I=i;J=j;K=k;L=l;
                find_X_Y_Z();
                find_U_V_W();
      }
       }
     }
   }
}


void find_X_Y_Z()
{
X0 = A[3*I];
Y0 = A[I*3+1];
Z0 = A[I*3+2];
X1 = A[J*3];
Y1 = A[J*3+1];
Z1 = A[J*3+2];
X2 = A[K*3];
Y2 = A[K*3+1];
Z2 = A[K*3+2];
X3 = A[L*3];
Y3 = A[L*3+1];
Z3 = A[L*3+2];
}



void find_U_V_W()
{
for(int i=0; i<size/3;i++)
```

```
  if(i != I && i != J && i != K && i != L)
    {  U = A[3*i];
       V = A[i*3+1];
       W = A[i*3+2];
       find_New_UVW();
       find_key();
       base=I;
       x=J;
       y=K;
       z=L;
       insert(key);
    }
}


double cal2 (double matrix[][2])
{
  return (matrix[0][0]*matrix[1][1]-matrix[0][1]*matrix[1][0]);
}


double cal3 (double matrix[][3])
{
   double a,b;

   a = matrix[0][0]*matrix[1][1]*matrix[2][2]+
       matrix[0][1]*matrix[1][2]*matrix[2][0]+
       matrix[0][2]*matrix[1][0]*matrix[2][1];
   b = matrix[0][0]*matrix[1][2]*matrix[2][1]+
       matrix[0][1]*matrix[1][0]*matrix[2][2]+
       matrix[0][2]*matrix[1][1]*matrix[2][0];
   return (a -b);
}


void rev_matrix3 (double matrix[][3])
{
  double matrix3[3][3];
  double matrix2[2][2];
  int i,j,k,l;
  double matrix_value;

  matrix_value = cal3(matrix);
  for (i =0; i<=2;i++)
for (j=0;j<=2;j++)
{
/* row 0 */
    if (i ==0 && j==0)
    {
       matrix2[0][0] = matrix[1][1];
       matrix2[0][1] = matrix[1][2];
       matrix2[1][0] = matrix[2][1];
       matrix2[1][1] = matrix[2][2];
    }
    else
    if (i ==0 && j==1)
    {
       matrix2[0][0] = matrix[0][1];
       matrix2[0][1] = matrix[0][2];
       matrix2[1][0] = matrix[2][1];
       matrix2[1][1] = matrix[2][2];
    }
    else
    if (i ==0 && j==2)
    {
       matrix2[0][0] = matrix[0][1];
       matrix2[0][1] = matrix[0][2];
       matrix2[1][0] = matrix[1][1];
       matrix2[1][1] = matrix[1][2];
    }
  else
  /* row 1 */
    if (i ==1 && j==0)
    {
       matrix2[0][0] = matrix[1][0];
       matrix2[0][1] = matrix[1][2];
       matrix2[1][0] = matrix[2][0];
       matrix2[1][1] = matrix[2][2];
    }
    else
    if (i ==1 && j==1)
    {
       matrix2[0][0] = matrix[0][0];
       matrix2[0][1] = matrix[0][2];
       matrix2[1][0] = matrix[2][0];
       matrix2[1][1] = matrix[2][2];
    }
```

```
          else
          if (i ==1 && j==2)
          {
             matrix2[0][0] = matrix[0][0];
             matrix2[0][1] = matrix[0][2];
             matrix2[1][0] = matrix[1][0];
             matrix2[1][1] = matrix[1][2];
          }
          else
          /* row 2 */
          if (i ==2 && j==0)
          {
             matrix2[0][0] = matrix[1][0];
             matrix2[0][1] = matrix[1][1];
             matrix2[1][0] = matrix[2][0];
             matrix2[1][1] = matrix[2][1];
          }
          else
  if (i ==2 && j==1)
          {
             matrix2[0][0] = matrix[0][0];
      matrix2[0][1] = matrix[0][1];
    matrix2[1][0] = matrix[2][0];
    matrix2[1][1] = matrix[2][1];
  }
  else
          if (i ==2 && j==2)
          {
             matrix2[0][0] = matrix[0][0];
             matrix2[0][1] = matrix[0][1];
             matrix2[1][0] = matrix[1][0];
             matrix2[1][1] = matrix[1][1];
          }

     matrix3[i][j] = cal2(matrix2)/matrix_value;
     if (((i+j)/2)*2 != i+j)
        matrix3[i][j] = -matrix3[i][j];
   }
  }
   for (i=0;i<=2;i++)
      for (j=0;j<=2;j++)
         matrix[i][j]= matrix3[i][j];
}
```

```
void find_New_UVW()
{
     int i,j,k;
     double matrix[3][3] = { X1-X0, X2-X0, X3-X0,
                             Y1-Y0, Y2-Y0, Y3-Y0,
                             Z1-Z0, Z2-Z0, Z3-Z0 } ;
     double matrix1[3][1] = { U-X0,
                             V-Y0,
                             W-Z0};
     double result[3][1];

     rev_matrix3(matrix);

     for (i=0;i<=2;i++)
for (j=0;j<1;j++)
{
   result[i][j] = 0;
   for(k=0; k<=2; k++)
       result[i][j]= matrix[i][k]*matrix1[k][j]+result[i][j];
}
     NewU=result[0][0];
     NewV=result[1][0];
     NewW=result[2][0];
}



void find_key()
{
   key= (int) (NewU*210 + NewV*120 + NewW*212);
}

void form_hashtable()
{
  Item *tmp;
  char i[5],base[15],x[15],y[15],z[15],n[15];
  char line[100];

  for (int j=0;j<100;j++)
     hash[j]=NULL;

  FILE *fp;
  fp = fopen("hashtable.dat","r");
  while ( (fgets(line,100,fp))!=NULL)
  {
        sscanf(line,"%s%s%s%s%s%s",i,base,x,y,z,n);
        tmp = new Item;
        tmp->Base = atoi(base);
        tmp->X = atoi(x);
        tmp->Y = atoi(y);
        tmp->Z = atoi(z);
        tmp->num = atoi(n);
        int index=atoi(i);
        tmp->next = hash[index];
        hash[index]=tmp;

  }
  fclose(fp);
}


void insert(int n)
{
int index =abs( n) % 100;
Item *tmp;

tmp=new Item;
tmp->X = x;
tmp->Y = y;
tmp->Z = z;
tmp->Base = base;
tmp->num=model_num;

tmp->next=hash[index];
hash[index]=tmp;
}


void display()
{
Item *tmp;

char Outputfilename[] = "hashtable.dat";

fstream Outstream;
Outstream.open(Outputfilename, ios::out);
```

```
for(int i=0;i<100;i++)
  {
    tmp=hash[i];

while(tmp != NULL)
    {
        Outstream<<i<<" ";
        Outstream<<tmp->Base<<" "<<tmp->X<<" "<<tmp->Y<<" "<<tmp->Z
                 <<" "<<tmp->num<<endl;
        tmp=tmp->next;
    }
  }
}
```

```
/*******************************************************************/
/*Program: 3d_match.C                                              */
/*Author : Joyce Ye Lu                                             */
/*                                                                 */
/*This program compares the target objet with all the database    */
/*objects and finds out the closest one.                           */
/*******************************************************************/



#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>

#define Maxsize 100
#define model_num 6

struct Item
       {
           int Base,X,Y,Z;
           int num;
           Item *next;
       };

typedef Item *Itempointer;
Itempointer hash[100];

int I, J, K,L, size, key;
float X0,Y0,Z0,X1,Y1,Z1,X2,Y2,Z2, X3, Y3, Z3,U,V,W, NewU, NewV, NewW;
float A[Maxsize];
int model[model_num];

void getdata();
void find_U_V_W();
void find_I_J_K_L(int n);
void find_X_Y_Z();
void find_New_UVW();
void find_key();
void form_hashtable();
void display();
void find_model();
double cal2(double[][2]);
double cal3(double[][3]);
```

```
void rev_matrix3(double[][3]);


main()
{
  form_hashtable();

  getdata();
  find_I_J_K_L(size/3);
  display();
}


void getdata()
{
  char Inputfilename[25];
  cout<<"Please enter your input file name:";
  cin>> Inputfilename;

  fstream Instream;
  Instream.open(Inputfilename,ios::in);

  int i=0;
  while(!Instream.eof())
  {
     Instream>>A[i];
     i++;
  }

  Instream.close();
  size = i;
}


void find_I_J_K_L(int n)
{
      for(int i=0;i<n;i++)
          for(int j=0;j<n ;j++)
          {
           if(j != i)
             for( int k=0;k<n ;k++)
             {
               if ( k != i && k != j)
          for( int l=0;l<n; l++)
```

```
                        {
        if ( l!=i && l!=j && l!=k)
        {
           I=i;J=j;K=k;L=l;
                       find_X_Y_Z();
                       find_U_V_W();
         }
          }
        }
     }
  }


  void find_X_Y_Z()
  {
  X0 = A[3*I];
  Y0 = A[I*3+1];
  Z0 = A[I*3+2];
  X1 = A[J*3];
  Y1 = A[J*3+1];
  Z1 = A[J*3+2];
  X2 = A[K*3];
  Y2 = A[K*3+1];
  Z2 = A[K*3+2];
  X3 = A[L*3];
  Y3 = A[L*3+1];
  Z3 = A[L*3+2];
  }



  void find_U_V_W()
  {
  for(int i=0; i<size/3;i++)
   if(i != I && i != J && i != K && i != L)
     {  U = A[3*i];
        V = A[i*3+1];
        W = A[i*3+2];
        find_New_UVW();
        find_key();
        find_model();
     }
  }
```

```
double cal2 (double matrix[][2])
{
  return (matrix[0][0]*matrix[1][1]-matrix[0][1]*matrix[1][0]);
}


double cal3 (double matrix[][3])
{
   double a,b;

   a = matrix[0][0]*matrix[1][1]*matrix[2][2]+
       matrix[0][1]*matrix[1][2]*matrix[2][0]+
       matrix[0][2]*matrix[1][0]*matrix[2][1];
   b = matrix[0][0]*matrix[1][2]*matrix[2][1]+
       matrix[0][1]*matrix[1][0]*matrix[2][2]+
       matrix[0][2]*matrix[1][1]*matrix[2][0];
   return (a -b);
}


void rev_matrix3 (double matrix[][3])
{
  double matrix3[3][3];
  double matrix2[2][2];
  int i,j,k,l;
  double matrix_value;

  matrix_value = cal3(matrix);
  for (i =0; i<=2;i++)
  {
      for (j=0;j<=2;j++)
      {
      /* row 0 */
          if (i ==0 && j==0)
          {
             matrix2[0][0] = matrix[1][1];
             matrix2[0][1] = matrix[1][2];
             matrix2[1][0] = matrix[2][1];
             matrix2[1][1] = matrix[2][2];
          }
          else
          if (i ==0 && j==1)
          {
```

```
      matrix2[0][0] = matrix[0][1];
      matrix2[0][1] = matrix[0][2];
      matrix2[1][0] = matrix[2][1];
      matrix2[1][1] = matrix[2][2];
  }
  else
  if (i ==0 && j==2)
  {
      matrix2[0][0] = matrix[0][1];
      matrix2[0][1] = matrix[0][2];
      matrix2[1][0] = matrix[1][1];
      matrix2[1][1] = matrix[1][2];
  }
else
/* row 1 */
  if (i ==1 && j==0)
  {
      matrix2[0][0] = matrix[1][0];
      matrix2[0][1] = matrix[1][2];
      matrix2[1][0] = matrix[2][0];
      matrix2[1][1] = matrix[2][2];
  }
  else
  if (i ==1 && j==1)
  {
      matrix2[0][0] = matrix[0][0];
      matrix2[0][1] = matrix[0][2];
      matrix2[1][0] = matrix[2][0];
      matrix2[1][1] = matrix[2][2];
  }
  else
  if (i ==1 && j==2)
  {
      matrix2[0][0] = matrix[0][0];
      matrix2[0][1] = matrix[0][2];
      matrix2[1][0] = matrix[1][0];
      matrix2[1][1] = matrix[1][2];
  }
  else
  /* row 2 */
  if (i ==2 && j==0)
  {
      matrix2[0][0] = matrix[1][0];
      matrix2[0][1] = matrix[1][1];
```

```
               matrix2[1][0] = matrix[2][0];
               matrix2[1][1] = matrix[2][1];
            }
            else
   if (i ==2 && j==1)
            {
               matrix2[0][0] = matrix[0][0];
         matrix2[0][1] = matrix[0][1];
      matrix2[1][0] = matrix[2][0];
      matrix2[1][1] = matrix[2][1];
   }
   else
            if (i ==2 && j==2)
            {
               matrix2[0][0] = matrix[0][0];
               matrix2[0][1] = matrix[0][1];
               matrix2[1][0] = matrix[1][0];
               matrix2[1][1] = matrix[1][1];
            }

       matrix3[i][j] = cal2(matrix2)/matrix_value;
       if (((i+j)/2)*2 != i+j)
          matrix3[i][j] = -matrix3[i][j];
    }
   }
     for (i=0;i<=2;i++)
        for (j=0;j<=2;j++)
           matrix[i][j]= matrix3[i][j];
 }


void find_New_UVW()
{
      int i,j,k;
      double matrix[3][3] = { X1-X0, X2-X0, X3-X0,
                              Y1-Y0, Y2-Y0, Y3-Y0,
                              Z1-Z0, Z2-Z0, Z3-Z0 } ;
      double matrix1[3][1] = { U-X0,
                              V-Y0,
                              W-Z0};
      double result[3][1];

      rev_matrix3(matrix);
     for (i=0;i<=2;i++)
for (j=0;j<1;j++)
{
   result[i][j] = 0;
   for(k=0; k<=2; k++)
       result[i][j]= matrix[i][k]*matrix1[k][j]+result[i][j];
}
     NewU=result[0][0];
     NewV=result[1][0];
     NewW=result[2][0];
}



void find_key()
{
   key= (int) (NewU*210 + NewV*120 + NewW*212);
}


void form_hashtable()
{
  Item *tmp;
  char i[5],base[15],x[15],y[15],z[15],n[15];
  char line[100];

  for (int j=0;j<100;j++)
     hash[j]=NULL;

  FILE *fp;
  fp = fopen("hashtable.dat","r");
  while ( (fgets(line,100,fp))!=NULL)
  {
        sscanf(line,"%s%s%s%s%s%s",i,base,x,y,z,n);
        tmp = new Item;
        tmp->Base = atoi(base);
        tmp->X = atoi(x);
        tmp->Y = atoi(y);
        tmp->Z = atoi(z);
        tmp->num = atoi(n);
        int index=atoi(i);
        tmp->next = hash[index];
        hash[index]=tmp;
  }
  fclose(fp);
}


void find_model()
{
 Item *current;
 int i;
 int index = abs(key) % 100;

 current = hash[index];
 while(current != NULL)
 {
   if (current->Base==I && current->X==J && current->Y==K && current->Z==L)
       model[current->num]++;
   current = current->next;
  }
}



void display()
{
  int largest=model[1];
  int largest_model=1;

  for (int i=1;i<model_num;i++)
    if(model[i]>largest)
    {
      largest = model[i];
      largest_model=i;
    }

  cout<<"The closest model is model "<<largest_model<<endl;

  for(int j=1;j<model_num;j++)
    cout<<model[j]<<endl;
}
```

REFERENCES

1. Y. Lamdan, "Object Recognition by Geometric Hashing," Ph.D. dissertation, Dept. Computer Science, New York University, New York, NY, May 1989.

2. J. S. Vitter and W. Chen, *Design and Analysis of Coalesced Hashing*, Oxford University Press, New York, NY, 1987.

3. Y. Lamdan, J. T. Schwartz and H. J. Wolfson, "Affine Invariant Model-Based Object Recognition," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 5, pp. 578-590, October 1990.

4. D.P. Huttenlocher and S. Ullman, "Object Recognition Using Alignment," in *Proceedings of the first Int. Conf. on Computer Vision*, 1987, pp. 102-111, London.

5. J. T. Schwartz and M. Sharir, "Identification of Partially Obscured Objects in Two Dimensions by Matching of Noisy 'Characteristic Curves'," *The Int. J. of Robotics Research*, vol. 6, no. 2, pp. 29-44, 1987.

6. R. Cyganski and J. A. Orr, "Application of Tensor Theory to Object Recognition and Orientation Determination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, no. 6, pp. 662-673, Nov. 1985.

7. E. Kishon and H. J. Wolfson, "Three Dimensional Curve Matching," *Proceedings of the AAAI Workshop on Spatial Reasoning and Multisensor Fusion*, October 1987, pp. 250-261, St. Charles, Illinois.

8. J. Hong and H. J. Wolfson, "An Improved Model-Based Matching Method Using Footprints," *Proceedings of the Int. Conf. on Pattern Recognition*, October 1988, Beijing, P.R.China.

9. M. R. Lindeburg, *Engineer-in-Training Reference Manual, 8th Edition*, Professional Publications Inc., Belmont, CA, 1992.

10. I. Rigoutsos, D. Platt and A. Califano, "Flexible Substructure Matching in Very Large Databases of 3-D Molecular Information," Computational Biology and Pattern Matching Group, I.B.M., T. J. Watson Research Center, Yorktown Heights, N.Y., May 1996.