School of Computing

The Implementation of Geometric Hashing on Parallel Architecture using a Single Instruction Multiple Data (SIMD) Parallel Computer.

Michael N Leishman

This thesis is presented as part of the requirements for the award of

Degree of Master of Science

Curtin University of Technology

Western Australia

December 1995

1. Abstract

This paper covers issues in the implementation of Geometric Hashing for recognition of the stable states of a 3D model from features in an image on a Massively Parallel Single Instruction Multiple Processor (SIMD) computer. Geometric Hashing is a two phase technique in which the first or learning phase processes model features to create the hash table, and the second or recognition phase processes image features and uses the hash table to determine if the model matches the image. The similarity is indicated by the number of votes for a particular model and geometric transformation.

The algorithm lends itself to parallel implementation because (1) access to all the image features and (2) indexing into the hash table, can be performed independently. To improve efficiency and to be able to implement on a limited number of processors with limited memory, the image space is virtualised and the hash table is remapped to obtain a pseudo-uniform distribution when scaled and distributed over the processors. An analysis is presented for the recognition of one or more instances of a stable state of the model within an image taking into account noise, spurious data and occlusion.

Geometric Hashing is implemented on a 2k processor DECmpp 12000 computer. The mapping of the image data is in place and is distributed over the processor array using hierarchical virtualisation. The hash table data is distributed over the processor array using cut and stack virtualisation. Each processor is capable of recognising one or more instances for each image feature stored on it. Results are given for the recognition of instances of the stable states of a 3D object.

2. Acknowledgment

The research and the writing of this paper has not been without difficulty. It is the culmination of 12 years of my life in which I have formally studied computer science, my children have become adults and my wife and I have become older. (Overnight it seams).

To begin with, I would like to thank my children, now adults, for giving me the time that was really theirs. I cannot replace the days in the park, the camps, the bike rides and the time in front to the TV with them that did not happen. However, I only hope that they will get some inspiration and know that it is never to late to do something worthwhile (academically that is!).

Next, thankyou to my wife who thinks I am insane. Her love and sufferance is more than a mere mortal can expect. Thankyou Jan.

My supervisors, Dr. Geoff West and Dr Svetha Vankatesh always believed that what I was doing was of the correct calibre and worthwhile. I enjoyed my arguments with Geoff and Svetha and valued their commitment, humour and input (in that order!).

I would also like to formally thank Curtin University, the Dean of Computer Science Dennis Moore and Head of School Terri Caelli and the school of Computer Science generally for allowing me the opportunity to undertake this course of study. Thanks also to Mary, Michelle, Mike and Peng and the others who wondered who I was when I occasionally visited the hallowed halls of CS.

Finally, thank you to all of my colleagues who believed in me.

3. Table of Contents

4. List of Figures

5. List of Tables

1. General Introduction

1.1 Introduction

Object recognition is an area of considerable research. The exponential increase in the number of robots, and the need for them to interact with the environment demands that they can recognise objects within their domain. Two major areas of research in terms of object recognition are (1) evidence based (Caelli & Dreier, 1994) and (2) model based (Chin & Dyer, 1986). An evidence based system relies on intensities defined by depth maps and models defined as aspect graphs or view dependant CAD data. A model based system requires criteria to be met for the recognition of models in an image. Generally, a model can be recognised when there is a correspondence between points in the image and points in the model, and where there is a mapping of parameters of the model to the object that allows alignment which is defined as pose.

The criteria for a good recognition system is founded on the following attributes of speed, accuracy and flexibility (Chin & Dyer, 1986). These criteria rely upon a high speed computer system that can process the large amount of data required in real time and have software that is able to correctly find the required object in terms of location and orientation, and be flexible enough to find a variety of different object in a wide variety of scenes. The systems and method this thesis supports endeavours to fulfil these objectives and are discussed in detail within its body.

Thus, in this thesis, I will be investigating a method for model based object recognition that can accurately determine the existence of an instance of a 3D model in real time using a massively parallel Single Instruction Multiple Data computer system.

1.2 Background

Computer vision systems have been in existence for about forty years and more recently, the number of systems to enable the recognition of objects has increased. These systems rely on the existence of a model of the object, or multiple instances of the object to be recognised. A model instance is compared with an image potentially containing at least one model instance and the objective is to recognise that model instance. There are three recognised approaches to model based object recognition (Chin & Dyer, 1986). One of these methods uses global features such as texture and area to determination the existence of a model, another method uses structural features which are abstract geometrical representations of images using points, line and curves, and the third method uses relational graphs which relate geometric structures such as points and lines. The method best suited to a parallel implementation is the structural feature method as features can be extracted and an identical process be performed on each feature independently to provide recognition. As well, most systems only cope with determining the existence of one model instance at a time so where more than one model instance in an image is possible, the time for recognition becomes dependant on the number of models. A structural feature methods employing Footprints (Hong & Wolfsen, 1988), Affine Transforms (Lamden, Schwartz & Wolfsen, 1988) and Geometric Hashing (Lamden & Wolfsen, 1988) are able to recognise more than one model instance at a time. The reason for this is made evident within the body of the thesis.

The capabilities of these systems are still very limited and processor intensive. With the decrease in the price of computer memory and the proliferation of inexpensive processors, the use of massively parallel super-computers to increase the speed of recognition has become a reality. The DECmpp 12000 (acker Maspar) and the CM-5 Connection Machine are two such systems that offer large numbers of processors and appropriate memory on each processor to allow the realistic implementation of Single Instruction Multiple Data (SIMD) processing of large images for the propose of model based object recognition. The DECmpp 12000 can have up to 16k processors with 16k bytes of RAM memory on each processor.

1.3 Approaches

The structural feature method was chosen for the implementation of model based object recognition on the DECmpp 12000 due to its applicability to a parallel implementation. A recently proposed method for 2D and 3D object recognition that is suitable for parallel implementation is Geometric Hashing. It is the extension to earlier methods using Footprints and Affine Transforms. This method uses a two phase technique in which the first (learning) phase processes combinations of model features to create a hash table where geometric data is stored, and the second (recognition) phase processes image features and uses the hash table data to determine if the model matches the image by counting the number of votes for a particular model. These votes are stored in an accumulator. The image being processed is stored in place as opposed to storing and processing the vertex set. The hash table is used to compactly store those feature coordinates which are used in the voting process and vote for a particular model. This thesis discusses the concepts of Geometric Hashing in the context of mapping the algorithm onto a SIMD architecture computer system. It differs from other implementations on SIMD machines such as the CM-5 in that it determines the best way to store the hash data on the array ie the hash table which is distributed across the array, it processes features in place and in parallel and performs parallel look-up into the hash table. Other implementations process features sequentially and use the array to do parallel look-ups into the hash table.

1.4 Thesis Outline

Chapter 1 is an introduction and discusses the broad objectives of the thesis.

Chapter 2 provides a definition to the basic problem of model based object recognition, defines correspondence and pose in relation to recognition and provides details of global matching, relational graphing and the use of structural features in recognition. It also provides the basis for which Geometric Hashing was chosen as a method for implementing model based object recognition.

Chapter 3 describes the background to hashing techniques and parallel processing. It also provides a summary of the attributes of the DECmpp 12000 computer system. Although not directly related to object recognition, this chapter is necessary to give the reader the background to the concepts of hashing and parallel programming allowing an understanding of the techniques used in the parallel implementation of Geometric Hashing on the DECmpp 12000 computer.

Chapter 4 is a literature survey of Geometric Hashing of both the serial and parallel implementation of the technique. The first instance of Geometric Hashing was referenced by Lamden & Wolfsen (1988) although work by Lamden, Schwartz & Wolfsen (1988) relating to Affine Invariant Matching and by Hong & Wolfsen

(1988) relating to Footprints are seen as precursors to Geometric Hashing. Later work by Prassana & Wang (1993) and that of Rigoutsos & Hummel (1992) describe the techniques used to map the Geometric Hashing algorithm onto SIMD parallel computers.

Chapter 5 relates the implementation of Geometric Hashing on a serial computer. One view of the 3D widgit, later used for parallel implementation, is used to test the algorithm employed in this implementation. Hash table generation, scale determination, image representation, threshold determination and error analyses are discussed in detail. Finally, the results of this serial implementation are presented and discussed.

Chapter 6 provided the method of the parallel implementation of the algorithm on the DECmpp 12000. Important issues discussed primarily relate to the mapping of the algorithm, the image data and the hash table onto a SIMD parallel architecture. Other issues discussed are methods to prevent serialisation of algorithm, threshold determination, access of image data and the accumulation of votes for model instances. The results obtained for a system to recognise one or more of the seven stable states of the widgit used are presented and discussed.

Chapter 7 is the conclusion. In this, the thesis is summarised and conclusions are drawn. As well, future directions in terms of the parallel implementation of Geometric Hashing are presented.

Appendix 1 discusses the preprocessing of image date that is assumed to be carried out in place on the parallel array.

Appendix 2 discusses the distance transform used and the techniques employed to verify and confirm the existence of model instances.

2. Object Recognition

2.1 Introduction

In an industrial situation where there are limited numbers of objects to be recognised, it has been shown (Chin & Dyer, 1986) that model based recognition systems are able to recognise one or more objects within an image, even in the presence of noise, spurious data and in situations where the object has a degree of occlusion. The mechanism of recognition allows the user to employ geometric constraints of the object's model as a means to recognise that object's correspondence and pose within a given image data set.

Model-driven object recognition algorithms employs object models to predict image features and then seek to find these features in the image data or in the low level data extracted from the image (Perro & Hamley, 1991).

Three approaches to model based object recognition will be analysed. From this analysis, and that of the method of Geometric Hashing, it should become evident as to the reasons why a parallel implementation of Geometric Hashing using affine transforms has been chosen as the means to simply and quickly recognise objects within an image.

2.2 Problem Definition

A representation of an object or objects is contained within an image, the data of which is available to a computer system. Models of the objects of interest are available and is stored in machine readable form. The models describe some of the properties of the objects such as shape, length of lines, end points of lines and angle between lines of importance. Thus, the description may not need to be complete or exact for recognition to take place. The problem of recognition is to use this geometric information to produce a set of *feasible interpretations* of the data with respect to the objects (Grimson, 1990).

For a solution to the recognition problem to be considered good, the following criteria must be satisfied:

2.2.1 Speed and Efficiency

The recognition of the object is nominally done in real time. Efficient algorithms need to be employed that will allow recognition to take place in a fraction of a second. Efficiency is measured in terms of run time and algorithm complexity.

2.2.2 Accuracy and Correctness

The algorithm must produce the correct interpretation of the data. Accurate determination of the object's correspondence and pose in the image is essential to the task. The algorithm must not decide the existence of the wrong object or an object in the wrong place, and it must not fail to find an object when it exists.

2.2.3 Flexibility and Robustness

In useful system, the presence of noise and spurious data should not prevent recognition from occurring. Noise occurs when the features of the object are not exactly in the position expected. This can be a consequence of the optical system used, measurement error and preprocessing. The affect of this is to require the system to broaden the error bounds with a consequential increase in processing time and complexity. Spurious data result from many sources and is data that is not associated with the object under consideration. Such sources include shadows and features of other nearby objects within the scene. This increases the set of possible features used in recognition with a subsequent increase in processing time.

Further, occlusion of parts of the object should not prevent recognition from taking place. Occlusion occurs when part of the object is obscured from view. This usually occurs when some other object or objects obscures part of the object or objects to be recognised. The consequence of this is that some of the features of the objects are removed from consideration in the recognition process.

As the amount of noise and spurious data in the data set and the level of occlusion increases, the recognition method must degrade smoothly. That is, the system must not be sensitive to small changes in conditions.

2.3 Search Problem - matching problem

For all model based object recognition systems, correspondence and pose need to be determined before recognition can be deemed to be effective. Many early systems employed global matching using properties such as volume, surface area and centre of gravity. These could not meet the criteria of a good solution (Grimson, 1990) on their own but can be useful in the final verification stage. Methods employing only global features will be considered to a limited extent within this dissertation. Those systems that allow determination of correspondence and pose will be considered in more detail. Such a system employs feature matching to infer correspondence and pose by mapping local data feature to model feature data. These features include edges, corners, holes to name a few and these systems can handle occlusion and spurious data.

Correspondence Definition

The pairing of individual data features with model features, each such pairing representing an assertion that the identified data feature is an instance of the corresponding model feature (Grimson, 1990).

Pose Definition

The space of all possible transformations of the model onto the data. It is a parameter space whose axes represent possible values for the parameters that determine the transformation (Grimson, 1990).

The steps that are involved in the search process for model based object recognition are outlined below:

- Define and structure search space.
- Define the data-model mapping.
- Map data and model to common representation.
- Define means of determining the worth of a feature in search space.
- Define means of exploring search space.
- Define method of determining pose of matching.

Most systems fall into two basic categories based upon the searching and verification techniques employed. These are (1) the relational graph method and (2) the structural feature method. Whereas the relational graph method usually requires all possible model and image matches to be considered before recognition takes place, the structural feature method employs a hypothesis and verify method whereby not all possible combinations need to be considered before verification can occur. This method is the one that eventually leads to the development of the Geometric Hashing method (Lamden & Wolfsen, 1988).

2.3.1 Global matching Systems

Global parameters used in global matching systems can include parameters such as perimeter, centroid, distance of contour points from the centroid, curvature, area, moments of inertia and texture. The most common method is to use a set of 2D global shape features for each stable view of the object and the features of the object in the image is directly compared to that of the model. The major problems which occur in recognition include the lack of ability to cope with occlusion and the requirement that model data must be exactly extracted in terms of orientation and lighting. Some exemplar recognition systems follow:

The first description is a general example of global parameter matching (Grimson, 1990). The model is considered as a flat 2D object which is represented as a binary image. The image contains the model and is also represented as a 2D binary image *b*(*x*,*y*). As the binary image can be represented as a set of pixels having values of 1 or 0, the model has a characteristic function, defined as follows:

$$b(x,y) = \begin{cases} 1 \text{ if the point (x, y) is on the object} \\ 0 \qquad\qquad \text{otherwise} \end{cases} \quad (1)$$

Each model is processed to produce a set of moments which is used to create a set of vector parameters that can be compared.

The zeroth order moment, which indicates the area, is

$$A = \sum_x \sum_y b(x,y) \quad (2)$$

Higher order moments are define by

$$M_{p,q} = \sum_x \sum_y x^p y^q b(x,y) \quad (3)$$

A set of moments can now be computed that can be used to form a vector of parameters which will identify the model. One such set, described by Bamieh & De Figueiredo (1986) is claimed to form a complete system. By comparing the vectors of the image and the model, the best match can be chosen.

Once the match is chosen, some of the higher moments can be used to determine the pose of the object in the image. The scaling parameter can be determined by finding the difference between the zeroth moment of the model and that of the object in the image. The translational parameter can be determined by using the zeroth moment and the first two moments. The moment about the *x*-axis can be shown to be $M_{0,0} x_c = M_{1,0}$ and the moment about the *y*-axis can be shown to be $M_{0,0} y_c = M_{0,1}$ where the centre of area is given by the point (x_c, y_c). The rotational parameter is obtained by assuming the object has some elongation which will allow the determination of orientation. After transformation of the image to a coordinate frame about the centre of mass, the second moments $M_{2,0}$, $M_{1,1}$ and $M_{0,2}$ are used to determine the angle of rotation θ of the image object with respect to the model.

Some of the problems of this method arise in the binarisation of the image. This requires strict control in terms of the orientation of the object and the lighting to ensure a consistency in shadow, reflectance and light intensity. Extending the first moment to include a grey scale does not assist in this area. Although the method shows robustness in the presence of noise, it cannot effectively deal with clutter and, in particular, occlusion. The computation of global measurements ensures that any overlapping objects are treated as the same model.

Some exemplar systems using global parameters are described below:

System using connected components (Gleason & Agin, 1979). The SRI Vision Module uses sets of global features which are selected by the user to construct an object's model as feature vectors. For the image, each connected component is extracted for independent analysis. Gross scalar descriptors (number of holes, area, perimeter, boundary chain code, compactness, number of corners and moments of inertia) are determined for each component to be used in the matching phase.

In the matching phase, one of two methods are used. The first employs a decision tree method and is based on a list of global features associated with each model. At the root level, the feature that give the largest separation between models is used. The two children of the root node are models with features less than or equal to a threshold on the left and the remainder on right. This is repeated recursively until the leaf node contains a single model. The second method uses a statistical pattern recognition scheme in feature space to match a model to a given list of global features extracted from an image.

Identifying grasp-points of randomly oriented cylinders (Kelley et al., 1982). The image contains many cylinders all of the same type but with different positions and orientations. Regions in the image are reduced to components of connected pixels. These are sorted by size and the largest selected as the *grasp-point*. The centroid and axes of minimum and maximum moments of inertia through the centroid are calculated to determine the location and angle of orientation. The ratio of the eigenvectors of the axes of minimum and maximum moments of inertia is used to determine angle of inclination of the cylinder.

Another system (Birk, Kelly & Martins, 1981) uses the centroid and minimum moment of inertia to orient the image where the object is overlaid with a grid pattern, the center of which is at the centroid and the orientation is in relation to the minimum moment of inertia. A simple count of pixels determines existence of the object. The CONSIGHT-1 system (Holland, Rossel & Ward, 1979) determines the centroid and area of objects as they pass along a conveyer belt by illuminating each object with a special light source and capturing the image as it passes a line scan camera.

2.3.2 Relational Graph method

Geometric relations are represented structurally as a graph representation of a part. The graph is constructed using local features and relationships between these features. Local features include such features as points and corners and are used because they are simpler and can be selectively detected. Recognition can still take place even in the presence of noise, spurious data and occlusion.

In the graph, each node represents a local feature which is labelled with its properties. Recognition is a graph matching process where a search is carried out for the largest possible pairing of model and image features for which there exists a transformation mapping each model feature to its corresponding image feature. (Huttenlocher & Ullman, 1987) In the correspondence space, all possible matching of data and model features are considered. The focus is on the matching process which attempts to find an association between image features and model features. Model features are given a unique label and attribute set as is the image data. All possible matches of both sets of data are considered and are deemed consistent if there exists a rigid coordinate frame transformation that maps each model feature onto a data feature.

A basic algorithm (Grimson, 1990) is described for a given image space and one model. The first step is to choose the features that are to be matched. Next, one feature f_1 in the image is chosen and it is matched with a feature F_1 in the model. The same is done for all features in the model to get a single level tree structure such that f_1 in the image is matched with model features $F_{j,\, j=1,..j}$. At the next level in the tree, the second image feature f_2 is chosen and again matched for model features $F_{j,\, j=1,..j}$. The same is done for all image features $f_{m,\, m=1..m}$ to create a tree structure which exponentially grows with a depth of m where m is the number of features in the image.

An interpretation tree is used to allow for a guided search without treating the entire search space and uses a depth-first, backtracking search. A hypothesis is made and it is tested to verify the hypothesis once a leaf on the tree is reached.

Exemplars of systems using relational graph matching follow:

Two level model of coarse and fine features method (Yachida & Tsuji, 1977). This system uses a simple feature graph representation plus a two level model (course-to-fine) to speed the search process. Each object is described by a set of viewpoints, each containing a coarse representation using global features plus a description of the boundary using polar coordinates. The fine level is based on a high resolution image and includes features such as holes, edges, texture and boundary. An attribute list containing location and extraction likelihood is associated with each feature and is ordered in terms of the reliability of each feature.

In the matching phase, the image data and model graph are examined to allow the proposal of the next matching step. Course and fine resolution allow matching using simple features as cues. Object boundary matching provides a match angle to allow for rotation. For a given feature extracted, a dissimilarity measure is made and tallied. If the measure is above a threshold, the model is rejected. Successive comparisons with feature analysers provide further cues as to the next feature to be employed.

Local feature focus method (Bolles & Cain, 1982). An object is considered to consists of three parts: (1) polygonal approximation of border, (2) list of local features specified by name, type, position, orientation and rotational symmetry, and (3) for each distinct feature, an unambiguous description of feature in all models. Each feature type can be used as a focus feature.

In matching, the largest cluster of image features is found using a graph matching technique. Models with lists of focal features and nearby features are loaded. For each image, all potentially useful local features are located and a cluster algorithm is used to hypothesise part occurrences. Template matching verifies the hypothesis.

After all the features in the image are located, the focus feature is selected and the system searches for a cluster of consistent secondary features. As it finds matching features, a list of possible model-to-image feature assignments is determined. This is transformed into a graph by creating a node for each assignment pair and adding an arc between pairs of nodes referencing the same model. Finally, the object is verified by testing the object features and checking the boundary.

2.3.3 Structural feature method

This employs a precise geometric representation of model features. These features are local, describe a part of the object and include ordered lists of points, lines, arcs and corners. The ordering of the features, in early methods, was important and often included ordered lists and sequences of equations and usually related to the object's boundary. In later methods, indexing and voting systems that allowed searching and verification of the image without the need to exhaustively search the entire image space before concluding the existence of an objects existence were employed. Geometric Hashing using footprints and affine transforms fall into this category.

Grimson considers this as searching the pose space in contrast to the relational graph method which he considers as searching the correspondence space. The pose space is sampled at some regular interval and a transformation is used to map the model onto the image. Values are scored or tallied to determine if matching takes place. The highest score is used to hypothesise the existence of the object in the image. In some instances, it is considered computationally expensive as there is a need to match one model at a time, and, in the event that the model can be oriented in terms of translation, rotation and scale, can have many instances to compare in determining the pose.

Recognition uses a hypothesis-verification method. The model's features are used to predict the existence of it in the image. A verification stage is then employed which measures parameters of the model against the image to confirm its existence. If verification is not concluded, the process is continued until existence is confirmed or the image set is exhausted.

With all of the earlier methods mentioned below, the image space is searched for the existence of one model at a time. This has the affect of making the algorithm timing dependent on the number of models. With the methods that employ a form of Geometric Hashing, the voting system allows for the potential recognition of any model instance concurrently. This greatly enhances the speed at which recognition can take place. As well, these systems lend themselves to parallel implementation as the same process is applied to all features of the image.

The following are examples of this method and lead to the development of Geometric Hashing.

Line and Arc Boundary Segment Method. Perkins (1978) uses boundary segments such as line segments and arcs to construct a 2D model called a concurve. A property list for each concurve is constructed. This includes such types as circles, arcs, line segments and complex curves as well as total length or radius, bending energy, compactness and rotational symmetry. A concurve is divided into multisectors which are short line segments placed at equal intervals along the curve and are oriented perpendicular to the tangent to the curve.

A model of an object is divided into groups of concurves which represents shape and which, because of curve smoothing, are relatively insensitive to noise.

Matching involves the following two stages. (1) Extraction of scalar measurements (lengths, area) from the image. Model concurves are then compared using exhaustive matching algorithm to return an ordered list. (2) Once a concurve of a model is matched against one image concurve, the transform from model to image can be determined. Successive comparisons take place using the list data until a tentative transformation is found. A global check is used to confirm matching of the complete model in the image.

Hierarchical Boundary Segment. Shirai (1978) represents edges by a description of curvature in terms of an equation and the end points of the curve. These consist of long, connected edge segments that describe part of the boundary of the object. The main feature which is considered the most obvious is used to imply presence in the image. The secondary features are details of the object. In the matching process the main feature is found in the image first. Secondary features are then searched for to give extra evidence and verify the main feature. Finally, recognition is confirmed by matching other features such as lines.

Image and Model Boundary Segment Pair Clustering (Stockman, Kopstein & Benett, 1982) uses real vectors to describe boundary segments between extracted point pairs. These contain direction, position and size information. Abstract vectors are used to link primitive features such as holes, curves and intersections. The holes are detected by using a mask and the other features detected by an edge linking algorithm. The set of vectors uses an object centred coordinate system and is defined by modelling rules that permit only certain combinations of features to be linked.

A clustering procedure is used to match the model to the image. All possible pairs of image and model features are matched on the basis of local evidence. In the cluster space points are used to indicate a match. Where there is a cluster of points, it is assumed there is a good matching of features in the model and image. Randomly placed objects require rotation, translation and scaling to derive parameters for all possible pairs of features. This method appears to be roust as it integrates all local information before confirmation of model presence in the image is made.

Alignment (Huttenlocher & Ulman, 1987). This is the first method that effectively uses a two stage algorithm, although it only allows the recognition of one model at a time.

A preprocessing stage identifies features in the model, and later in the image, that can be used for initial alignment. In the first stage, a number of chosen model features are aligned with the same number of image features. For example, two points on the model (a_m, b_m) are aligned with two points in a 2D image (a_i,b_i) after appropriate translation, rotation and scaling.

In the second stage, the model is transformed into image coordinates. Once the position and orientation has been determined, the model can be directly compared with the image. Recognition is deemed to have occurred when an alignment defines the transformation that best maps the model into the image. The scoring is done by transforming the model to image and mapping model edges to image edges. The best score is the one that maps most model edges onto image edges.

The system used for feature extraction and verification is similar to that used in this thesis in that a labelling scheme reduces the search and minimal information is used for alignment and recognition. The process can be summarised in the following steps.

- Extract the edges.
- Create a set of single connected contours.
- Determine the local orientation and curvature of the contours.
- Class the contours as line, corner or arc.
- Combine connected or nearby segments to form alignment features.
- Match alignment features in the image against each possible alignment feature of the model. For each pairing, calculate the pose.
- Verify the existence of the model by aligning the model with the image using an edge correspondence method.

The major variation to our method is the creation of the data base once the features are extracted, and the use of the database to determine the existence of any model in the database simultaneously. This is dealt with in more detail in the following chapter.

Footprint Method (Hong & Wolfsen, 1988) This method is also a two component method which introduces a hashing technique in the recognition of model instances in an image which may have undergone translation and rotation, but not scale. Footprints are shape signatures which allow comparison between segments of boundary curves. The characteristics of the curves are defined as local, translation and rotation invariant and stable. It is also desirable that an approximation of the observed curve can be constructed from the footprint and that different local curve segments will produce different footprints.

The optimal footprint is one which is based on arc-length *vs* turning-angle graph where the boundary is sampled at regular intervals and, at these positions, the average turning angle difference computed. The footprint is identical for curves which undergo translation and differs by a constant for the curves which undergo rotation.

The characteristic function or graph $f(u)$ of a polygonal curve C is derived. Given the constants a, b, c and d, the following transformation

$$s=au$$

$$\theta(s)=bf(u)+cu+d$$

translates $f(u)$ to the standard arc-length *vs* turning-angle graph $\theta(s)$.

In the preprocessing phase, all model objects are processed off-line to determine characteristic functions and footprints. The boundary curve is scanned and footprints generated at equally spaced points. The footprints are used as a key to the hash table which records the object and sample point for the generated footprint.

In the recognition phase, the following steps are carried out with respects to the image.

- The boundary curves are scanned and footprints generated at equally spaced points.
- For each footprint, the hash table is checked and for each pair of points (object, sample point), a vote is accumulated for the object and the relative shift between the object and the scene.
- Pairs with the most votes are chosen and the start/end points between footprints of the scene and the model for a given shift are chosen using alignment. Minor discrepancies are allowed for.

- Matching substrings are used to compute the appropriate endpoint of matching subcurves on the boundary curve of the model and scene. These are then matched using a least square match algorithm.
- An object with the longest matching subcurve is considered as part of the scene. The remaining subcurves are discarded and the process is repeat for the remaining subcurves in the reduced scene.

The method described above is similar to our implementation in that a two phase algorithm is used in which a geometric description of all models to be considered is placed in a database, which is then used to rapidly select those features in the image that indicate the presence of model instances. A final verification stage is required to determine both correspondence and pose. In this case, instances where the object undergoes scale and skew cannot be recognised.

Affine Invariant Model Based Object Recognition. Lamden, Schwatz & Wolfsen (1988) uses three non-collinear points to determine consistency and pose of a flat, rigid 3D images which has an arbitrary position in space and is essentially an affine approximation to the perspective viewing transformation. The transform is invariant under translation, rotation, scale and skew. This method employs a two step process which is essentially the same as Geometric Hashing. It uses 3 non-collinear model points to create a hash table with (m-3) model points. This gives a unique transformation between the model and the scene. If the viewing angle for both model and image is constant, then a similarity transformation can be applied where the transform is invariant under translation, rotation and scale. In this situation, a two point basis can be used which significantly reduces the complexity of the algorithm.

Recognition is by a voting process whereby for each non-collinear triplet of points in the image chosen as a basis, all other features are used to compute a coordinate in the hash table. Voting takes place for each (model, basis-triplet) and if the tally is above a threshold, the triplet is assumed to be in the image. A verification process confirms the existence of the model in the image. This process is called as the Geometric Hashing process.

2.3.4 Comparison

In the relational graph method, the model contains local and relational data. Matching relies on the presence of certain boundary features as well as features and properties of the interrelation between them. Recognition requires that a sufficient set of key local features are visible and in the correct relative position. The matching procedure depends on graph searching techniques that are computationally expensive and slow.

The structural feature method needs a training method and thus is computationally expensive but is executed off line. Local and extended boundaries are used to represent smoothed, intermediate symbolic descriptions. Methods employing image enhancement, smoothing and best line fit techniques give systems that are less sensitive to noise and have greater flexibility. Boundary features are invariant under translation and rotation. Partial occlusion can be handled but does require a significant proportion of the object to be unobscured.

By employing two phase algorithms and Geometric Hashing (Footprints and Affine Invariant transforms), the situation is such that, unlike their predecessors which could only recognise the existence of one model at a time, simultaneous recognition of all models in the data base as well as multiple objects in then image can be realised. This is a significant improvement over the earlier methods described which could only recognise one object at a time. It can also be seen as a merging of the two basic methods where a limited feature set, and a description of feature relationships can be employed in the process of recognition of a variety of models contained in the database in a 2D and 3D relationship. The system can easily handle noise, spurious data and occlusion while relating to a relatively small feature set.

2.4 Controlling the Search Explosion

2.4.1 Geometric Constraints

The presence of noise and spurious data in the image means some measures need to be used to control the search.. The primary aim is to reduce the number of points to be searched. Various methods employing constraints have been employed to limit the search space and consequently reduce the search time.

The objectives of employing constraints are to:

- Reduce the amount of search required,
- Minimise run time by employing an efficient algorithm,
- Characterise local shape of the object,
- Directly relate to the degree of error in data, and
- Check for global consistency.

The following constraints need to be considered:

- Unary constraint $C(i,p)=true$ iff the pairing of the i^{th} image feature to the p^{th} model feature is locally consistent. The pairing of model feature and image feature must be locally consistent. These constraints include length constraints and can also allow for error in the measurement of features.
- Binary Constraint $C(i,j,p,q)$=true if the pairing of the i^{th} image feature to the p^{th} model feature and the pairing of the j^{th} image feature to the q^{th} model feature are mutually locally consistent.
- Angle constraint is where the angle between image normals and model normals are within some error bound.
- Distance constraint relates to vectors having one component (head) on one image edge and the other component (tail) on another edge.
- Component constraints relate to the separation of two edge fragments. If the distance is not within given bounds, then the feature association can be discarded thus reducing the search.

2.4.2 Focal or privilege feature

This method attempts to deduce which features or clusters of features are most salient in recognising objects. The algorithm start by looking at a focal feature thus reducing the search space. Not all data is treated equally in terms of distinguishing the object and determining the pose. Information about model shape is used to remove inconsistencies.

The *Local focal feature method* assigns levels of importance to different features and uses partial pose information to guide the search for additional matching features.

The algorithm finds a feature in the image that identifies the model and allows the prediction of nearby features. Once matches for these nearby features have been found, the biggest cluster of image features matching object features is determined. This reduces the search to a limited set of features.

The basic method is:

- Select the focal feature.
- Find possible matching image features (unary constraints).
- For each match, find nearby features.
- Collect a list of possible matching features.
- Construct a graph of possible matches.
- Find maximal association in the graph to identify the best possible pairing.
- A rigid transformation is computed which is then verified.

2.4.3 Heuristic termination - measure of goodness of match with threshold

An attempt to exhaustively search the entire solution space can be a waste of time. Heuristics can be applied based upon *a priori* knowledge that can significantly reduce the search time. A measure of the "goodness of fit" can be applied to consistent interpretations of the data and if that measure exceeds a given threshold, the interpretation can be accepted and the search terminated. Measures that can be used include the number of features found, the percentage of the model accounted for and the sum of lengths of the features found compared to the total length possible.

2.5 Conclusion

Three methods of model based object recognition have been explored. The first method termed global matching uses the global features of the object as a means of recognition which has been shown to be inadequate. The second uses relational graphs which can prove to be computationally expensive, especially without the use of constraints. The third uses structural features in a select and hypothesise manner which eventually leads to the method of Geometric Hashing. Various methods have been employed to restrict the search space in an attempt to make the recognition process fast, efficient and able to recognise objects in images where a similarity transform is applied to create the data base and collect evidence for the existence of a model in an image.

In chapter 4, Geometric Hashing is discussed in more detail. This method logically follows from the structural feature method discussed above.

3. Hashing and Parallel Systems

3.1 Introduction

This chapter is divided into three parts which describe the general concepts of hashing and parallel processing related to this thesis. The first section describes the concepts of hashing and provides the background to the basis of the techniques used for model data storage. The second section discusses the concepts of parallel algorithms, and provides the reason for the choice of a SIMD architecture in the implementation of the parallel version of the Geometric Hashing algorithm. The third section give an overview of the DECmpp 12000 SIMD computer on which this work is implemented.

3.2 General Hashing Method

The primary objective of Hashing (Maurer & Lewis, 1975; Sedgewick, 1988; Date, 1990) is to search a file of n records independent of n. If n is large, the Hashing method is faster than a Linear Search method, which starts at the beginning of the data file and linearly searches until the required data is found, as opposed to seeking to the data directly. This is achieved in hashing by creating a different file that uses some aspect of the record to create an index. When the record is to be retrieved, the index value is generated from the request and used to return the record required without a linear search of the data base.

The method is used when n records R_1, $R_2,\ldots,R_n$ each of which consists of data items R_{i1}, $R_{i2},\cdots,R_{ij}$ for some j, and for some record R_i for which R_{i1} is equal to some predetermined value where R_{i1} is called a *key*.

The major problem that arises is when two keys produce the same address. The records involved are known as *synonyms* and this produces a *collision*. One method of dealing with this is to allow an address to store more than one record. The space for this is known as a *bucket* or *bin* and can be created using linked lists or arrays. When an attempt is made to put more than the expected number of records in the bucket, an *overflow* occurs. Another method is to place the record in the next available position, requiring the use of searching techniques and creating a limited Linear Search.

3.2.1 Hashing Functions

A hashing function $H(k)$ is used to create the key. The following methods are described.

- **Division:** A predetermined key k is divided by some value n. The remainder r is some integer value in the range $0 \leq r < n$. This can be used as an index to the data base. If T is the starting address of the hash table and K is the arbitrary key, then the hashing function is $h(K)=T+\mathrm{MOD}(K,n)$.

- **Random:** Key K is a seed to a pseudo-random number generator. As pseudo-random number generators always generate the same sequence of numbers for a given key, the first random number is considered the address and the remaining sequence is used for collision handling and thus deciding as to where to place the overflow.

- **Midsquare:** Key K is multiplied by a constant and it is assumed that the bit values are unique for the number generated. A set of ten (10) bits are taken from the middle of the value and assume to be random. This number is used as the index.

- **Radix:** Key K is considered as a string of octal digits. This string is then considered as another base (e.g. base 11). This is then converted to a base 10 decimal and used as an address. A set of bits are taken as in the midsquare method above, and then used as the index.

- **Algebraic Coding:** Key K is n bits long. Consider a polynomial of degree $(n-1)$ where each of the coefficients are one of the bits of K (either 0 or 1). The polynomial produced is divided by a k^{th} degree polynomial where division is by modulo 2. The remainder is a $(k-1)^{th}$ degree polynomial and can be considered as a string of k bits, which is used as the index.

- **Folding:** Key *K*, an *n* bit key, is divided into several *k* bit fields. These are added together or exclusive ORed to produce the index. This can be viewed as taking the key as a string of digits on a piece of paper that is folded several times to overlap the numbers. These overlapped numbers are used to create the index.

- **Digit Analysis:** This method usually requires a decimal number key and a decimal number index. Each digit of the key is considered and a value representing the amount of skew calculated. The index is calculated from this by *crossing out* all but those digits from the key.

The most significant problem with hashing is when bucket overflow occurs. This means that more than one instance of an index is created for a subset of records requiring some action to be taken.

3.2.2 Collision Handling and Bucket Overflow

A collision problem arises when two keys happen to have the same home address. Overflow arises when the assumption that no more than *n* keys will have the same address but more than *n* keys address it. In this case, each bucket will contain *n* records. The major problem arises when the records have to be retrieved. If the required record is not at the address given by the key, searching of the table for the record must be undertaken. Several methods for handling this are described:

- **Linear method**: Assume that the home address or index position of key *K* is *a*. Now assume another key *K′* has the address of *a.* As this address is already used, the record is now stored at the next available address *a*+*n* where *n* is the distance to the next available location. Wrapping allows for an address at the end of the table. In the event that a bucket at the address is full, the bucket at the next available address is used.

- **Overflow-area method:** The assumption made is that there is a very low probability of overflow. A general area is then kept to store all overflow records.

- **List method:** A linked list is maintained in main memory which allows the dynamic creation of the bucket. The bucket can then be of any size. In the event of overflow, a pointer to the next bucket can be maintained.

- **Random Method:** If the random method for addressing is used, a pseudo-random number generator is used such that synonyms will produce the same set of addresses which are randomly distributed and so can be used to find the record thus preventing a need to search the hash table. Given that the same seed is used to generate the pseudo-random number, the same set of numbers will be produced and thus generate the same key sequence.

- **Quadratic Method:** Records with similar keys will appear in a cluster in the hash table. If the linear method of handling overflow is used, much of this cluster must be searched. The quadratic method provides a means of calculating the subsequent address $h_i(K)$ as $h(K)+m_i+n_i$ for some fixed values of m and n thus putting the subsequent synonyms outside the cluster.

For a normal database search, the contents of the key which forms part of the required record is known before hand. In the case of Geometric Hashing, a number is calculated to create the key. No part of the stored record contains this data. Thus, the methods employed to create the hash table to allow for collision and overflow, as well as the search methods have to be a modified form of the above.

3.3 Parallel methods

A number of different methods of processing data using parallel systems are defined. The choice of which method to choose is predetermined by the data to be processed and the architecture and software available. (Alk, 1989; Gibson & Rytter, 1988) suggests the following definitions.

A Single Instruction Multiple Data (SIMD) machine consists of parallel data processors acting in unison and using one instruction at a time but on different data stored nominally on each processor. Instructions are executed serially but parallel instructions are executed on each active processor simultaneously. Figure 3-1 shows a data stream being processed by n identical processors all carrying out the same instruction on the variant data.

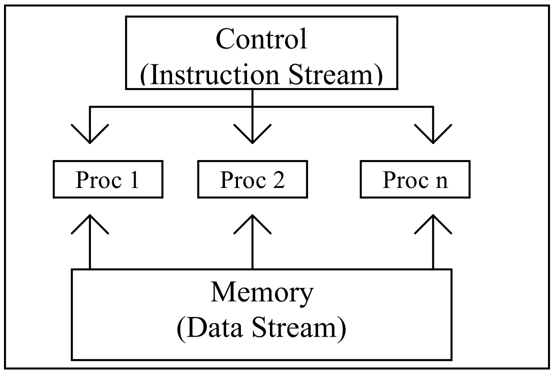


Figure 3-1 SIMD Architecture

In Multiple Instruction Single Data (MISD) machine, the same data item must be processed in many ways. Each processor has its own instruction set and it acts on the data item. The process embodies the concept of *pipelining*. Figure 3-2 depicts this architecture.

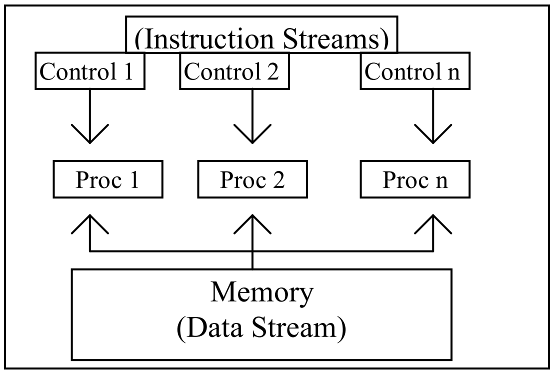


Figure 3-2 MISD Architecture

Multiple Data Multiple Instruction (MIMD) is where each processor has its own data set and instruction set. Each processor acts independently of each other and communication in some asynchronous user defined way. Figure 3-3 depicts this architecture.

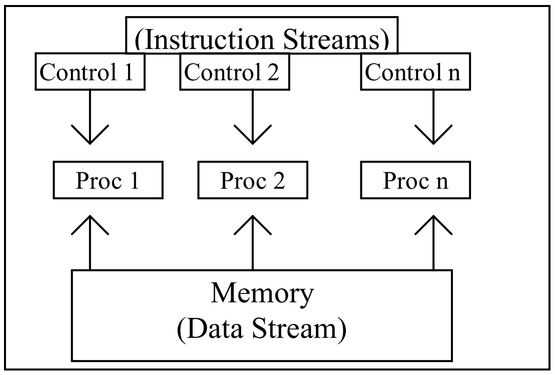


Figure 3-3 MIMD Architecture

3.3.1 Virtualisation

The number of processors on a parallel system can be considered as small or limited for most applications. However, the system can be considered as containing many more processors. The process of virtualisation allows a limited system to be considered as consisting of as many processors as are required. The major problems associated with this are: (1) data mapping, (2) system degradation in terms of speed of processing, (3) memory limitations and (4) algorithm considerations. Each of these factors are discussed below.

Data mapping embodies the process of mapping the data onto the processor array and has significant implications in terms of data access and speed of processing. Mapping of the problem onto the processor array can be considered as Hierarchical or Cut and Stack. Both of these mappings can be carried out in 1D or 2D. In this implementation, 1D mappings were used.

In the Hierarchical method, neighbouring data elements are stored on the same processor where possible. (Figure 3-4). This minimises Processor Element (PE) communication for processes that require interaction with nearest neighbours which will appear on the same PE. An example of this is pixel averaging where the central pixel is replaced by the average of itself and the surrounding eight neighbours.

In the **Cut and Stack** method, neighbouring data elements are stored on different processors. (Figure 3-5). Thus, virtual neighbours usually become actual neighbours where they are stored on neighbouring physical PEs. This maximises load balancing where the image processing is localised, thus distributing the processing as much as possible.

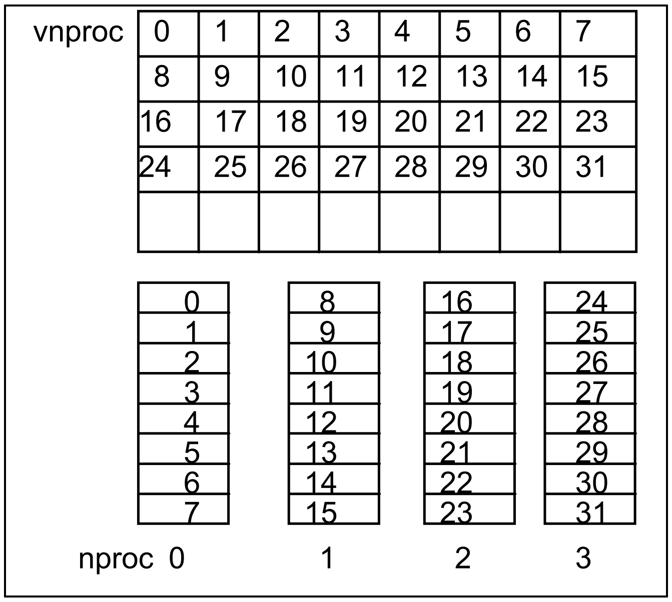


Figure 3-4 One Dimensional Hierarchical Distribution

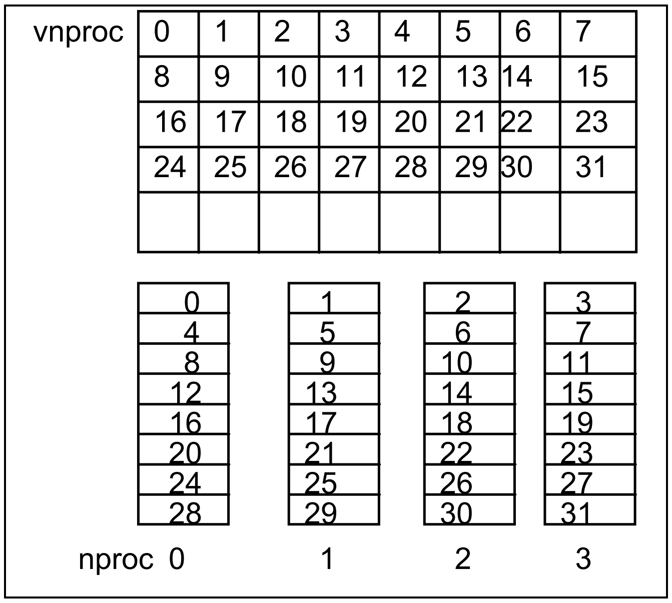


Figure 3-5 One Dimensional Cut and Stack Distribution

System degradation occurs with virtualisation because software is required to map the virtual processor to the actual processor. Each method of virtualisation requires a different algorithms to be implemented.

Memory limitations cause each processor to act as a number of virtual processors. All memory on that processor must be equally shared among all virtual processors. The virtual base data types are implemented as arrays with indices being virtual processor keys. With a greater degree of virtualisation, a smaller amount of memory for each virtual processor is implied.

Algorithmic considerations include the maintenance of global knowledge of the activities of virtual processors. When an attempt is made to access the same real processor by a number of virtual processors acting in parallel, contention occurs. The major problem arrises as a consequence of using multiprocess algorithms where the addressing of indexes on different processes is attempted. The system is forced to act in serial mode when this occurs. This will be dealt with in detail in Section 5.

3.4 DECmpp 12000

The DECmpp 12000 is a massively parallel computer. It consists of two major components, these being a front end graphics workstation that runs the Unix operating system and a Data Parallel Unit (DPU). At Curtin University, there is a 2k processor machine, although a 64k machine is possible. In our implementation, the 2k processor machine is virtualised to resemble a 64k processor machine, or any other sized machine dependent on the image size. The consequence of virtualisation is a proportional reduction in speed.

The front end is nominally an DEC Workstation that runs the DEC Unix Operating System (Ultrix) with Windowing capability and standard I/O. However, any Unix workstation, or the ability to access a workstation via a communications network is sufficient to control the system. For debugging and graphical display purposes, a Unix workstation is required.

The Data Parallel Unit (DPU) is a fine grained, single instruction multiple data (SIMD), massively parallel processor that does all of the parallel processing. It contains an Array Control Unit (ACU) and Processor Element (PE) Array.

The prime purpose of the ACU is to control the PEs. This is a register-based load/store 14 mips processor. It has its own registers, data memory and instruction memory. In the configuration used, it has thirty two 32-bit registers, 128K Bytes of data memory, and 1M Byte RAM (expandable to 4G Bytes of virtual memory) for storage of instructions. It is, however, slow compared to the front end and many of the singular instructions should be executed on the front end when speed demands it.

It has functions to perform the following:

1. Control the PE array and send data and instructions to each PE simultaneously.
2. Process singular data, which is data stored on the ACU and not processed by the PEs.

Part of the program implemented executes on the ACU. Although it may not be as fast as the front end, it is convenient in terms of processing.

The PE Array is a two dimensional array of load/store arithmetic processors with dedicated register space and RAM. Each PE has a 1.8 MIPS control processor, forty 32 bit registers and 16k bytes of RAM. The system used at Curtin University has a processor array with 64 columns and 32 rows thus making 2048 processing elements available.

Each non-overlapping square matrix of 16 processors is called a cluster which has one communication channel available for data transfer shared between the processing elements. This is an important consideration in router communications which is extensively used in this implementation of Geometric Hashing. Although the system can communicate with all clusters simultaneously, only one processor at a time within a cluster can be communicated with. Thus a process which requires access to all processors within a cluster effectively becomes serial.

The PE array allows the processing of data in parallel. Variables declared as parallel allow storage of data on each PE. Each PE receives the same instruction simultaneously from the ACU. If a PE is active, then the same instruction is performed on the appropriate data on that PE.

The ACU-PE bus allows communication between the ACU and PEs. Most communication is from the ACU to the PEs in broadcast mode. Instructions and data are sent from the ACU to all active PEs. This facility is used in our implementation of Geometric Hashing for tracking of global indices and for the storage of stable model point and line data, and current image line data.

If communication is from the PEs to the ACU, the data is logically reduced (Global ORed) to one value in the singular ACU variable. This is not used in this implementation of Geometric Hashing but is used in a method proposed using the Connection Machine (Rigoustos & Hummel, 1992).

There are two methods available to allow inter-processor communication. These are the X-NET and the Global Router.

1. The X-Net allows communication between PEs that lie in a straight line from the original PE along a predefined direction (N, S, E, W, NE, NW, SE, SW). As the PE access in Geometric Hashing is non-deterministic, this communications

method was not employed. However, an implementation that accesses this method would be significantly faster than by using the global router.

2. The global router allows simultaneous communication between any pair of PEs. This is the general purpose inter-processor communication method which is non directional. Due to the non deterministic nature of hashing and global image pixel addressing, global router communication is the method employed in this implementation for hash table access and image addressing.

With the DECmpp 12000, there are two languages available: MPL, the DECmpp Programming Language which is a superset of C, and DECmpp Parallel FORTRAN.

For the implementation of Geometric Hashing, MPL was chosen as this gives the greatest flexibility and greatest efficiency. MPL is a based on K&R C (DECmpp 12000, 1992) with added features to support parallel programming.

- Plural keyword

  Variables declared with this are allocated identically on each PE in the PE array. Those variables declared without the plural keyword are allocated on the ACU.

- Plural Expressions

  All arithmetic and addressing operations defined in K&R C (DECmpp 12000, 1992) are supported for plural data types.

- SIMD control statements

  SIMD control statements also control the active set of processors. The active set is that set of PEs which are enabled at any given time and participate in the processing. The active set is determined by conditional statements, and can be no larger than the total number of PEs present.

- SIMD Communications

  X-NET and Global Router commands are defined to make explicit data transfer between processors in the PE array.

When declaring parallel variables, the plural keyword is a type qualifier which tells the DECmpp system that the variable data is to be stored on each PE and not allocated to the ACU where singular variable data is stored. The plural keyword can be specified with any basic type such as char, short, int, long and float, as well as user defined structures.

The address space of singular and plural pointers is independent. The method of declaration determines the space pointed to. The following declarations are valid.

- Singular pointers to singular data.
- Singular pointers to plural data.
- Plural pointers to singular data.
- Plural pointers to plural data.

A singular pointer is a pointer variable allocated on the ACU and a plural pointer is a pointer variable allocated on all PEs. When a singular pointer to plural data or plural pointer to plural data is dereferenced, a plural value is returned.

Plural pointers to plural data is only possible on each processor in isolation. That is, a PE can only point to its own memory. When accessing data on different PEs, the XNET or global router must be used. This is an important factor when accessing hash tables and image components and explains why array access is necessary in most cases.

MPL supports plural arrays with the array space being replicated across all PEs. MPL allows singular arrays to be indexed by plural expressions. Reverse assignment is undefined when multiple assignments to the one array are made because potentially all processors containing different data can attempt to assign data to a singular variable.

In a plural array, all addressing by a variable is by the value held on that PE. This is important and a problem when the router is used to access the contents of an array stored on another PE. It also affects the virtualisation process. The consequences will be discussed in chapter 5.

3.5 Conclusion

This chapter has provided the background required to allow an understanding of the implementation of Geometric Hashing, both on a serial architecture computer as well as a parallel SIMD system.

The general techniques of hashing have been explained. A description of the common methods of implementing hashing and determining the key to the hash table were provided. The common methods of handling the problem of collision handling were detailed.

SIMD, MISD and MIMD methods of parallel processing were described and a reason for choosing SIMD provided. The DECmpp 120000 system was described, along with the methods of Cut and Stack and Hierarchical data distribution techniques.

The following chapter discusses the work carried out by other authors both in a serial and parallel environment.

4. Geometric Hashing

A number of different algorithms have been proposed to solve the problem of 2D and 3D object recognition. A recently proposed method that is suitable for parallel implementation is Geometric Hashing (Lamden & Wolfsen, 1988; Lamden, Schwartz & Wolfsen, 1990). This algorithm uses a two phase technique in which the first or learning phase processes model features to create the hash table, and the second or recognition phase processes image features and uses the hash table to determine if the model matches the image. The similarity is indicated by the number of votes for a particular model and geometric transformation. This chapter discusses the concepts of Hashing and its application to Geometric Hashing, the concepts of parallel processing and the application of SIMD parallel processing to model based object recognition using parallel algorithms. The DECmpp 12000 massively parallel SIMD computer is also described in detail.

4.1 The Basic Concept of Geometric Hashing

4.1.1 Problem Definition

Geometric Hashing is a method of model based object recognition. For the purpose of this thesis, the model or models may be contained in a scene which may contain instances of one or more objects that may be occluded or be a subset of a larger model that may have undergone an affine transformation (translation, rotation, scale and skew). The method is a two stage process. The first stage involves the creation of a redundant database of geometric data from a set of model features which, in most cases, are points that represent the ends of lines. This is achieved by defining an orthogonal coordinate frame based on an ordered set of points and then representing all other coordinates by their coordinates in the defined frame. The method requires the creation of the database off-line. The second stage is that of recognition, where an image is processed and, using the data stored, an hypothesis of the model and basis is determined. This can be fast requiring only a small number of hash table lookups. The final process, verification, is an addendum to the Geometric Hashing process and is used to confirm or deny the existence of a model by the hypothesis.

4.1.2 Learning (Preprocessing)

A set of interest features, which can be used to create an affine invariant transform to a set basis, are extracted from the model. A minimum number of these make up a feature set. All sets are individually selected in turn as a basis and processed to produce an expression which can then be applied to all other interest features. For each set, an expression allows a transformation of the other features which is affine invariant to produce a key that relates to the model and the selected feature set. The key is used to create an index to the hash table and points to a bin which holds {model, feature} set data.

A key is a value determined by a transformation of one of the remaining interest features. The key points to a position in the database that holds data relating to potential (model basis) pairs. In the creation of this key, the issues of collision, overflow, quantisation and distribution of the data within the hash table needs to be addressed. The index created points to a bin which stores zero or more (model basis) data. For a model with *n* features, *n*-2 bins store data relating to the particular model and basis pair as a redundant data base.

In Figure 4-1 below, the points *a* and *b* are selected as a basis. They undergo translation, rotation and scale to align with the frame (0,0) and (1,0). The remaining features undergo the same translation, rotation and scale and the coordinates of these become the key used to create the index to the hash table. Once all remaining points for this basis have been processed, another basis is selected and the process is repeated. After all combinations of feature pairs to form a basis have been selected, a new model is processed. This continues until all models have been processed.

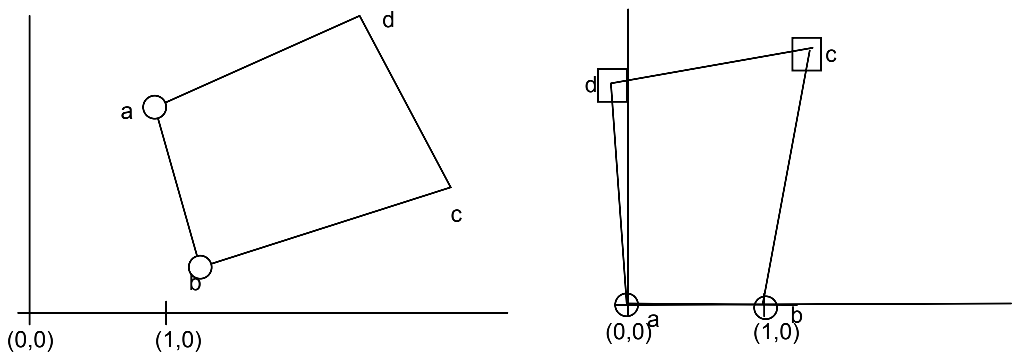


Figure 4-1 Basis selection and key creation

4.1.3 Recognition

The image is processed such that it is reduced to a collection of features. As in the learning phase, the features required for a basis are selected and an expression is created which is then applied to all other features of the image in turn. The key and subsequent index created is used to access the data base of (model basis) data. For all data in the selected bin, appropriate accumulators account for each (model feature) set encountered.

After all image features have been processed for the selected basis, the final tally list is searched and all (model basis) sets that lie above a threshold are considered as candidates for determining the model and model transformation used in the verification stage.

This process can be repeated for all permutations of appropriate image feature pairs to determine the existence of one or more (model basis) sets.

4.1.4 Verification

This stage lies outside the general description of the algorithm, however it is necessary for recognition. The general application of Geometric Hashing to simple model recognition in ideal image space is sufficient for recognition, however when there is noise and occlusion, the algorithm produces evidence of the existence of the model in the image but is not sufficient to guarantee recognition. Comparison needs to be made between the proposed model and its orientation, with the image to confirm or deny the existence of the model or models present.

4.1.5 Application of the general paradigm

In many of the papers that specifically deal with Geometric Hashing, (Bourdon, Medioni, 1990; Hong & Wolfsen, 1988; Khokhar, Prasanna & Kim, 1993; Lamden & Wolfsen, 1988; Lamden, Schwartz & Wolfsen, 1990; Lamden, Schwartz & Wolfsen, 1988; Leishman, Vankatesh & West, 1993; Prasanna & Wang, 1993; Rigoustos & Hummel, 1992; Rigoutsos & Hummel, 1991; Wolfsen, 1990), the features extracted were points which can represent such features as the end of a lines, centre of a circle, centre of curvature etc. Thus, the following descriptions of the algorithm relates to 2D models and images where points have been extracted and selected as the feature set. The following cases are examined:

1. The model represented in the image is a simple translation. (Lamden & Wolfsen, 1988) This requires the selection of one point of the model, translating it to coordinate (0,0) and then applying the same translation to the remaining (*n*-1) points to create a set of numbers that become the key for hash table index creation. The (model basis) data pointed to by the index is stored in the bin. In recognition, one point in turn is chosen from the image as a basis and the translation is calculated to align this to the coordinate (0,0). All other points of the image are chosen in succession and the translation applied to produce a set of indices. The bins are accessed using the indices and a tally made of each (model basis) found in the bins for all (*n*-1) image points. A threshold is applied to select the most likely (model basis).

2. The model represented in the image undergoes translation and rotation (Lamden & Wolfsen, 1988). To enable the effects of rotation to be accounted for, a two point basis is selected. This requires the selection of a pair of points of the model, applying an affine transformation to map the points to ((0,0), (x,0)) and then applying the same transformation to the remaining *(n-2)* points to create a set of numbers that become the keys to the hash table. The (model basis) data indexed by each key is stored in a bin of the hash table. In recognition, two points are chosen from the image as a basis and the transformation is calculated to align these to the coordinate frame of ((0,0), (1,0)). All other points of the image are chosen in succession and the transformation applied to produce a set of keys. The bins are accessed using the keys and a tally made of each (model basis) found in the bins for all *(n-2)* image points. A threshold is applied to select the most likely (model basis) pair.

3. The model undergoes a similarity transformation which involves translation, rotation and scale but not skew. (Lamden & Wolfsen, 1988) This requires a two point coordinate frame. The algorithm is identical to than in (2) above, except that scaling also needs to be considered in calculation of the transformation.

4. The model undergoes a transformation that is a combination of translation, rotation and scale and skew, and thus a three point coordinate frame must be used. (Lamden & Wolfsen, 1988) This requires the selection of triplets of points of the model, applying an affine transformation to achieve a coordinate frame of ((0,0), (1,0), (0,1)) and then applying the same transformation to the remaining (*n*-3) points to create a set of numbers that become the key for hash table index creation. All (model basis) data pointed to by the key are stored in the appropriate bin in the hash table.

For 3D models where it can be assumed that there is a parallel projection approximation to the perspective transformation, a 2D affine transformation can be applied. Without skew, the basis for (3) above can be used. The 3D model can be treated as a finite set of 2D models. Where the model undergoes a projective transformation, a four point basis is required with a consequent increase in the complexity and storage requirements (Lamden & Wolfsen, 1988).

4.1.6 Interest Features other than points

As stated earlier, points are chosen as the interest features. However, lines, planes and any other feature that can be proved to be affine invariant, can be used.

For points, the number of interest features required for a basis depends on the transformation. For the case of a transformation involving translation, rotation, scale and skew, a basis of three points is required. The normalised basis becomes ((0,0), (1,0), (0,1)). The remaining *(n-3)* points undergo transformation to create the hash table key. For a transformation involving translation, rotation and scale, a two point basis is required. The normalised basis becomes ((0,0) and (1,0)). The remaining *(n-2)* points undergo transformation to create the hash table key. This is the case where the viewing angle is the same for the model and image of the scene containing a model. Where the transformation is only translation, then a one point basis is required.

The major difficulty in using points is that the system is subject to noise which, in the recognition phase, leads to reduced discrimination and a significant increase in processing time.

Lines present more stable features than points and are not subject to the same level of noise. The concept applied to points can also be applied to lines since lines can be viewed as points in the dual space (Rigoutsos & Hummel, 1991). Three lines, of which there is no parallel pair, can form a basis of the affine space where each line has a unique coordinate in the basis. If the end points of the line segments are known, then a line segment and an additional point can be used. This reduces the complexity significantly.

The shape of planar rigid bodies can be described by their boundary curves. Thus object recognition can be performed by matching these curves. Curves can be represented by vertices of a polygonal approximation of the curve. The above methods using points or lines can be then used by extracting specific features for the curve such as concavity points and concavity entrance lines. This relies on there being a good correspondence between the polygonal description of the model and the image contours for the same object. This can be difficult to achieve.

4.1.7 Sensitivity

The process of Geometric Hashing relies on there being a high degree of redundancy. Measurement error can cause a miss in recognition and thus there is a need to have a number of affine coordinates (Hash Keys) each of which will enable recognition.

Analysis of the error bounds can be used to define a range of hash bins for the histogramming stage. The major problems occur when points are almost collinear or very close together.

Error analysis has been performed (Grimson & Huttenlocher, 1990) that demonstrates that a redundancy of a hash key increases with an increase in measurement error. It was shown that the fraction of the range of values of the hash key increases as the size of the basis vector increases, relative to the size of the overall object. Thus there is a corresponding increase in the radius of error. For the model, error due to quantisation of a continuous data set leads to error in the creation of the key to the hash table. During recognition, access to the hash table using a calculated key requires a search over the appropriate volume of the hash table to allow for error in measurement of the image points and subsequent error in calculation of the key. The average fraction of the dimension of the hash table covered by the range of feasible values was presented and this shows an increase in error as the inter-basis pair distance decreases as well as an increase in error as the measurement error increases. In our implementation, based upon the quantisation used, the measurement error is assumed to be small and the error volume maintained such that the hash key error is adjustable, and set initially at ±1 in both the *x* and *y* directions giving a volume with 9 locations.

The second factor to be considered is that of noise. Analysis that determines the probability that a set of random points will produce a high enough count in the histogramming phase to indicate the existence of a model and specific basis has been carried out (Grimson & Huttenlocher, 1990). It was clearly demonstrated that, given a small number of features (in this case, less than fifty), the error is low only when there is a small number of sensor features or a small measurement error. To have no error at all requires the existence of no spurious data.

In the parallel implementation discussed later, there are no more than eighteen model features and some of the models have as few as eight features. Using a similarity transform (translation, rotation and scale), the maximum number of features, without occlusion is six. To reduce the likelihood of spurious model-basis counts becoming excessive, a small amount of feature position measurement errors can be tolerated.

The method works well when little measurement noise and limited spurious data are present. When noise levels are moderate, the method degrades and the probability of a false positive or incorrect recognition becomes significant.

4.1.8 Error Analysis

The analysis of error revolves around two aspects: (1) the level of quantisation used, and (2) positional accuracy. The level of quantisation will determine the number of bins available for hash table storage as well as the number of records stored in each bin. Poor positional accuracy increases the number of bins for a given coordinate that needs to be accessed resulting in an increase in the number of candidates participating in the voting procedure.

The effect of error in positional estimation and inter-point distance was studied (Lamden, Wolfsen, 1991). The redundancy factor, or the fraction of the hash table, is calculated for different interpoint distances and error levels. They conclude that the redundancy factor decreases as the interpoint distance increases, or as the positional error value diminishes. For small errors of 1.5 pixels or interpoint distances of 200 pixels, a volume of 9 bins is suggested. For errors of 5 pixels or interpoint distances of 25 pixels, a volume of 30 bins is suggested.

Errors introduced by the choice of quantisation of the bins and the measurement error was analysis by Gavrila & Groen (1992). They determined that large error arises when the length of the basis vectors is small with respect to the error ε and if $|x|$ in the linear system $Ax = c$ is big. They also suggest that a threshold be applied in the recognition process that would effectively skip those bases.

The accuracy in measuring the position of each point also has an effect on the degree of error. It was also demonstrated (Lamden & Wolfsen, 1991) that, as the positional error increases, there is a greater probability of a random basis producing a high histogram count.

In experiments (Lamden & Wolfsen, 1991) and for the case of a similarity transform, with the number of models M=1, the number of features in the model m=12 and the number of features in the image n=20, and an error value of ε=2.5, one basis returned a count of 6 where the possible value is m-2 or 10 in this case. For the case of the affine transformation with the same model and image parameters, and a lower error value of ε=1.5, 1438 basis returned a count greater than 9, the potential threshold. It was noted that especially for an affine transformation, recognition by Geometric Hashing alone was insufficient and that a method of verification was essential to determine the true and false matches.

4.1.9 Complexity Analysis

The method uses a two phase system of learning and recognition. As learning takes place off-line, complexity is not an issue. The recognition component needs to be considered for complexity analysis only. The complexity depends on the number of scene points and the selection of the basis components and is independent of the number of potential models in the image or the number of models represented in the data base. For a scene consisting of *n* points, if a translation requiring the selection of a one point basis is used, the complexity is O(n^2) and for an affine transformation using a three point basis, the complexity is O(n^4). For a similarity transformation with a two point basis, the algorithm would be O(n^3).

4.1.10 Experimental Results

Point data from 2D images of pliers and wrenches was studied (Lamden & Wolfsen, 1988). A 2 pixel error in position was assumed. The typical number of model points in the scene was 16 which, using a 3 point basis, should have resulted in a maximum of 13 votes. They were able to correctly recognise the models in four instances where basis triplets accumulated 13 votes. Good recognition also took place at the 12 vote level, but degraded to the point where only 19% of the triplets were correct at the 6 vote level.

CAD models of cars and cranes were studied (Lamden, Schwartz & Wolfsen, 1990). The features used in learning and recognition were the end-point of line segments. They reported relatively high numbers of votes for the correct solutions and that initial matches were quite accurate even when segment end-points were quite noisy.

Gavrila & Groen (1992) used models and scenes consisting of random dots. The number of points in the models were 15 and the scenes had 25 points. The number of models in the database were *N*=50, *N*=100 and *N*=250. Their results attempted to demonstrate the effect of hash bin quantisation and distribution, as well as that of measurement error. They concluded that the larger bin size increased the number of random solutions and consequently, the expense of verification processing. There was the need to keep the bin size as small as possible but this required greater memory requirements (more bins) and longer processing time (searching). To avoid random errors effecting the count, the threshold needed to be set at greater than 60%, but this affected the amount of occlusion that could be tolerated.

Gavrila & Groen (1992) also used 3D images using the letters M, A, R, L, E, N as wire frame polyhedral objects. The database consisted of 2D projections of these images tessellated at discrete angles. Interest points were vertices and the bases were vertices connected by edges. The algorithm was able to distinguish models in different scenes which included occlusion and implicitly provided the correct solution before verification. Correct point correspondence between model and scene was achieved and the system was considered to be robust.

4.2 Parallel Implementation

4.2.1 Introduction

There have been a limited number of parallel implementations of Geometric Hashing on parallel computers. A Connection Machine (Prasanna & Wang, 1993; Rigoustos & Hummel, 1992; Rigoustos & Hummel, 1993) and a MASPAR (also known as a DECmpp 12000) (Khokhar, Prasanna & Kim, 1993) have been used.

In the two implementations of Geometric Hashing on a Connection Machine, the problem was defined as recognising dot patterns embedded in a scene where each dot can represent a feature location extracted from an image. Two methods of implementation were described (Rigoustos & Hummel, 1992) in some detail whereas three algorithms with similar scene and hash table distribution with and without sorting to determine the number of votes were described (Prasanna & Wang, 1993). The methods of Rigoutsos & Hummel (1992) are treated in more detail below.

In the first instance, a Parallel Hypercube configuration is used where the information is routed through a series of maps. For a detailed description of the configuration refer to Rigoustos & Hummel (1992). The method is parallel over the hash-table entries and scene points but serial over the bases. That is, the bases are selected serially at the front end and passed to the parallel array for processing. They use M models with an associated set S containing the coordinate pairs of the mth model's n points.

In the preprocessing stage, each model's data is serially treated using a *Building Block* algorithm to create the hash bin number. A two phase algorithm is used to efficiently communicate model and basis-point information to the hash bin. In the first pass, the number of entries that will occur in each hash bin is counted. This allows the organisation of the processor pool into a one dimensional array. Each hash bin will occupy a contiguous block of processors with the block length being equal to the expected number of entries. A map gives an index to the head processor for each block. In the second pass, the hash table entry is calculated and sent to the appropriate processor with the map entry point being incremented to maintain appropriate processor location. Collision is handled by a SIMD version of a parallel fetch-and-add instruction.

The hash table is contained in two data structures. The first is the map where one processor is used for each hash bin. These contain pointers to the processor containing the block entries in the second structure. The second structure consists of the hash bin entries of the form (m,i,j) where m is model number, i is the first point of the basis and j is the second point of the basis.

In the recognition phase, four sets of virtual processors are used. V_1 stores the set of S interest points on S processors. V_2 contains the pointers to the heads of the block of entries and V_3 is the 1D array of concatenated lists of hash entries. V_4 is used for histogramming. A basis pair is selected at the front end and broadcast to the S processors. The hash bin index is then calculated on each of the S processors and a message is sent to the appropriate processor in V_2 using additive writes and general routing. Multiple votes destined for the same processor combine in the routers. Next, each processor in V_2 that has received one or more votes relays the number of votes to each of the subset of processors in V_3. Additive writes are used in the final phase of histogramming. Each processor in V_4 is associated with a triplet (m,i,j). The processors of V_3 vote for their entries by sending additive writes to the appropriate histogram bin. These are incremented by the value of the votes received in the third phase. A global-max operation of the votes is used to determine the winning (m,i,j) combinations. These combinations are sent to the front-end where verification takes place.

Time complexity for the recognition phase (excluding the histogramming phase), per broadcast basis pair, is $O(\log(Mn^3)$. The histogramming dominates the time complexity and is dependant on the method used.

The second method implements Hash Location Broadcast. This uses large memory resources and Front End Broadcast. The hash table structure is considered as a collection of records of the form (m,i,j,k,x,y) where (x,y) is the hash location that point k maps to under basis (i,j) in model m. It is stored as a multi-dimensional table indexed by (m,i,j,k) where i,j,k are integer values between 1 and n.

In the preprocessing phase, the model points are loaded onto appropriate processors and M simultaneous triple products are computed to create the four dimensional array. The hash locations are then computed for all appropriate triples and stored as (x,y) coordinates in the array.

In the recognition phase, the interest points are held in V_1 while the hash table is held in V_2. The front end broadcasts sequentially a basis pair in the scene and each processor in V_1 (except those in the basis pair) calculates a hash index (u,v). Each processor in V_2 compares (u,v) to its (x,y) and where a close match is obtained, a tally is made for the (m,i,j,k). This continues for all S-2 points in the image. The third stage requires a segmented, parallel, tree-based sum operation to add the votes over k among the locations (m,i,j,k) to give the total number of votes for model m with basis (i,j). Thresholding is then used to determine the model-basis combinations that are to be used in the final verification stage.

The Hash Table is structured and distributed as follows. Each virtual processor is assigned an index $(m,i,j)$and local memory stores the n entries associated with k=1,2..n. This reduces the virtual processor ratio. Other efficiencies are achieved by accounting for the symmetric properties of the hash index.

The hash data is broadcast to all processors which need to search the entire hash space. The image is distributed such that S points are on S processors and the V_4 processor set is used as a set of bins for histogramming purposes. The front end selects the basis pair and S-2 processors work with the data.

The implementation of Geometric Hashing for the Maspar is described by Khokhar, Prasanna & Kim (1993). In this implementation, results for a 1K, 2K, 4K and 8K processor machine have been quoted although it appears that a 1K/2K processor machine was used. The algorithms differ in the partitioning and mapping of the hash table onto the processor array.

The processors are configured such that it is assumed that the number of processors available is P where $1 \leq P \leq S$ and S is the number of feature points in a scene. The mapping of the virtual processors onto the actual processors is not discussed by the authors.

The Hash Table is structured and distributed in three different ways:

1. Each processor is assigned $\frac{Mn^3}{P}$ distinct hash table locations.

2. Each sub-array of processors is assigned a complete copy of the hash table. Each processor in a sub-array of size s^2, where $1 \leq s \leq \sqrt{P}$ has $\frac{Mn^3}{s^2}$ distinct entries of the hash table.

3. Concurrent processing of multiple probes is carried out. The array is divided into disjoint sets of *S* processors.

The methods and processes involved in image distribution, bin distribution, basis selection and virtualisation were not discussed by the authors. A basis is selected by the front end or DPU and broadcast to each active virtual PE although it is not specifically stated.

4.3 Conclusion

The method of Geometric Hashing, and in, particular, the current implementations of the technique using singular and parallel methods have been introduced. Geometric Hashing is a two process algorithm, where learning takes place off line and recognition can be carried out in real time. A verification stage is usually also required. The method, as described by Lamden & Wolfsen (1988), can be applied to the recognition of 2D and 3D objects using a variety of features.

The parallel implementation of the algorithm principally revolves around (1) the mapping of the data onto the processor array, (2) the techniques involved in selecting features to form a basis for recognition and (3) the calculation of the key to the hash table.

In the following two chapters, the singular and parallel implementation of Geometric Hashing is described.

5. Implementation of Geometric Hashing (non parallel)

This chapter describes a non parallel implementation of Geometric Hashing which was created as the precursor to the parallel implementation. The issues that follow relate to this implementation of Geometric Hashing and are fundamental to the issues in the parallel implementation of this concept.

In this implementation, one model is used in the creation of the hash table and only one representation of this model is used in the image at recognition time as this is an initial examination of the algorithm. The full implementation using all model instances and a verification stage is presented when the algorithm is parallelised.. The model (Figure 5-19) is one view of the widgit and contains 12 points or vertices. The points of the model are defined as the ends of line segments and are the features used in the creation of the hash table. This data was obtained from the CAD description of view two of the 3D widgit used in the parallel implementation of the algorithm (Figure 6-1).

5.1 Hash table generation

The endpoints of lines of the model are chosen as the features to be used in the hashing process. All points are chosen in turn and all combinations of pairs of points are chosen as a basis. For each basis, a transformation is applied that translates, rotates and scales the points to fit the coordinate frame [(0,0), (1,0)]. The same transformation is applied to all other points of the model in turn to create a coordinate (*x,y*) or key that is used to generate the hash index.

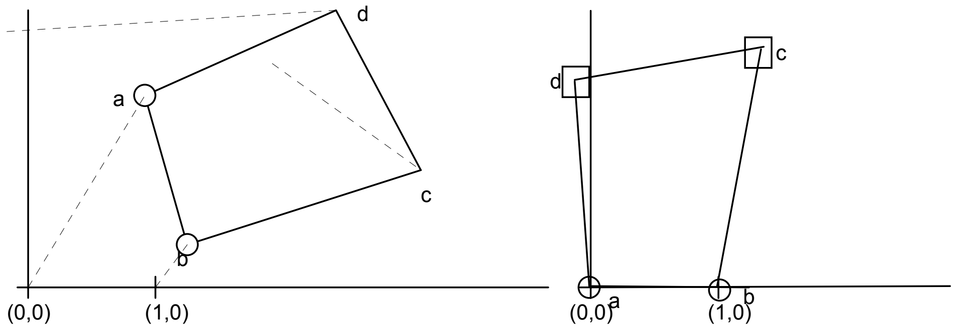


Figure 5-1 Basis and feature transformation

Figure 5-1 depicts a model where the chosen basis is represented by points *a* and *b*. These are transformed to (0,0) and (1,0) respectively. Each of the remaining features (points *c* and *d*) undergo the same transformation in turn to create indexes to the hash table where model and basis data is stored.

The algorithm shown in Figure 5-2 describes this process. Variables *p*1 and *p*2 make up the basis and represent basepoint 1 and basepoint 2 in the model. Variable *p*3 represent the third point. The angle of the line between *p*1 and *p*2 relative to the x axis, followed by the scaling factor required to transform the line to unit length when rotated to the coordinate frame [(0,0) (1,0)] is determined. All other points of the model *p*3 is chosen in turn and the transformation, based on translation, scale and rotation, is applied to return a coordinate (x,y) or key from which the hash index is created. This process is repeated for all combination of points.

```
FOR p1<= 0 TO number of model points
  FOR p2 <= 0 TO number of model points
     IF p2 != p1
        GET angle between p1 and p2
        GET relative scale reducing [p1 p2] to [(0,0) (1,0)]
        FOR p3 <= 0 TO number of model points
        IF p3 != p1 and p3 != p2 THEN
           determine x,y coordinate using angle
           and scale factor relative to p1
        ENDIF
        ENDFOR
     ENDIF
  ENDFOR
ENDFOR
```

Figure 5-2 Transformation of feature points

During this process, a scale is determined and applied to transform the coordinate frame from [(0,0) (1,0)] to [[(0,0) (*n*,0)] and consequently the third point is transformed by this scale factor to create the (x,y) coordinate. As the distance between basepoint 1 and basepoint 2 may be smaller than the distance between point 1 and point 3, the returned coordinate may have values for x and y that are greater than n. Because all combination of point pairs are chosen to form the basis, the relative positions of basepoint 1 and basepoint 2 compared to point 3 will reverse so the returned coordinate may have values for x and y that are negative. Thus the ranges $-\infty < x < +\infty$ and $-\infty < y < +\infty$ are possible and are dependent on the relative positions of the three points and the scaling factor used.

A hashing algorithm uses the coordinates to create the index. The data is stored as a 3D array in terms of the index (*x*,*y*) which determines the bin for data storage for $0<x<32$ and $0<y<32$ and the bin depth which allows storage of more than one data item in each bin.

An index in terms of an (*x*,*y*) coordinate is obtained by applying a function to the key (*x*,*y*) from the previous process which wraps the coordinate into an absolute coordinate frame by applying modulo arithmetic. The algorithm in Figure 5-3 shows the process of determining the hash index. To map the coordinates onto an array (32×32), modulo arithmetic (*x* mod 16 and *y* mod 16) is applied to each coordinate. This wraps values greater than 16 or less than -16 to the range $-16<x<16$ and $-16<y<16$. A constant (16) is then added to shift the values right to fit within the coordinate frame of $0<x<32$ and $0<y<32$. This become the index to the hash table.

```
index x <= coordinate x % 16 + 16
index y <= coordinate y % 16 + 16
```

Figure 5-3 Wrapping function

The *bin depth* determines the number of bin entries and is also used to check for bin overflow. The points, as (*x,y*) coordinates, are stored in the array at the location indicated by key (*x,y*). In the event that bin overflow occurs, the system is aborted and a deeper bin is created manually. In an ideal, non parallel implementation, a linked list would be used to implement the bin to allow dynamic allocation of memory for bin depth, however this cannot be the case in the parallel implementation and the issue will be further discussed in chapter 6.

The spread of the data over the available array is determined by the scaling factor and the hashing function used. The scaling factor determines the extent of spread and the hashing function performs wrapping and a shift right. With a small scale factor such as 5, the basepoints are scaled to [0,5] and so the coordinates obtained concentrate about the central [0,0] location, with minimum and maximum values of about 15. This value is dependent on the relative distances between the basepoint pair and the distance to the third coordinate from the first coordinate of the basepoint pair. Wrapping has little or no effect as the values determined rarely extend beyond the maximum range of 32. Large sized bins are required as the total number of entries need to be accommodated within this limited region.

With a scale factor of 10, the basepoints are scaled to [0,10] and so the coordinates obtained center about [0,0] but have minimum and maximum values of about 30. When shifted right, a larger proportion of the keys map outside the range 0-32 and so wrapping remaps these to an index within the hash table. This produces a better distribution of hash data and so reduces the bin size significantly.

As the scale factor increases, the spread increases and wrapping effectively distributes the data over the hash table. There is a limit to the effectiveness of scaling and wrapping.

The scale chosen spreads the key data and effectively leaves unused index values spread over the range. Because modulo arithmetic is applied to the key before the shift right, those indexes created remap into the unused places to allow a more even spread. This is effectively an interleaving process where the degree of interleaving is associated with the scale factor. A factor which is a factor of 32 will not allow interleaving whereas the prime number 19 provides the best interleaving.

Figure 5-4 shows the maximum bin size and the total number of bins with data for different scale factors. It shows two aspects of hash table distribution.

1. Overall distribution of data over the array. An increase in the scale factor generally causes more bins to contain less data and hence produce a better overall distribution of the data over the hash table. Improvements appear up to a scale value of 19, leveling out at about 600 bins containing data out of a total 1024. This gives an approximate 60% hit rate. As expected, there is a greater distribution of the data over the bins as the scale factor increases.

2. The maximum number of entries in a bin. This rapidly decreases until a scale factor of 10 is achieved. This is due to the scaling. Once beyond this, the value fluctuates as a consequence of applying modulo arithmetic which introduces the wrapping effect. This is particularly noticeable with scale factors of 16 and 32 which are factors of the hash table dimensions. The maximum bin content over the hash table gets less as the scale factor is increased again up to 26, and then no more gains after this are noticeable.

It appears from these observations and data in

Figure 5-4 that the choice of the scale value should be 19 or 26 as the spread of data over the hash table is maximised and the bin content minimised.

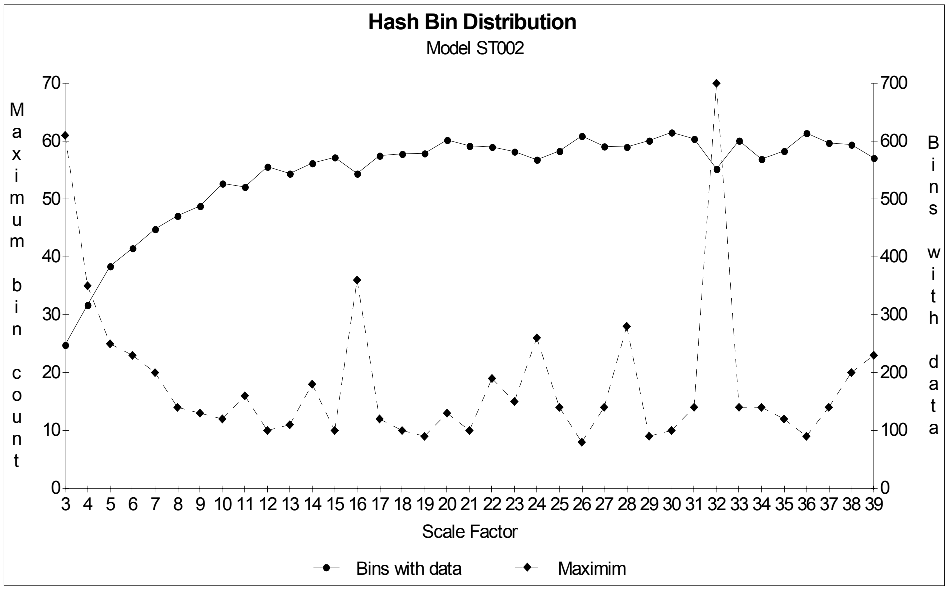


Figure 5-4 Effects of scale factor on Hash bin distribution

Figure 5-5 also demonstrates the effect of scaling and wrapping by displaying the spread of the data for predetermined scale factors. With the scale factor of 5, the data is concentrated about the center of the hash table with a few entries near the edges.

Figure 5-6 demonstrates the changes in the distribution of the hash table as the scale increases. The data is distributed more evenly but still has a concentration of data about the centre of the hash table.

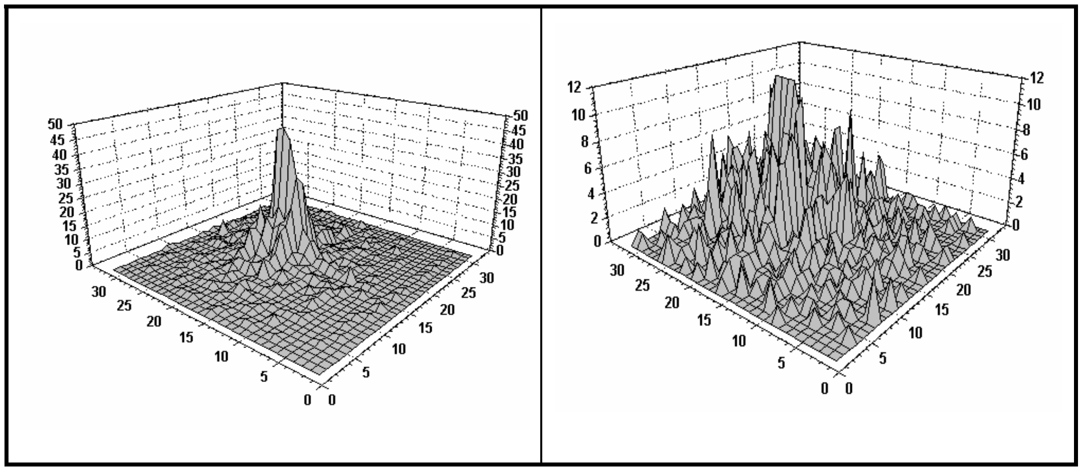
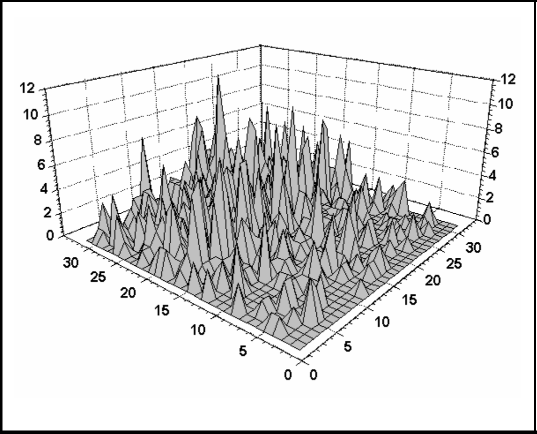


Figure 5-5 Model with scale factor 5 **Figure 5-6 Model with scale factor 10**

Figure 5-7 in particular, and Figure 5-8 shows a much better spread of the hash table over the array such that it is spread over the entire array and the peaks are not as high. The difference in the spread of the hash data with a scale of 19 compared with that of 26 is such that the spread for a scale of 26 in Figure 5-8 tends to have a higher concentration of the bins with larger peaks concentrated in one half of the bins.



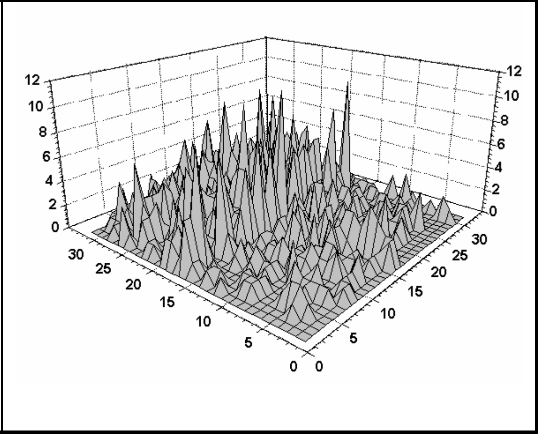


Figure 5-7 Model with scale factor 19 **Figure 5-8 Model with scale factor 26**

As the data is derived from CAD sources, it can be assumed that there is no measurement error in the position of the points of the model. Consequently, scaling does not cause error in hash bin placement. However, this is not the case with image data and hash bin access. The error in the position of the point data in the image will be affected by a change in the scale factor. Error in the image point location δp will cause a proportional error in the determination of the hash table coordinate δh. Given that the initial coordinate frame of [(0,0) (1,0)], a scale factor of n will result in a key error of $n\delta h$. To minimise the effect of this error, the scale factor must be kept to a minimum, thus reducing the hash table search area in recognition. For this reason, a scale factor of 19 was chosen as it gives bins with the lowest maximum number of entries in any bin for each scale factor and a lower $n\delta h$ compared to a scale factor of 26.

5.2 Image for recognition

The image used is a binary image where each byte with a value of 1 represents a visible pixel. The features that are extracted from the image are endpoints of lines.

It is possible to apply the search algorithm to images of any size, however the speed of recognition is dependent on the size of the image and the number of interest points in the image. The search algorithm is O(n^3) where n is the number of potential image points. It was decided that a practical image size for the non-parallel implementation was 32×32 or 1024k as this was adequate to test the algorithm.

Figure 5-9 shows the first test image that was created using the model CAD data. Figure 5-10 and Figure 5-11 show the images of the model where the CAD data has undergone translation, rotation and scale. Perturbation of some of the points was used in the creation of the data for the model resulting in the image shown in Figure 5-11. The circles indicate the location of the points used in the creation of the test images. Figure 5-12 shows the fourth test image and this was created from an image of the model instance which had been processed to extract line data and provide end point data as a file of points. The method of extraction is detailed in chapter 5.

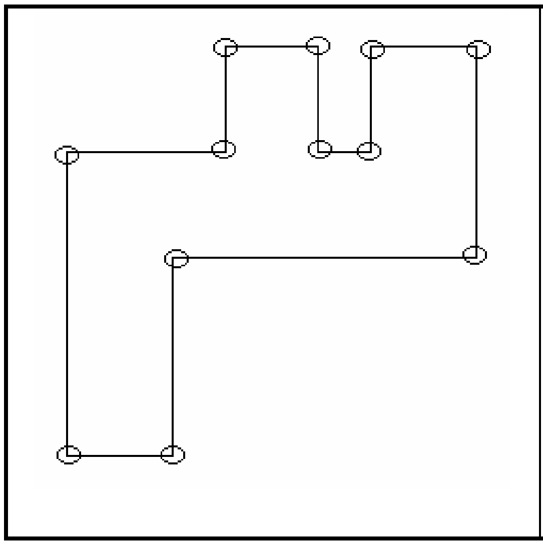


Figure 5-9 Test Image A

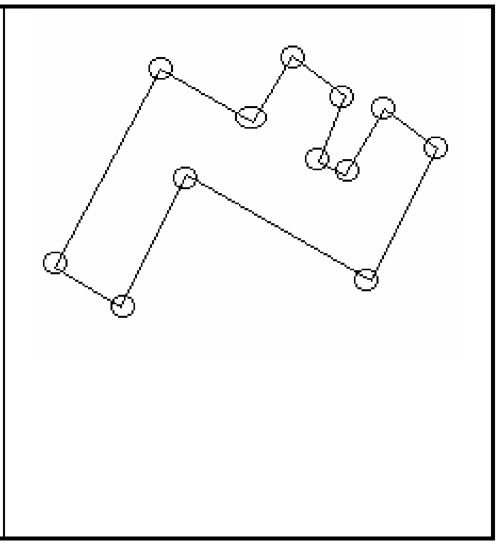


Figure 5-10 Test Image B

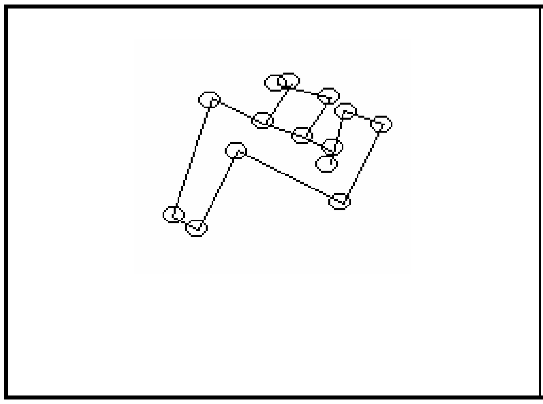


Figure 5-11 Test Image C

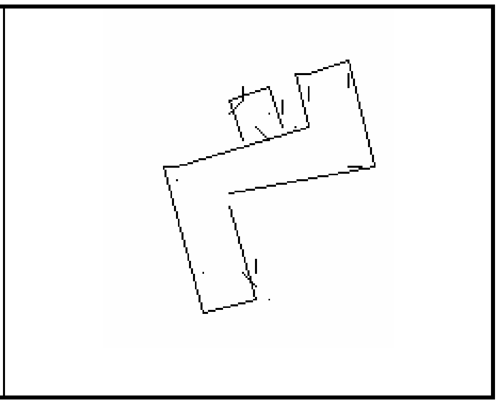


Figure 5-12 Test Image D

5.3 Recognition

The recognition begins by selecting two points of the image as a basis pair. A transformation is applied to the basis pair to map to [(0,0) (1,0)]. All other points of the image are selected in turn, and are transformed using the same transformation. Figure 5-13 shows this process where *p*1 and *p*2 are the points selected as a basis and *p*3 is some other point in the image. *p*1 and *p*2 are transformed to [(0,0) and (1,0)] and *p*3 undergoes the same transformation to become (*x*,*y*).

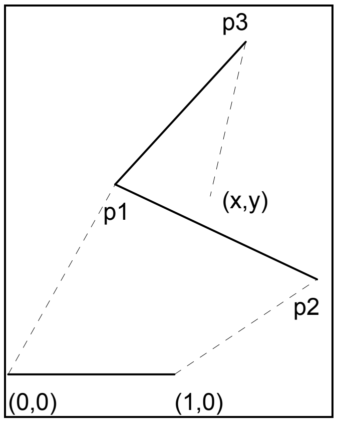


Figure 5-13 Basis and Coordinate transformation

The hashing algorithm is applied to produce a hash table index which is used to access the hash table and extract the basepoint pair data in that bin. An accumulator for each basepoint pair found is then incremented. After all image points are processed, the accumulators are searched to find any basepoint pair that have a value greater than the threshold. If any are found, those basepoint pairs are presented as potentially belonging to the model being sought. The above process is repeated for all combinations of two image points which form a basis pair.

5.4 Allowing for error

The major source of error in the recognition phase is that of feature positional error which can be a consequence of shadows, camera angle, aspect ratio and image processing techniques employed to reduce the image data to point data

5.4.1 Image point positions

Grimson (1990) introduced the concept of positional uncertainty to produce key values that are erroneous. Model data is considered accurate as it is taken from a

CAD definition of the model. However, the data obtained from the image after processing will not necessarily return point positions identical to the theoretical value from the CAD data. To account for this potential error in position, the following processes could be used:

1. Process all point positions within a given range about the image point. This would significantly affect the speed of processing as each point of the image would effectively be reprocessed by the factor relative to the range chosen. This would need to be done in the selection of the basis pair as well as the set of points determining the third point.

2. Use all entries in the hash table within a given range. This also increases the time taken for processing. However, in an ideal case, the hash data is only in the bin pointed to and thus processing time is hardly affected. In a noisy image, many of the bins pointed to by non-model data, as well as the neighbouring bins, may be empty and model data slightly off line will point to empty bins with the correct bin being a neighbour. This is the method chosen in this implementation.

An increase in the scale factor leads to magnification of the error as a scale factor of n will result in a key error of $n\delta h$. This reduces the likelihood of obtaining a hit for the correct bin and increases that for an incorrect bin if single bins are selected. To overcome this, a bin selection range needs to be established that relates to the degree of uncertainty.

All calculations relating to the translation, rotation and scaling of the image points to create the hash key are carried out in double precision. After the scaling factor and the modulo arithmetic is applied, the hash key returned is used to access the hash table. A simple expansion of $-1,0,+1$ is applied in the x and y directions to the hash key, H, and the hash table accessed at each of these 9 locations as shown in Figure 5-14. The values within these are used for accumulation purposes. For the purpose of this paper, this is defined Hash Table range searching, and is used in the recognition phase where there is noisy image data.

$H_{(x-1,y-1)}$	$H_{(x,y-1)}$	$H_{(x+1,y-1)}$
$H_{(x-1,y)}$	$H_{(x,y)}$	$H_{(x+1,y)}$
$H_{(x-1,y+1)}$	$H_{(x,y+1)}$	$H_{(x+1,y+1)}$

Figure 5-14 Hash table locations accessed

There are three major influences that affect the size of the search space. These are:

1. the error in the positions δp_1, δp_2 of the basepoint pair (P_1, P_2),
2. the error in the position δp_3 of the third point P_3,
3. the scale factor used.

Experiments are performed to empirically determine the extent of error in determining the hash key. The method is described below.

For an image space of 32×32, the error in the basepoint pair position was determined by setting the first point (x_1, y_1) at (0,0) and the second point (x_2, y_2) at values ranging from (1,0) to (27,5). The difference in the hash key obtained from pairs of adjacent second points was then determined where adjacency is defined as: $|x_{21} - x_{22}| \leq 1, |y_{21} - y_{22}| \leq 1$.

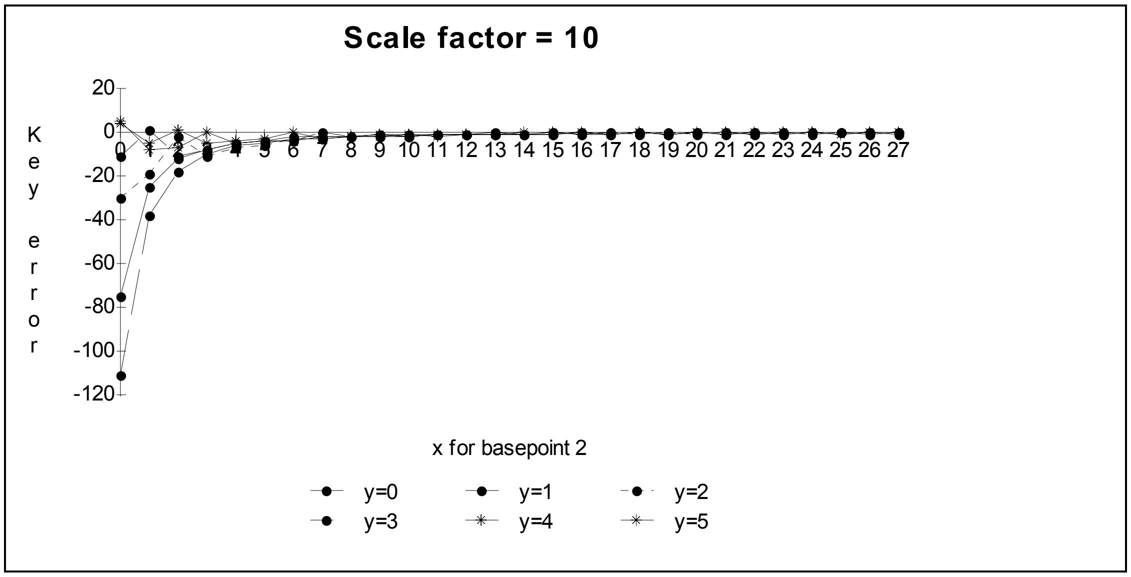


Figure 5-15 Radius of Error (Delta x)

Figure 5-15 shows the amount of error in the *x* key for a scale factor of 10, when $0 \le x_2 \le 27$ and $0 \le y_2 \le 5$. Six sets of data are shown for six values of y_2. With a basepoint-pair differences in excess of 10, the error in the hash table key is very small (of the order of 1). However, this error increases as the basepoint-pair difference is reduced.

Figure 5-16 shows a similar trend where a scale factor of 50 is considered. As a consequence, choosing basepoint pairs that are relatively far apart will limit the radius of error in determining the hash key. This will reduce the number of hash locations to be searched and reduce the number of false potential recognitions.

It was found that by ensuring that the gap between the two basepoint values was kept above a minimum value, the error involved in the calculation of the hash key δk was very small compared to the actual value. If the third point was considered to be in a slightly different position, the error in the hash key was of a similar value. Thus assuming the image pixel position was displaced by one, the hash key was similarly displaced. As a consequence, the hash table is searched minimally within a range of ±1. If the pixel error is considered greater than ±1, a larger hash table search space is required which then influences the discriminatory ability when selecting the correct model. As a consequence, the threshold must be reduced and the verification stage would take longer due to the larger number of possible models.

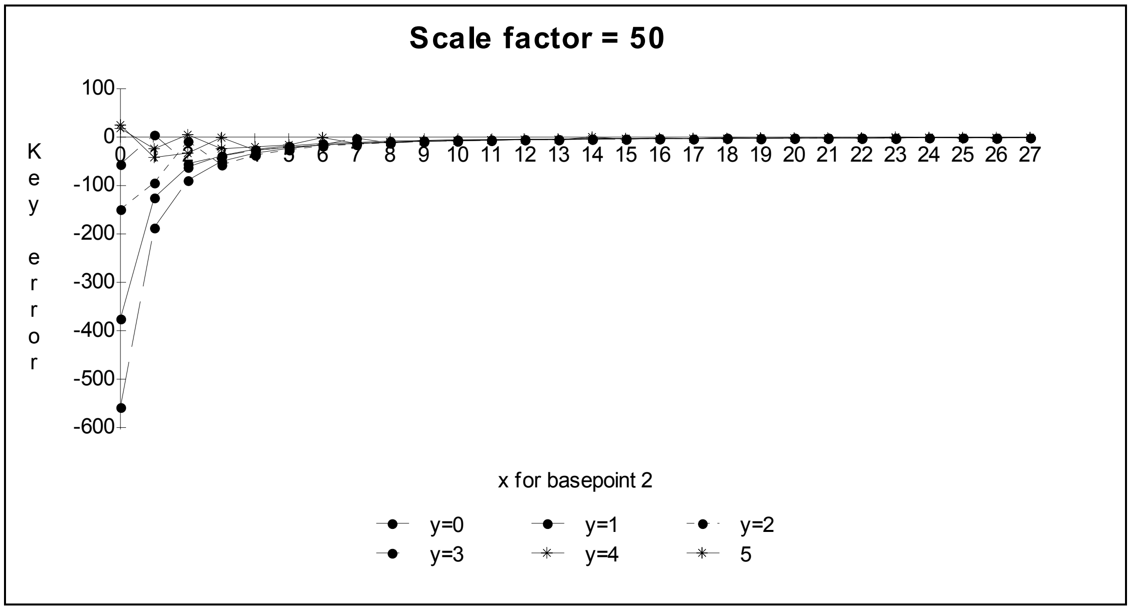


Figure 5-16 Radius of Error (Delta x)

5.5 Threshold

Theoretically, if a model has *n* points, then the number of votes for any given basis for the model needs to be (*n*-2) for recognition. For a given image, if a pixel is misplaced by more than one pixel, or is missing altogether because of occlusion or omission due to processing, then the number of votes needed becomes (*n*-2-*i*) where *i* is the number of missing or misplaced pixels. As a consequence, a threshold needs to be set that will account for these, as well as account for false votes due to lack of discrimination. In this implementation, the threshold is set at (*n*-2), which is 10, as there was no occlusion of the model in the image.

5.6 Bin sizes - Vote accumulation

An accumulator is assigned for each basis pair. For each basis selection, the hash keys are used to access the hash table at positions where there may be one or more basis-pair data located. For each of these, the appropriate accumulator is incremented. At the completion of the loop, the accumulators are searched and those with a count greater than or equal to the threshold are then considered as candidates for verification. The accumulators are implemented as a 2D array indexed by the basepoint pair (*bp1*,*bp2*).

The algorithm in Figure 5-17 searches the selected bin in the hash table and obtains a basepoint pair (*bp1, bp2*) for each entry. The array variable *count* is incremented for each of the entries (*bp1, bp2*).

```
Do for depth <- 0 to bindepth-1
  bp1 <- hashtable[depth][y_key][x_key].bp1
  bp2 = hashtable[depth][y_key][x_key].bp2
  Increment count[bp1][bp2]
Endfor
```

Figure 5-17 Increment counters

The algorithm shown in Figure 5-18 is used to select the most likely basis for model recognition. The array variable *count* is accessed. If any entry is greater than the threshold, the basepoints pair (*bp1, bp2*) is passed to the verification module.

```
Do for bp1 <- 0 to BASEPOINTS
  Do for bp2 <- 0 to BASEPOINTS
     If count[bp1][bp2] > THRESHOLD
       Verify(bp1, bp2)
     Endif
  Endfor
Endfor
```

Figure 5-18 Basis selection

5.7 Applying Heuristics to speed up search time

The knowledge that feature points of a given model are a predetermined distance allows some heuristics to be applied. This allows for a significant speedup in the recognition time as not all combinations of points in the image need to be considered. Those basepoint pairs that produce an inter-basepoint distance greater than the largest model inter-basepoint distance and smaller than the smallest model inter-basepoint distance can automatically be discounted. To choose the basepoint pairs to initiate a transformation, the size of the expected model in the image needs to be considered. Knowledge of the model shape and thus the distance of the closest or furthermost model points gives an indication of the minimum or maximum inter-basepoint distance.

Points of the model that are very close together give less accurate hash table key values. It was shown earlier that the relative error is inversely proportional to the inter-basepoint distance assuming that the pixel positional error is constant. Points that are close together in the model can be discounted.

5.8 Results

The following tables show the output of the recognition phase for each of the four images used to test the algorithm. They list the coordinates in the image and the corresponding basepoint pair of the model. As well, the accumulated votes for the basepoint pair above the threshold is displayed. Arrows, when present, indicate that not all data is presented. Shaded areas indicate that the presented basepoint pair was an incorrect recognition.

The original model data used as an image could be considered without noise or error. Figure 5-19 indicates the numbering used for basepoint identification for the model chosen. The image coordinate values are $0 \le x < 32$ and $0 \le y < 32$ and Table 5-1 gives the relative coordinates.

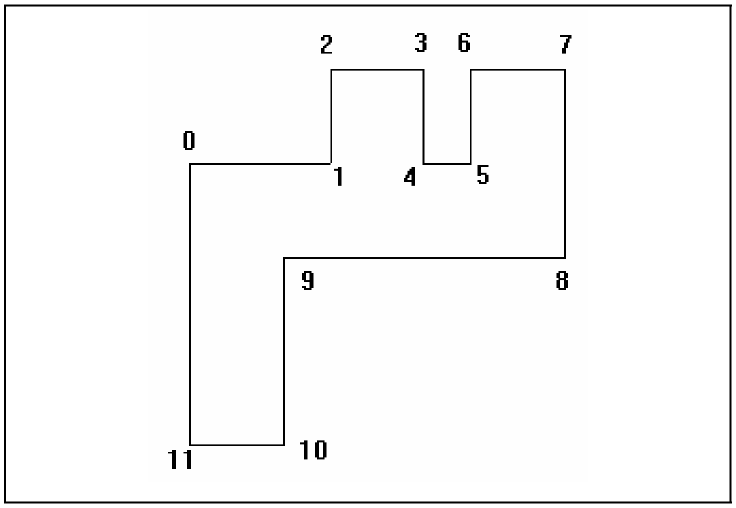


Figure 5-19 Model showing coordinate numbering

Coordinate Number	Coordinate
0	(0 8)
1	(12 8)
2	(12 0)
3	(19 0)
4	(19 8)
5	(23 8)
6	(23 0)
7	(31 0)
8	(31 16)
9	(8 16)
10	(8 31)
11	(0 31)

Table 5-1 Coordinates for Model

Coordinate 1	Coordinate 2	Basepoint Pair	Count
(12 0)	(19 0)	(2 3)	12
(12 0)	(23 0)	(2 6)	10
(12 0)	(31 0)	(2 7)	10
(12 0)	(0 8)	(2 0)	10
(12 0)	(12 8)	(2 1)	10
↓	↓	↓	↓
(8 16)	(31 16)	(9 8)	12
(8 16)	(0 31)	(9 11)	10
(8 16)	(8 31)	(9 10)	26
(31 16)	(0 31)	(8 11)	10
(31 16)	(8 31)	(8 10)	12
(0 31)	(8 31)	(11 10)	10

Table 5-2 Image 1 with Hash Table range searching off

Table 5-2 shows the part of the output obtained when image 1 is used and Hash Table range searching turned off. The model in the image was able to located for all basepoint pair combinations when the threshold was set to 10. As some point triples with the same basis pair map to the same bins in the hash table, some instances received a count greater than 10 with the maximum being 26. There were no incorrect detections as the image contained no noise.

Table 5-3 shows some of the data obtained and indicates the differences in accumulator value when Hash Table range searching is on. It shows that the number of instances of the model found increased. The count for each basepoint pair increased due to the effect of neighbouring hash locations containing basepoint pair data which affected the overall accumulator count. Note that there is an increase in count for the same positions when compared to Table 5-2.

Coordinate 1	Coordinate 2	Basepoint Pair	Count
(12 0)	(19 0)	(2 3)	14
(12 0)	(23 0)	(2 6)	18
(12 0)	(31 0)	(2 7)	12
(12 0)	(0 8)	(2 0)	12
(12 0)	(12 8)	(2 1)	14
(12 0)	(19 8)	(2 4)	14
↓	↓	↓	↓
(8 16)	(8 31)	(9 10)	26
(31 16)	(0 31)	(8 11)	12
(31 16)	(8 31)	(8 10)	14
(0 31)	(8 31)	(11 10)	14

Table 5-3 Image 1 with Hash Table range searching on

Coordinate 1	Coordinate 2	Basepoint Pair	Count
(20 10)	(2 25)	(2 11)	10
(10 11)	(31 17)	(0 7)	10
(10 11)	(24 19)	(0 5)	10
(10 11)	(2 25)	(0 11)	11
↓	↓	↓	↓
(31 17)	(26 26)	(10 9)	26
(31 17)	(7 28)	(7 10)	11
(12 18)	(26 26)	(9 8)	11
(24 19)	(2 25)	(5 11)	10
(24 19)	(7 28)	(5 10)	10
(2 25)	(26 26)	(11 8)	10
(26 26)	(7 28)	(8 10)	12

Table 5-4 Image 2 with Hash Table range searching on

Table 5-4 shows the output for the second image with Hash Table range searching on. Many instances of the model were found. The count above the threshold is similar to that for the original model data indicating that the key to the hash table was within the range of ±1, but there were less instances indicated. With Hash Table range searching off, the inaccuracy in the location of the points resulted in failure to detect the model.

Table 5-5 shows the output when a threshold of 10 is used. The incorrect bases are shaded. The count above the threshold is similar to that for the original model data indicating that the key to the hash table was within the range of ±1, but there are even less instances indicated than in model 2. It indicates that the inaccuracy in the location of the points causes the program to present few instances of the model. As well, one of those basepoints was found to be incorrect and thus a verification stage would need to be implemented to determine the correct basis-pair for the model at a particular location.

By lowering the threshold, more instances to be presented but many of these are incorrect. Thus, as the threshold is reduced, the number of incorrect instances generated increases and consequently there is a greater need for verification.

Coordinate 1	Coordinate 2	Basepoint Pair	Count
(11 6)	(24 8)	(0 7)	11
(11 6)	(24 8)	(9 6)	12
(11 6)	(21 14)	(0 8)	10
(21 7)	(8 15)	(6 11)	10
(21 7)	(10 16)	(6 10)	10
(24 8)	(13 10)	(7 9)	11
(24 8)	(8 15)	(7 11)	11
(24 8)	(10 16)	(7 10)	10
(13 10)	(21 14)	(9 8)	11

Table 5-5 Image 3 with Hash Table range searching on

Table 5-6 shows data for the fourth image with a threshold set at 10. The noise in the image caused the program to present many instances of the model. Also, many of these was found to be incorrect and thus a verification stage would need to be implemented to determine the correct basis-pair for the model at a particular location. The high count and consequent high basis pair instances is a direct result of noise points effectively randomly accessing the hash table and incrementing the accumulator for any basepoint pair. To cope with noise, the threshold needed to be increased, but this reduced the ability of the program to cope with occlusion.

The shaded areas in the table indicate those positions that are erroneous. As can be seen from the data, the program returns many alternate basepoint pairs as existing at the same coordinate location. The only way to determine the existence of the model at this stage is to use a verification process. The program cannot be used by itself and, in this case, only acts as a filtering mechanism to reduce the need for searching.

The above results indicate that for images created from CAD data, and for images with little noise and variation from the theoretical point positions, Geometric Hashing techniques alone are sufficient for recognition. However, for noisy images and those where occlusion and positional variations occur, more processing is required for recognition. The major areas of interest are (1) an increase in the number of points to be processed, (2) Hash Table range searching and (3) the inclusion of a verification stage. Each of these increase the processing time required for recognition.

Coordinate 1	Coordinate 2	Basepoint Pair	Count
(19 10)	(5 18)	(7 0)	16
(19 10)	(5 18)	(11 6)	17
(19 10)	(6 18)	(1 4)	19
(19 10)	(6 18)	(4 1)	17
(19 10)	(6 18)	(7 0)	17
(19 10)	(12 26)	(7 10)	18
(19 10)	(12 27)	(7 5)	19
(19 10)	(12 27)	(7 10)	20
(19 10)	(12 28)	(7 10)	22
(19 10)	(13 28)	(7 10)	21
(19 10)	(8 29)	(7 11)	27
(15 11)	(8 29)	(6 11)	18
(16 11)	(8 29)	(6 11)	22
(19 11)	(5 18)	(7 0)	16
↓	↓	↓	↓
(19 12)	(8 29)	(8 9)	19
(20 18)	(8 29)	(8 11)	17
(21 18)	(8 29)	(8 11)	22

Table 5-6 Image 4 with Hash Table range searching on

5.9 Conclusion

An implementation of Geometric Hashing was created that allows the recognition of a 2D shape within an image, in situations where perturbation of the image points and noise occurred. The major issues in this implementation relate to (1) the creation of the hash data, (2) the distribution of the data over the hash table using a system that allows a maximum even distribution of the data over the table, (3) mechanisms that allow for positional error of the image points and (4) the key issues in the voting process.

The following chapter takes these issues and applies them in the context of a SIMD parallel environment to the recognition of the 2D views of a 3D widgit. The same issues are dealt with, but in the context of the distribution and access to the Hash table data over the process array and the need to virtualise the array to cope with the image size and volume of hash table data.

6. Implementation of Geometric Hashing (parallel)

This implementation of Geometric Hashing on a parallel system involves the recognition of 2D views of a 3D model which are the stable states of the model. To begin, the issues relating to the collection and representation of the model data, and other issues specific to our implementation of Geometric Hashing on a DECmpp 12000 computer, are discussed in detail. Discussion relating to the virtualisation and distribution of the image data and hash table data follows, with inter-processor communication being a major component. The data accumulation and model hypothesis stage is then presented. The verification stages needed to guarantee model recognition and its implementation in a parallel environment are also discussed and finally, the results of a typical recognition are presented.

6.1 Models and Images

The model data were taken from CAD descriptions of the seven stable states of the model. This was represented as line end point data in the range of $0 \le x \le 1.0$ and $0 \le y \le 1.0$. This data was used to create images of size of 64×64 and then to create the hash table which was stored on disk.

Figure 6-1 is a pictorial representation of the seven stable states of the model chosen as described by the CAD data. The features used for the learning phase are the line endpoints or vertices.

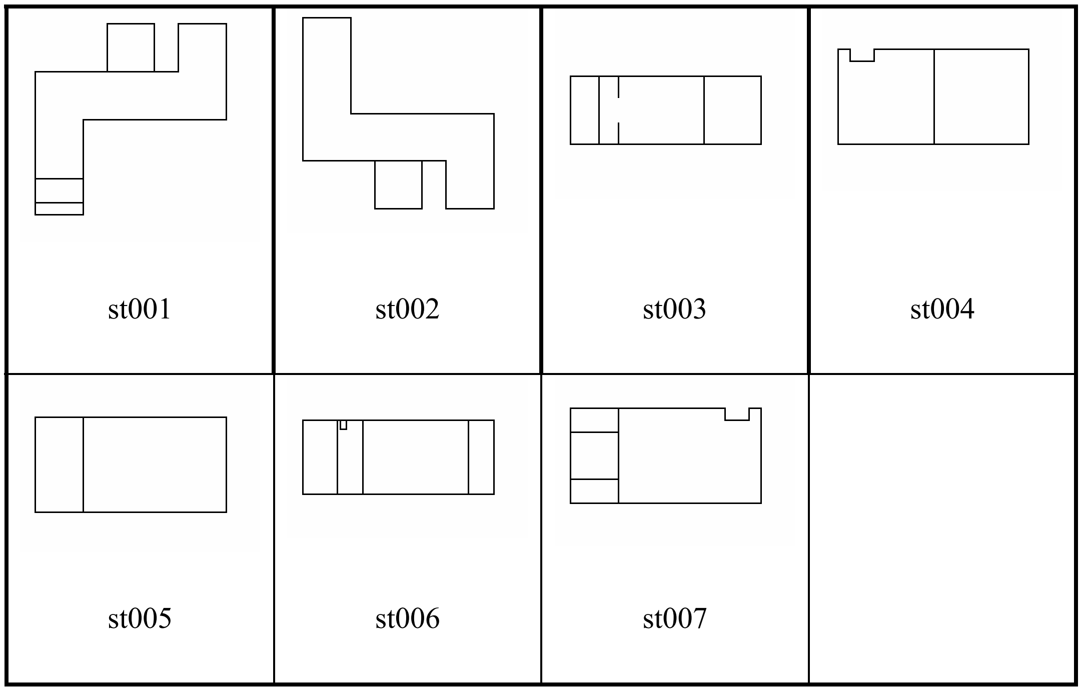


Figure 6-1 Seven stable states of Model

Figure 6-2 shows original images used in the testing of the recognition phase, as well as line images of the same images after processing in preparation for recognition.

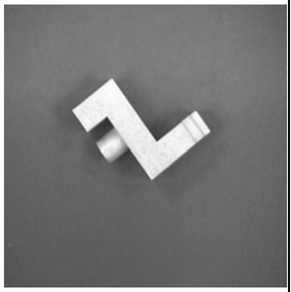
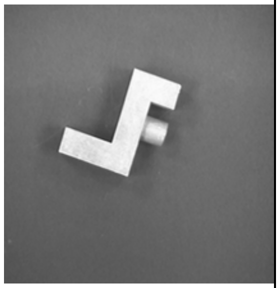
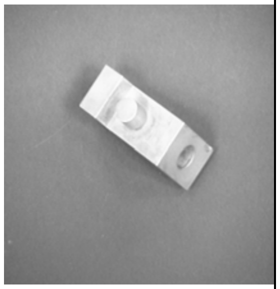
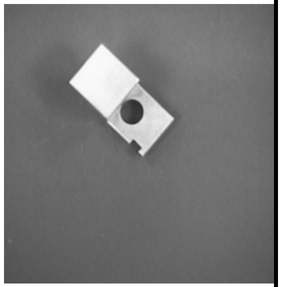
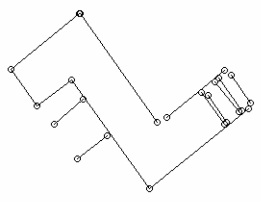
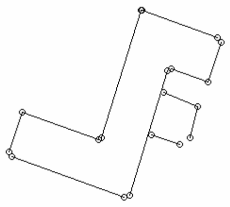
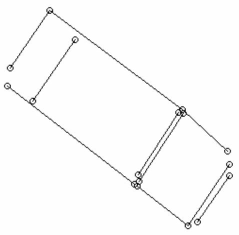
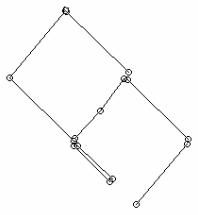
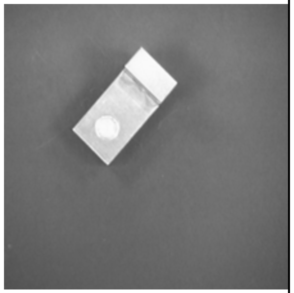
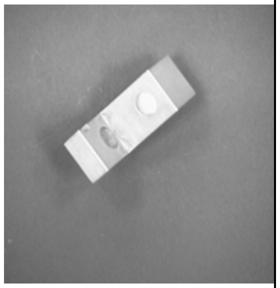
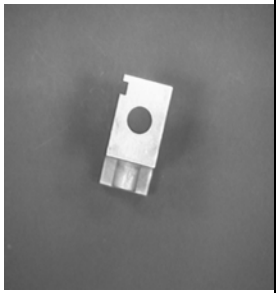
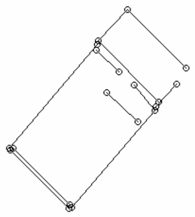
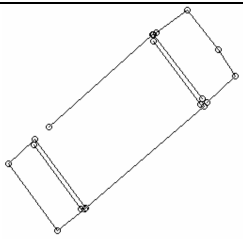
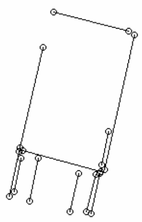
Picture 1 	Picture 2 	Picture 3 	Picture 4 
			
Picture 5 	Picture 6 	Picture 7 	
			

Figure 6-2 Images used in recognition

The original images were obtained using video equipment where the orientation of the camera was directly above the model placed on a light table. They were saved as a 512×512 grey scale image. In a real system, the preprocessing would be carried out in place on the processor array. However, the scope of this thesis does not cover preprocessing, this is also done off line.

The images obtained were processed (see Appendix 1) to produce 256×256 binary images where points or pixels at the ends of lines are labelled as grey level one and the rest of the image pixels are labelled as zero. The images shown in the figure are line images and have the point used in the recognition phase marked by a small circle. The lines with the figure are presented to allow visualisation of the data which is only stored as points at the end of lines.

Distance transforms of model and image data is required in the verification stage. These were produced off line and loaded onto the processor array before recognition was attempted. The distance transform for the image data would be produced in place in a real system. A description of the structure of a these files is discussed in Appendix 1.

Data required for recognition and stored in the data memory area of the DECmpp 12000 is as follows:

- Binary image of the model for recognition (stored on the Processor Element (PE) Array).
- Hash Table data (stored on the PE Array).

Data required for verification, and stored in the data memory area of the DECmpp 12000 is as follows:

- Distance transform for each model in data base (stored on the PE Array).
- Distance transform for image (stored on the PE Array).
- Point data for all models in the data base. (Stored on the Array Control Unit (ACU)).
- Line data for all models in the data base. (Stored on the ACU).
- Line data for image. (Stored on the ACU).

Point and line data is in the form of a list of numbers, and can easily be stored and accessed by any PE when stored on the ACU. The distance transforms are 256×256 grey scale images and are stored using hierarchical stack method on the PEs in the same manner as the image file containing the data for recognition. This is discussed in more detail later. It should be noted here that a distance transform is required for each model in the data base as well as for the image. This is because the verification stage is a two stage process that compares model against image and image against model. Consequently 32 bytes of memory is required on each PE for each model and for the image.

6.2 Virtualisation

The DECmpp 12000 used is a 2k processor machine, with each Processor Element (PE) configured as part of a 64×32 array. The initial 32×32 image and hash table of 32×32 as used in the non parallel implementation would allow each PE to hold one image pixel or one hash bin. However, the captured images would eventually be of varying sizes up to 512x512 and the number of hash bins would need to be increased to cater for the large amount of data from multiple models. Thus there was the need to virtualise the processor array to cater for (1) image placement and (2) hash bin allocation. This implementation allows the processing of images from a 256×256 image and 128×128 hash bins each with a depth of 100 entries.

The basic concept is as follows. A parallel array is assigned to each PE, where one item in the array represents a virtual processor. A mapping scheme is then used to distribute the data in the array on each PE and to allow access to the data based on the processing needs during recognition. There are four basic mapping procedures used for this. These are (1) 1D Hierarchical, (2) 2D Hierarchical, (3) 1D Cut and Stack and (4) 2D Cut and Stack (DECmpp 12000, 1992).

6.3 Placement of the Image data on the processor array

An important issue in using parallelism is the processing required to extract image features and to handle them in the context of geometric hashing. Ideally, feature extraction should be performed on the processor array as this removes the overhead of preprocessing the captured data on another system, and then loading the feature data onto the processor array. The capturing process would directly load image features onto the PEs for immediate processing. In this implementation, however, feature extraction is not implemented *in situ* and feature points are loaded onto the PEs after processing of the image off line. Further study would involve the in-place treatment of data which would include edge detection, feature extraction and creation of distance transforms in parallel.

The features of interest are end points of lines located in the image and should represent the vertices of the model. This image data has to be virtualised because the captured image has more pixels than the PE array has processors. A one dimensional hierarchical mapping scheme is used in which a contiguous block of image data is placed on each PE. For the case of an image of 256×256 and an array of 64×32 PEs, there are 32 pixels per PE. Hence the first 32 horizontal pixels are placed on the first PE, the second 32 placed on the second PE and so on. This scheme was detailed in chapter 3.

Figure 6-3 is the algorithm that shows the method by which feature data is loaded onto the processor array using a parallel read from the disk system. The image file is opened for reading and 32 contiguous bytes of data are placed onto each active processor using a parallel read from the data file stored on disk.

```
1 Open image file for read only
2 Parallel read 32 byte block of image data
3 Close image file
```

Figure 6-3 Algorithm for parallel read of feature data

The distribution of the end points of the lines in the image is non deterministic as the position, orientation and scale of the model in the image is unknown. It is generally sparse because the end points of lines are usually far apart. The objective is to have a uniform distribution of image feature points on each PE and so, with a sparse distribution, few feature points will be on any one PE. The following problems have been identified and hierarchical distribution offers a possible solution.

- Most parallel communications will be between PEs some distance apart, hence the need to use the Global Router for interprocessor communications. The process will serialise when there is communication between PEs within a cluster.

- Global router communications become serialised due to communications between different PEs in a given cluster. This is also minimised as there is a small number of feature points within each cluster thus reducing the probability for the need to access relatively close data.

- Communications within one PE where virtual processors are instantiated on the same PE. This has not been addressed and the global router is used for this communication. In this case, the overhead of communications and serialisation is high but is kept to a minimum as most virtual processors have no model points stored ie image values of zero. For an image generated from CAD data, the maximum number of feature points is 16, which represents 0.024% of the image space and, as such, 0.024% of the virtual processors and 0.78% of the PEs. When processing real images, noise producing false vertices will increase this to some extent.

Hierarchical distribution is also used in this implementation as it allows the use of search techniques that activate as many PEs as possible given a normal set of image data at the one time. The algorithm also uses a data acquisition technique that minimises the probability that two or more PEs would address data from within the same cluster.

6.4 Selection of the Image data on the processor array

For each PE containing interest points or features, a basis consisting of two different features must be determined and then the remaining features must be selected in turn to be able to recognise a model instance with a given basis pair. This process is described below and is termed a probe.

Figure 6-4 is the algorithm that ensure as many PEs as possible are simultaneously active. The array on the PE representing the virtual processors may contain no interest points, one interest point or many interest points.

The first WHILE loop finds all PEs with at least one interest point and eliminates all other PEs without an interest point. As each PE must execute the same code, once a PE has determined that there is an interest point to process, it must wait until all other PEs have found an interest point or it has exhausted its list. If a processor has an index value of MAXLAYER, then it has no interest point stored.

Within the second WHILE loop, the selected PEs are activated to select the second interest point to form a basis. The selection of the second feature and processing of the remaining points to achieve recognition with this basis pair is performed in the following way.

```
On each PE do
  index = 0
  WHILE index<MAXLAYER and image[index] not an interest_point do
      increment index
  ENDWHILE
  WHILE ((index<MAXLAYER) do
      Select element with a feature
      select 2nd interest_points and
      determine if part of model
      increment index
      WHILE index<MAXLAYER and image[index] not an interest_point do
         increment index
      ENDWHILE
ENDWHILE
```

Figure 6-4 Algorithm for selection of processor with first interest point

The third and inner WHILE loop finds the next set of PEs with at least one remaining interest point and the process is repeated, and continues until all interest features stored on each PE have been processed. It should be noted that the system must wait until all PEs with interest features have selected the feature or exhausted its list, and then all active PEs process their data in parallel. As the WHILE loop continues, more PEs will be deactivated until no PEs have interest points remaining.

PE	Layer	PE	Layer	PE	Layer
2	7	686	13	908	24
464	0	714	11	911	0
478	30	751	25	921	13
519	5	768	27	926	30
542	24	799	23	1019	4
553	31	815	16	1210	5
574	20	823	13	1460	16
662	6	869	2		

Table 6-1 Result of processor selection

Table 6-1 illustrates a typical application of the algorithm in Figure 6-4 when the first WHILE loop finishes. It shows the active processors and the array index when image data is processed. The column labelled **PE** contains the real processor number and the column labelled **Layer** contains the array index (virtual processor stored on this PE) for the location of the image point data. Each of these PEs have one or more interest points stored on them.

The algorithm in ***Error! Reference source not found.*** is used to ensure that, in the selection of the second interest point for the creation of the basepoint pair, no other PE is able to select it as well. Although this does not guarantee that the second points will not be from a PE within the same cluster, the probability that they are is the same as that for the selection of the first point. This is because clusters are in a local 4×4 block of PEs and the selection process does not account for this.

The significant difference between this algorithm and the previous one is that all feature points from the current PE to the last PE are processed in turn to determine the coordinate of the second feature to form a basepoint pair. In searching forward from the current PE, all other PEs act in unison and search from their respective positions, stepping forward until there are no more feature points to process. This process ensures that all feature point combinations are selected as basepoint pairs, and no PEs will simultaneously use the same feature point as the second point to form a basis.

One disadvantage with this is that the PEs at the beginning of the processor array must process more basepoint pairs than those at the tail. However, recognition should occur after a small number of probes and the process can be terminated. An alternative method would be to allow wrapping similar to the algorithm for the selection of the features to create the indexes.

```
For each active PE do
  FOR bp2<- current PE to last PE do
     IF bp2 = Current PE THEN
        layer2 <- layer1 + 1
     ELSE
        layer2 = 0
     ENDIF
     WHILE layer2 < MAXLAYER) and image[layer2] = 0 do
        Increment layer2
     ENDWHILE
     WHILE layer2 < MAXLAYER) do
        Select element with a feature
        determine if part of model
        increment index
        WHILE index<MAXLAYER and image[index] not an interest_point do
           increment index
        ENDWHILE
     ENDWHILE
  ENDFOR
```

Figure 6-5 Algorithm for selection of the processor with the second interestpoint

The description of the algorithm in **Error! Reference source not found.** is as follows. All PEs from the current active PE to the last PE need to be searched for potential second feature. The FOR loop selects the PEs in turn from the current PE to the last. The IF condition determines the image array index to start searching from to find a feature. If the PE is the same as for the first feature, the index starts at the next index value to ensure the basepoint pair represent different features. If not on the same PE, then the index starts at zero. The next WHILE loop searches the array for elements containing features and uses the basis pair selected to determine the existence of the model instance at the selected basis. This process is discussed in more detail below. The inner WHILE loop determines the next array element that contains an image point.

The algorithm in **Error! Reference source not found.** is used to select the third feature. This is used in conjunction with the basepoint pair already selected to determine the hash key and subsequent incrementing of the appropriate accumulators for recognition. All features need to be accessed by the active PEs which represent each one of the selected basis pairs. If each active PE was to search for these from the start of the processor array, the algorithm would serialise as each PE would try to access the same PE. By starting at the active PE, going to the end of the processor array and then starting from the beginning, each active PE accesses a different PE to obtain the third point. The further away the PE containing the third feature point is from the cluster containing the active PE, the less probability there is of serialisation. However, if two or more active PEs are within the one cluster, then their processes will always be serialised.

```
For each active PE do
  FOR bp3_start <- 0 to nproc do
      bp3 <- (bp3_start+iproc) MOD nproc
      layer3 <- 0
      WHILE layer3 < MAXLAYER and image[layer3] = 0 do
         increment layer 3
      ENDWHILE
      WHILE layer3 < MAXLAYER do
         Determine key and increment accumulators
         increment layer3
         WHILE layer3 < MAXLAYER and image[layer3] = 0
            increment layer3
         ENDWHILE
      ENDWHILE
```

Figure 6-6 Algorithm for selection of the processor with the third interest point

The algorithm achieves this in the following way. The first loop counter is added to the PE number and the sum is modulo *nproc* to return a cyclic processor sequence. The WHILE loop determines the image array index on the selected processor that contains an image point and is used to determine the keys and increment the appropriate accumulators. The inner WHILE loop is used to detect the next array index containing a feature.

6.5 Placement of hash table data on the processor array

An important issue is that of placing the hash bins onto the processor array in order to improve timing of recognition. Different methods have been proposed by Khokhar, Prasanna & Kim, (1993), Prasanna & Wang, (1993) and Rigoustos & Hummel, (1992). Inappropriate placement will potentially cause serialisation which needs to be avoided in a parallel implementation. These methods, described below, are (1) the hash data is distributed over sub-arrays of processors, (2) a complete copy of the hash data is stored on each PE and (3) the hash data is equally distributed over all PEs.

6.5.1 Data distributed on each sub array of processors

In Khokhar, Prasanna & Kim (1993), the hash table is placed in its entirety on each PE sub-array. This is only possible when there is sufficient memory as this requires multiple copies of the hash table. One probe is handled by the processors in the sub-array. The major advantage of this technique is when the sub-array is a cluster and the PEs within the cluster are responsible for the entire processing of one probe. In this case, PE contention would be eliminated. This method would also require a complete copy of the image distributed over the cluster to ensure serialisation of the process was not a problem.

6.5.2 Data on each PE

Prasanna & Wang (1993) placed the hash table is in its entirety on each processors. This is an ideal situation but is only possible if sufficient memory is available on each PE. Router communication is reduced and PE contention eliminated during hash bin access, thus significantly increasing the speed of recognition. Given the redundant nature of the hash table data, this would work fine for a small number of model instances, but would be impractical for real world model recognition. The space requirement to store the hash table data for the seven stable views of the model used in our experiments is in excess of 3 megabytes. Each PE on the DECmpp 12000 has 128k for data storage. Each virtual processor shares a portion of this memory and so there would be insufficient memory for this method.

6.5.3 Equal distribution of hash data over all PEs

Khokhar, Prasanna & Kim (1993) equally distributed the hash table data over all PEs. This requires only one copy of the hash table saving memory but increases communication times. The processor array is virtualised and the hash bins are stored equally over all PEs. As the hash table access is random and communications is by the global router and not X-NET, serialisation of the process occurs in some cases as a consequence of different PEs simultaneously requiring data stored on one PE or within a given cluster of PEs.

The amount of data stored on each PE is determined by the degree of virtualisation, the number of models and the number of feature points in each model.

The degree of virtualisation has two major effects, these being (1) the quantisation of the hash space and (2) the level of interprocessor communication. This is only limited by the amount of memory available on each PE within the processor array.

In our method, the hash table is distributed such that each virtual processor contains one hash bin. The quantisation of the hash space, and the degree of virtualisation, depends on the scale factor used to change the basis from [(0,0) (1,0)] to [(0,0) (*n*,0)] where *n* is the scale factor. Thus for a scale factor of 30, the hash index values would have the range $-30 \le x \le 30$ and $-30 \le y \le 30$. After a shift right of 30, the hash space would need to be 60×60 processors in size. Thus the processor array would need to be virtualised from 64×32 to a minimum of 64×64 PEs.

Each bin may contain a number of model basis-pair records because of the number of models required to be recognised and the number of feature points. That is, there may be numerous triples that have the same geometric configuration which are subsequently stored in the same bin. During recognition, voting is based on the hash key which points to a particular bin and consequently, the accumulator for each of the model-basis present is increased. Thus, the degree of virtualisation affects the total number of bins available and consequently the degree of discrimination. With a large number of model basis-pair in each bin, there would be an increase in the accumulation of votes for the model-basis in the bin, and a subsequent possibility of incorrect model recognition and lack of discrimination.

There is also the parallel issue in that potentially more PEs will be attempting to access the same bins, or bins in the same cluster simultaneously. As only one PE at a time can access a second PE or a PE in the same cluster, all other PEs must wait until the first PE has finished. This leads to the serialisation of the search process. Virtualisation of the PE should allow a more even spread of the hash table and a consequent reduction in the instances of model-basis in each bin, thus reducing the chance of serialisation.

The choice of scale factor used will influence the distribution of the data in the bins, with the objective of getting the minimum amount of data in as many bins as possible. Figure 6-7 shows the effect on the number of entries in the hash bins when data for all the model instances are stored. Scale factors are from 10 to 39 with hash table dimensions of 64×64, 128×128 and 256×256. As discussed in the previous chapter, the objective is, for a given scale factor, to determine the maximum number of hash entries in each bin and then to determine the minimum value over the range of scale factors, thus determining the scale factor that gives the greatest spread of the hash data over the hash table. A minimum number of bin entries implies less processor contention during hash table access in recognition, and consequent faster recognition.

As can be seen, generally as the scale factor increases above 18 the lower the bin count. However, the values of 27 for a table of size 128×128 provides the lowest maximum number of entries. Beyond a scale value of 27, there appears to be an increase in the lowest maximum. Also, as the scale factor increases, the radius of error in the recognition phase increases proportionally, thus the scale factor must be kept to a minimum.

The process of determining the best scale factor for the model instances used can be automated during the learning phase, as the criteria is the lowest maximum bin content over a given range of scale factors. The optimal scale factor can then be passed onto the recognition phase when the hash table is loaded.

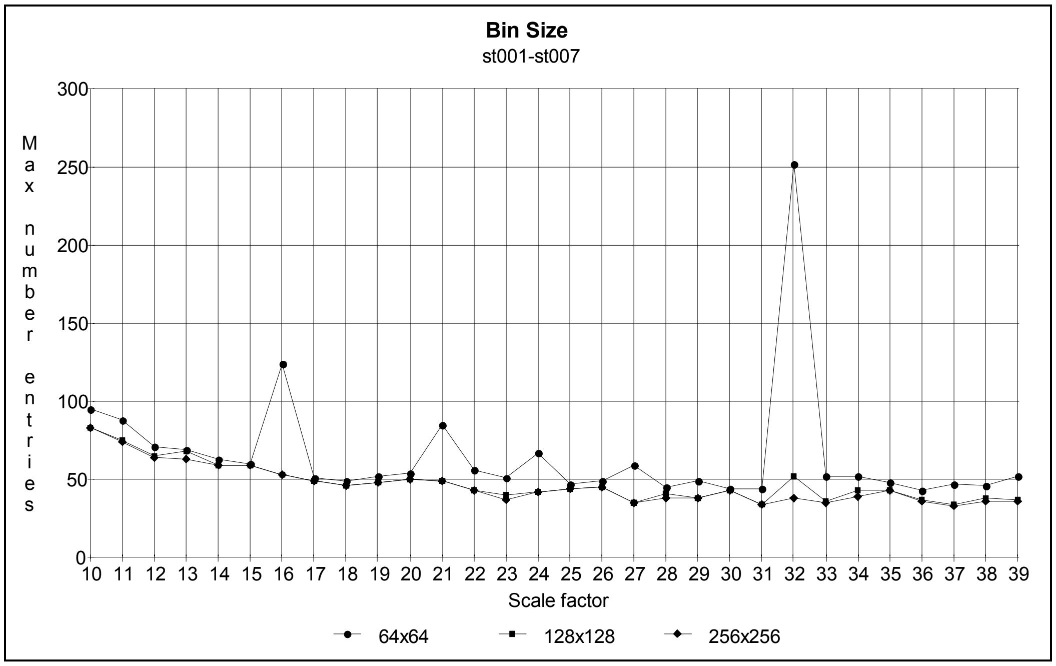


Figure 6-7 Hash table distribution with variant scale factor

Table 6-2 shows the affect on memory requirement, memory utilisation and probe time for different virtualisation ratios. Absolute fill is the maximum number of entries in a bin and Percent fill is the percentage of the total possible entries in a bin. Using a scale factor of 28, and the data for the seven stable state models instances of the 3D model chosen, the maximum number of entries in a bin was similar for virtualisation values of 64, 128 and 256. The hash table data space is $N \times N$ where N is the virtualisation value. The number of bins containing data did increase with an increase in N and timings for one probe did decrease slightly when the virtualisation value was increased from 64 to 128. However, the percentage of bins with data decreased significantly, where only 7% of the bins in the hash table contained any data, and the timing did not decrease with a virtualisation of 256.

Virtualisation (N)	Maximum Bin size	Absolute Fill	Percent Fill	Probe Time (sec)
64	45	2720	66%	4.80
128	41	4189	26%	4.56
256	38	4665	7%	4.60

Table 6-2 Effect of changing Virtualisation ratio

Figure 6-8 shows the absolute and percentage usage of the hash bins with different virtualisation ratios for hash table storage. As the ratio increases, the data is distributed more over the space available allowing for a lower maximum amount of data in each bin. This, however, is at a memory cost, and the percentage utilisation of the bin space decreases. The objective is to distribute the data as best as possible within the finite memory of the system. As a consequence, the virtualisation chosen for this implementation was set to 128 which gave almost optimum distribution without being to wasteful on resources. size

Table 6-3 shows the effect of different virtualisation numbers on the number of recognition instances or votes for a model instance after a probe. This value is compared to the threshold and indicates the existence of a model instance. The number of potential recognition instances is *p-2* where *p* is the number of model points. For the data shown, the number of model points was 12 thus the number of potential recognition instances was 10. When *N* is small, the quantisation is such that there are less bins and so more data must be placed into each bin. This is shown in Table 6-2 where the maximum bin size is 45 with a virtualisation of 64 as compared to 41 at 128. As a consequence, more model-basis sets are placed into the one bin. In the recognition phase, this causes a larger number of potential false hits and an increase in PE contention causing an increase in serialisation. Thus, for a virtualisation ratio of 64, there are an excessive number of false votes. This causes longer processing times due to the need for extra processing to cater for the larger number of instances above the threshold, and a greater demand for a verification stage to filter out the incorrect instance recognitions.

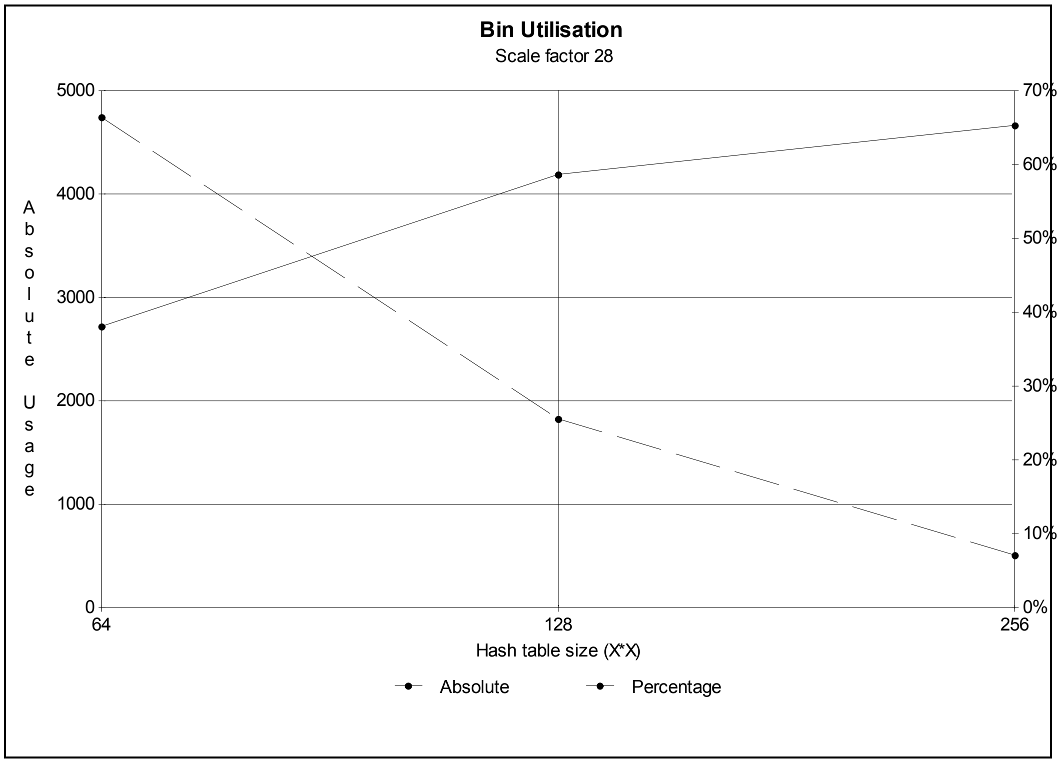


Figure 6-8 Bin utilisation with variant hash table

Without virtualisation, the data space for the DECmpp 12000 is 2K (64×32). Virtualisation of 16K (N=128) gives the correct number (10) of potential recognition instances or bin accesses for a model with 12 feature points. At 4K (N=64) there is an increase in the number of potential recognition instances and an increase in the timing due to the increase in the number of collisions.

Virtualisation (N)	Instances
64	14
128	10
256	10

Table 6-3 Recognition instances with variant virtualisation

For each model instance in the database, there is a memory requirement based upon the number of feature points. An increase in the number of model instances requires a linear increase in the memory required assuming they have the same number of feature points. The number of features in the models will also affect the amount of memory required. The database is a redundant description of each model based on the number of feature points. All permutations of feature points are used, so an increase in the number of feature points for a model requires an exponential increase in the memory requirement. Thus, to ensure a minimal number of entries in each bin, the number of virtual processors needs to be considered.

The virtualisation of the processor array to store the hash table is implemented using a 1D Cut and Stack method. The method is used so that no consecutive bin is located on the same PE. It is recommended (DECmpp 12000, 1992) for load balancing where the activity is localised and gives better performance because the PEs can share the work more equitably.

The processing of the hash data produces a distribution that can be described as non-optimal, with data peaking about boundary areas. Figure 6-9 shows how the data is distributed when placed in an array. Although there is a reasonable degree of distribution over the array, there are still instances of high position data content.

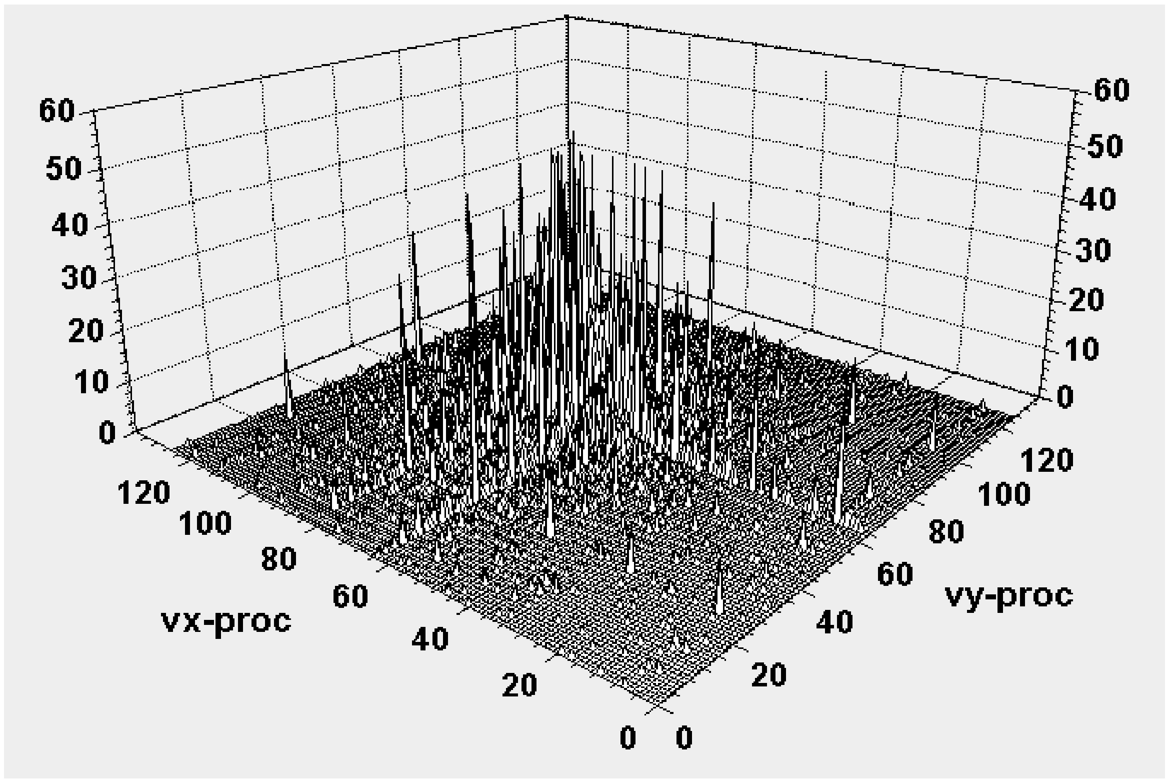


Figure 6-9 Distribution of Hash data (non parallel)

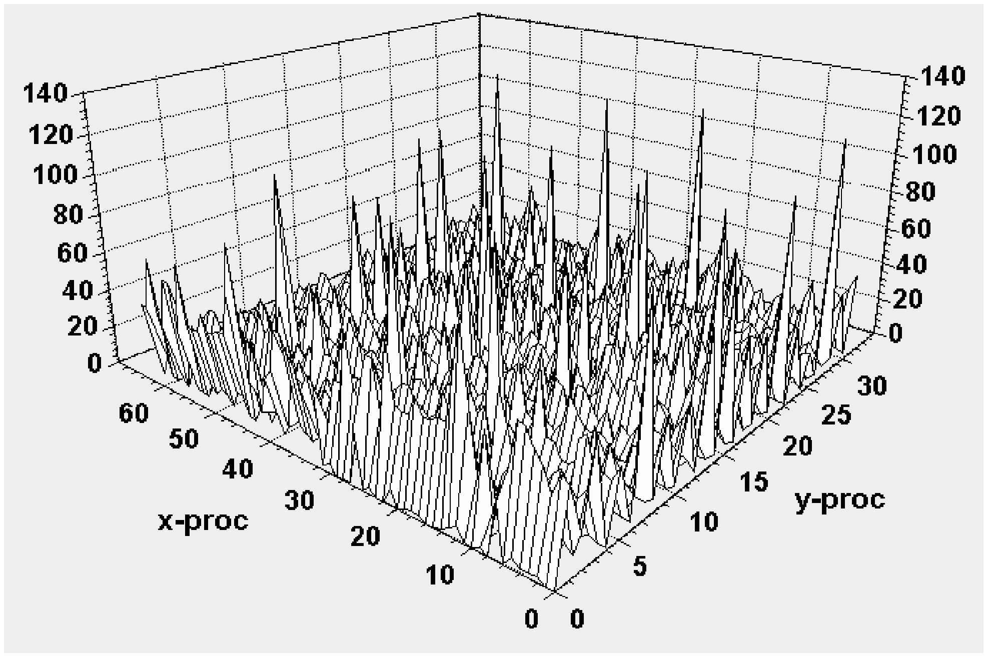


Figure 6-10 Hash Table Distribution - Cut and Stack

Figure 6-10 and Figure 6-11 show the distribution of the hash table over the processor array using a 1 dimensional Cut and Stack method as opposed to a 1 dimensional Hierarchical method. Comparing each figure, the Cut and Stack distribution produced a more even spread of data over the real processor array whereas the Hierarchical distribution tended to concentrate data.

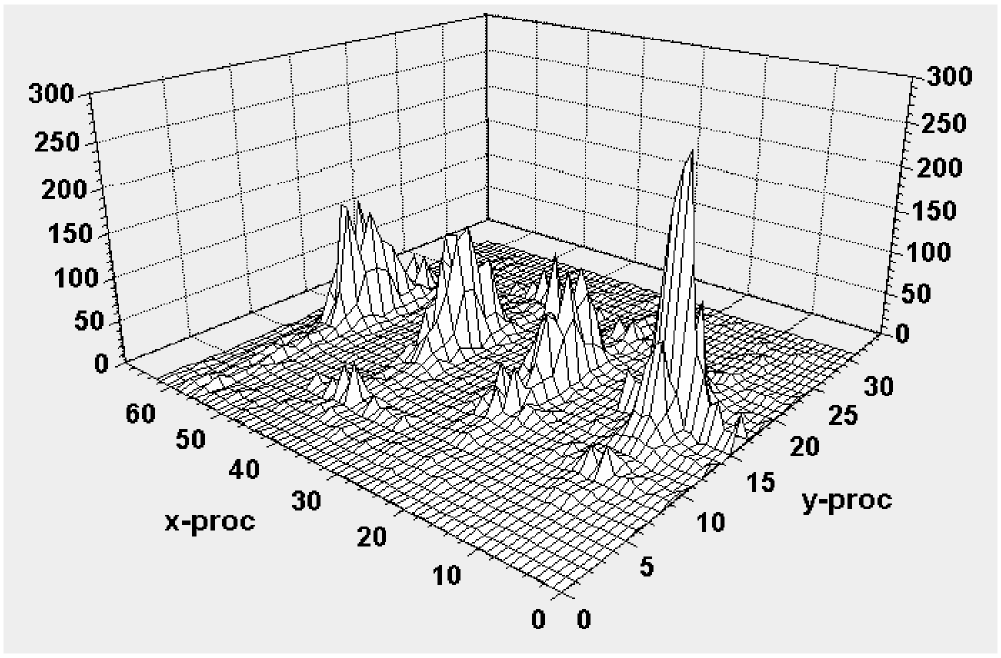


Figure 6-11 Hash Table Distribution - Hierarchical

Algorithm 1 is used to load the hash data from disk onto the processor array. It does this by loading one layer or bin at a time on every processor in parallel. As the number of virtual processors is a factor times the number of PEs, each PE has that many layers and is set with MAXHASHLAYERS. Each layer is loaded onto the PE in sequence until the entire hash table is loaded.

```
OPEN hash data file for reading
For each PE
  FOR layer <- 1 to MAXHASHLAYER
      Parallel read hash_table[layer]
  ENDFOR
```

Algorithm 1 Parallel Load of Hash Table

Ideally, the number of entries at each hash location would be the same. The distribution of the hash entries in the hash bins has been found to be based upon a Gaussian distribution (Rigoustos & Hummel, 1992). In (Rigoustos & Hummel, 1992 and Rigoutsos and Hummel, 1991), the equalisation technique of applying the function $f = (1 - e\frac{-u^2 + v^2}{3\sigma^2}, a\tan 2(v, u))$ is used to uniformly redistribute the hash table data such that each hash space has nearly the same length. With scaling and a modulo hashing function, the hash table data is relatively uniformly distributed without the expense of the redistribution function. The maximum number of entries is dependant on the number of models being considered and the number of feature points for each model. For the implementation of 7 model instances with as much as 16 feature points, the maximum entries in each bin is preset. For a virtualisation value of 128 and a scale factor set at 27, the maximum depth required was found to be 41.

Due to the virtualisation method used and the processes required for passing data using the global router, the bins are implemented as an array on each virtual processor, the index for which must be stored on the PE containing the array. Stored with this is a value which indicates the number of entries in the bin on that virtual processor. Under non-parallel circumstances, the relevant index would be calculated and then used to access the array. However, the process is different under a parallel system as the index is calculated on the PE of the first basepoint pair, and that is passed using the global router to the PE containing that portion of the hash table. The hash table is accessed and the data contained in the bin passed back to the PE of the first basepoint pair and used to increment appropriate accumulators in its memory.

On the DECmpp 12000, which is a SIMD system, each PE carries out one instruction at a time. When serialisation take place, all processors must wait until all others have finished the process. In this case, when more than one PE needs to access the same PE containing a portion of the hash table, all active PEs must wait until the process is complete. As the process of hash bin access requires more than one step, any other active PE requiring access to the same bin tries to do the same thing. All PEs then queue at each step until they have all completed that step. Data passed by previous PEs to the PE with the required portion of the hash table is lost except that from the last active PE. To overcome this, all PEs requiring access to the same PE must wait until each active PE completes all of the steps necessary for recognition. The following algorithm achieves this by setting a variable called *activSet* which is set to 0 for all PEs except the active PEs. The active PEs then queue for the common process and only the currently connected PE is allowed to continue. When that PE has completed the required task, it is deactivated by setting the activSet variable to 0. The next active PE then connects and the process continues until all active processes have completed the task. In this situation, a degree of serialisation takes place. It should be noted however, that there may be many active PEs and only a few are trying to access the same PE. Only the processes on those active PEs that try to access the same PE become serialised. Also, there may be many different groups of active PEs undergoing some degree of serialisation.

```
For each active PE (PE1)
  Calculate index to Virtual PE
  Calculate PE number (PE3)
  all activSet <- 0 on all PEs
  activSet <- 1 on all PE1
  WHILE activSet = 1
      IF connected to the required PE (PE3)
        Pass index to PE3 to get virtual processor
        k <- bin depth from PE3
        FOR i <- 0 to k-1 do
          Pass i to PE3
          Pass model data from bin at i to PE1
          Increment counter for model/basepoint
        ENDFOR
        activSet <- 0
      ENDIF
  ENDWHILE
```

Figure 6-12 Algorithm for Multiple Processor access handling

The algorithm in Figure 6-1 handles the above structure. The index to the virtual processor and the PE number containing the portion of the hash table are calculated. All PEs are turned off by setting the variable *activSet* to 0. Currently active PEs are turned back on. The WHILE loop ensures that only active PEs take part in the process which will eventually turn off all the PEs. If the active PE is connected to the required PE, data from the array will be available. PEs requiring access to the data on the same processor take turns in acquiring the data and turn off once this is accomplished. It is worth repeating that serialisation only occurs when PEs require data from the same PE, and parallel access between other PEs take place in parallel.

6.6 Data accumulation and the indication of model presence

As any PE can potentially contain the feature used to initiate a basepoint, it also can be the particular PE that processes the correct model instance and hence determine the existence of one or more models in the image. That being the case, each PE has an accumulator on it which there is a 3D array representing all (model basepoint-pairs). During the processing of the third point, all bins accessed may have one or more instances of (model, basepoint-pair) data and each will increment the value at this location of the array on that PE. The accumulator will only be incremented with data relevant to that PE because of the hierarchical virtualisation of image data, and the algorithm allows access only to one virtual processor on any one PE at a time.

This allows all active PEs to simultaneously determine and verify the existence of models within the image.

Once all image points have been processed for a given probe, each active PE compares the values in each accumulator to a threshold value. If the total is above the predetermined value, the model with basepoint-pair data at that image location is hypothesised and the (model, basepoint-pair) data is referred to the verification stage for further processing. A major problem that arrises with trying to determine a variety of models is the number of feature points in each. The threshold T is set as a function of the number of feature points n, the number of missing or misplaced image points m and some other factor f that allows for noise, point shift and extraneous points in the image that cause more or less bin access and so affect the final tally. The function is $T = n - 2 - m \pm f$ and T must be determined for each model. This is not so much of a problem with the model data containing 16 feature points but become a significant problem when the model instance has only 6 feature points as in st005 (See Figure 6-1).

6.7 Verification

In an ideal image, the image data after processing would be a rotated, scaled and translated version of the CAD data. This, however, is not always the case: the returned image points are possibly displaced and missing, and the image has a degree of noise. As indicated in the chapter 5, for these cases, a verification stage is necessary.

Also, one of the major attributes of the Geometric Hashing method is the ability to cope with a degree of occlusion. This means that by lowering the threshold for a particular model instance, part of the model is sufficient to determine the existence of that model in the image. Closer inspection of the stable state model data will indicate a further problem with trying to determine the existence of any one of these model instances. Many of the instances could be considered as subsets or supersets of other instances, other instances with some of the feature points missing, or instances with noise points.

To overcome some of these difficulties, a double verification stage using line data and distance transforms of the image and models are used. When a model is proposed, the line data for the model is used to trace over the image distance transform to determine if there is a potential fit (See Appendix 2). If this is successful, the image line data is then used to trace over the specific model distance transform. If there is a positive indication from this process, it is concluded that the particular model is in the image at the location given by the basepoint pairs, and has a size and orientation as proposed in the recognition phase.

6.8 Results

6.8.1 Identification and verification

Table 6-4 shows a typical set of the results for an image. It is for model instance 1 which contains 14 feature points, none of which were occluded. This image is a good representation of the model to be recognised in that it has a relatively high number of feature points as compared to some of the other instances. The shaded region indicates unrecognised model basepoint instances. *M* indicates the proposed model instance, *BP1* and *BP2* indicate the proposed basepoint pair from model data and *BP1 Coord* and *BP2 Coord* indicated the position of the relative basepoints in the image. *Count* is that value in the accumulator after voting takes place that is above the threshold for that model. Note that all model instances have different threshold levels based upon the number of feature points in the model instance (eg model instance 4), and those values falling below the threshold have been excluded. *Verify* is the value obtained when the model line data is compared to the image chamfer file and must be close to zero to allow verification. A threshold is also set for this and any data for a model instance below the threshold is passed to the confirmation routine where the image line data is compared to the particular model chamfer file. *Confirm* is a value that must be close to zero to indicate a close fit. Low values for *Verify* and *Confirm* indicate the presence of the model instance at an orientation determined by the basepoint pair. Thus the model, instance can be confirmed. The empty rows indicate the completion of a probe. It must be noted that not all data is presented in the table as many model instance and orientations are presented to the verification stage.

M	BP1	BP2	BP1 Coord	BP 2 Coord	Count	Verify	Confirm
1	12	3	(45 115)	(222 115)	24	8	0
1	14	3	(45 115)	(222 115)	14	16	
4	0	4	(45 115)	(222 115)	7	167	
6	12	9	(45 115)	(222 115)	12	69	

1	3	1	(224 113)	(45 115)	13	21	
1	3	12	(224 113)	(45 115)	24	7	0
1	3	14	(224 113)	(45 115)	13	16	
6	9	12	(224 113)	(45 115)	13	72	
1	5	13	(216 67)	(63 69)	13	11	
1	6	13	(216 67)	(63 69)	19	5	0
1	8	12	(152 113)	(45 115)	13	10	
1	1	5	(0 58)	(222 59)	20	7	0
1	1	6	(0 58)	(222 59)	21	12	
1	0	1	(249 93)	(27 96)	14	18	
1	0	14	(249 93)	(27 96)	24	7	0
1	2	1	(249 93)	(27 96)	14	21	
1	2	14	(249 93)	(27 96)	20	13	
2	8	1	(249 93)	(27 96)	9	59	
4	1	7	(249 93)	(27 96)	7	114	

5	0	5	(249 93)	(27 96)	4	38	
5	5	0	(249 93)	(27 96)	4	37	
6	4	13	(249 93)	(27 96)	12	134	

Table 6-4 Results - Image of model instance 1

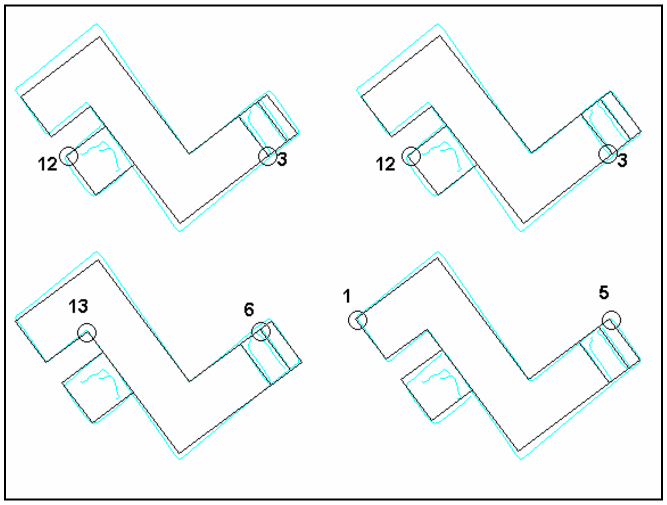


Figure 6-13 Results of recognition

Figure 6-13 shows the position of the model instance, and superimposed on these are the position of the model as recognised in the first four confirmation outcomes. The circled points are the basepoint pair, selected during recognition, that were used for the transformation and subsequent vote accumulation. As can be seen from the figure, the model instance is correct and the relative positions are reasonably accurate given errors in aspect ratio, vertex position, noise etc.

6.8.2 Speed of the algorithm

As indicated in chapter 4, the recognition phase is capable of relatively fast recognition of model instances, for features extracted from an image and subject to errors such as noise. Timings for the first probe was 4.7 seconds and a further 8.0 seconds for the second. Given that the PEs are virtualised, this can be equated to less than less than 0.6 seconds a probe and a further 0.4 seconds for verification on a 64k machine for which virtualisation would be necessary.

The first verification of recognition took place during the second probe. Without verification, one probe takes 4.7 seconds, but with the extra burden of verification, this increases to 8.0 seconds. There is no guarantee that recognition will take place within any set of probes, but as many probes occur simultaneously, the chances of early recognition is high.

6.9 Conclusion

The system is robust and capable of treating image data sourced from video capture. It can cope with the detection of models with a different number of features even where the number of features is almost such that the threshold needs to be set at zero. In this case, threshold t=2 when number of model feature point n=4 using t=n-2.

It can also recognise more than one model instance at a time as all processors independently take part in recognition. As well, each processor has the potential to recognise more than one model instance in a probe. Occluded models can be recognised, as well as models that are subset and superset to another model in the data base.

Although timing initially does not appear to be fast, when it is considered that all processors can carry out independent probes concurrently, and that the computer used had only 2k processors hence requiring a high degree of virtualisation, the timing required to recognise a model instance in an image is better than other parallel implementations.

7. Conclusion

In this thesis, it has been demonstrated that it is viable to implement Geometric Hashing as a means to implement model based object recognition in a single instruction multiple data (SIMD) parallel environment. The method has shown that it is possible to recognise a 2D model instance of a 3D model in real time using the DECmpp 12000 massively parallel super computer in a time of less than 0.5 os a second (relative to virtualisation).

Important issues discussed in relation to this were:

- General application of the Geometric Hashing algorithm on a single processor system.
- Determination of the appropriate size of the hash table.
- Use of scaling and modulo arithmetic to disperse the hash table data equitably over the hash table array.
- Application of error analysis to allow for maximum recognition in the presence of noise, spurious data and occlusion.
- Determination of the vote threshold and the implications of this in terms of recognition. This includes incorrect recognition as a consequence of low threshold and limited recognition as a consequence of the threshold being to high.
- Automatic determination of the optimal scale for hash table data distribution.
- Virtualisation and placement of the image on the processor array to affect *in place* processing of data to allow optimal data access to limit serialisation.
- Virtualisation and placement of the hash table on the processor array to affect minimal data on individual processor and provide maximum distribution of hash data over the hash table.
- Verification of the proposed model instance in view of instances being similar to, superset to, and subset of other model instances. This was particularly difficult in view of the minimal number of feature points in some of the model instances.

As well, a general overview of the systems for model based object recognition over a period of 15 years has been given. This has clearly demonstrated the impetus to the development of Geometric Hashing as a significant development in model based object recognition. In particular, the potential for parallel implementation has been discussed, and the work of other authors using the CM-5 Connection machine and the Maspar have been detailed. An important aspect of this work is the system's ability to recognise more than one instance of a model in one probe where the instance can be of the same model instance or another, separate model instance.

In the light of this research, there are some specific future directions that need to be explored. These are:

- The parallel implementation of preprocessing of the image data in place.
- The parallel implementation of the distance transform used in the verification stage in parallel.
- An investigation into the speed differential in processing the image data as an indexed list as opposed to *in place* processing. This would include the timing to unload and reload the PEs with the image data.
- The recognition of a true 3D model as a 3D image as opposed to using 2D stable state model instance to represent the 3D model.

8. Appendix I Preprocessing of the Image data

All software associated with the preprocessing of images is available from Curtin University, Department of Computing Science.

The preprocessing of the image involved

1. Capture the image as a 512×512 256 grey scale image (image.raw).
2. Apply the Canny Edge detector and create a magnitude file (image.mag).
3. Create a text file of points to represent the line data using **linknew** (image.lnk)
4. Extract out the image data and modify to create a 256×256 image list (image256.lnk). Better resolution of edge detection was obtained by applying the edge detector to the 512×512 image rather than beginning with a 256×256 image.
5. Create a set of end points of lines using **line** (image256.line).
6. Reduce the set of points so as those with interpoint distances greater than a threshold (30 pixels) are considered.
7. Create a 256×256 binary image consisting of feature points and load this onto the processor array.

9. Appendix II Verification

A distance transform of the model data and the image used for recognition were created. These files were known as chamfer files. They were created as follows.

1. A line image was created from the data files. This was a file consisting of lines joining the features used in recognition.

2. The distance transform program ‘chamfer34’ converted the line data to a chamfer file, as depicted in Figure 9-1 for model instance 1 and for image 1.

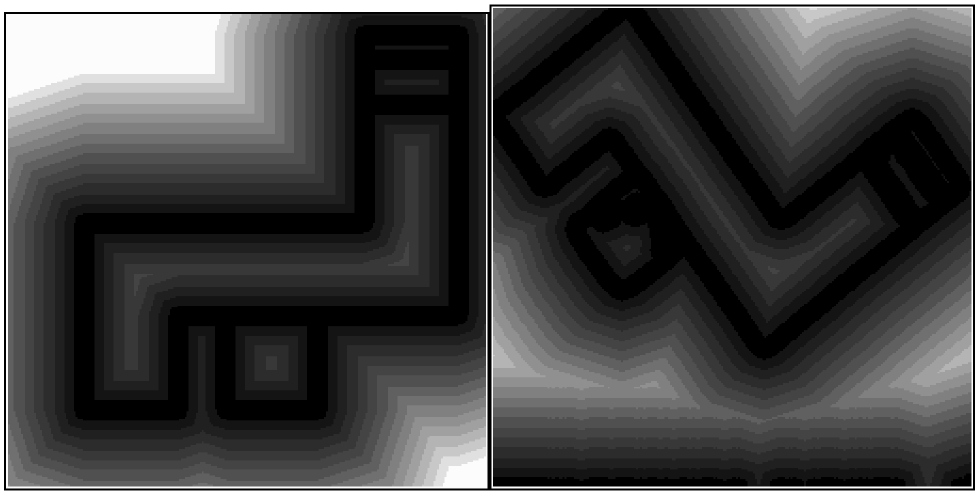


Figure 9-1 Distance transforms for model 1 and Image 1

In the verification stage, a proposed model instance was compared to the image distance transform. A line draw algorithm used the model data and the transformation provided by the recognition algorithm to draw the model in image space, and accumulate a value representative of the distance from the true image. The verification process required an accumulated value to be close to zero. The reverse process also was carried out in the event that the returned value was below a threshold. A line draw algorithm used the image line data and the transformation provided by the recognition algorithm draw the image in the proposed space, and accumulate a value representative of the distance from the true image. The confirmation process required an accumulated value to be close to zero also. This recognition and confirmation system was necessary as some of the model instance could be considered as subsets of or super set to other model instances. As well as occlusion being considered, this was the only sure mechanism that would allow for the identification of any model instance.

10. Bibliography

Bourdon, O. & Medioni, G. 1990, *Object Recognition Using Geometric Hashing on the Connection Machine,* Proceedings of the International Conference on Pattern Recognition, pp 596-600.

Birk, J.R., Kelley, R.B. & Martins, H. 1981, *An Orienting robot for feeding workpieces stored in bins,* IEEE Trans Syst Man Cybern. 11 (2) pp 151-160.

Bamieh, B. & De Figueiredo R.J.P. 1986, *A general moment invariants/attribute-graph method for three dimensional object recognition from a single image*, IEEE J. Robotics Automation, vol 2 (1) pp 31-41.

Bolles, R.C. & Cain, R.A. 1982, *Recognising and locating partially visible objects. The local focus-feature method*, Int. J. Robotics Research ,1 (3) pp 57-82.

Besl, P. & Jain, R. 1985, *Three Dimensional Object Recognition,* ACM Computing Surveys, 17 (1) pp 75-154.

Chin, R. & Dyer, C. 1986, *Model Based Recognition in Robot Vision,* ACM Computing Surveys, 18 (1) pp 67-108.

Costa, M.S. Haralick, R.M. & Shapiro, L.G. 1992, *Optimal affine-invariant matching: performance characterization,* SPIE Vol 1662.

Date C.J. 1990, *Database Systems,* 1, 5th Ed, Addison-Wesley.

DECmpp 12000, 1992, *DECmpp Parallel programming Environment Tutorial,* Digital Equipment Corporation.

Farrell, C. & Kieronska, D. 1992, *On Implementation of Automatic Virtualisation for Parallel Algorithms on SIMD Architectures,* Technical Report No 17, School of Computing Curtin University of Technology Western Australia.

Fischer, D. Nussinov, R &, Wolfsen, H.J. 1992 *3-D Substructure Matching in Protein Molecules* Procs of Conf. Combinatorial Pattern Matching.

Gavrila, D.M. & Groen, F.C.A. 1992, *3D Object Recognition from 2D Images using Geometric Hashing,* Pattern Recognition Letters 13 pp 263-278.

Gleason, G. & Agin, G.J. 1979, *A modular vision system for sensor-controlled manipulation and inspection,* Proceedings of 9th International Symposium on Industrial Robots, pp 57-70.

Gonzales, R.C & Wintz, P. 1987, *Digital Image Processing,* 2nd Ed Addison-Wesley.

Grimson, W.E.L. 1990, *Object Recognition by Computer: The role of Geometric Constraints*, MIT Press, Cambridge, Mass.

Grimson, W. & Huttenlocher, D. 1990, *On the Sensitivity of Geometric Hashing,* IEEE International Conference on Computer Vision at Osaka.

Huttenlocher, D.P. & Ullman, S. 1987, *Object Recognition using Allignment*. Proc. of the 1st International Conference on Computer Vision, pp 102-111.

Huttenlocher, D.P. & Ullman, S. 1988, Recognising Solid Objects by Alignment, Proc. DARPA Image Understanding Workshop, Morgan Kaufman Publishers: San Mateo, CA, pp1114-1124.

Huttenlocher, D.P. & Ullman, S. 1990, Recognizing Solid Objects by Alignment with an Image, Int. Journal of Computer Vision 5(2) pp195-212.

Holland, S.W. Rossol, L & Ward, M.R. 1979, 'CONSIGHT-1*:* A vision controlled robot system for transferring parts from belt conveyors' in *Computer Vision and Sensor-Based Robots* eds Dodd & Rossol, Plenum, New York pp 81-97

Hong, J. & Wolfsen H.J. 1988, *An Improved Model-Based Matching Method using Footprints* Proc. of ICPR Rome.

Illingworth, J. & Kittler, J. 1988 *A Survey of the Hough Transforms,* Academic Press.

Kelley, R.B. Birk, J.R. Martins, H.A.S. & Tellar, R. 1982*, A robot system which acquires cylindrical workpieces from bins,* IEEE Trans. Syst. Man Cybern 12 (2) pp 204-213.

Kelley, R.B. Martins, H.A.S. Birk, J.R. & Dessimoz, J.D. 1983, *Three vision algorithms for acquiring workpieces from bins*, Proc. IEEE 71 (7) 803-820.

Khokhar, A.A. Prasanna, V.K. & Kim, H.J. 1993, *Scalable Geometric Hashing on Maspar Machines,* CVPR

Kingspor. F. Lohofer, D. & Rottke, T. 1992, *Parallel Searching for 3D-Objects*, Joint International Conference on Vector and Parallel Processing, 2nd Ed.

Lamden, J. & Wolfsen H.J. 1991, *On the Error Analysis of 'Geometric Hashing"* IEEE Computer Society conference on Computer Vision and Pattern Recognition

Lamden, J. & Wolfsen, H.J. 1988, *Geometric Hashing: A General and Efficient Model-based Recognition Scheme,* IEEE Second International Conference on Computer Vision.

Lamden, Y. Schwartz, & J. Wolfsen, H.J. 1990, *Affine Invariant Model-Based Recognition,* IEEE Transactions on Robotics and Automation, 6(5).

Lamden, Y. Schwartz, J. & Wolfsen, H.J. 1988, *Object Recognition by Affine Invariant Matching,* Computer Society Conference on Computer Vision and Pattern Recognition.

Leishman, M. West, G.A.W. & Vankatesh, S. 1993, *Implementing Geometric Hashing on the DECmpp 12000*, Proc. DICTA-93 Australian Pattern Recognition Society.

Leishman, M. West, G.A.W. & Vankatesh, S. 1995, *Geometric hashing on SIMD Architectures*, Proceedings of the Australasian Conference on Parallel and Real-Time Systems pp279-286.

Lindley, C.A. 1991, *Practical Image Processing in C,* Wiley.

Maurer, W. & Lewis T. 1975, *Hash Table Methods,* ACM Computer Surveys 7(1).

Mauler, W.A.D. 1992, *Hash Table Methods,* Proceeding of the SPIE Vol. 1069.

Murdoch, D.C. 1970, *Linear Algebra,* Wiley.

Moukas, P. West, G.A.W. 1988, *Instrumentation for Computer Integrated Visual Inspection of 3-Dimensional Industrial Components*, Proceedings IMEKO-88 Huston Texas pp423-435.

Newman, W.M. & Sproul, R.F. 1979, *Principles of Interactive Computer Graphics*, 2nd Ed. McGraw-Hill.

Perkins, W.A. 1978, *A model-based vision system for industrial parts*, IEEE Trans. Computer 27(2) pp126-143.

Perkins, W.A. 1980, A*rea segmentation of images using edge points,* IEEE Trans. Pattern Anal. Mach. Intell. 2(1) pp8-15.

Perrott, C.G. & Hamey, L.G. 1991, *Object Recognition, A Survey of the Literature,* Macquarie Computing Reports.

Prasanna, V.K. & Wang, C. 1993, *Scalable Data Parallel Object Recognition using Geometric Hashing on CM-5*, Technical Report, Dept. of EE Systems, USC.

Press, W.H. Teukoisky, S.A. Vetterling, W.T. & Flannery, B.P. 1992, *Numerical Recipes in C The Art of Scientific Computing,* 2nd Ed. Cambridge Uni. Press.

Rigoustos, I.; & Hummel, R. 1992, *Massively parallel model matching: Geometric Hashing on the Connection Machine* Computer, 25(2) pp33-42.

Rigoutsos, I. Hummel, R. 1991, *Implementation of Geometric Hashing on the Connection Machine,* Workshop on Direction in Automatic CAD Based Vision, MAUI.

Rigoutsos, & I. Hummel, R. 1992, *Massively Parallel Model Matching Geometric Hashing on the Connection Machine,* Computer, pp33-42.

Sarachik, K. & Grimson W. 1993, *Gaussian Error Models for Object recognition,* Conference on Computer Vision and Pattern Recognition

Sedgewick, R. 1988, *Algorithms.* (2nd Ed.) Adison Wesley Publishing.

Shirai, Y. 1975, *Edge finding, segmentation of edges and recognition of complex objects,* Proc. 4th International Joint Conference on Artificial Intelligence, pp674-681

Shirai, Y. 1978, 'Recognition of real world objects using edge cues', *Computer Vision Systems, eds* Hanson & Riseman, Academic Press New York pp353-362

Stockman, G.C. Kopstein, K. & Benett, S. 1982, *Matching images to models for recognition and object detection via clustering*, IEEE Trans. Pattern Annal. Mach. Intell. 4(3) pp229-241.

Suetens, P. Fua, P. & Hanson, A.J. 1992, *Computational Strategies for Object Recognition.* ACM Computing Surveys 24(1).

Tsai, F. 1994, *Geometric Hashing with line features,* Pattern Recognition 27 pp377-389.

Turbo C++, 1992, *User's Guide,* Borland.

Wolfsen, H.J. 1990, *Model Based Object Recognition by Geometric Hashing,* Proceeding of the European Conference on Computer Vision, France, pp 526-536.

Yachida, M. & Tsuji, 1977, *A versatile machine vision system for complex industrial parts,* IEEE Trans. Computer 26(9) pp882-894.

Yachida, M. & Tsuji, S. 1980, *Industrial computer vision in Japan.* Computer 13(5) pp50-63.