Parallel Process Techniques for 3D Model-Based Vision

Usoh, Martin

692

UNIVERSITY OF LONDON

Parallel Process Techniques for 3D Model-Based Vision

by

Martin Usoh

A thesis submitted in accordance with the requirements for the degree of

Doctor of Philosophy

Queen Mary and Westfield College,
Department of Computer Science,
Mile End Road,
London, E1 4NS.

QMW Vision Group

December 1991

Abstract

Visualisation and closed-form testing of algorithms are vitally important in the development of model-based vision systems. The general idea is that, if we can specify the models in their own coordinate frames, their subsequent transformation (in terms of their translation, rotation and scaling), and the camera geometry, we can then generate images for which the results of our visual recognition and localisation algorithms are known *a priori*. The fully tested and validated models and algorithms can then be used to analyse scenes where the details of the identity, position, orientation, scale and motion are not known, but where we have a range of models with which to form our interpretation of the scene, that is, model-based vision.

The work we describe is concerned with the parallel implementation of techniques involved in 3D model matching. Essentially, these are carried out on the AMT DAP (Distributed Array of Processors), a fine grain, highly parallel array computer. It provides a natural solution for many problems which cannot easily be solved on conventional computers with associated serial languages due to their computational complexity.

For the closed-form testing of our matching algorithms, we implement a generalised parallel ray tracer which can render Constructive Solid Geometry models and incorporates spatial subdivision and anti-aliasing for the fast generation of high resolution data images.

The geometrical model matching approach of Grimson and Lozano-Pérez has been used for the recognition of polyhedral objects and scenes containing planar curves. We describe a method using the data edges and surface normals of planar curves recovered using structure from stereo. We use constraint-based pruning of an interpretation tree to generate the feasible interpretations which are then validated. The problem is mapped onto the DAP architecture which requires the reformulation of the matching to use logical matrices rather than trees. Results have been obtained from closed-form testing of curved surfaces and projections of developable surfaces. The subgraph matching to deal with large models, parallel data validation and refinement, and dealing with spurious data using the bin are then elaborated with results from the closed-form testing of models.

Acknowledgements

In completing this research I am forever grateful to Dr Hilary Buxton who has proved a first rate supervisor by still finding time to discuss problems when her schedule has been very tight. Also, for her suggestions, support, patience, and seemingly endless proof-reading, I am indebted.

Many thanks also go out to Dr Bernard Buxton for his mathematical support and enlightening conversations.

To Shu Chang and Jan Wysocki of the Vision Group I am grateful for their help and suggestions offered including support on the DAP. Also, to the other members of the Vision Group I wish 'clear sight'.

In originally taking up this PhD[*] I very much wish to say "thank you" to Anna Bara. Without her initial suggestion I may now be wearing a suit and not a metaphorical 'white coat'.

[*] This research project was supported by a Cooperative Award in Science and Engineering (CASE) from the Science and Engineering Research Council in conjunction with British Petroleum, Information Technology Division.

To my parents who have always shown
me love and encouragement.

To my brothers.

To Anna.

Table of Contents

List of Figures

List of Tables

List of Plates

Chapter 1

Introduction and Overview

1.1 Introduction

Computer Vision, as a subfield of Artificial Intelligence, is still in its evolutionary infancy since being pioneered by the work of Roberts in the early 1960s. It is often the case that the phrases "computer recognition" coupled with "artificial intelligence" conjures up visions of intelligent robots and machines which are able to recognise and interact with humans. Behind the scenes of a film production studio, this may be easily achieved. However, in reality, such a system is far from the norm. For example, systems have been produced which can determine or recognise instances of human faces using invariant features [WLT89, CJCM86, Kana77]. However, these are currently not as robust or as fast as if the task was undertaken *by* a human!

The recognition of objects is an essential step towards the development of intelligent autonomous machines. Immediate applications for such systems are apparent. The general class of factory robots are unable to perceive their environment and can only exist in a highly defined, static domain. The problem with the determination of full scene descriptions is the computational bottleneck which exists due to the numerous possible interpretations of a set of recognition features. A possible widening of this *bottleneck* is by the use of a number of sophisticated pruning constraints to reduce the interpretation search space. An alternative, though not a direct method, is the use of a faster machine architectures and, hence, parallel computing. Using this, as reported in this dissertation, means that recognition tasks may be remapped to achieve a faster, more efficient, solution procedure.

The field of Computer Vision includes not only the determination of objects in a scene but also the understanding of the scene or an event in the scene. This may range from locating simple geometric objects to the interpretation of incidents for use in automatic surveillance systems. Model matching is at the lower-level of such complex event understanding systems and is the main area of work discussed in this dissertation. As a subset of recognition, it is the determination of an instance of an object from a scene using an internal model representation.

Such queries may include *"Is model X present in the view?"* and, *"What is the 3D location of model Y?"*. For these to be successfully answered, we require a range of models in a database and a method for determining their presence in an image. They may be represented using low-level features such as 3D edges, points, or vertices, or some combination which may be efficiently computed. From the database, models may be retrieved for comparison with image data, or for unidentified objects, constructed from image data and added back to the database for later matching. As the process is dependent on the use of models, it may, in general, also be classed as model-based.

1.2 Bottom-Up or Top-Down?

In recognising objects, two basic strategies exist - *bottom-up* and *top-down*. The bottom-up approach to recognition is data-driven and is the most common method. This involves the matching of instances of the data to the model to determine higher-level full 3D volumetric or surface based descriptions. Top-down, however, uses knowledge to drive the recognition process from high-level model hypothesis to low-level instantiation. This includes the model primitives being matched to the scene primitives. We note that different combinatorics exist in the two methods. Given m model entities and s data features, m^s possible interpretations exist using the bottom-up approach and s^m interpretations exist using the top-down method. As an examplar, if $m = 10$ and $s = 2$ then for bottom-up matching $10^2 = 100$ interpretations exist and for top-down $2^{10} = 1024$ interpretations are possible. Also, as a data item is not permitted to pair with more than one model item in a single interpretation for bottom-up matching, and *vice versa* using a top-down approach, there exist interpretations in one approach which are not present in the other given that $m \neq s$. Thus, assuming that model entities are labelled alphabetically and data items are labelled numerically, the interpretation

$$\{ (1,a), (2,a), (3,b) \}$$

is not possible using a top-down approach and similarly

$$\{ (1,a), (2,b), (1,c) \}$$

is not valid with bottom-up matching.

With knowledge of the difference in combinatorics, a bottom-up or a top-down approach may be taken. Grimson and Lozano-Pérez [GL-P84*b*] in their work using sparse data obtained from tactile sensors or the like assumed that $s<m$ and grew their scene interpretation tree in a data depthwise manner so that the maximum number of leaf nodes

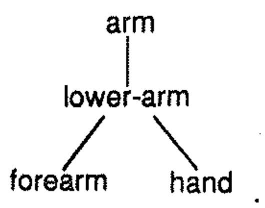
(interpretations) was m^s. Faugeras and Hébert [FH83], however, opted for the top-down tree structure when using data obtained from a laser range-finder. By taking advantage of $s^m<m^s$ for $s>m>2$ they grew the tree in a model feature depthwise manner to constrain the search. However, several paths in the tree may be mutually consistent, and these must be combined to produce a final interpretation. This is best done after a globally consistent transformation has been found for each path (interpretation) and the improvement in combinatorics must outweigh this overhead.

1.3 Classic Early Model-Based Systems

The early literature in the field contains some classic examples of model-based programs which give some idea of what can be achieved. Brooks' ACRONYM is a "comprehensive domain independent model-based system for vision and manipulation related tasks" [Broo83]. It represented a significant advance over earlier matching technologies such as that of Roberts and used a hierarchical description of models in terms of their subparts and interrelationships. Using a top-down approach, high-level descriptions of the data may be matched to high-level descriptions of the model. The use of these high-level descriptions has the advantage that the combinatorics are reduced due to a relatively small number of primitives.

The models in ACRONYM were constructed using generalized cones or cylinders as primitives in the hierarchy. These are defined by a *spine* (a 3D sweep axis), a 2D cross-section, and a *sweeping rule* with which the cross-section sweeps along the spine. However, in this case, they were restricted to straight line and circular arc spines. From the image domain, ACRONYM was able to construct a similar description of the data as for the model using the same rule for generalized cylinders. The edge elements were identified and linked into extended edges with descriptions such as *ribbons* and ellipses, where a ribbon is a 2D analog of a generalized cylinder. Again, these descriptions were restricted, but here, to straight lines and linear sweeping rules. The model and data descriptions were fed into a matching process which used interrelationship constraints between the model and data subparts to drive the search. The system has been applied to several tasks including the identification of stationary airplanes from aerial views. It's performance has been reported as robust to occlusions due to the use of the high-level descriptions. However, the computation of such representations is expensive.

Similar work to Brooks' ACRONYM, but allowing image sequences, is the WALKER model due to Hogg [Hogg83]. This again used a hierarchical approach to determine instances of a walking person from a sequence of camera images. The hierarchy is described with a structure such as



In the choice of modelling approaches, the scheme should be capable of representing major aspects of a person walking such as the flexibility at limbs and the constraints of posture. These limited the mappings of say, an arm to a leg and, thus, pruned out inconsistent associations. Movement constraints were also incorporated by matching the rate of change for variable positional parameters. Thus, the 3D structure of the WALKER model is composed of elliptical cylinders which are constrained indirectly with respect to one another through the model hierarchy. Like ACRONYM, these were recovered from the image data and the positional and posture constraints were used to drive a tree search procedure.

1.4 Objectives

As the thesis title suggests ("Parallel Process Techniques for 3D Model-Based Vision"), the project is concerned with the implementation of parallel methods for tasks in the model-based recognition of three dimensional objects. However, the solutions are also easily applicable to 2D. The main reason for using parallelism for the solution of any task is usually the quest for speed although much image data is intrinsically parallel and parallel solutions can be more satisfying. Parallel architectures have a great speed advantage over uni-processor Von Neumann machines. Such speed of processing may not be necessary in many circumstances but, in the real-time application of a process which requires a vast number of computations, it can be very useful. As this work is concerned with the recognition of real-world objects with a view to real-time applications, the parallelism is essential. We have chosen a Single Instruction Multiple Data (SIMD) architecture for fast, effective computations. In particular, we used, the AMT Distributed Array of Processors (DAP) for the implementations. This is a workstation based parallel array computer with 1024 processors and fast video output for program development.

Our prime objective is to implement a self-contained parallel system which is driven by (non-sensor specific) data in the matching process. In pursuing this, we also propose a means of closed-form testing of the recognition algorithm by the fast generation of image data. This 3D visualisation process is achieved using the technique of 'ray tracing'. The method is very effective as a means of rendering realistic images but suffers greatly from a vast computational overhead. We attempt to reduce this rendering time by a significant factor by mapping the algorithm onto the processors of the DAP. To further reduce the overall processing time, we will adopt a two dimensional spatial subdivision of the image space. This procedure, along with the mapping of the algorithm onto the DAP processors, is then an effective method of reducing the total rendering time as we show by comparison to an equivalent sequential implementation of the technique.

For the recognition system, we will adopt the serial matching paradigm of Grimson and Lozano-Pérez [GL-P84*b*] and its subsequent parallel implementation due to Holder and Buxton [HB89*a*]. This is shown to map well onto the DAP processors with significant speed ups when compared to its serial counterpart. We will improve on the parallel implementation by introducing a new data sorting technique and show how this reduces the search space considerably when compared to the sorting implemented by Holder and Buxton. The recognition process we will describe is bottom-up and uses the 3D edge features of target objects. The edge features are used so that the process may easily be extended from polyhedral objects to objects containing planar curves. We present this extension and show that the adaptation of planar curve surface normals in an edge-based fashion is robust in determining model to sensor space transformation parameters. Finally, in confirming the stability of the system, we present results obtained from closed-form testing of the matching process using data derived from artificial scenes by means of the parallel ray tracer. Being artificial, however, it can only be guaranteed that the system is stable under synthetic conditions. Thus, we introduce a further degree of stringency by testing with data obtained from a real-world environment.

1.5 Thesis Outline

The thesis is organised so as to initially introduce the reader to relevant concepts and related work before detailing the approaches taken. In an attempt to lay down appropriate background and to give a wider view of theories in model-based recognition, we present a selective review in Chapter 2. This puts forward early concepts and ideas with arguments expressed by workers in the field. We discuss the work of Grimson and Lozano-Pérez upon which the research in this dissertation is based. In addition, we discuss adaptations of this method by Murray and Cook and introduce authors who have attempted to remap the algorithm onto parallel architectures. These have included Holder and Buxton on the DAP and Flynn and Harris on the Connection Machine.

Chapter 3 is a second background chapter, here concerned with tools used in this work as well as the idea of parallelism. We present the DAP architecture along with a brief outline of its programming languages. This is necessary as examples and implementation details may be more easily understood by the reader when acquainted with this style of parallelism. We also present the Connection Machine (CM), a fine-grain, massively parallel computer and discuss works in the area of parallel processing for vision and display. The WINSOM, ISOR, and TINA toolsets, all of which are complete systems, are also introduced. These serve as tools for 3D visualisation and recognition and may be used, as here, for three dimensional data recovery. We also introduce basic edge detection algorithms and briefly discuss the Canny edge detector which is implemented in both the ISOR and TINA systems. A basic description of the principles of motion and stereo reconstruction algorithms is also presented.

After Chapter 3, the reader should have gained enough background for the visualisation process which is discussed in Chapter 4. This is the parallel implementation of the ray tracing algorithm due to Whitted [Whit80]. We present this as a generalised ray tracer which includes a Constructive Solid Geometry datatype and an extension for Digital Terrain Maps for the fast generation of realistic images. We show that the DAP is well suited for such a task (especially ray casting) as each ray, being independent of any other, can be mapped onto a processor in the array structure. We also show that a two dimensional spatial subdivision of the image space further improves the image generation time. This is confirmed with experimental results from serial and parallel implementations, and from parallel brute force and parallel subdivision approaches.

In Chapter 5, we investigate the Grimson and Lozano-Pérez matching paradigm and its parallel implementation on the DAP. We discuss the local pruning constraints and derive a new data sorting technique which we show to produce a more effective traversal of the interpretation tree. Verification of hypothesized interpretations are also discussed along with the parallel implementation of the validation process to compute model to sensor space transformations. As a major hindrance to the recognition process, spurious data can lead to the failure of the matcher in returning successful interpretations. To overcome this, the idea of a 'bin' is introduced. We show, through experimental results, that the new sorting technique, when applied to the bin, is more effective than the data sorting adopted in earlier works. Also, through experimental results, we show the accuracy and time efficiencies of the parallel method using comparative tests with a serially implemented version. We discuss the costs of using edges rather than faces as matching features and present a novel procedure for the successful matching of models which are too large to map onto the array processors directly.

An extension of the polyhedral algorithm discussed in Chapter 5 is presented in Chapter 6 with an overview of systems implementing curved object recognition. We discuss the use of the surface normal at the centre of circles and ellipses and their treatment in an edge-based manner for incorporation into the polyhedral matcher. The axis of rotation of cylinders and cones are used as additional edge matching features. We show that these pseudo-edges may be effectively used in edge-based recognition and confirm this through experimental results. We also present an experiment based on a real-world scene in order to show the effectiveness of the system under such conditions.

Chapter 7, the concluding chapter summarises the contributions offered in the preceding chapters. We discuss the advantages and pitfalls of SIMD parallelism and put forward future directions for improving and extending the system.

Chapter 2

Model-Based Recognition: Activities and Advances

2.1 Introduction

In order to recognise an object one must have some prior imagery or knowledge of it. An account of this is given by Gregory [Greg78] in which a man, blind from birth, has his sight restored by a surgical technique. When shown a simple lathe, similar to one he worked on when blind, he could not comprehend its structure. However, when allowed to feel it with his hands he was able to *see* it replying, "Now that I have felt it, I can see". This also shows that the representation of an object for a recognition system may be of many forms. Although, when shown the lathe, the man could see it, he was not able to recognise it as his representation of it (by touch) was different from his new found 'sensors'. This points to the importance of the 'internal model' or representation of an object.

In machine vision, the task of recognising objects and scenes has generally been investigated using the example of human perception. Processing is conducted on several levels; a strategy used for recognising pebbles is generally different from that required to determine a coastline. For a dinner table scene, high-level abstractions are needed to reason about the relationship between the crockery and cutlery, or cutlery to napkins, and more low-level processes undertake the task of distinguishing between the different classes of objects. In machine recognition these low-level processes perform tasks such as edge and feature detection [Cann86, NB86] and scene segmentation [MCB89].

The grouping of features to form perceptual structures is basic to many machine vision applications. Model matching, a subset of recognition, is typically a 'low-level' approach to the recognition problem using low-level features to make high-level abstractions. Much work has been done in this field [GL-P84*b*, MC88, HB89*a*] and avenues of investigation have been wide and varied. A hypothesis that an object is in the scene is generated and then a search is performed to find a combination of orientation and viewpoint which gives a close approximation to the actual image. Matching can converge efficiently if the object has strong features. When dealing with geometrically well defined objects, analytical techniques can be employed. At other times, more pragmatic rules of thumb are used, for example an object might be matched by its colour and image area. Whatever the method, matching is truly model-based: segments of the image are matched against an accurate representation of the model.

The work on object recognition and matching may be split into two classes:

1) Bottom-up - Those that use low-level matching features to make high-level interpretations, e.g. [GL-P84*b*]

2) Top-down - Those that use high-level hypotheses to match to low-level features, e.g. [Hogg83].

The more commonly used strategy, however, is bottom-up. By keeping the features simple, the algorithm itself can be kept simple which increases its efficiency in terms of speed. This speed factor is an essential consideration especially when the system is expected to perform interactively or in real-time. Along with simplifying the algorithm, other methods for reducing the time spent on the problem have been developed [HB89*a*, SRS90, FH85]. These have been by implementation on parallel architectures. However, the most established algorithm is that of Grimson and Lozano-Pérez [GL-P84*b*] originating from work by Gaston and Lozano-Pérez [GL-P84*a*]. This has itself been adopted and extended by other workers in a variety of ways. Although the algorithm is essentially serial, workers such as Holder and Buxton [HB89*a*] have attempted to parallelise it.

In recent years, much of the work on model matching has been on polyhedra with up to six degrees of freedom. However, in the literature, numerous applications to curved surfaces can be found [OS83, FJ90, BCZ90]. In the following sections we will outline the base work, that of Grimson and Lozano-Pérez, and its adaptation. We will also review the recent literature which attempts, or sets out the foundation, for the solution of the problem of curve matching.

2.2 The Problem in Perspective

The problem definition reads thus:

i) Given a representation of a known object, identify and recover it from a scene containing other objects (known or unknown).

ii) Recover its pose, i.e. the rotation and translation that would transform the object from the model world to the scene world.

Or in a more colloquial form:

"Here is an object. Find it and tell me where it is".

For humans, this instruction, along with its solution, is fairly simple. For a machine, however, it is very complex. Given an object, a person would survey it and see if there are any special markings or features to distinguish it before searching for it. If the object is of a simple geometry (according to the individual's perception) or has strong features, this is a trivial task. For a machine, even a simple scene can have a large number of possible interpretations. For a model with m model features and s recovered data features, m^s possible interpretations exist [GL-P84*b*]. We can therefore see the extent of the problem faced by a vision system. According to Faugeras *et al.* [FHPP83] the solution may be divided into three main problem groups. The first is the representation of *a priori* knowledge, i.e. an accurate geometric description of the model world. This is generally very complex as few things are known about ways of representing and organizing the corresponding database. The second problem group is the reconstruction from the sensor's output of a symbolic (and fairly accurate) description of the information necessary for solving the problem. Finally, the output of the first two tasks must be combined to find a solution for the recognition of the object. Separating the original problem in this way allows us to identify potential bottlenecks in the solution. The complexity of each individual task can be defined by a number of parameters such as; 1) the quality and accuracy of the sensor signal, 2) what *a priori* knowledge is available and, 3) how different the objects or observed phenomena are. We can see that problem (2) is mainly a knowledge representation problem, whereas (1) is mostly a signal processing problem, and (3) has a bearing on the search or control strategy problem. By fixing one or several of the task parameters and varying the others in a controlled manner, a methodology for solving the corresponding problems in a variety of situations in each group may be outlined. A number of workers have tried to keep the knowledge representation problem as simple as possible by working with objects with a relatively simple geometric characteristic [SRS90, MCB89]. Also, with the availability of accurate sensing equipment (tactile or laser range-finder), problem (1) becomes easier with a direct consequence in making problem (3) generally simpler.

Representation of the models adopted by different workers have again been varied. Planar objects may be reconstructed from their 3D edges or faces [MB90]. Patterns of tactile contacts have also been taken as a representation scheme [Broi79, OT77]. Shankar *et al.* [SRS90] have shown the feasibility of vertex representations and Faugeras and Hébert [FH83] illustrate how their matching algorithm may be extended to quadrics. Matching approaches have also included clustering in parameter space [MT87, Stoc87], searching a tree structure for corresponding model and image features [GL-P87, MC88, FH86, FJ90], and the direct searching of model-to-image transformations [HU90]. Verifying that the hypothesized matches of data to model features hold under transformation of the model is addressed by [GH90, FH83, HB89*b*]. What now follows is a survey of the approaches taken by different researchers and extensions for a solution to the matching problem.

2.3 Sensing Techniques

The sensing techniques adopted by various workers for the recovery of data features have been diverse. Gaston *et al.* [GL-P84*a*] and Grimson *et al.* [GL-P84*b*] have obtained data using tactile sensing apparatus. The simplest of these tactile devices is a microswitch which is able to detect when the force over a small area exceeds a threshold. An important extension of this is the matrix tactile sensor and, as the name suggests, is composed of an array of sensitive points. A simple form is an array of microswitches. These produce a 2D map which is a function of the pressure distribution over the sensors. A review of tactile sensors may be found in [Harm82] and their designs in [Hill82].

Tactile information is useful for identifying and locating objects in terms of the texture, hardness, temperature, and even slipage of the grasped objects. Such immediate circumstances are when visual information is not available such as in underwater or reduced lighting conditions. Information from a number of tactile sensors may be used to determine which object from amongst a set of known objects has been grasped and to determine the objects position and orientation relative to the hand [GL-P84*a*]. This is still a definition of our original problem. Workers such as Broit [Broi79] and Okada *et al.* [OT77] have relied on the contact patterns of matrix sensors. Their assumptions being that the individual data elements are not repeatable and only their statistical patterns remain stable. The measured statistics are compared to the reference statistics for known models. These, however, are limited to discriminations between a few simple types of objects. A second approach to statistical tactile recognition uses patterns of the positional sensors which are mounted on the fingers of articulated hands which come to rest on the objects. A number of workers have used the joint angles of the fingers as their sensory data [Mari81, OT77]. Gaston and Lozano-Pérez [GL-P84*a*] illustrate that tactile recognition and localization may be achieved without resorting to statistical pattern recognition or global feature-finding. Also, their method uses very sparse tactile data - one point for each sensor.

Visual range sensing has also received a large amount of interest. Jarvis [Jarv83] gives a detailed review of techniques as applicable to vision. Possible methods include edge-based stereo methods [MF81] which provides three-dimensional positions of sparse sets of points in the image. This may be used to reconstruct a surface representation from which surface normal estimations may be made. Laser range-finding techniques, used by Faugeras *et al.*, have shown a fair accuracy in the data reconstruction [FHPP83]. Others have concentrated on techniques of structure from motion [MB90, MCB89] and structure from stereo [PMF85, CPPMF90].

2.4 The Grimson and Lozano-Pérez Matching Paradigm

Many of the recognition algorithms in recent years have adopted a tree search procedure [GL-P84*b*, MC88, HB89*a*, SRS90, FH83]. Many are based on the Grimson and Lozano-Pérez matching paradigm [MC88, HB89*a*]. The algorithm [GL-P84*b*] is non-specific to the sensing technique and is illustrated by analyses of the surfaces of simple polyhedral objects. The aim was to use local measurements of 3D positions and surface normals to identify and locate objects from amongst a set of known objects. The objects are modelled as polyhedra with up to six degrees of freedom relative to the sensor. This was an advance on that of Gaston and Lozano-Pérez [GL-P84*a*] where objects were permitted 3 degrees of freedom (2 translational, 1 rotational). By using local constraints, inconsistent hypothesis about pairings between sensed points and object surfaces can be efficiently discarded. The local constraints chosen act on the distances between faces, angles between face normals, and angles of vectors between sensed points relative to the surface normal. The number of possible interpretations consistent with the constraints is very small relative to the number of potential interpretations. Given s sensed points and n surface normals this would result in n^s potential interpretations [GL-P84*b*]. The local information about sensed points is also used to determine the set of positions and orientations of an object that are consistent with the sensed data. The object is considered unmatched if there are no consistent positions and orientations.

As a key to their matching algorithm, they generate a set of *feasible* interpretations. These are generated by pairing each sensed point with some object surface of the known object. Inconsistent pairings are discarded using local constraints. The recovered interpretations are further tested for global consistency with the surface equations of the model requiring that an interpretation is only legal if it is possible to solve for a rotation and translation that would place each sensed point on the appropriate object surface. The approach is an example of the classic paradigm of AI - *generate and test.*

The range of possible pairings of sensed points and model faces for an object are represented in the form of an interpretation tree (IT). The number of descendants at each node of the tree is equivalent to the number of model faces. The sensed data points are then required to pair onto these branches, one at each level. It is noted that, from the structure of the tree, it is possible for several data points to be paired onto the same model face which means that it deals well with fragmented data. The top-down, model driven approach, however, is not appropriate for such data since, here, several data features cannot pair onto the same model feature.

The local constraints used to prune the IT have been adopted and modified by other workers [MC88, HB89*a*]. Four are used, namely an angle constraint, a distance constraint, a direction constraint, and a triple product constraint (see GL-P84*b* for details). These typically serve to prune out the non-symmetric interpretations of the data. Murray [Murr87] approaches the matching problem as Grimson and Lozano-Pérez. Using 3D information obtained from most visual motion systems, the depth/speed scaling ambiguity means that absolute depth and size is unavailable. He shows that shape data alone, without absolute size, is enough for reducing the search space of the interpretation tree. By modifying the local constraints of Grimson and Lozano-Pérez he shows how this is possible. The angle constraint, however, is identical. The unit vector of the surface normals are taken in pairs to determine the angle between them. The direction constraints of Grimson and Lozano-Pérez are concerned with the range of values for the component of a vector between sensed points on faces i and j in the direction of the sensed normal at i and j. This must intersect the range of components of possible vectors between points on the faces assigned to i and j. This requires the full displacement vector between the points on the faces. Murray gives alternative direction constraints (classified as direction constraints 1, 2, and 3) using unit direction vectors obtained from the normal of face i, the normal of face j, and the unit vector in the direction of their vector product.

Grimson and Lozano-Pérez point out that the efficiency of the IT pruning is improved with presorted data. This ensures that the most effective pruning occurs near the root of the tree. They sorted the data in terms of distance (pairwise separation) with the furthest points encountered first. This, along with the ordering of the local constraints in terms of effectiveness (distance constraint first), greatly improved the pruning procedure as detailed in their results [GL-P84*b*]. Although the distance constraint of Grimson and Lozano-Pérez was not used by Murray, he was able to show a similar result when sorting by pairwise scaled separation. Of course, the loss of absolute distance or range, and its replacement with scaled depth, makes the matching paradigm less effective but not disastrously so. Also, as may be expected, Murray shows that sensing data from the edges rather than the centre of the faces, proved more effective. This is because more extreme measurements of angles and directions are being considered.

Estimation of the rotation by Grimson and Lozano-Pérez is performed using an averaging technique in preference to the least-squares method adopted by Faugeras and Hébert [FH83]. This, they consider as computationally expensive. Murray adopts an equivalent rotation estimation but chooses a least-squares method for estimation of the translation and scale [Murr87]. A modification of the method of Murray to an edge-based system can be found in [MC88]. Here 3D fragmentary segments of polyhedra recovered by structure from motion techniques are acted on using the same local constraints as in the surface-based algorithm. However, an important feature of the constraints was their ability to attach direction signs to the sensed data fragments corresponding to those of the model edges. This proved an overhead during the early stages of matching but progressively increases the power of the constraints and assists in determining the transformation between model and sensor spaces. Again, by presorting the sensed edge fragments by length so that the longest fragment would be encountered first, they found that the overall search space is reduced. In addition, presorting the model edges in a similar fashion increased the likelihood of the search delivering a valid interpretation sooner, although, the overall quasi-exhaustive search time remains unchanged. The combination of several visualisation algorithms (including Murray and Cook) into a complete recognition system (ISOR) can be found in [MCB89]. Using 3D matching cues (obtained from visual motion) from a sequence of images, the system is able to recognise the recovered geometry of a polyhedral object from a database of 3D wireframe models. The difficulties and ambiguities in going *'from image sequences to recognized moving polyhedral objects'* are clearly shown even though the algorithms presented perform well.

Faugeras and Hébert's work on recognition and positioning using geometrical matching between primitive surfaces [FH83] uses data from an accurate laser range-finder in a recognition algorithm which is essentially a simple tree-search. This, again, is an example of the class of algorithms which rely on dynamically growing and pruning an interpretation tree structure that describes the correspondence between model and data features. Bray [Bray90] criticizes these tree-search formulations and concludes that their performance does not degrade gracefully with respect to robustness and speed under noisy conditions. That is to say, a single missing feature causes substantial problems for the search, and multiple missing features greatly increase the search space with the possibility that an interpretation may not be found within a given period of time. Solutions to the problem of missing edges or features have been suggested. One such idea is the null pairing or null face hypothesis [CD86, Goad83] where an unmatched feature is permitted to remain unpaired or pair to a non-existent feature. This, however, has the disadvantage of expanding the search space which is wholly undesirable.

The tree-search paradigm is essentially sequential and well suited to conventional Von Neumann machines. Attempts have been made to parallelise them [HB89*a*, SRS90] as Bray recognises saying that "it is certain that a tree-search can always be performed using a parallel algorithm in such a way as to exploit parallel facilities" but he duly points out that the solution is easier and more flexible when dealing with an intrinsically parallel algorithm [Bray90]. Goad [Goad83] describes how his tree-search algorithm may be unwound and how optimal search paths can be pre-computed from models off-line. Bray maintains that this is simply a modification to what is 'essentially a depth-first sequential search'. Bray also argues on the predictability of tree-search formulations. The time taken for a correct solution to be found can vary enormously depending on the location (of the solution) within the tree. Heuristics may be used to guide these searches but, as heuristics, they will prove to be effective some of the time. There would be an uncertainty in whether the system can arrive at a valid solution within a given time which can cause a problem in real-time systems requiring the 'best solution so far' after a given degree of processing. With heuristics, it is also true that they require some form of parameterization. The bounds of these are generally a matter of trial and error which is always time consuming. Bray further presents a new formulation for object recognition as an alternative to the tree-search paradigm which does not rely heavily on heuristics. It uses the same 2D information recovered from an image and transforms the problem from tree-search to signal detection. The algorithm is inherently parallel and is demonstrated to cope well with poorly segmented images that would cause major problems with conventional algorithms. This parallelism makes it potentially fast and contributes to its robustness. It uses local geometric constraints between 2D line segments obtained from segmented 2D images in order to recognise 3D polyhedral objects and adapts to use with either 3D line data or 2D polyhedral objects which, in either case, increased its efficiency. By accumulating information available from the local constraints (an angle constraint and a direction constraint), it forms match hypotheses subject to two global consistency checks that are enforced using the competitive paradigm. The local constraints can cope with the types of errors in the data commonly associated with occlusion, bad edge detection, and poor lighting.

2.5 Parallel Implementations in Recognition

As mentioned above, any sequential tree-search algorithm is exploitable on a parallel computer architecture. An early algorithm, proposed by Grimson and Lozano-Pérez, allowed the search space to be explored in parallel on a Connection Machine [GL-P84*c*]. They exploited the router feature of the Connection Machine because there is no inherent locality to the recognition problem. New levels of the tree were generated and pruned in parallel by having processors hold bit arrays which represent consistent pairing of points to faces. Flynn and Harris have improved on this approach [FH85]. By having a Connection Machine processor hold bit arrays which represent consistent pairings of points to faces means that a separate processing element is required for every interpretation which results in the algorithm being highly dependent on the size of the interpretation tree. Given s data points and m model faces, m^s interpretations are possible and this is, therefore, the number of processors required by the algorithm. When compared to the small number of data points generated when using tactile sensors, laser range-finders can produce as much as a 100 times the number of data points. This, along with fairly complicated objects, quickly makes the algorithm redundant on even a large machine with 256,000 processors. They propose a dynamic algorithm (unimplemented) which copes with the inevitable problem of processors being overrun. The idea was to generate enough levels of the tree to fill the machine, perform the pruning step, generate new levels of the tree to fill the tree, perform the pruning step, and so on. The machine is required to find unused processors to represent new branches of the tree and deallocate processors which no longer represent consistent pairings.

Holder and Buxton [HB89*a*] in their work on the DAP parallel architecture took a different view of the problem. Efficient parallelism, in their algorithm, comes about from the initial determination of consistent matches between model and data features. This is the key step to the algorithm and is performed in parallel for all data points. Therefore, given m data points on a polyhedral model with n faces, the method exploits $n \times n$ parallelism in establishing the possibilities of a geometric match. The computation is performed off-line and stored in binary 2D look-up tables. These are very efficient to manipulate during the tree-search local consistency phase. The search is still sequential but now acts on a very heavily pruned interpretation tree. The effectiveness of the algorithm outweighs the array memory required to stored the look-up table matches. For an object having 32 faces and 8 data points a store of 28 32×32 two dimensional arrays are required. The serial search becomes a recursive process which simply checks that at any level of the search the matched pairs must be consistent with the preceeding partial interpretation. Parallelism can again be exploited here as the test must be performed on matched pairs at that level. This algorithm maps very well onto the processors of the DAP with the (partial) limiting factor being the dimensions of the 2D array store.

In accord with Grimson and Lozano-Pérez and, Murray and Cook [MC88], Holder and Buxton have observed that the efficiency of interpretation tree search is improved when the data is presorted so that the most effective pruning occurs near the root of the tree. However, because all data points are checked against model faces in parallel, sorting these in order of pairwise separation will make little difference to the search time. They proceed to show that the most effective pruning constraint is likely to result in the least number of paired matches in the look-up tables. The solution (they suggest) is therefore simply to sort the data into ascending order of geometrical matches before commencing to check for consistent assignments of data to model faces. However, we will show (in Chapter 5) that sorting in order of ascending number of matches is only effective for small amounts of data. They note (conveniently) the efficient implementation of sorting functions capable of performing such operations on large array data structures in the DAP programming language.

A demonstration of the algorithm is compared with that of Flynn and Harris on data obtained from a chipped brick. This showed very large gains in speed after seven pairwise comparisons even though the method of Flynn and Harris was reported to run four times faster than an implementation of the sequential algorithm [FH85]. The null pairing hypothesis [CD86, Goad83] may be efficiently implemented using the technique of Holder and Buxton. This is simply a matter of increasing the dimensions of the array look-up table by one (this being the null pairer). The obvious combinatorics observed by other workers has a reduced effect here. The heavy pruning action of the algorithm, along with the sorting in order of most effective constraint, ensures the extra overheads are kept to a minimum.

A parallel implementation of a vertex-pair matching algorithm [MT87] can be found in [SRS90]. The implementation is on the Connection Machine and simultaneously computes the transform between a vertex-pair in the model and all vertex-pairs in the scene in parallel. The vertex-pair feature is defined as two vertices and two edges intersecting at one of the vertices. Scene vertex-pairs are stored one per Connection Machine processor. These are able to be compared in parallel against each model vertex-pair. A further parallel method is described in [NLC90]. This is an implementation on a Hopfield neural network which uses the parallelism and distributed processing of the network to globally match all the object data in the sensor space to all the object models in the model space simultaneously. The models are constructed as a graph structure with nodes (model features) connected by arcs which represent the relationship or compatibility between them. Object recognition results in matching a global model graph (representing all the models) with a scene graph (representing a single object or several overlapping objects). A 2D Hopfield binary neural network implementation obtains the optimal compatible matching features from the two graphs. This proceeds using excitatory and inhibitory supports at the synaptic interconnections between neurons. The state of each neuron in the network represents the possibility of a match between a node in the model graph and a node in the scene graph. That is to say, matched features belonging to the same model receive excitatory supports, and matched features belonging to different objects receive inhibitory or mutual support depending on whether the input scene is an isolated object or an overlap of several objects.

2.6 The Method of Geometric Hashing

While many recognition algorithms have adopted a tree search procedure, others have used the idea of look-up tables to determine instances of a model from image data. Advantages exist for either method. A particular approach which uses the scheme of look-up tables is *Geometric Hashing*. This is based on the indexing into a *hash-table* to recover precomputed transformation invariants of models in the database. The method [Wolf90, LW88, LSW88, RH91] is a general model-based recognition approach for partially occluded objects with noisy data. It is applicable in both 2D and 3D and is divided into a *representation phase* and a *matching phase.*

The key step lies in the off-line precompilation and storage of representations of the models in the database in terms of all possible transformation invariant coordinate frames. This representation uses groups of *interest points* in the model such as vertices or lines and the procedure builds a hash-table based on these representations. During the matching phase, interest points in the image are used in the same manner as in the representation phase to determine an *index* with which to enter the hash-table.

The approach is viewpoint independent and is able to recover the model to sensor space transformation. Lamdan and Wolfson [LW88] illustrate the methodology for the representation scheme. The interest points chosen must be invariant to rotation, translation, and scale. These are taken in pairs (for the 2D case) or triplets (for the 3D case) and a local coordinate frame is defined by them. For the 2D case the positions of the pair of points or *basis pair* are assigned the coordinates (0,0) and (1,0) for the first and second points respectively. This vector (from (0,0) to (1,0)) is taken as the unit vector in the x-direction of a new coordinate frame. The y-axis is now defined using the unit orthogonal vector of the same length in the anti-clockwise direction to the x-axis. After defining the new coordinate frame, the m-2 remaining interest points are determined with reference to this frame, where m is the number of model interest points. Each interest point coordinate, after appropriate quantization, is used as an entry into a hash-table where the basis pair at which the coordinate was derived and the model are recorded. In order to cope with the likelihood of one or other of the basis pair being occluded or absent in the image data, the process is repeated with all possibilities of basis pairs for all models in the database. The matching process is now reduced to determining interest points in the image, choosing an arbitrary basis pair, and determining a coordinate frame with which the s-2 remaining interest points can be located, where s is the number of sensed interest points. For each coordinate, the appropriate entry in the hash-table is checked and for every (*model, basis-pair*) record at that entry, a tally is maintained. After determining all the coordinates in all possible frames, the (*model,* *basis-pair*) records scoring a high number are taken as possible matches. The matches may then be confirmed by determining the best model to sensor space transformation.

Comparisons have been made between the method and other similar approaches [LW88] and the results have been favourable. As the hash-table is computed off-line, the method can be efficient in terms of search time. Also, extension to the 3D case simply involves defining the (x, y) plane using three point bases with the z-axis defined by the normal to the plane. However, in checking the sensitivity of such a method, Grimson has found it to perform well for simple images or exact data but rapidly degrades for cluttered scenes or in the presence of sensor uncertainty [Grim90]. This may be partly attributed to the method's dependence on a relatively sparse hash-table which quickly fills up and, thus, generates incorrect matches with the addition of spurious data features. The effects of this, however, can be reduced by coupling the process with a selection method that can reduce the ratio of spurious data to real data to a more manageable level. As the method is 'parallel in a straightforward manner' [LW88], an option also exists to exploit this parallelism on an SIMD or MIMD architecture [BM90].

2.7 The Invariant Theory

The theory of invariants has been around for many years [GY03, DC71, Spri77] but has recently received new attention [FMZB90, ZMFMR90, Wein89]. An invariant, in vision terms, is a factor which, as the name suggests, does not vary with respect to pose or camera position. It may be defined as a function which, when it acts on a point, has the same result irrespective of the coordinate frame [FMZ90]. Thus, area and curvature are invariant under translation and rotation in a plane but are not under projective transformation. Recent work by Forsyth *et al.* has demonstrated the theory using real data on planar and curved surfaces [FMZB90]. As the shape of objects vary with viewpoint, they demonstrate how invariants can be used to construct descriptors for such surfaces.

Many properties are invariant to projection. For example, straight lines project to straight lines and collinearities and intersections are preserved. It is noted that this invariance makes possible polyhedral model-based vision. For smooth curves and surfaces, invariants such as the zeros of curvature, cross ratio, and Gaussian curvature are available. These do not offer such a strong set of constraints as the edges of polyhedra for vision. Another publication by Forsyth *et al.* investigates a way of expanding the use of curves for vision by the use of invariants [FMZ90]. It is, at present, able to do this for coplanar curves but 3D extensions have been found to be more difficult. Other invariant schemes for the recognition of planar curves may be found in [Lin87, CCOD87].

A set of invariants for a model may be constructed and placed in a model database. This might be the point scalar invariant for a pair of coplanar conics (as might be found on a pair of scissors). The technique then proceeds using an image containing varying conic data. For two sets of coplanar data points, the joint scalar invariants are computed to produce a projectively invariant descriptor. This is done by fitting an approximation of a conic to the data using "the invariant fitting theorem". It is possible to determine if the curves are coplanar by constructing the invariants from two different views and checking to see if they have changed significantly. It must be noted that the object must have at least two coplanar curves that will both be visible at the same time. The technique then reduces to fitting a conic to every available curve, computing the projectively invariant descriptor and extracting the appropriate set from the model database. The system is fast as the search is merely a case of indexing into the model database. However, it suffers greatly from occlusions. In tests conducted, Forsyth *et al.* demonstrate how various image data have been correctly identified and labelled [FMZB90, FMZ90]. Examples such as gears, spanners and a pair of scissors were used. From the invariants of four parallel lines, they are able to recognise pallets in images, the invariants being the cross ratios of four collinear points. They also show how the system could be used to index into a parameterized model database to identify and recover various states of an open pair of scissors. Problems with ambiguities, however, have arisen using model representations. The representation is very sparse; one number for a pallet, two numbers for a gear and scissors. The robustness, however, may be increased by defining a model with a system of curves, for example, four or five conics in the description.

The extra constraints which are gained from an identified object may be exploited in determining the transformation parameters from model to sensor plane [ZMFMR90]. The invariant fitting allows a pair of coplanar curves to be modelled by a pair of coplanar conics. By this, the modelling conics undergo the same projective distortions as the original curves. Given this, the determination of the transformation becomes:

> **Given a known pair of conics on the world plane, and their corresponding conics in the image, determine the transformation between the two planes.**

This is achieved by back projecting the known conics. Zisserman *et al.* [ZMFMR90] give the results for a mouse, rotated by ~90° and translated slightly. There seems to be good agreement between the actual and computed rotation. They conclude by pointing out several generalisations of the work. To disambiguate models which have the same representations in terms of projective invariants, they suggest including values for pairs taken from a range of nearby features. To overcome the occlusion problem they suggest using the projective differential invariants (as in [FMZ90]) or exploiting the projectively invariant function associated with one of the conics. On the subject of 3D modification, it is viewed as "uncertain" in its practicality, as it either requires the elimination of an extremely large number of variables from a system of polynomials, or solving very large polynomial systems.

Weinshall [Wein89] puts forward the case that the sign of the Gaussian curvature of a surface is a more concise representation of a surface for storage and recognition than the relative depth map of the surface, i.e. the exact 3D coordinates of the surface. Koenderink and van Doorn have also acknowledged this [KD75, KD76] showing how various qualitative properties of objects and the motion field relate to the optical flow or stereo disparity field, i.e. invariants of the vector field. This includes the sign of the Gaussian curvature. However, the sign of the Gaussian curvature does not provide a complete classification of a surface with respect to the viewer. That is, it does not distinguish between concave and convex. The location of the focus of expansion (FOE) can be used to complete the classification of local surface patches and distinguish between patches which are convex, concave, parabolic (i.e. cylindrical), hyperbolic (saddle point), and planar. The FOE is the point towards which (or away from which) the camera's motion is directed. Thus, in perspective projection (where the motion is translational only), the optical flow takes the form of vectors which intersect at a single point, the FOE. Regardless of the location of the FOE Weinshall shows that the sign of the Gaussian curvature may be computed. Weinshall also shows that is is not necessary to perform computationally expensive processes to recover the exact function of the surface and the motion parameters in determining the sign of the Gaussian curvature.

These claims for invariants as "a new framework for vision" have recently been challenged by Moses and Ullman [MU91]. They investigate the limitations of non model-based recognition schemes with respect to viewing position and illumination conditions and propose that non model-based recognition schemes produce the same recognition function for every set of models. Here, a recognition function is defined as a function from 2D images to a space with an equivalent relation. This means the function is universally consistent, i.e. it has an identical value on all images of the objects within the recognition scheme's scope. They show that all consistent universal recognition functions, with respect to viewing position, are a constant function, in that they do not make any distinctions between objects. This has also been concluded by Burns *et al.* [BWR90] and Clemens and Jacobs [CJ90] and means that the function cannot be relied on for object recognition. Therefore, a scheme not limited to a specific class of objects must be model-based, and the invariant approach to recognition cannot be applied to generalised 3D objects viewed from arbitrary positions. To exemplify this, they demonstrate how a wire object projects in one direction as a triangle, and in another as a square. This means that any invariant function must have the same value for a pyramid and a box, which means that a non model-based recognition scheme with a universal scope will not be able to discriminate between any two objects. They go on to show that even if it is permitted to make errors on a significant number of images, a non model-based scheme with a universal scope still cannot discriminate between objects.

2.8 Work with Curved Surfaces

The utilisation of surfaces, thus far, has been limited to polygonal and planar curves. However, it is important that arbitrary curves be considered when dealing with real-world objects and data. Such curves can describe surface markings or curvatures of contours. Representation of these surface curvatures have been investigated by Tanaka and Lee [TL90] who point out that a good representation must, amongst other qualities, be stable under small perturbations, invariant under changes in view point, and local so as to be robust to occlusion. In determining matching features for curved object recognition, Fuageras [Faug90] reports on extending previous work on stereo and motion which were limited to points and lines, to curvilinear (non-planar) features. The work is primarily concerned with the 3D motion of a curve to its observed image motion. Several recognition systems using scene data obtained from range-finders have been implemented for the determination of curved objects. A brief overview of these may be found in the later section of 6.2.

The idea of motion parallax, that is, the relative motion of nearby image points due to camera motion, has been investigated by several workers for the realisation of surface curvatures [BC89, BCZ90, BB88, RL85, L-HP80]. It is known that qualitative information about curvature can be obtained from a static view. Also, from orthographic projection, under planar viewer-motion, quantitative information for the curvature is available from spatio-temporal derivatives of flow [BC89]. Earlier literature has shown that the surface orientation along an extremal boundary can be computed from image data. Koenderink [Koen84] has also found a relation between the curvature of an apparent contour and the intrinsic curvature of the surface (the Gaussian curvature). The sign of this Gaussian curvature is equal to the sign of the curvature of the contour. Given some known local motion of the viewer, the Gaussian curvature of a surface at a point on its extremal boundary can be determined. The curvature is computed from spatio-temporal derivatives of image measurable quantities. Blake and Cipolla [BC89] report on their extension of earlier theories to the general case of curvilinear viewer motion under perspective projection. As it is difficult to determine whether the Gaussian curvature along the contour is bounded or not while moving around a smooth featureless object, their work is carried out using feature-rich (patterned) objects. Image features are then seen being "sucked" over the extremal boundary at a rate which is dependent on the curvature of the surface. Differential measurements of curvature across two nearby points are shown to be independent of uncertainties in the viewer rotational velocity. Typically, the two points consists of one fixed surface point, and the other, possibly on the extremal boundary. Unlike single point measurements they are also independent of the viewers acceleration. In short, differential measurements based on two points are insensitive to errors in rotation and translational acceleration. This is because it is a function only of the differential motion of the image features.

The use of motion parallax is therefore revealed as a robust geometric cue for the computation of relative depth and surface curvature on specular surfaces and at extremal boundaries. Using the relative motion between a specular highlight and an image feature, Blake and Brelstaff [BB88] have shown that the parallax of specularity is also a robust geometric cue for the estimation of surface curvature. The human visual system appears to use this cue when dealing with curvature on specular surfaces [BB90]. Earlier, Reiger and Lawton [RL85] implemented an algorithm using motion parallax for estimation of relative depth and direction of translation from real image sequences. They were able to show the degradation of the estimations with increased separation of the two points.

Experimental results for differential measurements based on two points are reported in [BC89]. From a sequence of three images of a scene taken from a camera mounted on a robot-arm (calibrated), they were able to to estimate the radius of curvature for a point on the extremal boundary of a cup, B, and a surface marking on another cup, A. A surface marking is considered as a point with infinite curvature and, therefore, its radius of curvature can be stated as zero. The estimated curvatures agreed with the actual measured curvatures although the results were very sensitive to errors in the motion parameters. They show this: an error of 1mm in a translation of 55.3mm in the determination of the camera position produces an error of about 190% in the estimation of the curvature at point B. Similarly, an error of 1mrad in the camera rotation was found to produce an error of 70% in the estimation of the radius of curvature at point B. These are reduced to 17% per mm, and 8% per mrad using the differential measurements of curvature computed between points A and B. The errors in estimation of the curvatures were again found to be further reduced for ratios of differential curvatures. Compared with the values obtained for the differential curvatures, the sensitivities were further reduced to 1.5% per mm and 1.1% per mrad respectively. Additional results on a real-time tracking system based on deformable contours (snakes) can be found in [CB90].

2.9 Conclusion

We have presented a review of the main results to be found in 3D model-based recognition. However, since we have a primary experimental approach to the research, this has been very limited with a concentration on the Grimson and Lozano-Pérez based methodologies which are often referred to in our work. We have, however, attempted to give a general background to model-based vision both with polyhedral and arbitrary curved objects. For the reader wishing to gain a wider view, detailed reviews are abundant in the literature [CD86, BJ85, Bied87].

Chapter 3

Parallelism and Tools in Computer Vision

3.1 Introduction

A recognition system, apart from being highly dependent on an accurate matching process, is also dependent on a large class of processes which compose the system. Like any precise system, the accuracy of the scene interpretation relies on the accuracy and effectiveness of the processes at the lower end of the system pathway. These components may range from the hardware to the algorithms used to recover the matching features. In terms of the hardware, we must be careful that we do not constrain ourselves too much by the machine architecture. Several implementations of visual systems have been carried out on fast processor serial architectures. Due to the size of the search space and the fact that real-time applications are sought, speed of processing is a very important factor which cannot be overlooked. In the quest to reduce the processing time of the matching phase, our work is implemented in parallel. A highly parallel array computer, the AMT Distributed Array of Processors (DAP) is used. In choosing such an architecture, we were careful not to limit ourselves and to ensure that the algorithms and approaches taken may equally well be ported to any array structured machine. The use of the DAP was also advantageous in that many serial algorithms are inherently parallel and may easily be described in an array type manner.

Other important components to consider in the recognition system are the data recovery processes. These must be able to accurately segment the scene and reconstruct the appropriate matching features from image data. Popular techniques include structure from motion or stereo. They involve an initial recovery of the 2D data edges using some type of edge detection algorithm, e.g. Canny or Sobel, followed by a 2.5D reconstruction of the scene pulling out the 3D relationships of visible surfaces and edges.

In the following sections, we describe the architecture of the DAP along with a brief outline of its programming languages. We also introduce the Connection Machine, a powerful and popular SIMD machine, along with an SIMD/MIMD architecture which attempts to be more applicable to a variety of problems. Next, we introduce systems which have been used as tools in the development of various recognitions algorithms, namely, the WINSOM solid modeller, and the ISOR and TINA visualisation and recognition systems. Prior to the latter two, we introduce the Canny edge detection algorithm which is implemented in both systems and describe the basic principles of the motion and stereo reconstruction algorithms.

3.2 The Distributed Array of Processors

The AMT DAP (Distributed Array of Processors) is a fine-grain, highly parallel array computer. Current techniques of VLSI technology have made it possible to allow a large number of simple processors to be integrated with memory to achieve what is known as *active memory*, hence AMT (Active Memory Technology).

The DAP provides a natural solution for many large problems which cannot easily be solved on conventional computers with associated serial languages. Primarily, these were not constructed to handle arrays of data and/or the manipulation of variable word lengths. The DAP, therefore, targets problems which require the manipulation of large volumes of data which have inherent parallelism. This covers a wide area of tasks which may be found in fields such as graphics, image processing, neural network simulation, and computer aided design (CAD). The parallelism also proves useful in problems of computer vision where a real-time analysis is desirable.

At present there are two versions of the DAP. A 500 series and a 600 series, both with an operational speed of 10 MHz. They are constructed with an SIMD (Single Instruction Multiple Data) architecture and operate on vector and matrix arrays of memory. Therefore, the DAP is 'data-parallel' as opposed to 'task-parallel' which is the case for an MIMD (Multiple Instruction Multiple Data) machine. The difference in the DAP series is given by the number of processors in each machine; 1024 for the 500 series, and 4096 for the 600 series. These processors or, as they are better known, processing elements (PEs) are not as sophisticated as in single processor machines. They are simple single-bit processors which operate on bits of data simultaneously. This is the principle of the SIMD architecture in which the PEs execute the same instruction simultaneously, each in its own local memory. Processor elements can, however, be turned off and a particular advantage of the DAP, apart from its speed, is its ability to operate on a selection of data using conditional instructions. The DAP therefore overcomes the problem found in vector processors where conditional instructions interrupt the vector flow.

3.2.1 Internal Configuration

The processor elements of the DAP are arranged in a square array of ES × ES, where ES (the number of PEs in each row and column) denotes the edge size. Currently two edge sizes exist: 32 for the DAP 510, and 64 for the 610. The DAP 510 therefore has 32 × 32 = 1024 PEs, and the 610 64 × 64 = 4096. Each PE has connections to its four nearest neighbours in a North, East, West, South configuration. This provides rapid data broadcasting and fetching and gives the high level of connectivity required for many applications. A bus system also connects processors by rows and by columns. Connections of PEs at edges are wrapped around to appropriate PEs on the opposite end.

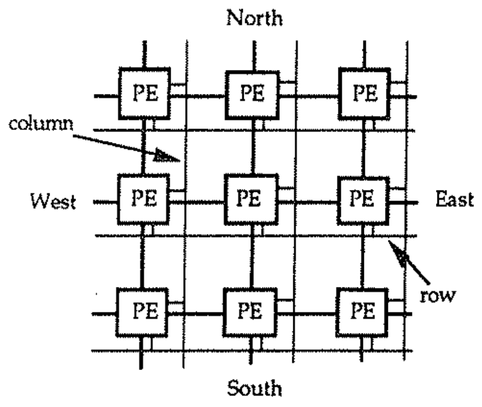


Figure 3.1: PE connections of DAP

Commands to the DAP are received by a Master Control Unit (MCU) which overlooks the PEs. This is, in fact, a conventional CPU except that it does not execute all the commands it receives as in single processor machines. Parallel instructions are received and decoded by the MCU from the code memory. These are, in turn, broadcast to the processor array to be executed in parallel by each PE. This execution is conducted in the local memory of the PE which is constrained by the present architecture to between 32 Kbits and 1 Mbit. This gives a potential memory range for the DAP 510 of between 4 Mbytes and 128 Mbytes (for 1024 processors).

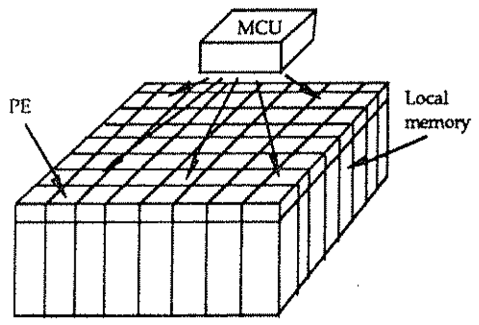


Figure 3.2: MCU and processor array

The MCU is a 32-bit central processing unit with many conventional features such as registers, instruction counter, branch instructions, arithmetic unit, etc. The object code of the DAP is loaded into the code memory from which the MCU fetches and interprets instructions. Some instructions will be executed wholly within the MCU (such as scalar operations using MCU registers and control instructions), others will be broadcast to the processor element array to be obeyed by the individual PEs in parallel.

There is an option to increase the processing power of both DAP series by using an array of 8-bit co-processor chips. The array is ES × ES and is reported to enhance the performance of the systems by an order of magnitude [AMT90]. The 8-bit co-processors work alongside their respective 1-bit processor for complex arithmetic such as floating point operations, whilst the single-bit processors continue to be used for memory access, fast input/output, and Boolean logic operations. As an example of the processing power of the 610 DAP with co-processor (610C), a factor of 40 in computing power is reported over the DAP 510 (without co-processor) as it can perform 8-bit integer multiplications at a rate of 2.4 billion per second and has a rate of 560 MFLOPS for 32-bit addition.

3.2.2 DAP Programming

A DAP program resides in the code memory of the MCU. The DAP may be accessed through a SUN or VAX computer which is termed the 'DAP host'. To the host, the DAP acts purely as extra memory. As two computers are used in DAP programming (i.e. the DAP and the host), a working program must be constructed in two parts. A host section which runs on the host machine, and a DAP section which runs using the DAP hardware. At run-time, the DAP program is initiated and controlled by the user program on the host. Depending on the application, it may be better to run a program almost wholly on the DAP or to run only highly data-parallel routines on the DAP leaving the host program predominantly in control.

The DAP unit may be programmed using one of two languages. These are an extended version of Fortran called Fortran-Plus and a powerful macro-assembler APAL (Array of Processors Assembly Language). Operations on vectors and matrices are constrained by the edge size of the DAP. Recently, however, an extended version of Fortran-Plus, Fortran* (Fortran star, or Fortran-Plus enhanced) has overcome this limitation. The most important feature of Fortran-Plus and Fortran* is their ability to manipulate complete data structures called *vectors* and *matrices*. These are extensions of Fortran arrays with parallel operations defined upon them. A 'matrix' corresponds to a two dimensional array, while a 'vector' corresponds to a one dimensional array. Data items in a vector or matrix are distributed over the DAP processors, each being processed by a single processor. Operations are (conditionally) performed by all processing elements at once. If the size of a vector exceeds the number of DAP processing elements, the system reorganizes the vector so that each processing element operates on more than one data item. With matrices, however, this happens if either of the sides exceeds the edge size of the DAP.

3.2.3 The Array Store

To facilitate parallel operations, scalars (non-parallel objects) are stored horizontally across a DAP bit-plane and vectors and matrices are stored vertically. For vectors and matrices this means that the bits of each component are stored at the same position in successive bit-planes. Therefore, the bits of each component are within the memory of the same processing element. A bit-plane is a plane in the array store. This means that each bit-plane is associated with the corresponding PE in the matrix of PEs and all bits in a plane have the same plane address. This is shown below.

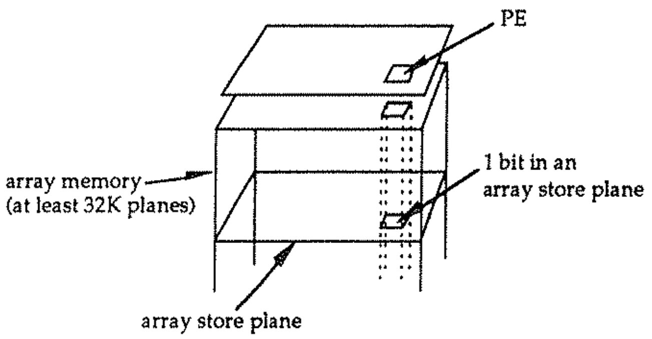

PE
1 bit in an array store plane
array memory (at least 32K planes)
array store plane

Figure 3.3: DAP array store

To touch briefly on the SIMD aspect of the DAP, consider the following. If we have a matrix A of dimensions 32 by 32, to sum its contents with the corresponding components of a matrix B of equal dimensions would result in a loop executing 1024 times on a conventional machine. On the DAP however, each component of the matrix is mapped onto a single PE which executes the command in its own local memory. This will mean that all components are executed simultaneously under the command: A + B. It is not necessary for every PE to execute the instruction. If we choose, we may turn PEs off with the aid of a 'mask'. Our problem may then be carried out on, say, every even column or where the components of A are less than the components of B.

3.2.4 Application of the DAP

In the work carried out, a DAP 510 was used in conjunction with a SUN 3/160M or SUN 4/260 host running under the UNIX operating system. The DAP had a local memory of 128 Kbits per processor and a high resolution colour monitor on the fast data channel. The host programs were usually written in C. Fortran was sometimes used as it made things easier when transferring data from the host to the DAP as they both had the same common block structure. As an early vision and graphics tool, the DAP is very useful. Its array structure makes it ideal for image processing and graphics techniques such as ray tracing where many pixels need similar processing operations.

3.3 The Connection Machine

Although many parallel machine architectures can be found on the market, some have been more popular than others in the research field. The Connection Machine (CM) is one such massively parallel supercomputer which has aided in many research works both in graphics and vision [CDHMS89, RH92, FH85]. The architecture is fine-grained SIMD and is so named because it is able to configure the topology of the machine to the topology of the problem [Hill85]. Being SIMD, it is often used for well-structured problems with regular patterns of control. Its power is achieved with the use of a very large amount of simple processing cells (PEs) e.g. the MIT Connection Machine contains 256,000 processors. These PEs can be connected to form *active data structures* such as sets, trees, butterflies, strings, arrays and graphs. These are then able to represent and process the data. Like the processors of the DAP, the CM also provides for the manipulation and switching off of individual processors for conditional instructions.

Currently two generations of the Connection Machine exist; the CM-1 and the CM-2. These differs mainly in the number of PEs and local memory available per PE. The CM-1 has 65,536 processor/memory cells each with 4096 bit of memory to give a total of 32 Mbytes. In contrast, the CM-2 has 8 Kbytes per processor which results in 128 to 512 Mbytes per machine. The architectures support *routers* which are interconnection networks constructed from autonomous switching elements. The routers are not all connected to each other. They are wired in a relatively sparse pattern which is the topology of the network and aid communication between processors by forwarding messages from router to router. The processors, which are connected in a nearest neighbour and router configuration, are then able to form a general intercommunications network that can connect to cells in any arbitrary pattern, i.e. a processor stores a pointer to the next processor cell. Being so small, the cells are incapable of computing a significant computation on their own. Therefore, by connecting multiple cells as active data structures, complex calculations can be achieved by interaction of thousands of cells through communications networks. For example, all cells in a certain state may be directed to add together two of their memory locations and pass the result to another cell.

As in a conventional computer, the Connection Machine has no processor/memory bottleneck since the PEs do the processing. That is, as larger computers are built, the processing power to memory ratio decreases due to the overhead in maintaining the larger memory reducing the processing power. It supports parallel data structures such as the *xector* which is a parallel form of a 1D array. The architecture also supports a *virtual processor* capability. Here, each real processors memory is divided into equal parts each devoted to a different virtual processor. Instructions are then repeatedly executed with a different base address until all virtual processors have been served. This technique is used when the amount of data strongly outweighs the number of processors. If we consider a 512 × 512 pixel image, 16 virtual processors per real processor on a 16K processor machine is sufficient for rendering on a pixel to processor basis.

As with most fine-grained architectures, the CM is designed to run in conjunction with a single processor host machine (here a DEC VAX or Symbolics Lisp Machine). This is due to parts of the code being able to run faster on a single processor machine. Like the DAP, the CM is programmed in an extension of a sequential language CmLisp (Connection Machine Lisp) a version of Common Lisp designed to support parallel operations of the CM.

Applications by different workers have been varied [RH92, FH85, BM90, BS88]. Crow *et al.* describes an implementation of ray tracing on the CM [CDHMS89]. The image is divided into 128 × 128 patches thus providing a pixel to processor mapping of 1:1 for a 16K processor machine. The PMF stereo reconstruction algorithm of Pollard, Mayhew, and Frisby [PMF85] is reported to have potential for implementation on a suitable large scale parallel computer [Poll88]. Such an implementation by Drumheller and Poggio on the CM is further reported in the introduction of [Page88*a*].

3.4 The Disputer: An SIMD/MIMD Hybrid Architecture

In view of its design, the SIMD architecture is more suited to data-parallel tasks where a large number of simple processors are used. With the processors arranged in an array, great performance advantages may be gained as only one processor controller is required for the whole array, e.g. the MCU for the DAP. However, the processors are constrained in that operations are performed in strict lock-step which means that the architecture is suited for tasks requiring similar computations over a number of regular data items. The MIMD idea on the other hand frees and relaxes some of the often severe restrictions imposed by the SIMD architecture by using a network of conventional von Neumann processors. This, however, is at the cost of the controller and program memory being replicated for each processor and the expensive irregular non-local processor communication. Such advantages and disadvantages of the two architectural paradigms have often acted in directing SIMD machines to be used in low-level, early vision tasks and MIMD machines for higher-level vision problems where a more natural, independent analysis, may be desirable. Page [Page88*b*], however, reports work on implementation of a dual SIMD/MIMD paradigm which attempts to bridge the gap between the two architectural extremes. This hybrid, called a Disputer, is also suited for other tasks such as graphics where the SIMD paradigm can exploit the data-parallel operations found in image processing and analysis, and the MIMD sector may exploit the task-parallel computations such as ray tracing.

This hybrid architecture consists of a 16×16 array of single-bit processing elements with bi-directional data communication to its North, South, East, and West nearest neighbours. This square array of PEs is coupled to a transputer network in a rectangular array of 6×7 transputers. It is reported to have an SIMD computational bandwidth of about 0.5 Gigabits per second and an aggregate MIMD execution instruction rate of 200 MIPS with 10 transputers. The whole system is programmed in Occam 2 with control overlooked via a control transputer.

Implementation of a Mandelbrot set browser on the Disputer is reported [Page88*b*]. Here, the transputer network is used as a linear pipeline of 42 processors. The control transputer sends work packets sequentially down the pipeline and these are taken up by the transputers in the network. They compute the Mandelbrot set for the pixel in the work packet and send a run-coded result packet to the control transputer. These are then directed to the SIMD processor array which renders the pixels. Up to 64 4-bit pixels in a row may be rendered in 3 SIMD instructions (about 500ns per instruction). Thus, with an iteration limit of 250 on the Mandelbrot equation, a 512×512 image may be computed and displayed in around 3 seconds. The worst case image, i.e. completely black, is reported to take around 25 seconds. In this implementation, however, the 42 transputers were found to be the bottleneck which makes fast rendering of the SIMD array somewhat superfluous. Page reports more recent examples of a dual-paradigm program for simulation of interaction of Newtonian particles, low pass filtering, and edge detection [Page89]. Depending on the nature of the algorithm, processing may weigh mostly on the SIMD or MIMD portion of the architecture.

For the reader interested in hybrid machines for image processing, a recent article reports on Adapt, an architecture-independent language based on the split-and-merge programming model [Webb92].

3.5 The Use of Parallelism in Vision and Display

In recent years, many workers have realised that parallelism is the step forward in many computer based tasks. Numerous machines now exist which are based on the SIMD or MIMD parallel architectures. Even more algorithms exist for mapping serial problems onto these architectures, thus, exploiting the inherent parallelism to achieve a faster, more efficient program execution. We note that although an algorithm may be mapped onto several parallel machine architectures, many workers have tried to dedicate such machines to a particular field, be it image processing, graphics, or vision. This is so as to exploit as much of the parallelism as possible in the research area. Such dedicated architectures in the field of image processing may be found in [TM89, NS89, Gree89].

Sleigh *et al.* [SRH88], in an attempt to analyse these varying methods, have examined three architectures in the context of computer vision systems. These are the DAP, the Reconfigurable Transputer Processor, and the DIPOD system which is a multi-processor MIMD architecture. They implement several typical computer vision algorithms on these amongst which include the Sobel, Roberts, and Marr-Hildreth operators and confirm that in each case, processing speeds were significantly faster than on conventional machines. They also note that low-level algorithms which involve local interaction between data and regular computations are suited to the SIMD architecture and MIMD architectures are generally more accommodating to higher-level data dependent algorithms. Manning *et al.* [MDW88] and Morrow *et al.* [MP88] have also reported on the analysis of different image processing algorithms when implemented on a programmable VLSI processor array such as an array of transputers or CMU Warp processing elements [AKMS85].

The scope of recent parallel machine vision implementations have been varied. Kriskelis and Lea [KL89] in the work on image convolution and histogramming, discuss the implementation of these using the parallel SIMD architecture of the Single Chip Array Processing Element (SCAPE) chip [Lea86*a*, JL87] which is an implementation of the Associative String Processor (ASP) [Lea86*b*, Lea86*c*] optimized for numerical computation. It is constructed as a string of 256 identical PEs each comprising of 37-bits of contents-addressable memory. SCAPE chips may be linked to form a chain which increases the processing power of the system linearly. In addition, multiple groups can support control configurations such as Single Instruction control of Multiple SIMDs (SIMSIMD) and Multiple Instruction control of Multiple SIMDs (MIMSIMD). However, a problem with the application of string architectures to computer vision tasks is primarily with the mapping of the structure onto the image data structure. Kriskelis and Lea propose such a method which divides the image into patches with concatenation of patch lines. Multiple patches may then be processed simultaneously on the same string.

Continuing the theme of concurrency in image processing, Forrest [Forr88] describes exploitation of the parallelism inherent in the image restoration algorithm of Geman and Geman [GG84] on the DAP. This is an algorithm to enhance the gray-scale pixel images which have been corrupted by some noise process and seeks to find the most probable estimate of the uncorrupted image based on the corrupted image. Murray, Buxton, and Kashko [MKB86] have similarly used the DAP to exploit the parallelism of the Geman and Geman algorithm.

3.5.1 Edge Detection and Model Representation

In the lower-level, early stage of vision systems, image processing tasks show inherent massive parallelism. This is due to the independent nature of the data which can lead to a natural mapping of the algorithm onto a parallel architecture. Early low-level parallel vision implementations have included gradient operators such as Sobel maximum difference filters. This requires fairly little computation. Implementations, however, of more computing intensive processes such as the Canny operator can be found in the literature. Ruff [Ruff88] describes a parallel implementation using a pipelined architecture. It is able to operate at video rate but the architecture is not applicable to higher level vision processing which involves unpredictable, long-range support of the data. Another parallel implementation of Canny is presented in [Wyso89]. This exploits SIMD parallelism on the DAP which is ideally suited in that the operator relies extensively on the neighbourhood pixel operations. The DAP version was reported to perform 64 times faster when compared to an implementation on a Sun 3/160 serial machine. Also, in contrast to the serial version , the DAP timings were found to be largely unchanged for a complex image containing many edges.

In working on image data, Reddaway [Redd88] examines possible mappings of such data onto processor arrays as this is critical when implementing algorithms on SIMD machines. Such mappings onto 2D processor arrays may be sheet or crinkled or a mixture of both. He describes implementations of this on the DAP and presents performance results for an example Sobel edge detector. The importance of the mapping technique is revealed here when we realise that in the experiment conducted, the Sobel operator when mapped onto a

512 × 512 image in a crinkled, combined, and sheet manner was executed in 2.4, 4.9, and 8.0 msec respectively.

As the effectiveness of any vision system is not only dependent on the recognition and reconstruction algorithms but also on an adequate means of representing the two and three dimensional models, Brady and Scott [BS88] review two such representation schemes: Symmetric Axis Transform (SAT) and Smoothed Local Symmetries (SLS) which was designed to overcome some of the problems of SAT. They note that difficulties arise in developing a parallel algorithm to compute the SLS and introduce an algorithm, implemented on the Connection Machine, to compute the reflectional symmetries of the SLS.

3.5.2 Towards Higher Levels

According to Lamdan and Wolfson [LW88], the Geometric Hashing algorithm has potential for parallel implementation. A recent article [RH92] describes such an implementation on the Connection Machine architecture. Here, Rigoutsos and Hummel explore two parallel hashing algorithms: 1) a parallel hypercube to route information through a series of maps, and 2) a building-block algorithm which used the CMs large memory resources to achieve parallelism through broadcast facilities from the front end. Rather than assigning each hash table entry to a separate processor, the entire list of entries for a hash bin is stored in a single processor's local memory. This makes the preprocessing phase of creating the hash tables far less efficient due to the fact that the processors need to randomly access local memory as entries are appended to the lists. However, as the this phase is computed off-line, it does not present too great a problem provided that no single list becomes so long as to demand an exceptional amount of memory. During recognition, the entries in the hash bins which receive votes are counted. Rather than histogramming (i.e. counting) by sorting, the authors choose a message passing strategy which involves each hash bin that receives one or more votes from the scene points concurrently traversing its list of entries and sending messages to the corresponding buckets [RH92]. This, however, accounts for 99% of the recognition phase execution time. They suggest two possible enhancements which employ a rehashing function along with the use of certain symmetries in the hash table to reduce the entry lists.

The system was enhanced by running C code on the front end with system calls to the CM using its Paris package which includes many of the building-block and routing algorithms used. The model examples used are dot patterns of 16 points each. These were to be located in scenes of approximately 200 dots after subjection to rotations, translations, and scalings.

Along the same lines, Wallace *et al.* [WMMW92] present recent work which exploits MIMD architectures in intermediate and high-level vision including segmentation and model-based interpretation. They analyse different approaches in developing parallel algorithms and illustrate direct implementation, i.e. implementation employing a parallel language specifically designed for a parallel environment. This is performed by implementation of the Hough Transform for feature detection in 2- and 3D images. As this is expensive in both computation and storage requirements, they further modify the basic algorithm to exploit the greater flexibility of an MIMD network of transputers.

Finally, an interesting report on the use of parallelism in high level vision systems is discussed in [MBLSB92]. Here, Marsh *et al.*, at the University of Rochester, present a checkers playing system in a multimodel parallel programming environment. This is an *active vision* system in that observer-controlled input sensor are used and action is taken depending on the visual reports. Each module in the multimodel program may be implemented under a different parallel environment including MultiLisp, Lynx, the Uniform system, or Uthread. This integrated vision architecture is controlled under the Psyche operating system which was specifically designed to support multimodel programming.

In concluding, it can be seen that the use of parallelism in the areas of vision and display has not been restricted to any particular level. The problems tackled have, however, been strongly affected by the machine architecture. Lower-level issues have generally been implemented on SIMD architectures, with higher-level control-dependent tasks usually, but not exclusively, implemented on MIMD networks. As parallelism has not become popular untill recent years, a lot of work is still necessary to bridge the gap between the two extremes of machine architecture. Future constructions such as the hybrid of Page could therefore offer a more flexible and natural basis for concurrency.

3.6 WINSOM - A 3D Solid Modeller

In designing and testing any recognition strategy, it is important for test data to be readily available. Efficiency and ease of generation are factors important in the production of the data. If a camera image is the source of the scene visualisation, a 3D solid modeller is an effective tool in the production of such images. As it generates images artificially, we are able to determine parameters of the camera geometry precisely. Also, the lack of noise in the image makes the task of segmentation and 3D reconstruction more robust, therefore providing more accurate data necessary for the matching process. As the penultimate step to real-world scene recognition, the solid modeller is a very effective tool.

WINSOM [Quar84] is such a solid modeller. It allows the user to build raster pictures of objects which are defined by Constructive Solid Geometry (CSG). This means that complex scenes may be created using basic primitives such as blocks, cylinders, and spheres. Objects may be combined using the binary algebra operators union, difference, and intersect, and manipulated under a number of transformations. For example,

```
pair = ball UNION tube AT (100, 20, 50)
```

will combine two predefined objects (ball and tube) using the union operator and translate the result (pair) by (x=100, y=20, z=50). Using this, it is therefore possible to determine precisely the pose of any defined object.

The rendering of the objects uses a light simulation technique known as 'ray tracing'. This is a powerful method in realising artificial scenes but suffers in that it is computationally expensive and, therefore, slow, especially when implemented sequentially (as in WINSOM). Using it, however, we are able to define and manipulate various light sources in the scene and assign surface properties to the objects. In determining images suitable for stereo processing, WINSOM provides two operators: STEREO and DISPARITY. In the case of structure from motion processing, the object may be translated step-wise for each image frame.

3.7 Edge Detection

The detection of edges is usually one of the first processes undertaken in any vision system implemented with camera image data. The need for this is obvious - it simplifies image analysis by significantly reducing the amount of input data to be processed. A careful balance must, however, be established between the amount of image data retained and the preservation of structural information pertaining to object boundaries. After the initial edge detection, the 3D reconstruction can be performed. It is, however, essential that the edge detector exhibits a low error rate as the performance of modules requiring its output are hampered by inaccurate or spurious responses. Thus, in consideration, three criteria exist for the effective performance of any edge detecting algorithm [Cann86]:

1) The detection must be of a standard that it does not fail to respond to true edge points and falsely detects non-existent edges. This criterion is maximized by maximising the signal-to-noise ratio.

2) A detected edge point must correspond to within a small deviation to the centre of the true edge point.

3) Multiple responses to a single edge are not permitted.

This extraction of edges from gray scale images is dependent on two procedures, first: the detection of edge elements - *edgels*, and second, the linking of these edgels to form edge strings or edges. The extraction may either be via a parallel method - whereby processing of parts of the image does not depend on any other, or sequential - where processing in one region is dependent on the results of an earlier region. The type of operator used in detecting the edge depends on the type of edge sought. The profiles produced by an edge are affected by the illumination and may be positive or negative *step*, *roof*, or *edge-effect* edge types depending on whether the object is illuminated from far away, from nearby, or by specular rays.

3.7.1 The Roberts and Sobel Edge Operators

Most edge detecting procedures consist of a mask or *operator* applied over the intensity map of a gray scale image. By computing a gradient function in respect to image intensity, intensity discontinuities produced at object boundaries may be identified on the basis that a high local intensity gradient, thus indicating a sudden intensity transition, is likely to be the cause of an edge discontinuity. Many edge operators for this task can be found in the literature. A one-dimensional operator may be used to find edgels at an edge whose direction is predicted. When the directions of the edges are not predicted, a two-dimensional operator is used to determine edges of all directions. The *Roberts* operator is a simple two-dimensional operator for step type edges and uses a 2×2 window to determine the gradient function $g(x,y)$ for each pixel at (x,y). The convolution involves the summing of neighbourhood intensities multiplied by the operator values to obtain weighted averages at each pixel location. The operator window is defined in terms of x and y using two templates g_x and g_y for gradients:

0	1
-1	0

g_x

1	0
0	-1

g_y

mapped to the pixels

(x,y+1)	(x+1,y+1)
(x,y)	(x+1,y)

This gives the image gradient function:

$$g(x,y) = \sqrt{[\{f(x,y) - f(x+1,y+1)\}^2 + \{f(x+1,y) - f(x,y+1)\}^2]}$$

where $f(x,y)$ is a function of the intensity. It can be seen that the operator is based on comparative intensities in orthogonal directions across the window diagonals with the gradient function ranging from zero - indicating that all four pixels are of equal intensity, to a large positive number dependent on the neighbourhood pixel intensities. Therefore, the larger the gradient function value, the greater the possibility of an edge boundary. However, a particular disadvantage of an operator with such restricted neighbourhood comparison is that it becomes more sensitive to noise and, therefore, prone to misidentification of edge pixels. The *Sobel* operator uses a larger processing window in an attempt to reduce the sensitivity to noise. Unlike one-dimensional operators, a degree of neighbourhood smoothing is achieved by applying the operator in both x and y directions. Like most operators, it is defined in a 3 × 3 window as this can contain all the neighbours of the centre pixel:

$$\underset{g_x}{\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}} \qquad \underset{g_y}{\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}}$$

and delivers the gradient function

$$\begin{aligned} g(x,y) = [\ &\{(f(x+1,y+1) + 2f(x+1,y) + f(x+1,y-1)) - (f(x-1,y+1) + 2f(x-1,y) + f(x-1,y-1))\}^2 \\ &+ \{(f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)) - (f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1))\}^2\ ]^{0.5}\ . \end{aligned}$$

At detected edges the convolution signal is used to derive estimates of edge magnitudes and direction. The direction gives an indication of the alignment of the edge and is measured as an angle α, given for the Roberts operator as

$$\alpha = \frac{g_y}{g_x} = \tan^{-1}\left\{\frac{f(x+1,y) - f(x,y+1)}{f(x,y) - f(x+1,y+1)}\right\},$$

and for Sobel (for large angles) as

$$\alpha = \tan^{-1}\left\{\frac{(f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)) - (f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1))}{(f(x+1,y+1) + 2f(x+1,y) + f(x+1,y-1)) - (f(x-1,y+1) + 2f(x-1,y) + f(x-1,y-1))}\right\}.$$

The Roberts operator is not appropriate for the detection of roof type or edge-effect edges. For these, Laplacian operators are used where the Laplacian is defined for the function $f(x,y)$ as

$$\nabla^2 f(x,y) = \partial^2 f(x,y)/\partial x^2 + \partial^2 f(x,y)/\partial y^2\ .$$

After applying the edge operator, a *thresholding* procedure is performed on the edge value image array. Thresholding attempts to pick out edgels by preferring (x,y) locations whose *edge values* are above a threshold level (therefore indicating the greater possibility of an object discontinuity being present). Being simple, this is not always effective in determining edgels as, if the threshold is too low compared with the contrast, many pixels are indicated as composing edgels. A more robust method is to detect the local maxima which corresponds to the centre of the edge when the edge intensity profile is smoothed and the operator applied to it. Since noisy profiles may have fluctuating positions of local maxima, the method is usually applied to a pre-smoothed image or by using an operator with a smoothing effect. The local maxima may then be detected by the zero-crossing method of Marr and Hildreth [MH80, Marr82]. In their work, they propose that after an initial application of the Gaussian operator to the image for smoothing, the local maxima may be determined by the application of the Laplacian* operator and then extracting the zero-crossings.

3.7.2 The Canny Edge Detector

The detector due to Canny has proved popular for the recovery of edge data. In [Cann86] he presents a procedure for the design of edge operators for arbitrary edge profiles and presents mathematical forms for the first two criteria of edge operators given in Section 3.4. Given an operator with a response $f(x)$ at an edge $G(x)$ and assuming that the edge is centred at $x = 0$, the signal to noise ratio is determined as

$$\mathrm{SNR} = \frac{\left| \int_{-W}^{+W} G(-x)\, f(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f^2(x)\, dx}} \qquad (1)$$

where the numerator is the is the response of the operator to the edge at its centre with the assumption that the operator has a finite impulse bounded by [-W, +W], and the denominator is the root-mean-squared response to the noise n(x). Here, n_0^2 is the mean-squared noise amplitude per unit length. The localization is given as

$$\text{Localization} = \frac{\left| \int_{-W}^{+W} G'(-x)\, f'(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f'^2(x)\, dx}} \qquad (2)$$

* This produces the procedure which is known as a *Laplacian of a Gaussian*.

where for the responses, a local maximum in the total response is assumed to be at $x = 0$. As simultaneous maximization of these two criteria is sought, it is achieved by maximizing their product

$$\left\{ \frac{\left| \int_{-W}^{+W} G(-x)\, f(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f^2(x)\, dx}} \cdot \frac{\left| \int_{-W}^{+W} G'(-x)\, f'(x)\, dx \right|}{n_0 \sqrt{\int_{-W}^{+W} f'^2(x)\, dx}} \right\} . \qquad (3)$$

which is achieved when $f(x) = G(-x)$ in $[-W, +W]$. However, Canny notes that it will be almost impossible to find a closed form for the function f which will maximise (3) given that the operator is only permitted to give one response to a single edge and shows that a numerical optimization may be performed directly on the sampled operator impulse response to find optimal roof and ridge edges.

In concluding this section, we feel that the work described by Canny cannot be summarized effectively without going into detail. We therefore direct the interested reader to the reference [Cann86].

3.8 Principles of Three Dimensional Image Reconstruction

Many methods exist for the determination or reconstruction of 3D information from camera image data. Depending on the constraints on the scene and the input data available, the structural determination may be from stereo, motion, shading, or texture. In this section we describe briefly the principles of the first two techniques - structure from stereo and structure from motion. These are very useful procedures in vision systems and their accuracy is important in such systems.

The advantage of stereoscopic vision is that the position of a point in three-space may be recovered from 2D images. The principal idea is the use of disparity data of points in the left and right images based on the geometry of stereo vision. The disparity is the difference in the position of a point in one image compared to when observed in the other image. Using it, and the properties of right-angled triangles, the depth information may be recovered from an image pair. Consider Figure 3.4 below.

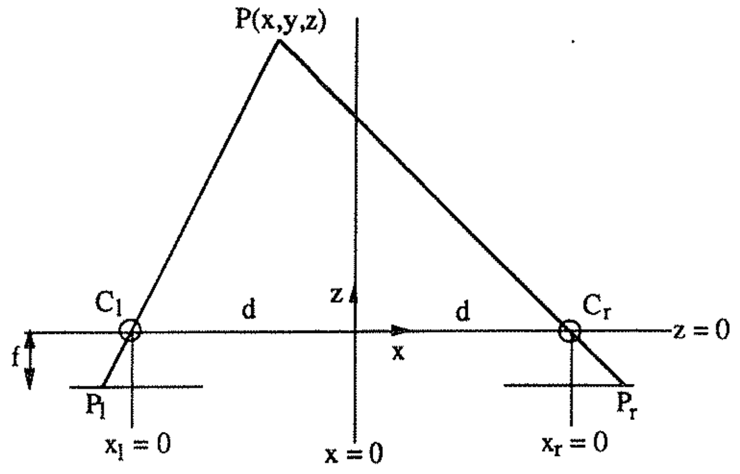


Figure 3.4: Principles of stereo geometry

Given two cameras C_l and C_r placed in a world coordinate space with positive x to the right, positive z into the scene, and positive y vertically upwards (out of the page), a point P at (x,y,z) will be projected onto the image plane of each camera as P_l and P_r for the left and right cameras respectively. If the interocular separation of the cameras is 2d and each is defined in its own local system, here, with the image centre at $x_l = 0$ and $x_r = 0$ for left and right cameras respectively, then the 3D coordinate of P may be determined from similar triangles. Note that P_l and P_r are measured in their respective camera coordinate systems. Note, also, that P_lC_l and C_lP form the hypotenuses of a set of similar triangles such that we can state for the left camera

$$P_l/f = -(d+x)/z \qquad (4)$$

where f is the camera focal length. A similar case exists for the right camera

$$P_r/f = (d-x)/z \qquad (5)$$

Combining (4) and (5) to eliminate x and then rearranging we obtain

$$z = 2df/P_r-P_l \qquad (6)$$

We can see from (6) that P_r-P_l is the disparity between the projections of the point P. As this approaches zero, $z \rightarrow \infty$ and becomes less reliable. Once z is determined, x and y may easily be obtained. The major problem, however, is finding corresponding points in the left and right images in order to compute the 3D positions of the points in space. The correspondence process usually involves extracting feature points in the left image and finding their corresponding points in the right image. A point is regarded as a feature point if, in the abscence of noise or specularities, gray scale intensities change considerably in its neighbourhood region. These points must be selected so that unambiguous matches can be determined.

Like the stereo matching algorithm, a similar correspondence problem may be found in determining structure from motion. Here, visual motion or *optical flow* is used. This is the velocity observed (from a set of image sequences) with each point on a 2D image plane when the scene, the observer, or both, move with respect to each other. If the velocity of the observer (or scene) is known, it is possible to recover 3D information from the images. Two techniques exist for the computation of this motion. The first, gradient or intensity-based schemes, use spatio-temporal processing of image irradiance gradients to compute the component normal of image motion along the gradient direction (or moving edge). This leads to the *aperture problem* which is the ambiguity experienced in determining the motion component of a selected area of the image. From Figure 3.5 we can see that although the motion of the triangle is from left to right, the motion component as viewed in the selected area is in a north-easterly direction perpendicular to the edge.

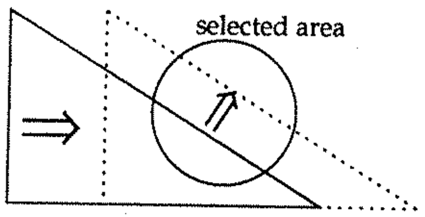


Figure 3.5: Example of the aperture problem

The second technique, token tracking, however, makes use of image features and suffers from the correspondence problem found in stereo matching in that a feature point in one image must be located (tracked) in a sequence of images. The displacement of feature points from frame to frame Δr is related as

$$\Delta r = \dot{r}\,\Delta t$$

to the visual motion $\dot{r}$, where Δt is the interframe time. Although this technique is much simpler than gradient schemes, it can yield the full visual motion [MB90].

3.9 The ISOR System - A Structure from Motion Toolset

The ISOR system (Image Sequence Object Recognition) developed at GEC Hirst Research Centre for the Alvey project: 'Spatio-Temporal Processing and Optical Flow for Computer Vision' is a complete system for the recognition of polyhedral objects from a set of image sequences [MCB89]. It is a 3D model-based, image recognition system that functions in a restricted blocks-world type of domain. Three dimensional data, recovered from a set of time varying images of a polyhedral object, is matched to an edge or face-based model representation. A single camera is used to generate this image sequence and both structure and motion are obtained from optic flow.

Three time varying image sequences of the data are fed into the system. Processing is maximal for the central image. The system is described in four main phases:

1) Low-level image processing and recovery of visual motion;

2) Image and visual motion segmentation;

3) Structure from motion;

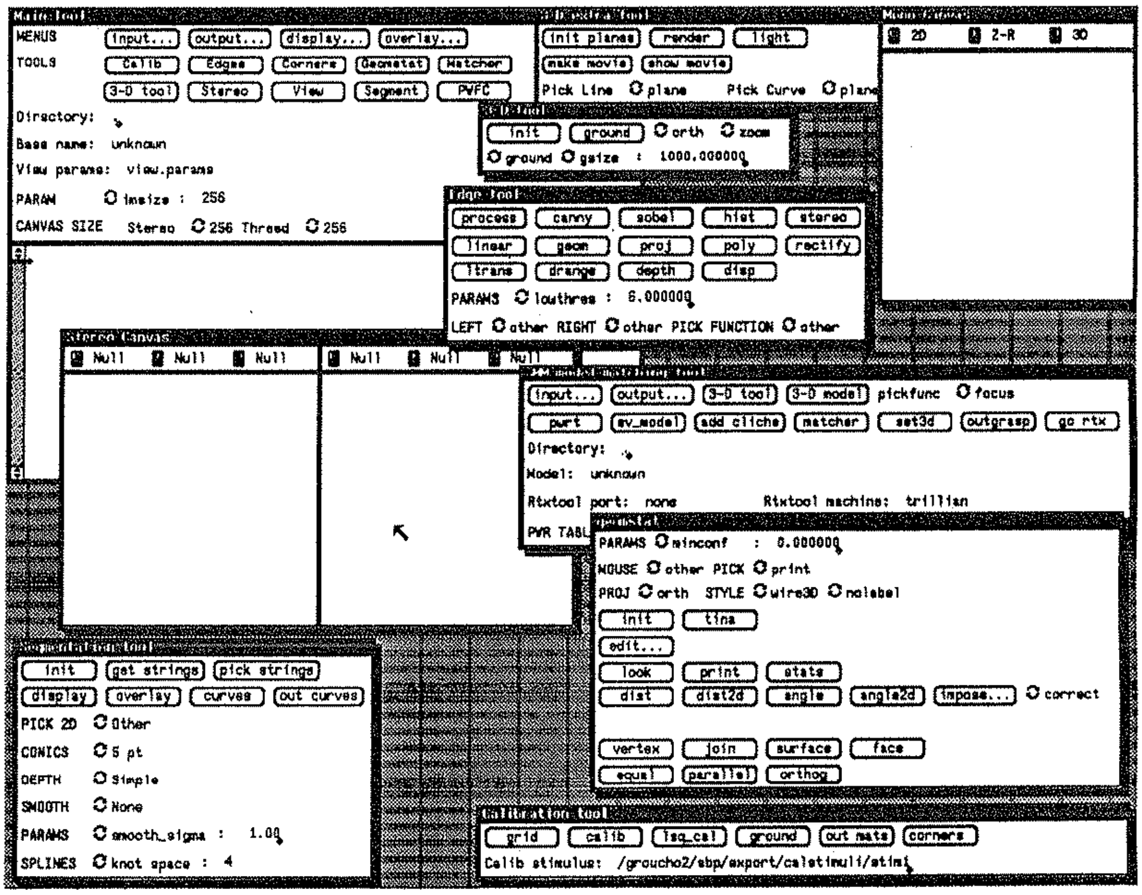
4) Model matching and validation of interpretations.

Since there is no stereo disparity, differential focus, or range-finder data, the depth/speed scaling ambiguity cannot be resolved bottom-up. Instead, the ambiguity is resolved at the stage of model matching. The matching is performed by a serial implementation of the Grimson and Lozano-Pérez algorithm using modifications of the pruning constraints. The matching data may be edges (for edge matching) or faces (for face matching). The models are stored as hand-coded vertex maps representing model edges and faces. For edge matching, model edges are sorted in order of decreasing length from which constraint look-up tables are produced. Four main constraints are calculated which use the angles between the edges and between an edge and a line drawn to a point on a second edge as the matching invariants which are used to prune the interpretation tree. For face matching, the constraints are applied to planar sections. Matching commences using the data edges (sorted in order of increasing length) or the data face sections, along with the constraint look-up tables to generate a set of feasible interpretations. These interpretations are refined by testing the global consistency of each match with the model. A program, *edgerst* for edges, or *facerst* for faces performs this step using a simple least-squares fitting to determine the rotation. Implementations of the quaternion method of Faugeras and Hébert [FH83] in the form of the programs *edgeqrst* and *faceqrst* offer a more accurate computation of the model to sensor space rotation. The translation and scale of each interpretation relative to the sensor are also computed. These are done by a simple averaging procedure. The interpretations are then passed or failed depending on how close the data features maps onto their respective model features.

3.10 TINA - A Stereopsis Based Recognition System

The TINA vision system (There Is No Alternative) is a complete visualisation and recognition system implemented in a SunView environment on a SUN workstation. It is built with the ability to recover 3D descriptions from binocular stereo images and, at present, these descriptions consist of point, straight edges, and planar curve primitives. The toolset is composed of a mouse driven front end consisting of a comprehensive set of tools for the processing and manipulation of stereo data and camera set up. Figure 3.6 illustrates the user interface with the main process tools.

Detailed usage of the system has proved it robust in the recovery of geometric descriptors and their subsequent construction into partial 3D wire frames. As opposed to full wire frames, these are permitted to include descriptors which arise from surface illuminations such as specular highlights. The system also includes a process for the segmentation of planar curves using both global and local behaviour analysis applied to edge-based stereo data [CPPMF90].



Main Tool
MENUS input... output... display... overlay...
TOOLS Calib Edges Corners Geomstat Matcher
3-D tool Stereo View Segment PVFC
Init planes render light
make movie show movie
2D 2-R 3D
3-D tool
init ground orth zoom
Edge tool
process canny sobel hist stereo
linear geom proj poly rectify
ltrans drange depth disp
Stereo Canvas
Null Null Null Null Null Null
3-D model matching tool
input... output... 3-D tool 3-D model pickfunc focus
pwrt sv_model add cliche matcher set3d outgrasp go rtx
geomstat
init tina
edit...
look print stats
dist dist2d angle angle2d impose... correct
vertex join surface face
equal parallel orthog
Segmentation tool
init get strings pick strings
display overlay curves out curves
Calibration tool
grid calib tsq_cal ground out mats corners

Directory:
Base name: unknown
View params: view.params
PARAM imsize : 256
CANVAS SIZE Stereo 256 Thread 256

Pick Line plane Pick Curve plane

ground gsize : 1000.000000

PARAMS lowthres : 6.000000
LEFT other RIGHT other PICK FUNCTION other

Directory:
Model: unknown
Rtxtool port: none Rtxtool machine: trillian
PVR TABL

PARAMS minconf : 0.000000
MOUSE other PICK print
PROJ orth STYLE wire3D nolabel

PICK 2D Other
CONICS 5 pt
DEPTH Simple
SMOOTH None
PARAMS smooth_sigma : 1.00
SPLINES knot space : 4

Calib stimulus: /groucho2/sbp/export/calstimuli/stimi

Figure 3.6: TINA tools user front end

The edges are generated to sub-pixel accuracy from gray scale images using a Canny edge operator with further improvement by thresholding. The stereo matching algorithm is an improved version of PMF [PMF85] which prefers matches between left and right image edges if the ratio of the difference in their disparity to their actual physical separation is below a threshold level (here taken as 0.5). They must also satisfy a number of higher level group constraints amongst which are uniqueness, figural continuity, and ordering along epipolar lines. The number of possible matches between edges in the left and right images is reduced using *a priori* information about the geometry of the cameras. In saying that, no *a priori* assumptions are made about the qualitative organisation of the cameras such as the need for the principal axis to intersect at a single point. However, permitting the field of view and the images obtained from each camera to be as similar as possible, and ensuring that the ratio of the viewing distance to the interocular camera separation is fairly large (about 5:1), greatly assists the matching process. It also improves the accuracy and robustness of recovered 3D features.

3.10.1 Reconstructing 3D Primitives

The strings of edge primitives obtained from the stereo processing are grouped into higher level primitives and combined with stereo matching and calibration data to obtain the 3D primitives. In recovering plane curve primitives, early versions of TINA began by obtaining estimates in a disparity space from sub-pixel estimations of individual edge primitives by way of their off-epipolar differences and, subsequently, fitting a plane to the disparity data by orthogonal regression [PPM87]. However, the algorithm did not recover an accurate estimate of the disparity between matched edges in the two images. In the current version, high-level grouping into 2D features such as edge strings, straight lines, and conic sections are performed prior to disparity detection [CPPMF90]. The disparity may then be obtained for each matched edge point from the intersection of the 2D descriptor with the epipolar corresponding to the sub-pixel location of the matched point.

The edge primitives are segmented with various parameterizations depending on whether straight lines or planar curves are sought. For straight lines, this is performed by a recursive fit and segment method, the fitting being carried out by orthogonal regression. This results in a set of 2D straight line sections which are combined with stereo matches to obtain 3D sections using least-squares fitting of disparity to line. The recovered 3D primitives are described with respect to the left-handed coordinate frame of the left parallel camera. They may subsequently be transformed to the coordinate frame of the physical left camera.

Various combinations of 2D segmentation and 2- and 3D fitting algorithms are permitted by the system for the realization of 3D planar curves. An initial 2D segmentation combines adjacent edge primitives into curves provided that their combined edge strings are sufficiently closely approximated by a circle. After this, the curved edge strings may be fitted by higher level primitives which, at present, are interpolating cubic splines and conic sections. The fitted 2D curves may then be interpreted in one of four ways into three-space by:

1) using the edge disparity data directly,
2) fitting an interpolating 3D spline,
3) projecting on the 'best fit' plane in 3D,
4) finding the best affine transformation between left and right images.

Combinations of 2D fitting with an interpolating conic fitted using a Bias Corrected Kalman Filter and 3D interpretation by projecting on the best fit plane in 3D tends to lead to much more accurate curve interpretations.

3.10.2 Model Matching and Verification

The model matcher of the TINA system is, at present, restricted to polygonal structures. It uses pairwise relationships to prune the interpretation tree and a *focus feature,* chosen from the group of features, which is required to have a good match at all times. This reduces the complexity of the search space. Cliques above a threshold cardinality between the match features are identified. These are used to determine transformations between model and sensor spaces. In building or correcting test models for the matcher, a statistical geometry package 'Geomstat' may be used. This allows wire frames to be built interactively such that geometric constraints such as orthogonality of faces, edge-vertex intersections, and parallelism of edges are satisfied. Corrections to the whole structure may be necessary to make the constraint linearised. This is true for the orthogonality constraint in which the corrections are optimally determined using an extended Kalman measurement filter.

Hitherto, the system has been limited to straight lines and circles as matching primitives. Cai *et al.* propose a planar curve segmentation method based on the analysis of curvature and the local and global behaviour of these curves [CPPMF90]. This would make it possible for matching to occur using conic and various forms of generic curve descriptors such as splines. By fitting descriptors in the left image, the 3D curves may be projected into space using the disparity data from the right image. This is more robust than the direct fitting in three-space of Pridmore *et al.* [PPM87]. This bottom-up approach aims to locate generic segmentation points (such as corners and smooth joins) and produces a symbolic description of the curve as a set of knots which are joined by arcs with descriptions such as *straight, concave,* or *convex.*

3.10.3 Recovering Curves

As curvature is invariant under rotation and translation operators, it is widely used as an indicator of corners, smooth joins, and curved objects. Sharp changes in orientation along a line produces a high curvature and indicates a corner at that point. These may be as a result of the intersections of two straight lines, a straight line and a curve, or two curves. However, because of corner-rounding effects due to data smoothing and finite differencing, this is not sufficient in locating the corners. Local behaviour analysis is used to overcome this. Local analysis cannot, however, foresee events accompanied by small and gradual shape changes. An example is a long curve segment with low curvature of which a short part results in a near straight line. Here, global analysis must be used to track the behaviour of the curve. Using the mean polygonal area enclosed by the curve and its chord (as the global analysis description), long running arcs with low curvature may be distinguished from straight lines. Note, however, that for corner segments formed by two straight lines, or a straight line and a curve, the mean polygonal area is nearly zero and these may be distinguished from the corners formed by two convex curves, two concave curves, or a convex and a concave curve.

The segmentation algorithm initially proposed reduces data noise with the B-spline smoothing of Cai [Cai89]. Using a high/low two-level curvature classifier, a rough segmentation of the curves is produced. This is refined with a high/low/zero three-level curvature classifier thus separating low curvature arcs from zero curvature straight lines. A set of semantic descriptions of the curve is then obtained using a knot/straight/convex /concave shape classifier. This also produces a set of critical knots for all segments. The final symbolic representation is obtained with a linear or quadric curve approximation applied to each segment. It is noted, however, that the segmentation method proposed is not always successful in extracting an elliptical curve as a whole, but experimental results have succeeded in several cases using the same choice of low and zero curvature thresholds.

3.11 Conclusion

In recent years the effect of parallelism in both vision and display has been most evident. We have reviewed some of the work describing implementations of once serial algorithms on parallel architectures. These architectures have included the DAP and the Connection Machine, both popular in the field in view of their massive parallelism. Networks of transputers have also been popular due to their flexibility in higher level tasks. We have also described some of the processes that are necessary in the visual interpretation of image scenes. These have been integrated into systems implementing 3D primitive recovery from both stereo and motion. However, we must stress that this is a qualitative rather than quantitative description of the tools available. A more detailed analysis of the procedures and tools available in Computer Vision can be found in the papers and technical reports listed in the bibliography at the end of this dissertation. We have concentrated here on the camera imaging and not, on other methodologies such as data retrieval using tactile or range-finding techniques as we are principally interested in vision.

A very important procedure in any system using camera images is an edge detection or feature finding process. This drastically reduces the amount of input data and, in doing so, contracts the search space for further processing. The Canny edge operator has become popular in view of its speed and accuracy. However, work is still progressing and improvements are sought as higher-level vision procedures may be highly dependent on this lower-level processing. This is true in bottom-up systems such as TINA and ISOR. However, other systems have used a top-down approach where a high-level control architecture regulates lower-level processing.

The procedure and tools reviewed in this chapter have been restricted to those encountered during the project. In conjunction with the model-based vision review of Chapter 2 they form a background for the research work which is to be described.

Chapter 4

Parallel Visualisation for Recognition

4.1 Introduction

Visualisation and closed-form testing of algorithms are vitally important in the development of model-based vision systems. The general idea is that, if we can specify the models in their own coordinate frames, their subsequent transformation in terms of their translation, rotation and scaling, and the camera geometry, we can then generate images for which the results of our visual recognition and localisation algorithms are known *a priori*. The fully tested and validated models and algorithms can then be used to analyse scenes where the details of the identity, position, orientation, scale and motion are not known, but where we have a range of models with which to form our interpretation of the scene, that is, model-based vision.

Ray tracing, or ray casting, is an effective method of generating such 3D scene images. However, this is very often time consuming and therefore is not practical for many image generation purposes. Introduction of parallelism is likely to effectively reduce the time impracticalities, and make the method more suitable for a system where several slightly different images of a scene must be generated efficiently. We present an SIMD parallel version of a generalised ray tracer which includes a Constructive Solid Geometry datatype, and an extension for Digital Terrain Maps.

In the following sections, we present the background for the work, its parallel implementation, and comparison with serial versions of the algorithm. We show that, for efficient image generation, the ray casting (and ray tracing) algorithm is suited to the SIMD architecture of the DAP. This is achieved by a 1:1 pixel to processor mapping which can readily be applied to other SIMD machines.

4.2 Ray Tracing and Ray Casting

Ray tracing [Appe68, Whit80], as the name suggests, is a technique in computer graphics where complex images can be generated by 'tracing' the path of light rays which are received by the eye. The early ray tracing algorithms of Appel have been extended by Whitted and other workers [Whit80]. This has resulted in probably the most powerful technique for rendering realistic pictures and can exhibit such light phenomena as soft shadows, reflected illumination (where an object is reflected onto another), and mirror-like, specular reflection. Transparency effects of refraction and fuzzy blur are also possible. Using this, naturally occurring phenomena such as clouds, fog, and flames may be realistically modelled [KV-H84].

The popularity of the technique also stems from its conceptual simplicity. The general principle is the projection of simulated rays emanating from a defined eye position through every pixel in the image into the scene world. The computer screen may be viewed as a window through which the observer is looking to see a collection of objects held behind it. Each ray may be reflected and refracted to some degree by the objects until they eventually reach the light source or reach the limit of the tracing. The fundamental idea is, then, to obtain the global illumination information for each pixel which is dependent on the degree to which the ray passing through the pixel has been absorbed on interaction with the various surfaces in the scene.

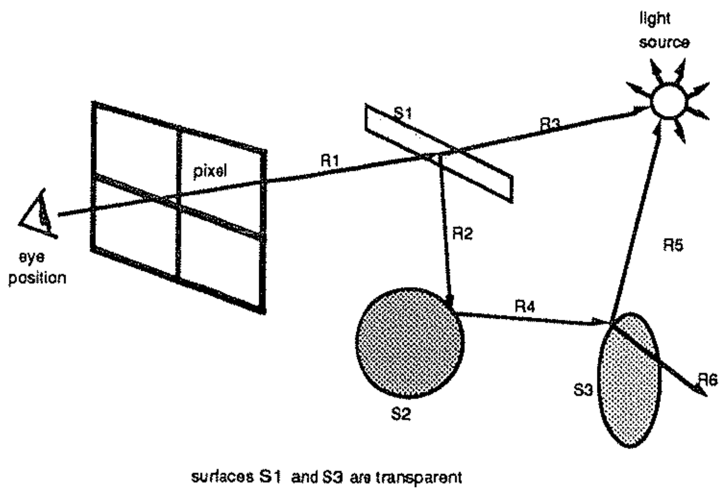


Figure 4.1: Path of light through pixel interacting with objects in the scene

This is analogous to the real world where rays emanating from a light source are repeatedly reflected and refracted by objects in the scene. The ray tracing problem then reduces to the fundamental calculation of finding the intersection of 3D lines (rays) with object surfaces.

On interaction with certain materials, a proportion of the ray will be refracted and also reflected. This will create two rays of a now reduced intensity. These rays will go on to make subsequent interactions with other objects in the scene. Eventually some will find their way to the eye position whilst the majority will not. However, an infinite number of rays will be produced by the light source and to follow up all of these proves unfeasible. Fortunately, this is not as fatal as it first seems. The key is to reverse the search and follow a relatively small number of rays - those that reach the observer. For this reason, the trace is worked from the eye position to the light source, immediately eliminating a huge number of 'dead end' traces.

The intensity of the ray reaching the eye position must be computed to set the colour/intensity of the pixel through which it passes. This is deduced from the intensity of the original ray and then by calculating the intensities of the subsequent rays after interaction with the objects. Of course, some rays may completely fail to collide with an object surface. The pixels of these rays must then assume the background colour. The ray trace and intensity determination may best be represented in the form of a tree structure. Each node of the tree represents a ray-surface intersection, and each branch of the tree represents the ray reaching the eye position. At each node of the tree, at most two sub-branches representing the specularly reflected and refracted rays are generated. We note that diffusely reflected rays are not to be traced since there are an infinite number of them. The effect of these diffusely reflected rays is approximated by an 'ambient' term in the lighting function. The radiosity method deals with such effects in a more systematic way and calculates the radiosity of surfaces in a viewpoint independent manner [GTGB84, NN85]. The tree is processed in a post order fashion and the intensity of the ray is not decided until all the intensities of the ray sub-branches have been computed.

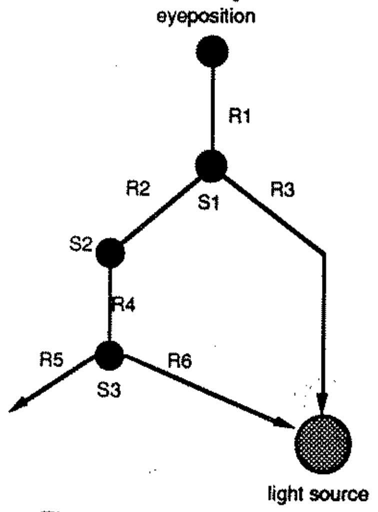


Figure 4.2: Reflection tree

From the reflection tree in Figure 4.2, all rays excluding R5 are required for the calculation of the colour/intensity of the light reaching the eye position.

A simpler form of ray tracing, which does not have the effects of reflections and refractions, is ray casting. Instead of following the paths of the rays as they collide with objects in the scene, the 'tracing' is stopped on the first collision. The illumination intensity of this object is then calculated. This will, therefore, give the effect of all objects in the scene as being opaque and non-reflective. As a consequence, this technique is much faster for rendering a scene but does not produce the realistic images seen in the full ray trace, a definition of which is given below.

```
procedure raytrace_scene
begin
  for each pixel from bottom_left to top_right do
     ray = make_ray(from_observer, to_pixel)
     first_object_hit = recursive_trace(from_observer, ray, object_rgb)
     if (first_object_hit is valid_object) then
        display_pixel(pixel, object_rgb)
     else
        display_pixel(pixel, background_rgb)
     fi
  od
end

function recursive_trace(from_point, ray, object_rgb) : object_number
begin
  min_t_value = FAR_AWAY
  for each object from first to last do
     current_t_value = t_value of intersection
     if (current_t_value < min_t_value) then min_t_value = current_t_value  fi
  od
  hitpoint = point of intersection of min_t_value
  normal = surface normal at hitpoint
  object_rgb = shade_object(hitpoint, ray, normal)
  return(object number of object with min_t_value)
end

function shade_object(hitpoint, ray, normal) : object_rgb
begin
  object_rgb = ambient light contribution at hitpoint
  object_rgb = object_rgb + diffuse light contribution at hitpoint
  object_rgb = object_rgb + specular light contribution at hitpoint
  if (ray hit object && maximum trace level not reached) then
  begin
     if (object surface is reflective) then
     begin
        increment trace level
        reflected_ray = calculate reflected ray
        first_object_hit = recursive_trace(hitpoint, reflected_ray, new_object_rgb)
        if (first_object_hit is valid object) then
           object_rgb = object_rgb + reflected light contribution of new_object_rgb
        else
           object_rgb = object_rgb + reflected light contribution of background_rgb
        fi
        decrement trace level
     end
     fi
     if (object surface is transparent) then
     begin
        increment trace level
        transparent_ray = ray
        first_object_hit = recursive_trace(hitpoint,transparent_ray,new_object_rgb)
        if (first_object_hit is valid object) then
           object_rgb = object_rgb + transparent light contribution of new_object_rgb
        else
           object_rgb = object_rgb + transparent light contribution of background_rgb
        fi
        decrement trace level
     end
     fi
  end
  fi
  return(object_rgb)
end
```

Figure 4.3: Serial ray tracing algorithm

Note that the shade_object function is defined recursively (indirectly via the recursive_trace function). Modification of this algorithm for ray casting will simply involve limiting the maximum trace level to one. This ensures that after initial collision with an object, the trace is terminated and the colour/intensity of that point computed. On tracing any reflected rays however, it is most important to realise that testing all the objects with the reflected ray (via recursive_trace) will inevitably produce a valid intersection point with the object from which the ray is reflected from! This is due to floating point precision and is overcome by simply ignoring any such intersection with the source object, for once a ray exits from one of the standard objects (box, sphere, cone, plane), it will not normally intersect again unless after a further reflection from another object.

Although ray tracing has been the primary technique where the production of superior images is concerned, it has often been rejected when there are considerations of speed as it is very expensive in terms of CPU usage. In Whitted's original 1980 paper he estimated that about 90% of the time is spent on intersection calculations. The time taken for complex, multi-object scenes can stretch into hours, or even days! For example, consider rendering a scene of dimensions 1024×1024 containing 100 primitives with an average of 2.75 intersection calculations per primitive (derived from a sphere, cone, box, and cylinder). Testing each pixel ray against all primitives results in 2.88×10^8 intersection calculations for the complete image! For this reason, attempts to reduce the number of intersection calculations as a means of reducing the overall ray trace time have been varied. Bounding volumes provide an effective method of decreasing the amount of time spent on intersection calculations for complex objects. Models which are relatively complex to test may be enclosed in a bounding volume whose intersection calculation is less expensive. Such volumes have included the sphere [Whit80], ellipsoid [Bouv85], and box [Roth82]. A tighter bounding volume is given by [KK86] which is defined as a set of infinite slabs each defined by a pair of parallel planes which bound the object. An object need not be tested for intersection if the ray fails to intersect with its bounding volume.

Attempts to reduce the large usage of CPU time have included the process of beam tracing [HH84], cone tracing [Aman84], and pencil tracing [STN87]. They are based on Whitted's suggestion that unweighted area sampling should be used to avoid the drawback of point sampling a regular grid as in conventional ray tracing. The algorithms tend to accomplish anti-aliasing (see later section), and at the same time reduce rendering time by taking advantage of coherence between neighbouring pixels. Williams, Buxton, and Buxton [WBB87] have also been successful with an SIMD implementation of a ray tracer which makes use of parallelism in reducing the scene rendering time.

Another approach proposed by Dippe and Swensen [DS84] uses a subdivision algorithm on the scene space. This was implemented on an MIMD (Multiple Instruction Multiple Data) architecture. The space is divided into cells, each of which is monitored by some working processor or processors. A processor is only responsible for those rays entering the cell. On intersection of a ray with a surface within the cell, the processor computes the intensity corresponding to the intersection and generates children for the ray. When a ray leaves the cell without intersection it is passed to the processor responsible for the region which it will next enter. However, the performance of this approach depends on the uniform distribution of load to each working processor [LS88].

4.3 Parallel Ray Tracing on the DAP

The ray tracing algorithm of Whitted has been shown to map well onto the processors of the DAP [WBB87]. This is due to the planar nature of the processor array where each processing element (PE) simultaneously obeys a stream of instructions broadcast by the Master Control Unit (MCU). The serial host machine is responsible for all I/O operations, reading in data files and setting up formats for the subsequent operations. The DAP is, itself, responsible for the computationally expensive calculations required for the ray tracing. These are the ray-surface intersections, determining shadows, and computing the illumination of visible surfaces. These are easily implemented as functions simultaneously operating on matrix arrays.

The 32 × 32 planar arrangement of processors makes it very convenient for the process of distributed ray tracing using sliced mapping [WBB87]. The principle is inherently simple. If each processor is assigned to a pixel, then the PE can ray trace the light through that pixel. In effect there is a 1:1 pixel to processor mapping in blocks of 32 × 32 = 1024. Assigning one ray per pixel in this manner permits 1024 rays to be sent into the scene simultaneously, thus, allowing the concurrent determination of colour/intensity for 1024 pixels. This is possible because the behaviour of each simulated light ray is independent of any other. Therefore, no local neighbourhood operations are required between the processors. On completion of each job, the processor may be shifted to the next patch. This is performed until the entire image is traced. The ray tracing algorithm of Figure 4.3 can easily be transported onto an SIMD architecture. However, although the DAP hardware can perform recursion, its organisation makes it inefficient owing to misuse of storage space. Iteration using a simple stacking system has been used to overcome this, i.e. data from each level in the recursion corresponds to data at an equivalent level in the stack. Unwinding the recursive segments in this manner ensures the parallelism is maintained as efficient as possible. This is demonstrated in Appendix A.

Using the mapping methodology of one pixel to one processor, large images may be traced with a great reduction in time. For example, if a single processor required 1 unit of time to trace a light source through one pixel, then in order to trace a 512×512 pixel image, 262,144 units are required. If $32 \times 32 = 1024$ pixels can, however, be traced in a single time unit (concurrently), then only 256 units of time are needed to trace the entire image! However, we should note that a factor of 1024 speed up is not obtained since the single bit processors used in the DAP 510 are not as sophisticated as those in single processor machines. Also, there is the additional overhead of uniform termination on an SIMD machine so many of the processors can be idle towards the end of the tracing process.

4.4 Representation of Ray

The viewing model used is the pinhole camera model (the standard in image processing) with the +Z axis inside the screen (left handed coordinate system). Rays projected out of the screen converge at the focal point - the eye position. A ray is simply represented as a line in 3D space with an origin and direction vector [Roth82]. The origin is given in a parameterized form as (Ox, Oy, Oz) with the direction as (Dx, Dy, Dz). Since the algorithm is performed in parallel, each processor holds the origin and direction vector for its individual ray. The t-value, which is used to reference a point along the ray, can be used to determine the position at any point (X, Y, Z) in world coordinates. Thus:

$$\begin{aligned} X &= Ox + t \times Dx \\ Y &= Oy + t \times Dy \\ Z &= Oz + t \times Dz \end{aligned}$$

Note that the t-value is the distance from the origin to the point (X, Y, Z) along the ray. The direction vector is maintained as a unit vector by normalisation. This simplifies later ray-surface intersection calculations.

4.5 Illumination and Shadows

A point in shadow is due to the path of a ray from the light source to that point being obstructed by another object in the scene. This is determined by firing a ray called a shadow feeler, from the point, towards the light source. If the ray collides with an object before reaching the light source then the point is in shadow. Self-shadowing occurs when the cosine of the angle between the surface normal and the ray to the light source is less than zero, i.e. the surface is facing away from the light source. When this occurs, the point must be taken as being in shadow.

The illumination model used is a modified Phong distribution [Kuch88]. It models ambient as well as diffuse and specular lighting effects. Ambient lighting is the effect produced by a diffuse, non-directional light source in the scene and is assumed to impinge on all surfaces from all directions. The amount of ambient light reflected by a surface is therefore dependent on the material properties of that surface. Diffuse and specular effects, in contrast, are dependent on the position of a point light source with the object surface. This positioning of the surface is related to the surface normal at the point in question.

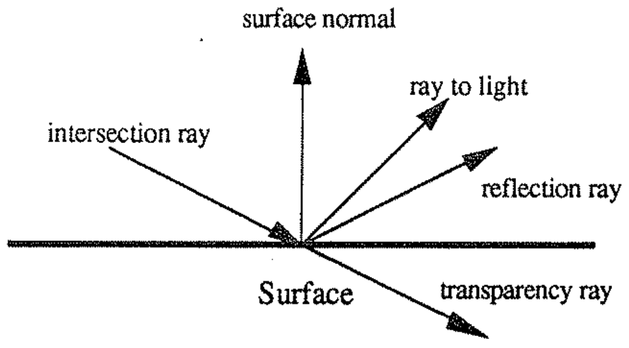


Figure 4.4: Illustration of light rays

The reflection ray from the surface is obtained from the intersection ray and the surface normal with the angle between the intersection ray and the surface normal being equal to that between the normal and the reflection ray.

The intensity/colour of the point is the summation of each component of the ambient, diffuse, and specular lighting. The initial colour of the intersection point is that of the ambient light. The diffuse component is then added to this as obtained from the dot product of the surface normal and the ray to light vector. If this is less than zero, then the surface is facing away from the light and so no diffuse light contribution is present. This dot product when multiplied by the brightness of the light (1.0 maximum, and 0.0 minimum) and the diffuse surface colour gives the diffuse light contribution.

The specular highlight is computed in a similar way, as the dot product of the reflection ray and the ray to the light. If greater than zero, this dot product is raised to the power of the surface specular coefficient which determines the 'shininess' of the surface. The specular lighting contribution is then the product with the specular surface colour.

If ray tracing, the colour/intensity of the transparency and reflection rays must also be included to the present evaluation of the colour/intensity at the point being shaded. For reflection,

$$Colour_p = Colour_p + \Re \times Colour_r$$

where $Colour_p$ is the present evaluation of the colour for the point and $Colour_r$ is the colour at the point of intersection of the reflected ray with another object. If no intersection is encountered, the background colour is used. $\Re$ is the reflectivity of the surface being shaded and ranges from 0.0 (non-reflecting) to 1.0 (a perfect mirror).

The transparency contribution is included in a similar fashion using

$$Colour_p = (1.0 - T) \times Colour_p + T \times Colour_t$$

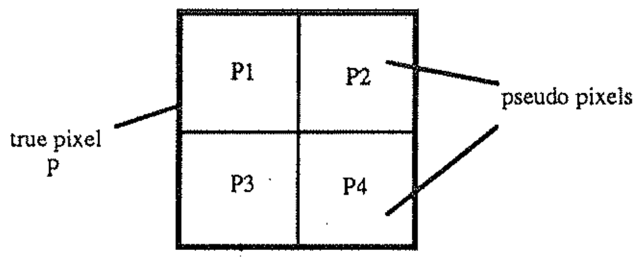
where again $Colour_p$ is that for the point being shaded, T is the transparency factor of the surface - 0.0 for opaque to 1.0 for completely transparent. As with reflection, $Colour_t$ is the colour at the intersection point of the transparency ray with a corresponding object. Again, if no intersection is detected, the background colour is used. Note that, in this particular research theme, no internal reflection is possible and a transparency ray passes directly through the object generating it.

As each lighting component is added, the illumination of the surface increases differentially, and a more realistic picture of the surface emerges. The image can be further enhanced with the addition of multiple light sources. They can create shadows and all round illumination of a scene which increases depth and realism. The illumination algorithm must then take into account the effect of each light source by iteratively testing the point of interest against them.

4.6 Alias and Anti-aliasing

An 'alias' is a term used in signal theory to describe the unwanted correlations in a signal produced when a high frequency signal is sampled at too low a rate. Alias is, therefore, produced in ray traced images when too few rays, with respect to the level of detail, are used to represent each pixel. The information returned by this ray will not contain enough detail for the colour/intensity of the object point with which it collides. This is especially relevant at object boundaries, where a pixel may represent the background and its immediate neighbour represents an object surface. The low sampling rate will result in a jagged look to the object boundaries.

Techniques for correcting this, 'anti-aliasing', generally involve increased sampling of the scene. This is well suited to the DAP's distributed ray tracing technique. A bundle of rays are sent out into the scene. These rays check for collisions with objects and return with the object-id of the object with which it collided, if any. Zero is returned if there was no collision. If this object array contains greater than one object-id the whole patch of scene is re-sampled at a higher resolution. This was originally achieved by sending a ray through the centre of each quadrant of a pixel. The average of the four pseudo-pixel's intensities was then taken as the intensity of the true pixel as shown below [SS87].



$$I(P) = \frac{I(P1) + I(P2) + I(P3) + I(P4)}{4}$$

Figure 4.5: Production of pseudo-pixels

Although this gives a more accurate intensity for the pixel, it suffers in that:

i) the computation involved increases more than linearly.
ii) the ratio of the increase in resolution to the work done is low.
iii) objects smaller than the pseudo-pixel size are not sampled correctly.
iv) straight forward averaging does not take into account the fact that pseudo-pixels nearer the centre of the true pixel should give a greater contribution to the intensity than those at the boundary.

For these reasons (especially (iv)), sampling was performed by a weighted averaging scheme put forward by [Crow81]. This was based on statistical theory to give a greater weighting for central pseudo-pixels. Each pixel is divided into a 3 × 3 pseudo pixel of the following weights.

1	2	1
2	4	2
1	2	1

The intensity of the true pixel will therefore be:

$$I = ((I_{11} + I_{13} + I_{31} + I_{33}) + 2(I_{12} + I_{21} + I_{23} + I_{32}) + 4I_{22}) \;/\; 16$$

Not only does this cater for point (iv), but the ratio of the increase
the work done becomes more favourable.

4.7 Primitive Representation and Intersection Calculat

Objects in computer graphics may be modelled in a numb
representation sought must be one suitable for manipulation by the comp
important when factors such as speed, storage, and simplicity co
representations which are frequently used include swept volume solids a
representations. Here we choose to model the primitives as quadric fur
the form:

$$f(x,y,z) = Ax^2 + 2Bxy + 2Cxz + 2Dx + Ey^2 + 2Fyz + 2Gy + Hz^2 +$$

or alternatively in matrix form

$$P^TQP=0 \qquad \text{where} \qquad P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad \text{and} \qquad Q = \begin{pmatrix} A & B \\ B & E \\ C & F \\ D & G \end{pmatrix}$$

They are quite straightforward to describe and manipulate
Our computations are made more convenient by the use of homogeneo
The representation also has the advantage that a wide range of pri
and defined in a uniform manner by altering the coefficients of m
planes, spheres and cylinders are a subset of the total number [Hecl
mations when applied to quadrics generate new quadrics. For insta
tion matrix,

$$\text{let} \qquad q = MP \qquad \text{and} \qquad P = M^{-1}q$$

then

$$q^T(M^T)^{-1}QM^{-1}q = 0$$

$$q^TQ'q = 0 \quad .$$

Note, however, that the class of quadrics may interch
instance, a cylinder may become a cone, and a sphere can become

The quadrics may be combined into other objects s
consisting of six quadric planes or three parallel plane pairs. E
coordinate system and intersections of a ray with a primitive
the ray from the world coordinate system to the coordinate sy

solving the equation of the quadric [WBB87, Roth82]. This transformation of the ray into the coordinate system of the primitive simplifies the calculations as it is easier to transform the ray than to transform the object. Here, only point and vector transformations are important. Consider a 4×4 transformation matrix T and a point in 3-space P = (X, Y, Z). Transformation of P by T would result in

$$P' = P * T, \quad \text{or} \quad (X', Y', Z', 1) = (X, Y, Z, 1) * T$$

where P' = (X', Y', Z'). Transformation of the vector V = (Dx, Dy, Dz) by T becomes

$$V' = V * T, \quad \text{or} \quad (Dx', Dy', Dz', 0) = (Dx, Dy, Dz, 0) * T$$

where V' = (Dx', Dy', Dz'). The transformation of a ray or line is achieved by simply transforming its origin O, and direction vector D. Thus

$$\begin{aligned}(Ox, Oy, Oz)\,(Dx, Dy, Dz) * T &= ((Ox, Oy, Oz, 1) * T)\,((Dx, Dy, Dz, 0) * T)\\ &= (Ox', Oy', Oz')\,(Dx', Dy', Dz')\end{aligned}$$

The line parameterizations, or t-values, are found to be independent of the coordinate system when transformed in this way. That is, a point P'(t) = P(t) * T for all t, where P(t) is a point in the original ray and P'(t) is a corresponding point on the transformed ray for the same value of t. The parameter independence ensures that we do not need to transform parameters or points between coordinate systems but just the rays. This relieves us of the greater problem of transforming object models or surface equations.

As already mentioned, each primitive is defined in its own local right-handed coordinate system as in Figure 4.6.

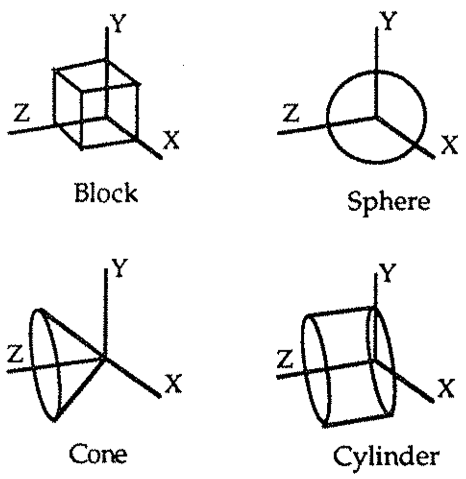


Figure 4.6: Representation of objects

Ray-quadric intersection calculations within the local coordinate frame of the primitive are simplified by defining them with unit dimensions and vertices, ends, or centres at the origin, i.e. all primitives are bounded. The block is defined as a unit cube in the positive octant with a vertex at the origin. The cone also has its vertex at the origin and lies along the positive Z axis with unit radius and length. The sphere is defined with unit radius and centre at the origin, and the cylinder primitive similarly has unit radius and length and again lies along the positive Z axis with one end at the origin. The effects of translating, rotating, and scaling a primitive is achieved by translating, rotating, and scaling the ray with the appropriate transformation matrix.

With the exception of the sphere, at least two ray-quadric intersection tests must be made for each object. For example, in order to determine the entry/exit points of a ray with a block, six ray-plane intersection tests must be made. For the cylinder, this reduces to two ray-plane and one ray-quadric intersection test. One ray-plane and one ray-quadric test is required for the cone. The intersection point where the ray meets the object (if any) is obtained by solving the surface equation for the quadric and applying the bounds test (as given in Table 4.1) where necessary.

Primitive	Surface Equations	Bounds Test
Block		
plane 1	$X = 0$	$0 \leq Y, Z \leq 1$
plane 2	$X = 1$	$0 \leq Y, Z \leq 1$
plane 3	$Y = 0$	$0 \leq X, Z \leq 1$
plane 4	$Y = 1$	$0 \leq X, Z \leq 1$
plane 5	$Z = 0$	$0 \leq X, Y \leq 1$
plane 6	$Z = 1$	$0 \leq X, Y \leq 1$
Cylinder		
plane 1	$Z = 0$	$X^2 + Y^2 \leq 1$
plane 2	$Z = 1$	$X^2 + Y^2 \leq 1$
curved surface	$X^2 + Y^2 = 1$	$0 \leq Z \leq 1$
Cone		
plane	$Z = 1$	$X^2 + Y^2 \leq 1$
curved surface	$X^2 + Y^2 - Z^2 = 0$	$0 \leq Z \leq 1$
Sphere	$X^2 + Y^2 + Z^2 = 1$	none

Table 4.1: Surface equations of primitives

Given a ray definition

$$P = O + tD$$

expressed as

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} Ox \\ Oy \\ Oz \\ 1 \end{bmatrix} + t \begin{bmatrix} Dx \\ Dy \\ Dz \\ 0 \end{bmatrix}$$

or in partitioned form:

$$\begin{bmatrix} R \\ 1 \end{bmatrix} = \begin{bmatrix} O \\ 1 \end{bmatrix} + t \begin{bmatrix} D \\ 0 \end{bmatrix},$$

then substituting into the equation of a quadric $P^T Q P = 0$ produces the expression

$$t^2(D^TQD) + t(D^TQO + O^TQD) + O^TQO = 0$$

since D_4 is zero by definition [WBB87]. This is a quadratic equation of the form $at^2 + 2bt + c = 0$ and has solutions

$$t_1 = \frac{-b + \sqrt{b^2 - ac}}{a} \quad \text{and} \quad t_2 = \frac{-b - \sqrt{b^2 - ac}}{a}.$$

Thus if

i) $(b^2 - ac < 0)$: the ray does not intersect the quadric.
ii) $(b^2 - ac = 0)$: the ray is tangential to the quadric.
iii) $(b^2 - ac > 0)$: the ray *may* intersect with the quadric - calculate t_1 and t_2.

For case (iii), if both t_1 and t_2 are less than zero, the surface is not visible as it is behind the viewpoint. If only one is less than zero then the viewpoint is inside the object and the positive t gives the ray-quadric intersection. And obviously, if both are greater than zero, the smaller t gives the entry intersection into the quadric, and the larger t the exit intersection point.

Thus, to intersect a parameterized ray:

$$(Ox, Oy, Oz)\ (Dx, Dy, Dz)$$

with the Y-Z plane of the block, simultaneously solve:

$$X = 0 \quad \text{and} \quad X = Ox + t * Dx \quad \text{for t.}$$

We can see that the result, $t = -Ox/Dx$ defines the point of intersection:

$$(0, Oy + t * Dy, Oz + t * Dz).$$

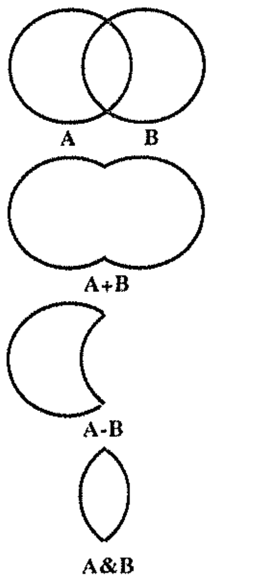
The ray would have only hit the solid if this point complies with the bounds test, i.e. it lies within the bounds of the primitives, where the bounds test for this point on the Y-Z plane of the block is:

$$(0 \le (Oy + t * Dy) \le 1) \text{ and } (0 \le (Oz + t * Dz) \le 1).$$

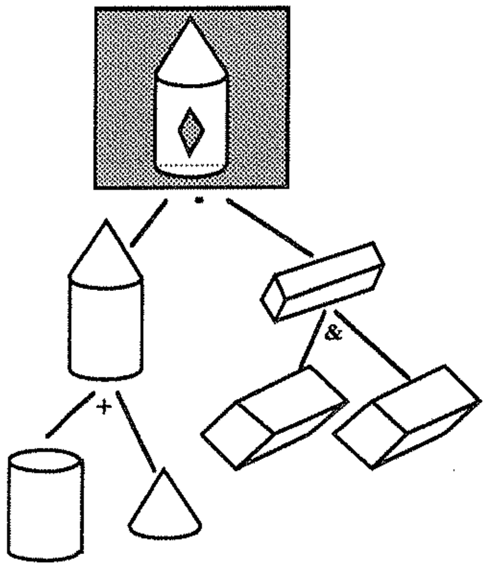
In order to determine the intensity of the surface, the outward surface normal at the point of intersection must be known. Once the t-value is available, this normal vector is easily calculated as the partial differential of the quadric function of the surface, thus the vector $[df/dx, df/dy, df/dz]$ for a function $f(x, y, z)$.

4.8 Constructive Solid Geometry (CSG)

Although various types of primitives may be created using quadric functions, a greater and more complex formation may be created by the application of Constructive Solid Geometry. This involves the combination of the primitives using the binary algebra operators UNION(+), INTERSECT(&), and DIFFERENCE(-). Each CSG operator requires two arguments, although either or both of them may be NULL. The order in which the operators are applied is determined by constructing a binary tree in which the terminal nodes represent primitives, and the non-terminal nodes the CSG operators. The composite object at the root of the tree will, therefore, represent the final object as shown below.



a) Application of CSG operators



b) Example of CSG Composition Tree

Figure 4.7: Formation of CSG tree

The CSG tree is defined recursively as follows:

<CSG tree> ::= *<primitive>* |
<CSG tree> *<operator>* <CSG tree> |
<CSG tree> *<rigid transformation>*

where *<primitive>* is a solid primitive such as sphere, cone, box, or plane, and is defined using the quadric representations. *<operator>* gives the binary operator +, &, -, and *<rigid transformation>* is a combination of the translation, rotation, and scaling transformations which can be applied to the tree. It is noted that, as the binary operations cannot destroy boundedness, the CSG models are guaranteed to define bounded sets.

The user determines the tree structure to be analysed. This is passed to the host module which, i) makes it left heavy, and ii) converts it to a reverse Polish format. The organisation of the DAP hardware means that these two transforms can reduce the number of planes used for stack space by the DAP. In addition, point ii) makes it possible for the implementation of a simple stacking system and the representation of the tree as two simple lists with one containing the CSG operators, and the other the primitive-ids. Traversing the tree now becomes a matter of running along the lists and pushing and popping the required data planes on and off the stack.

As mentioned above, the host module performs the reformatting of the user defined tree. This remodelled tree is passed to the DAP TREE_WALK module which is responsible for applying the combination operators. As the module parses the tree nodes' list, each element is checked for an operator. If the node represents a primitive, i.e. it is a terminal node, a bundle of rays (32 × 32 = 1024) is sent into the scene. These return with the entry and exit points of the ray with the primitive, if any. The information is placed on the user stacking system as a list of entry/exit points. If a ray did not intersect the object, then zero is returned. It is worth stressing that the only information that needs to be stored are the t-values for the entry and exit points of the ray with the object and the identification number (object-id) of that object. These are maintained in a vector list structure which can be efficiently manipulated in parallel on the DAP. Both the entry/exit and the object-id lists are sorted in order of ascending t-values. When a combination operator is encountered the top two entry/exit lists are popped off and passed to the COMBINE module which applies the operator to them. The primitives are combined according to an in-out classification [Roth82]. This means classifying the parts of the ray which are outside the object as *out*, and those parts that are inside as *in*. The resulting list is now pushed back onto the stack.

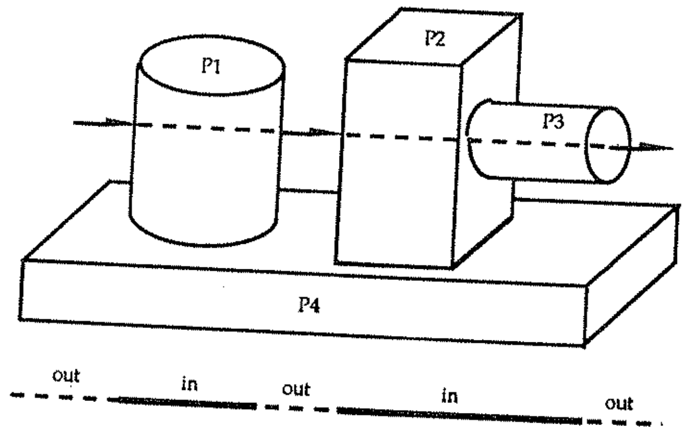


Figure 4.8: Classification of ray passing through 3 primitives

A ray classification must be generated for each branch of the CSG tree, i.e. one for the left branch and one for the right. Thus, for a simple case, the classification of two rays with the left ray passing through two objects, and the right through one are shown in Figure 4.9. Note that the solid lines represent the *in* classification and the dashed lines the *out* classification.

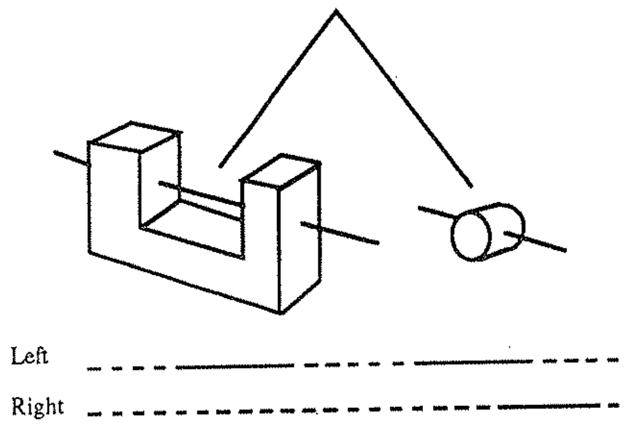


Figure 4.9: Classification of left and right rays

Using this in-out classification of rays, the composite object results from combination of the two rays with the appropriate CSG operator (Figure 4.10). Note that the difference operator (-) specifies the 'in' values which are present in the left ray but not the right.

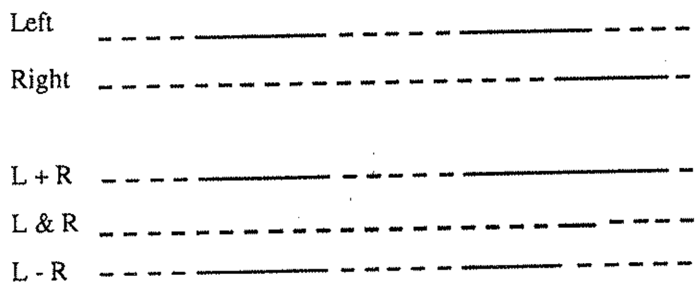


Figure 4.10: Combination of rays

The composite solid after applying each operator is shown below.

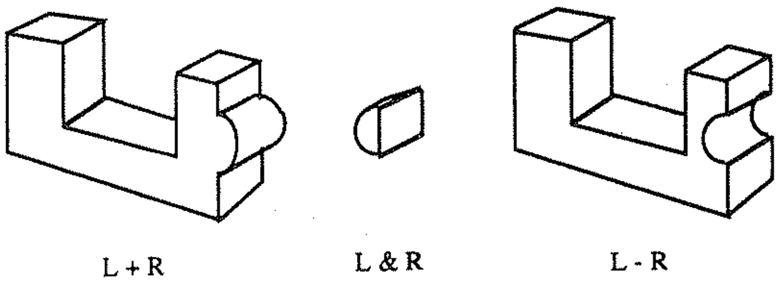


Figure 4.11: Resulting solids after application of operators

Combining the classifications is just simple boolean algebra with the composite result as in Table 4.2.

Operator	Left	Right	Composite
+	IN	IN	IN
	IN	OUT	IN
	OUT	IN	IN
	OUT	OUT	OUT
&	IN	IN	IN
	IN	OUT	OUT
	OUT	IN	OUT
	OUT	OUT	OUT
-	IN	IN	OUT
	IN	OUT	IN
	OUT	IN	OUT
	OUT	OUT	OUT

Table 4.2: Result of ray combinations

The combination of the rays is outlined in three steps. First, the lists are merged and sorted into ascending order, i.e. the entry and exit points of both lists are combined to determine the order with which objects are met. Second, the resulting list is then scanned from minimum entry point to maximum exit point and the points are classified in or out according to the operator. The third and final stage involves noting only the initial and final entry points of each composite solid. The resulting list obtained from the combination is then pushed back onto the stack. After the tree has been traversed, the final list of the entry/exit points determines that for the top-level object. From this list of 32 × 32 rays, the very first entry point of each ray is taken. This determines the object which is visible to the eye. This entry point, the object-id at that point, along with the surface normal are all passed to the shading routine of the ray tracer which returns the colour/intensity at the point. An example of the parallel combination procedure is demonstrated in Appendix B.

Points in shadow are determined by traversing the CSG tree again but this time sending rays out from the light source towards the point. If, after traversing, the object-id of the point is the same as that 'seen' from the light source, the point is not in shadow. If it is different, and the surface is not transparent, the point is classified as being in shadow as the light is clearly not reaching the nearest object surface to the observer. On interaction of the shadow feeler with a transparent object, the intensity of the light eventually reaching the point of interest is reduced by a factor dependent on the transparencies of any object encountered.

A simple optimization procedure for CSG traversal, termed as 'early outs' by Roth [Roth82] can be seen from Table 4.2. If at a composite mode the left ray subtree is classified as out, the composite classification will also be out irrespective of the right subtree classification. Thus, classifying the right subtree is unnecessary. This not only reduces the number of ray-surface intersections performed, but also the number of nodes traversed.

4.9 Spatial Subdivision

A major step in reducing the processing time during a ray tracing procedure is the reduction of the number of ray-surface intersection calculations [Whit80]. This may be achieved using *a priori* knowledge about which surfaces a ray is most likely to intersect and then testing only these. If reflection and refraction are required (ray tracing), then a 3 dimensional spatial subdivision is required to reduce the number of intersection calculations. For our closed-form testing, however, ray casting is sufficient and we may use a two dimensional space subdivision. This is the basis of the *Tiling* algorithm [Slat86] which was previously proposed as a solution for damage repair of geometrically defined objects on bit mapped display devices. It involves tessellating the two dimensional view plane into an $divx \times divy$ grid, where $divx$ is usually equal to $divy$. Each grid cell is referred to as a 'tile' and maintains a list of all primitives intersecting or contained by it from projections at the eye position. These are obtained by a preprocess stage where rays are sent from the eye position into the scene checking for possible intersections within a tile. Hence, during ray casting, only those primitives whose identities are held by the tile through which the ray is passing need be tested for a possible intersection. This immediately suggests that empty tiles are not tested for ray-primitive intersections and may be assigned the background colour. Since only a limited number of primitives are likely to pass through a tile, this greatly reduces the number of intersection calculations during ray casting and solves the problem of testing all primitives against each ray.

The previous SIMD implementation of the CSG rendering procedure can be enhanced by permitting tiles generated by the two dimensional spatial subdivision to be mapped onto the processors of the DAP. Since the space subdivision process produces a series of tiles, it would be appropriate to generate a much reduced CSG tree for each tile. As each tile will contain a subset of the primitives, these pruned trees will generally be much smaller than the overall CSG tree. By identifying which surfaces are local to the tiles, the overall combined rendering times for the tiled CSG trees is much less when compared to the untiled scene.

The CSG preprocess phase responsible for generating the pruned trees consists of one main procedure to localize primitives for each tile. It is possible to calculate the vertical and horizontal ranges over which the primitives are visible and consequently determine the tiles which are intersected without the need to scan the entire screen [Malh90*a*]. The intersected region will be within the silhouette of the primitive where the silhouette edge is formed when the Z-component of the normal is zero when projected into view space. The intersected primitives can then be stored per tile as appropriate. The preprocessor then traverses the main CSG tree for each tile checking for ray-surface intersections. At each leaf node, if the primitive is present in the tile it places a true value, T, at that node in a new tree. If not present, a false value, F, is inserted. When combining branches of the tree according to the operators, the result is dependent on whether the original primitives exist in that tile. If not, the whole branch and, hence, subtree may be pruned off. The efficiency of the Tiling algorithm is most evident when rendering large CSG trees, say, for example, upwards of 20 primitives. Each tile may contain 2-3 primitives (or more depending on the dimensions of the tile). This would immediately reduce the time devoted to ray-surface intersection calculations by a factor of 12. Serially, an optimal tile size has been shown to be 8 × 8 pixels [Malh90*b*]. However, this is not a universal tile size for all solid models, since this parameter is highly dependent on the composition of the CSG tree. On the DAP 510, it is best to use 32 × 32 tiles. Using Fortran*, 8 × 8 pixel tiles are possible, but this is futile as it will result in 960 redundant processors.

4.10 Modelling Language

As closed-form testing of a vision system was the main application of the ray tracer, we required a language which was simple to use even when modelling quite complex objects. A WINSOM type modelling language [Quar84] was developed for this. It allows the user to specify a complex CSG tree as a collection of simpler operations. The primitives BLOCK, CYLINDER, SPHERE, and CONE are understood and transformations such as rotations about the x-, y-, and z-axis are given by values of XROT, YROT, and ZROT respectively. Information such as the primitives' surface properties, including reflection and transparency, are also provided in the user specified data-file which is read in and processed by the host machine. A skeletal example for the model shown in Figure 4.8 is given below.

```
bigtube        = CYLINDER ( 8, 0, 0, 25) XROT (-90) AT (10, 0, 13);
smalltube      = CYLINDER ( 4, 0, 0, 10) YROT (90)  AT (45, 16, 18);
tallbox        = BLOCK ( 15, 25, 16) AT (30, 0, 10);
flatbox        = BLOCK ( 50, 5, 26);

pair1          = bigtube UNION flatbox;
pair2          = tallbox UNION smalltube;
object         = pair1 UNION pair2;

DRAW ( object ) AT (0, 0, 100)
```

Figure 4.12: CSG data file

As discussed earlier, the tree structure of interest is converted to a post ordered list of primitives and operators. Operations, or properties, such as surface type can easily be applied to any defined tree structure and tree structures may be combined using any of the CSG operators. The symbolic association of 'names' with the CSG constructions makes it possible to reuse a particular tree structure several times and maintains this as a powerful language for solid modelling.

4.11 Program Structure

The program is structured as modules, each one handling a different data-type. They are linked to the ray tracer via the TRACER module. This is responsible for controlling the generation of rays and their subsequent firing into the scene. Additional data-structures such as Digital Terrain Maps (DTM) may be included in the program structure. The module must, however, convert the data-structure to a list structure compatible with the tracer.

On entering the program, the module handling the appropriate data-structure is invoked. It processes local information to a standard format necessary for the TRACER module. This, then, determines which rays have intersected with which objects in the scene and returns the colour/intensity of the pixels at these points. Non-intersection points are masked with the background colour.

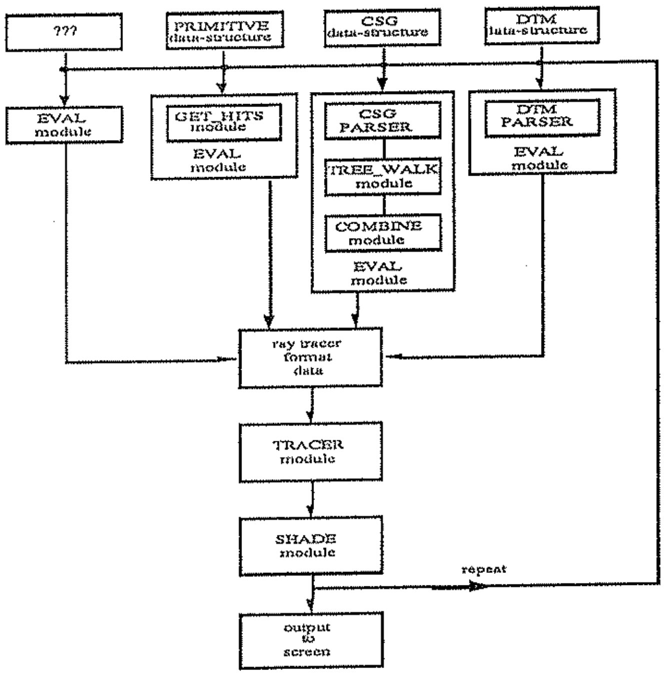


Figure 4.13: Program modules

From Figure 4.13, the data for each data-structure is processed by it's own evaluation function which is located in the EVAL module. This returns scene data in a format which is acceptable to the ray tracer. The TRACER module acts as a common link between the various evaluation modules. It sends rays into the scene as 32 × 32 bundles to be processed in parallel and returns with a map of t-value intersections. If no intersection was found a value of zero is returned. The normals at these points are computed and passed to the SHADE module which determines the colour for that pixel. This information is then directed to the screen. The process is cyclic and is repeated for subsequent 32 × 32 tiles.

The EVAL module of the CSG data-structure is constructed with three major modules; CSG PARSER, TREE_WALK, and COMBINE. The CSG PARSER module is responsible for the generation of the CSG tree and it's conversion to a postordered list of objects and operators. The list is scanned by the TREE_WALK module to produce left and right subtree lists of entry/exit points. These are combined according to the CSG operator by the COMBINE module. The key to the generality is the clear division between model specification, and generic surface rendering, where the depths and surface normals of the nearest surfaces have been calculated as above.

4.12 Closed-Form Testing for Vision

Images of polyhedral objects which are required for closed-form testing of the vision system can be produced efficiently by the parallel ray tracer. However, the ray tracer is limited to a single level search (i.e. ray casting) for the specific purpose of the closed-form testing. Image data may be produced and translated, rotated, and scaled at will. This makes for a very effective visualising technique for a system where several images of the same model, taken from slightly varying viewpoints, may be required. The ray casting (when compared with ray tracing) is very fast making it possible to produce sets of image sequences in a few minutes.

The classic chipped block model often found in computer vision is able to be produced using just two block primitives and the difference operator. It's matching features are the 7 faces or, where edge matching is utilised, 13 edges.

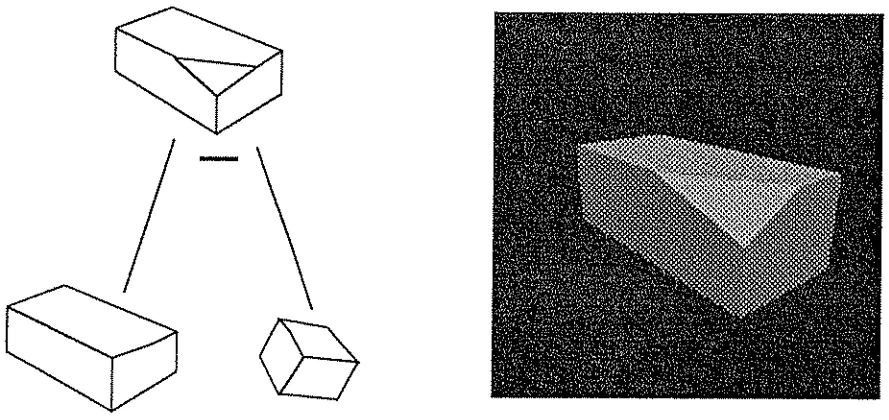


Figure 4.14: Chipped block CSG tree and model

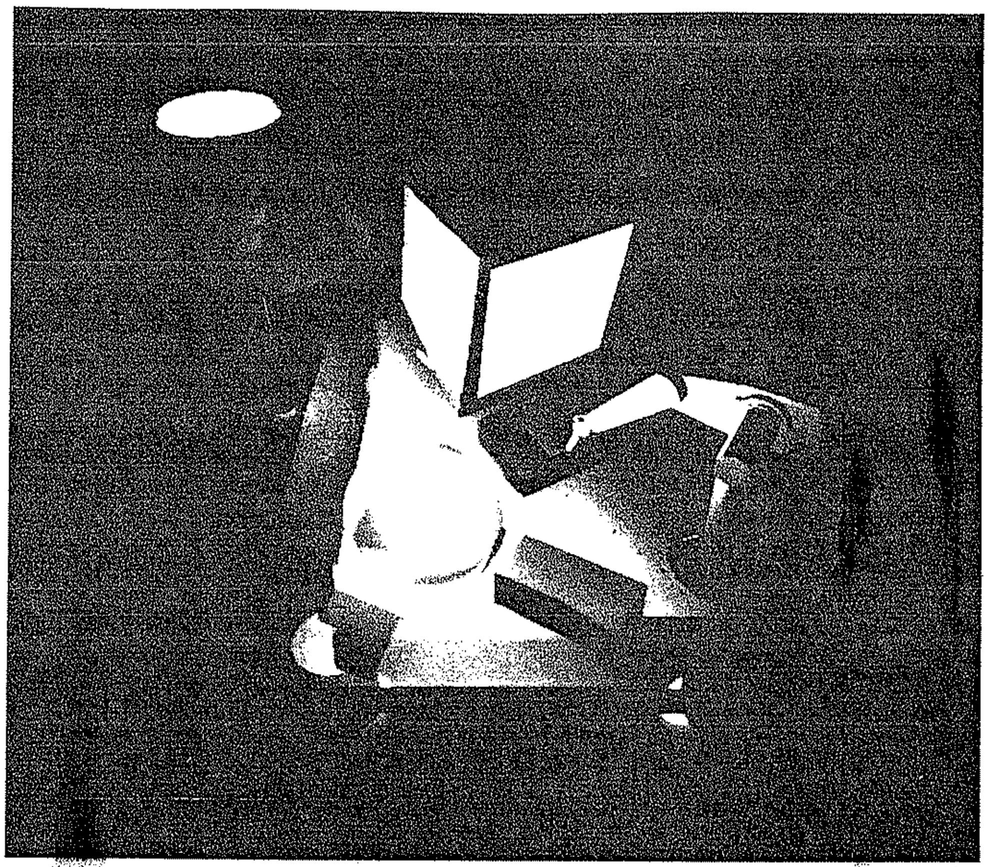
More complex models can be efficiently generated using the modelling language. Plate 4.4 (in Section 4.13) shows a model of an oil rig structure produced using 44 block type primitives. This has 498 edges and is a fairly large model where matching is concerned.

The images are stored in the form of *image file format* (iff) files. The model features such as vertices, edges, or faces, required for the model-based recognition algorithm must, at present, be obtained by manual computation. Automation of this step and the recovery of matching features directly from the CSG tree is possible but is beyond the scope of this thesis.

4.13 Experimental Results

We now present results obtained from the integrated solid modelling system described in the preceding sections. Plate 4.1 illustrates the power of the ray tracing algorithm. The ability to include the properties of reflection and transparency into a primitive surface is demonstrated in 4.1(a) with the addition of shadows in 4.1(b). They were generated using 100 primitives illuminated by 3 light sources. These images, produced using a 2 level deep trace are, however, too 'flamboyant' for our purpose which is specific to closed-form testing. The shadowing and properties of transparency are somewhat unnecessary. However, this does not mean that we cannot use such images in our tests. On the contrary, such scenes may be included in more detailed investigations to truly demonstrate the vision systems accuracy and robustness in a computer generated, 'real world' scene.

Aside from the unnecessary addition of reflections and transparencies in the Table scenes, the time required in rendering is of great importance for our application. Plate 4.1(a) was produced in 20 minutes and Plate 4.1(b), with the additional overhead of shadow computation, was generated in 75 minutes. They may, however, be reproduced (though less aesthetically) in a shorter period of time by limiting the trace level to a single level search (ray casting). Plates 4.2 to 4.6 show test scenes rendered by ray casting using the parallel ray tracer. Each scene is illuminated by three light sources and has a resolution of 512×512 pixels thus requiring 256 32×32 DAP matrices to render.


(a) Properties of reflection and transparency

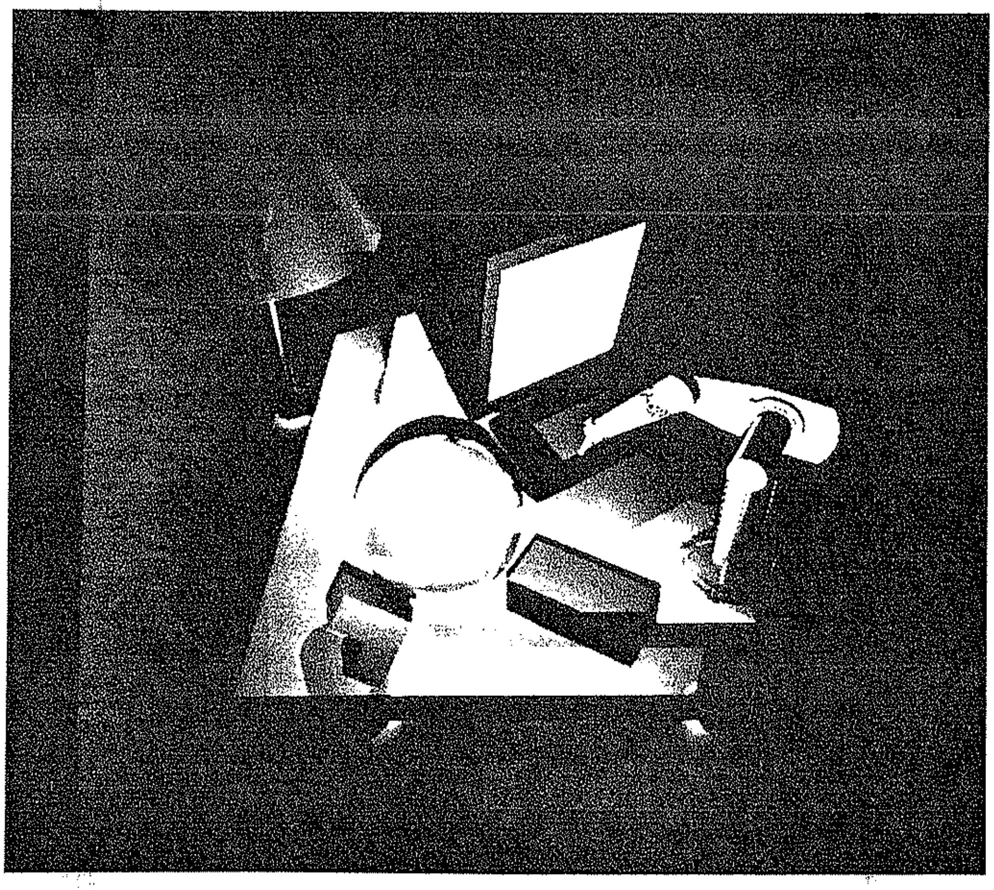

(b) Properties of reflection, transparency and shadowing

Plate 4.1: Table scene

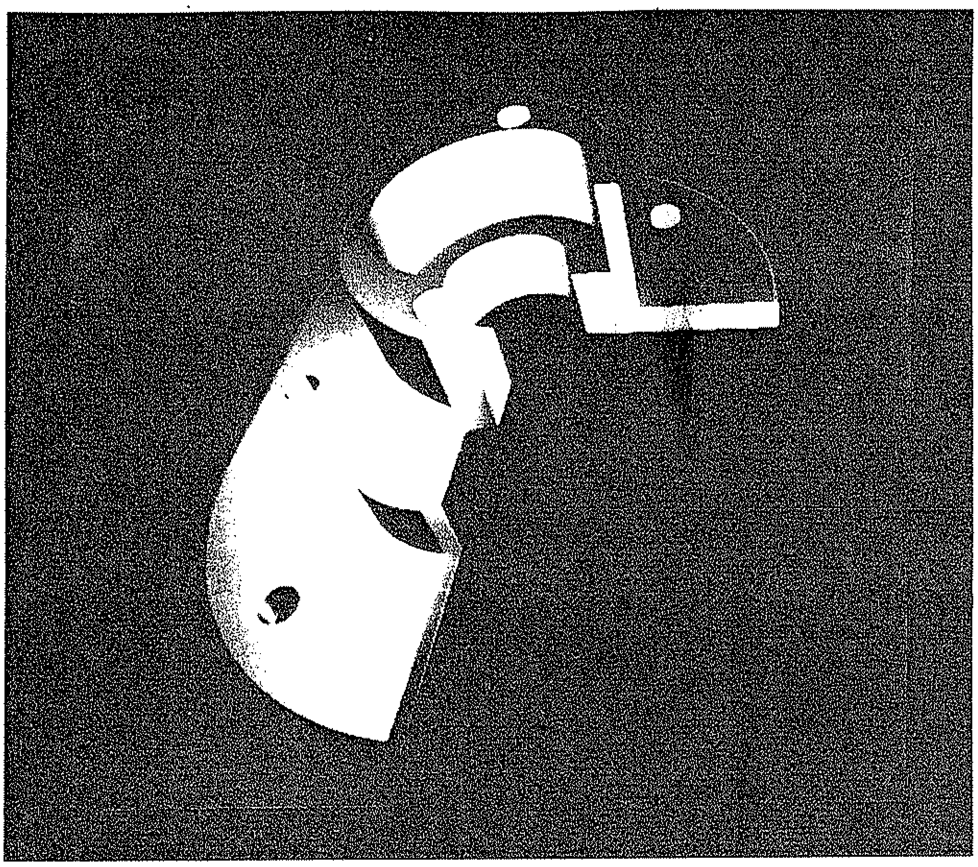


Plate 4.2: Coverplate scene

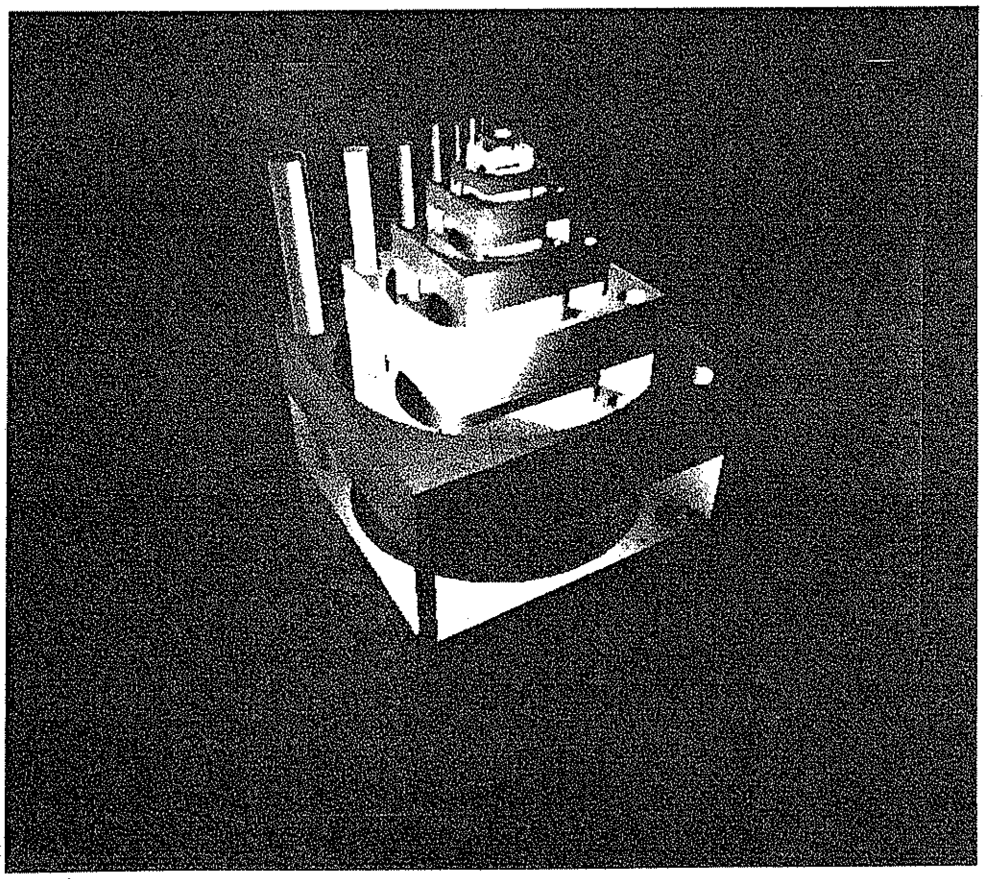


Plate 4.3: Recursive Cubes scene

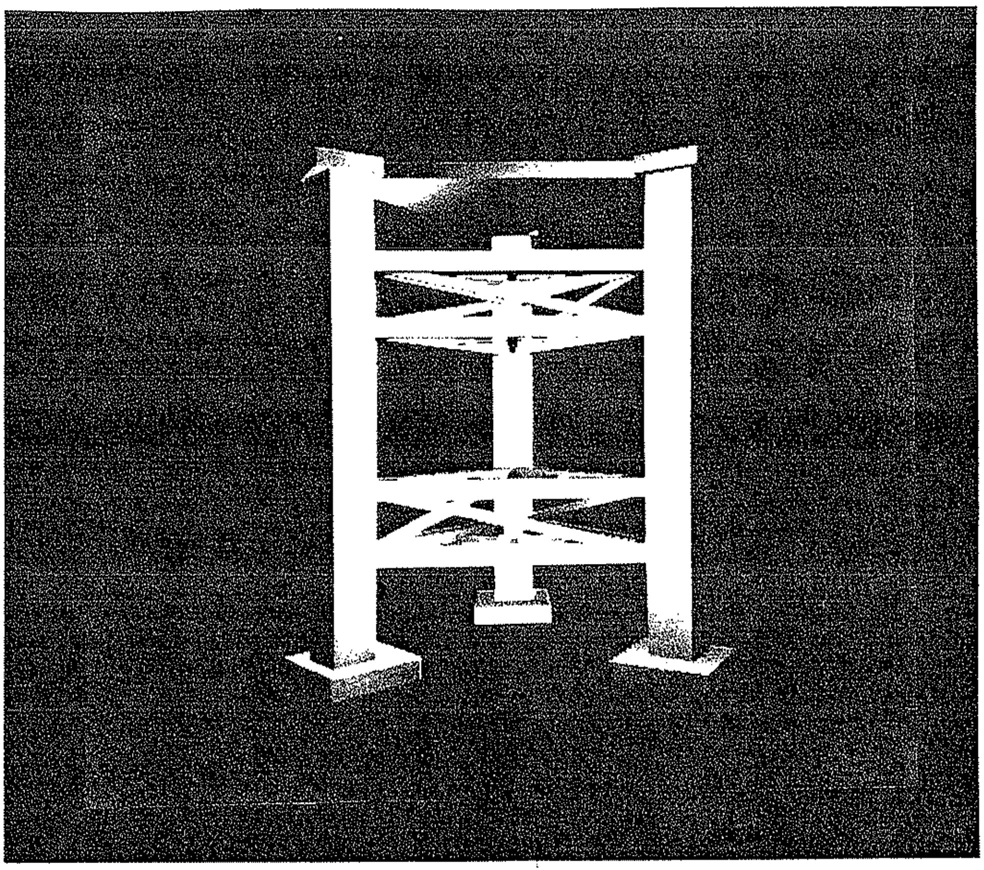


Plate 4.4: Oil Rig Structure scene

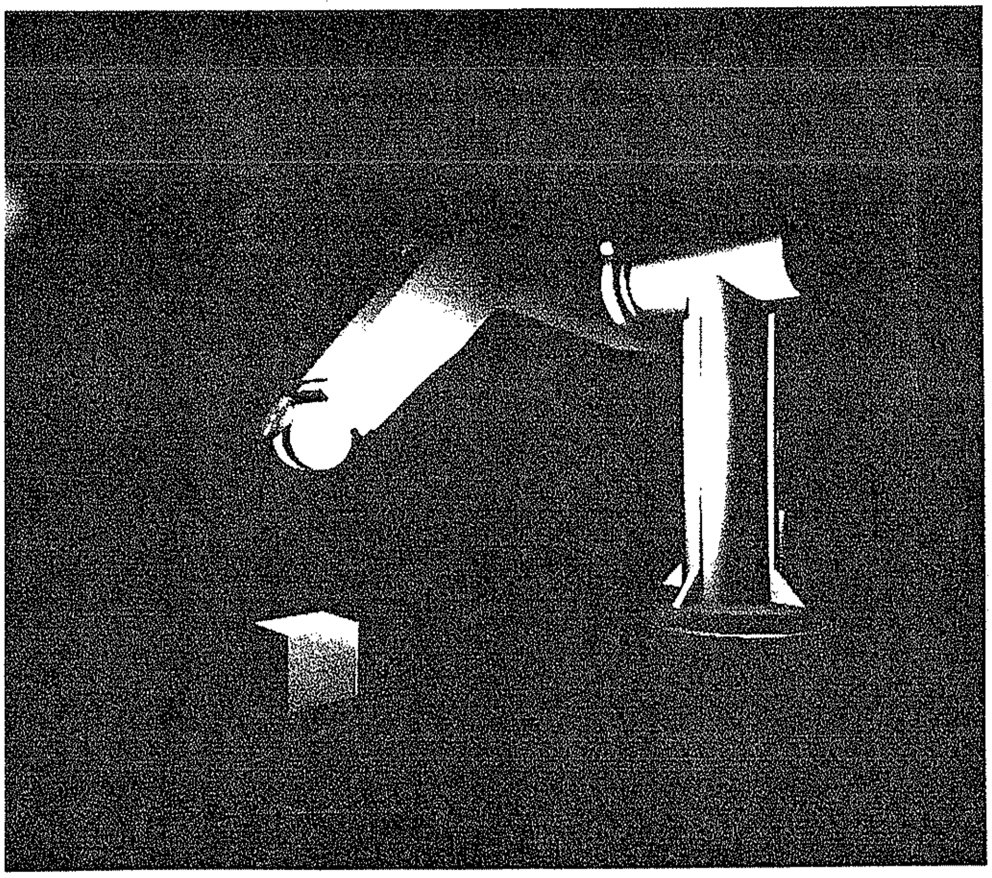


Plate 4.5: Robot Arm scene

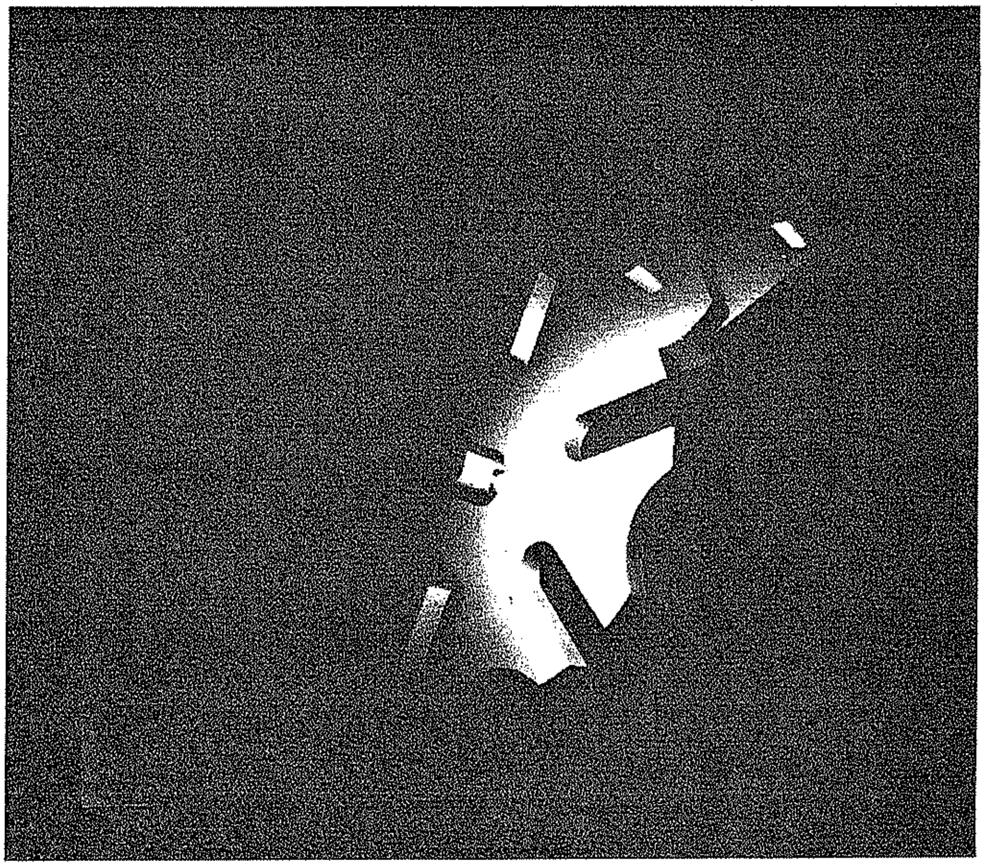


Plate 4.6: Geneva Wheel scene

The complexity of the scenes, in terms of primitive content and halfspaces, is shown in Table 4.3. Here, a 'halfspace' is the surface which is able to be represented by a quadric function. Figure 4.15 demonstrates the percentage of those pixels in a scene which are required to render it, i.e. the number of pixels containing a visible primitive.

Scene	Number of Primitives	Number of Halfspaces
Coverplate	25	93
Recursive Cubes	42	147
Oil Rig Structure	44	264
Robot Arm	63	258
Geneva Wheel	38	147

Table 4.3: Complexity of test scenes

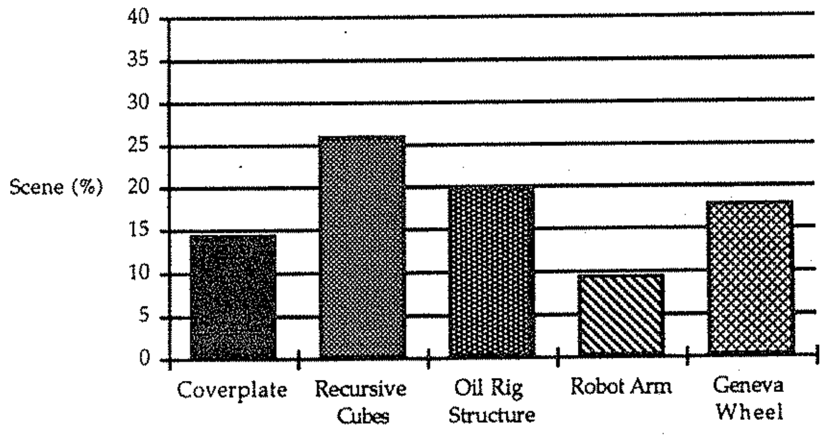


Figure 4.15: Percentage of visible pixels in scene

In the following text we compare and contrast the ray tracing algorithm as implemented on the parallel architecture of the DAP 510 to that implemented serially, in the C programming language, on a Sun 3/160 workstation running under the UNIX operating system. The two programs are coded as similarly as possible to ensure an accurate representation of the comparative times. Comparisons are also made between the two dimensional Tiling versions of the algorithm implemented on both architectures. We show that the introduction of the Tiling algorithm produces significant speed ups of the scene rendering times of each test image.

Note, however, that due to the large difference in timings of the serial and parallel implementations, many of the results have been plotted logarithmically for display purposes.

4.13.1 Serial vs Parallel - A Brute Force Approach

In this section we compare the experimental data obtained from test runs of the ray tracing algorithm as implemented on the Sun 3 workstation and on the parallel DAP 510. This is a Brute Force implementation (non-tiled) whereby a ray is fired into every screen pixel. As expected, the parallel implementation greatly out-performs its serial equivalent. Figure 4.16 demonstrates this for the five tests scenes. It can immediately be seen that the parallel version appears favourable for the visualisation task, rendering scenes in minutes rather than hours! On average, it shows an 83.8 fold speed up over the serial version. The minimum speed up is exhibited by the Recursive Cubes with a value of 56.6 times, and the maximum is shown by the Oil Rig structure with 145.4 times.

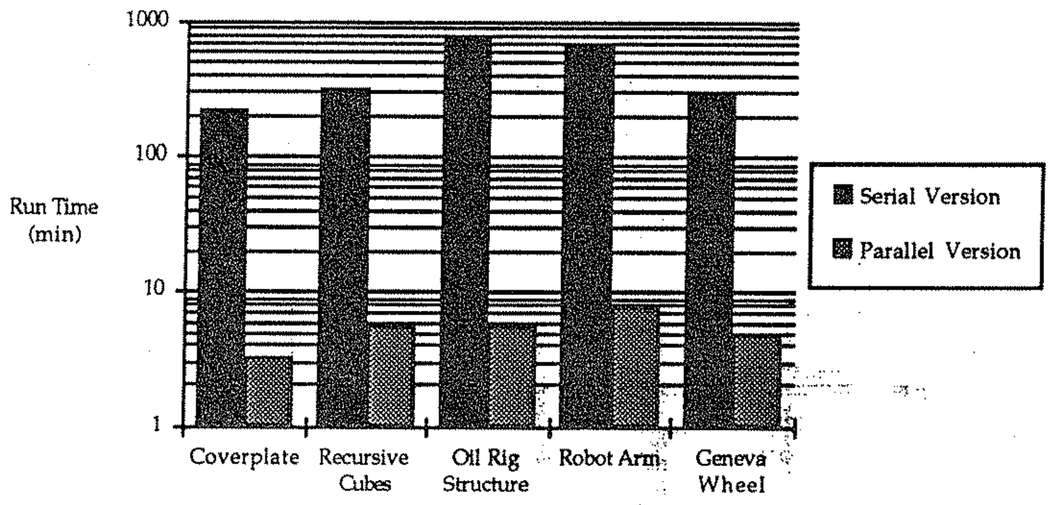


Figure 4.16: Total CPU expenditure of serial and parallel naive implementations

This very significant speed up for the Oil Rig scene is directly related to the vast number of intersection calculations made by the serial version. With an average of 5.7 ray-surface tests per primitive, this contributes greatly to the rendering time when compared to an average of 1.6 ray-surface tests per primitive for the Recursive Cubes. This large average value for the Oil Rig scene is due to the majority of the primitives being Blocks (which require 6 intersection calculations). Thus, even though the Oil Rig and Recursive Cubes have roughly the same number of primitives, 44 and 42 respectively, and the Recursive Cubes adopts a larger proportion of the scene than the Oil Rig, the difference in the total number of intersection tests made is most significant in affecting the rendering times. The vast speed up of the parallel version for the Oil Rig scene may therefore be explained by realising that during the intersection tests, 1024 intersections can be determined in parallel. For a Block requiring 6 intersection tests, this parallelism becomes more effective. Note, however, that although a speed up of 145 may be considered large, it is far from the 'target' of ~1024 which may be naively expected. This is because of the simple bit processors of the DAP which are not comparable with the serial machine's CPU. However, we note that the new DAP 510C, a 32 × 32 DAP with additional 8-bit parallel co-processor chips for complex arithmetic is able to deliver a times ten speed up over the single-bit DAP 510. In order to put the different architectures in perspective, we also note that a Sun 4/260 workstation was found to give a mean speed up of 17 over the Sun 3/160 for the test scenes.

4.13.2 Serial <u>vs</u> Parallel - The Tiling Approach

Pixels necessary to render a scene are generally considered to be those which contain 'visible' primitive surfaces. A primitive which vanishes from the scene using the difference operator may be thought as being non-existent as its visibility is the prime property in which we are interested. From Figure 4.15 in Section 4.13, it is obvious how much of a scene is *white space*, i.e. background. Time spent in these pixels is wasteful and futile, only being able to return the background colour. The two dimensional Tiling algorithm, however, can be used not only to reduce the size of the CSG tree traversed by each ray, but can also be used to prevent the unnecessary execution of 'empty' tiles.

A serial and parallel implementation of the Tiling algorithm is compared here for the five test scenes. Tiles of 32 pixels square are used. However, we also present results obtained using 8 × 8 tiles in the serial implementation as it was found to be an effective tile size (see Section 4.9). From the results obtained, the time expenditure of the preprocessing phase necessary to construct, and build, the much smaller tile trees is insignificant when compared to the total CPU time expenditure, especially when considered in conjunction with the speed up it produced. An average value of 1.01 seconds was recorded for the combined cost of building the tiling array, and the subsequent generation of the tile trees.

Figure 4.17 illustrates the results of the total CPU expenditure times of the two Tiling implementations. As expected, these run times are considerable less than the Brute Force method in the previous section. For the serial version, scenes are rendered in 10's rather than 100's of minutes. Also, as expected, the parallel version shows a significant speed up over its serial counterpart in all the test scenes. The average was found as a factor of 89.0

times, with the minimum being 63.8 times for the Robot Arm, and the maximum, 105.7 times for the Geneva Wheel (using 32 × 32 tiles).

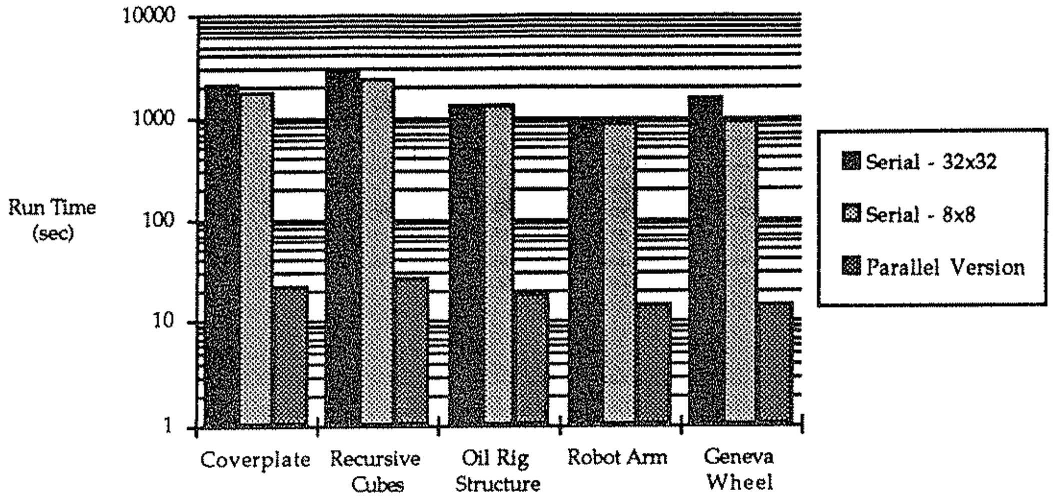


Figure 4.17: Total CPU expenditure of the serial and parallel Tiling implementations

Note, however, that the Oil Rig structure does not demonstrate the greatest speed up (74.4 times) unlike in the Brute Force experiment (145.4 times). This can be explained by noting that the Tiling algorithm does not any longer investigate 'empty' sections of the scene which before contributed so much to the Brute Force technique's drawback.

Also, even though using 8 × 8 tiles is found to improve the speed of scene rendering for the serial implementation, it still does not compare with the results delivered by the parallel version.

4.13.3 Parallel vs Parallel - Comparing Brute Force and Tiling

We have shown, in the previous sections, how a parallel implementation of the ray tracing algorithm is considerably superior to an equivalent serial method using both Brute Force and Tiling methodologies. We now investigate in more detail the implementation of the two techniques using SIMD parallelism in terms of those factors which most greatly effect their ability to render a scene efficiently.

Initially, we can compare the overall CPU time expenditure in the two techniques. Figure 4.18 demonstrates this. From these results we can see the effectiveness of the Tiling algorithm over the Brute Force method. Speed ups of a factor of 18.1 are evident. The minimum speed up factor of 8.7 is exhibited by the Coverplate, and the maximum, of 31.6, is shown by the Robot Arm.

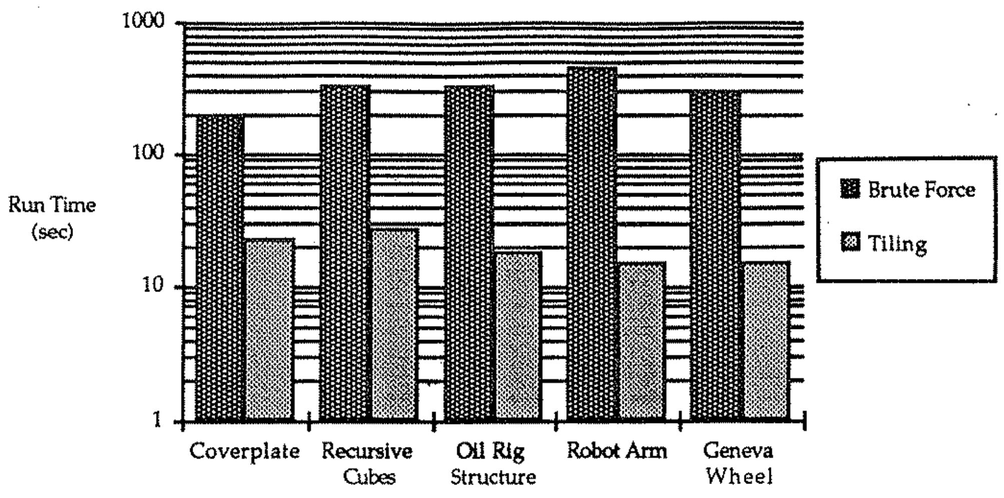


Figure 4.18: Total CPU expenditure of parallel Brute Force and Tiling implementations

In an attempt to explain the above findings, we consider the proportion of the scene actually rendered in the Tiling version. Table 4.4 shows the number of tiles rendered for each scene, and Figure 4.19 shows these values as a percentage of the 256 potential scene tiles.

Scene	Total Tiles Rendered
Coverplate	95
Recursive Cubes	93
Oil Rig Structure	90
Robot Arm	50
Geneva Wheel	77

Table 4.4: Total scene tiles rendered

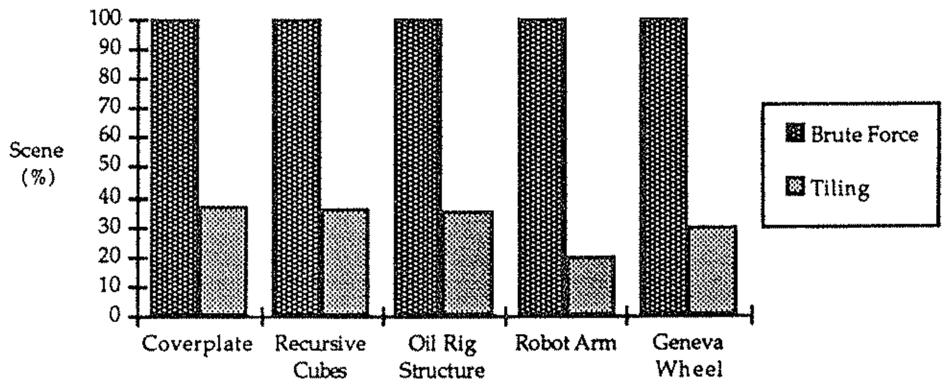


Figure 4.19: Total scene tiles rendered as percentage

We can now see (though, maybe naively) that the calculated speed ups have some relationship with the number of tiles rendered in a scene. Indeed, the tessellation of the scene into tiles, and the subsequent generation of tile trees (which are much smaller then the original scene CSG tree) proves very effective given the speed ups obtained, and considering that the tiling procedures account for an average of 1.01 seconds of the overall rendering time for each scene.

Figure 4.20 shows a comparison of the average number of primitives rendered per 32 × 32 DAP matrix for the two techniques. Figure 4.21 shows the mean rendering time for each rendered DAP matrix.

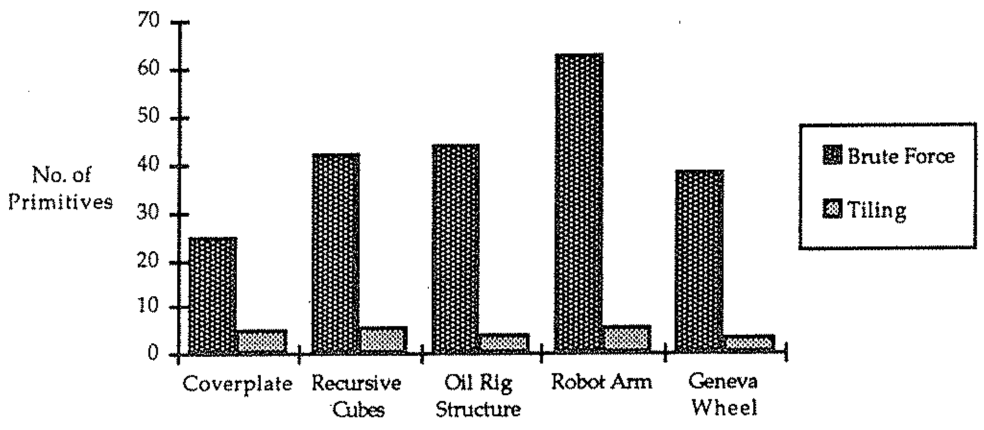


Figure 4.20: Average no. of primitives per 32 × 32 DAP matrix

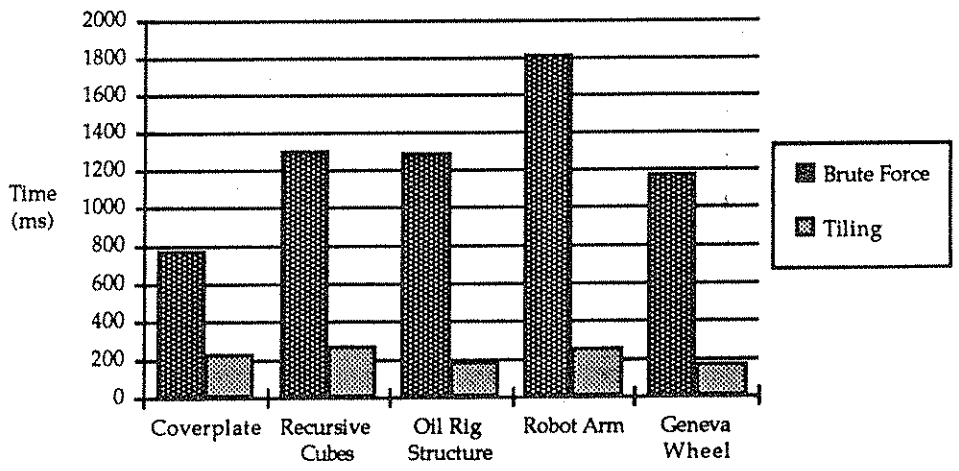


Figure 4.21: Mean rendering time (ms) per 32 × 32 DAP matrix

The Tiling method displays a much smaller tile tree compared to the original CSG tree - an average of 12.5% of the original CSG tree. This is significant when compared to the 100% value of the CSG trees rendered by each DAP matrix in the Brute Force version. This pruning preprocess of the Tiling algorithm proves very effective in reducing the overall size of the CSG trees traversed and, thus, the number of intersection calculations made. To show this, we consider the data given in Figures 4.22 and Figure 4.23.

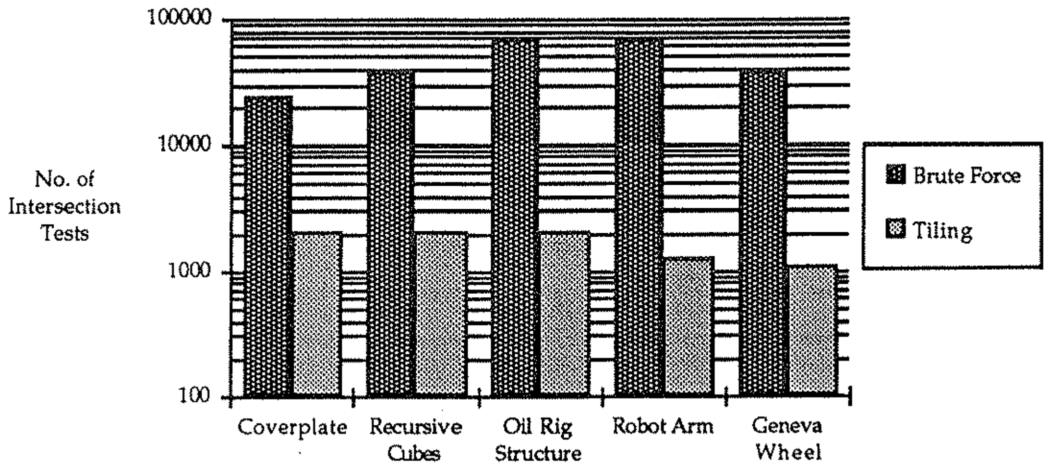


Figure 4.22: Total intersection calculations performed

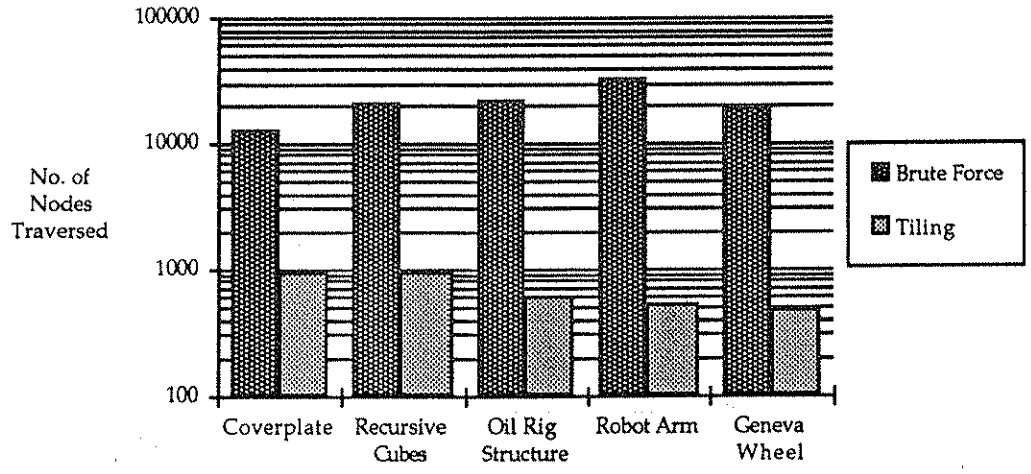


Figure 4.23: Total number of nodes traversed

The total number of intersection calculations have been greatly reduced in the Tiling method - by a factor of more than 90% in all cases, and 98.1% in the case of the Robot Arm. This clearly confirm the speed ups over the Brute Force method shown earlier. These values are a direct consequence of the heavily pruned CSG tree which results in far fewer nodes being traversed in the Tiling version. Again, reductions of the number of nodes traversed of greater than 90% of the Tiling method over the Brute Force approach are seen for all test models.

Also, the relative size of the CSG tree is related to the total number of combine calculations needed for evaluating it. We can see from Figure 4.24 the relationship of the total number of combine calculations made during rendering of the test scenes in the two techniques.

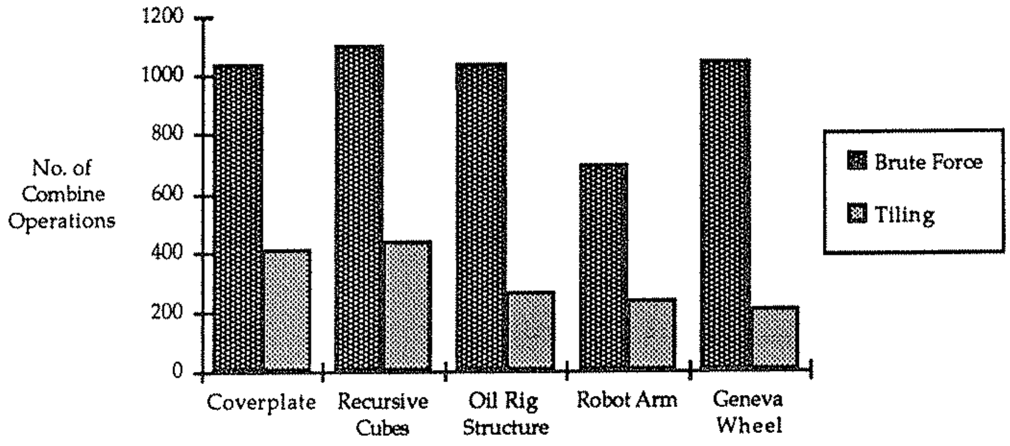
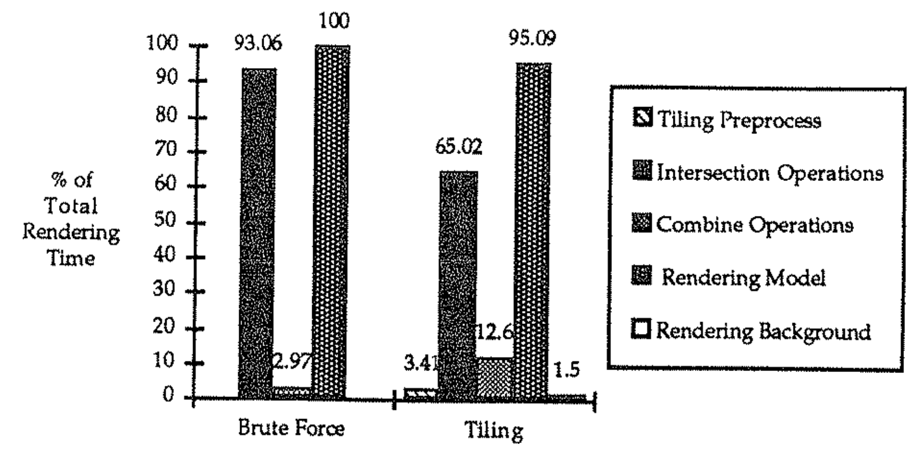


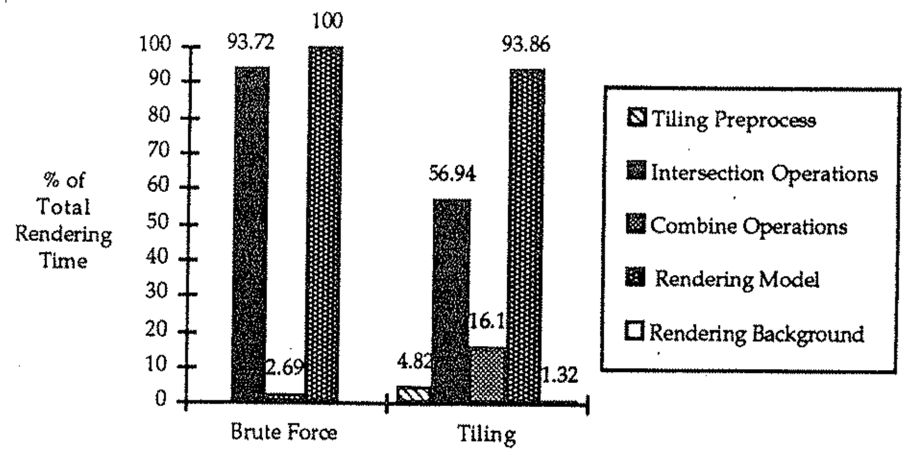
Figure 4.24: Total number of combine operations performed

An average percentage reduction of 68.6% of the total number of combine calculations of the Tiling method over the Brute Force procedure is found. A maximum reduction of 80.4% is demonstrated by the Geneva Wheel.

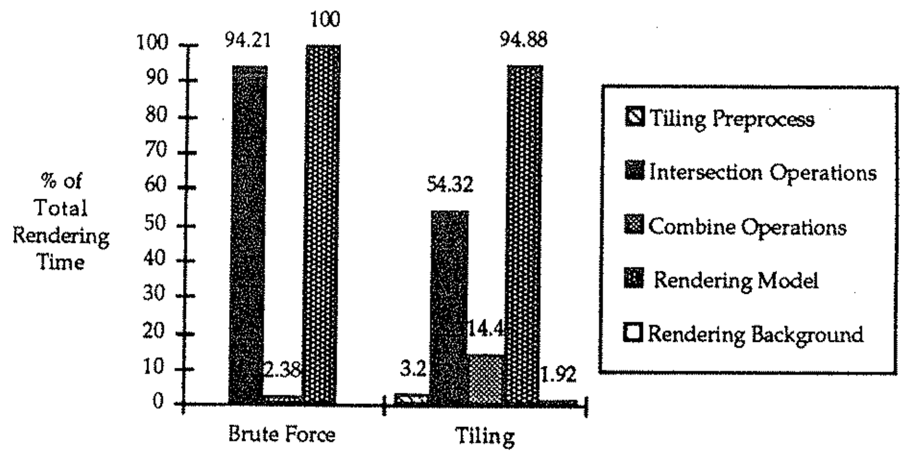
In order to accurately determine in what manner the Tiling algorithm is faster than the more naive, Brute Force approach, we must consider the time spent in the various procedures and the percentage of this over the total CPU expenditure (Figure 4.25). We note that the total rendering time for a scene is composed of Tiling Preprocess, Rendering Model, and Rendering Background. The time required for Intersection and Combine Operations account partly for the Rendering Model expenditure.



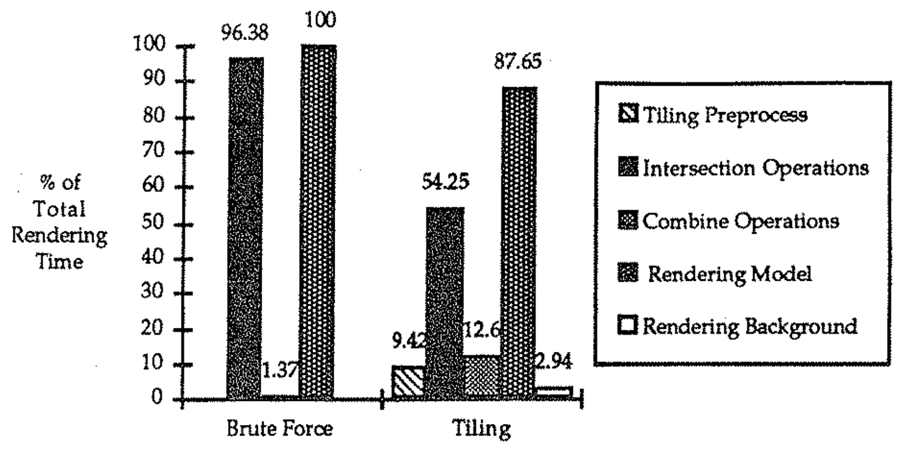
a) Coverplate Scene



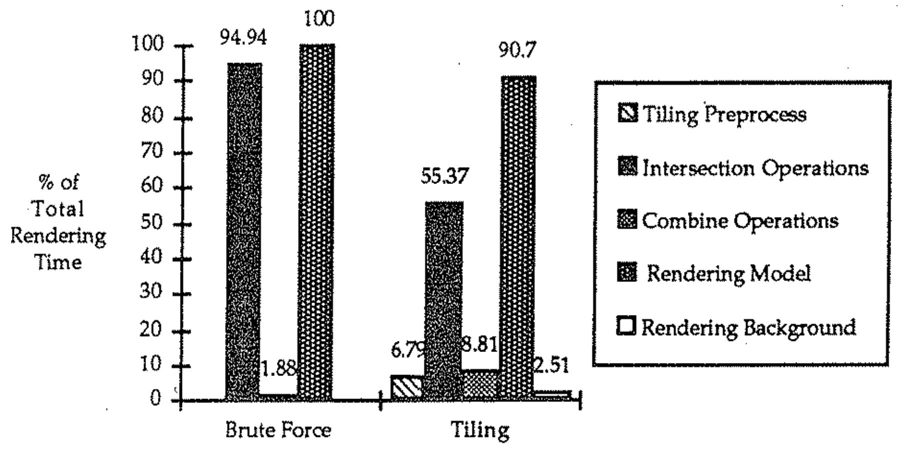
b) Recursive Cubes Scene



c) Oil Rig Scene



d) Robot Arm Scene



e) Geneva Wheel Scene

Figure 4.25: Profile of procedures of Brute Force and Tiling implementations

It can be seen, for the Tiling implementation of each scene, that the total time required to render the tile trees approaches 100% of the program execution time. This confirms that, in the case of Tiling, the preprocess and house keeping procedures necessary for the technique are, in fact, very conservative in their demands of the CPU. This is indeed evident from the graphs where the average percentage time required by the Tiling preprocessor is 5.5%.

We can see in all the test scenes, and in both ray tracing techniques, that the intersection procedure accounts for the largest expenditure of CPU processing time. This confirms Whitted's observation that about 90% of ray tracing time is spent on intersection calculations. For the Brute Force approach, their values are indeed close to this even though the trace is a single level search. In the case of the Tiling algorithm, this value is reduced primarily because of the tiling CSG tree pruning phase.

We must stress, at this point, that the values obtained for the proportion of the time required by the CSG tree combination procedure is somewhat deceptive. In actual CPU time, the Tiling implementation's combination procedure requires much less of the CPU than its Brute Force partner. However, because of the large increase of the total rendering time (i.e. Brute Force over Tiling), their values, as a percentage of the total rendering time, does not maintain the raw differences seen in the comparisons of the CPU expenditure.

4.14 Conclusion

It seems that SIMD distributed ray tracing is suited to the DAP by mapping each pixel to a processor but, in some ways, the technique is made more difficult due to the organisation of the DAP hardware. Recursion may be performed but at the risk of inefficiency owing to the mis-use of storage space. However, an iterative method using stacks has been successfully implemented here. A single level ray trace (ray casting) remains a very powerful method of image generation on SIMD processor arrays. In particular, for our closed-form testing which requires image sequences,.or pairs of images, this is the technique that is most appropriate as it produces rendered scenes of sufficient quality in a very reasonable length of time.

The developed program has been used to produce images of high quality especially when anti-aliased. As aliases are most evident at object boundaries it seemed most logical to apply the anti-alias function only at such points. The overall quality of the image is improved and, at the same time, the speed of rendering is not reduced as much as would have been the case if the whole scene were anti-aliased. As speed is of the essence, rays were not traced for longer than necessary. This was achieved by first examining each bundle of rays for intersections. If no intersections were found, then the bundle was immediately returned with the background colour. This was the case for the Brute Force implementation of the algorithm.

Also, a parallel implementation of the Tiling algorithm has been shown to exhibit significant speed ups over the more naive version. However, reducing the number of primitives per tile by reducing the edgesize of each tile is considered to produce a more efficient execution of the algorithm. This, however, would have left a significant number of processors idle.

A major hindrance of the parallel algorithms was the difficulty in uniform processor termination. Often, especially along model boundaries, a large number of processors are idle because a lesser proportion have active ray intersections. A method of overcoming this would significantly enhance the speed ups recorded.

Summarising, we have described a generalised ray tracer developed using the SIMD parallel technique of Williams, Buxton, and Buxton. We have enhanced the technique by introducing anti-aliasing and, implementing a version of the Tiling algorithm which was shown to produce significant speed ups. We have also included various surface and lighting properties such as transparency and reflection, and specular, diffuse, and an approximation of ambient lighting. The generalised ray tracer also includes bounded primitives which is essential for realistic image production but was not implemented by Williams, Buxton, and Buxton. The completed software represents a substantial improvement for parallel ray tracing and allows the synthesis of realistic image sequences for closed-form testing with a fast, effective tool.

Chapter 5

Parallel Aspects in Model Matching

5.1 Introduction

Computational vision may be thought of as the 3-dimensional interpretation of a 2-dimensional image or images. Many situations exist where it is advantageous for machines to accurately perceive the 3D nature of their environment and possibly take action according to the information received. An immediate example is robotic vision where 'the key task for the robot's vision system is to supply the control unit with a quantitative and symbolic description of its surroundings' [MC88]. This may range from the visualisation of simple geometric objects to complex multivarying scenes. Even a simple problem definition 'to identify an object from amongst a set of known objects and to locate it relative to a sensor' can break down into a set of extremely demanding operations [GL-P84*b*].

The major problem with any recognition system is the task of investigating a vast number of possible interpretations of a scene, and to do so in real-time. The ability of humans to do this is often thought of as trivial but, on consideration, this is definitely not the case. Attempts have been made to reduce the process time wasted on spurious scene interpretations by adopting a number of local constraints which are applied to an 'interpretation tree' of the scene space. Other methods have included the introduction of parallelism on SIMD machines, namely the AMT DAP [HB89*a*], and the Connection Machine [FH85, SRS90]. A problem of the present implementation of Holder and Buxton is that the matching algorithm is applied to the faces of polyhedral objects with a limitation of 32 faces. The aim here, however, is to match to edges, not faces, and increase the number to a value dependent on the memory limitations of the machine. This will be achieved by matching the data to portions of the model and the information coalesced in the form of subgraph matching. The data edges may be recovered using the ISOR system [MCB89], or the TINA toolset [TINA] or an appropriate data recovery system. Viz, the data may be presented in a form with, or without, absolute length information. Another problem with the Holder and Buxton implementation is found in the ordering of matches between data and model which determines their subsequent traversal order during the search for consistency. We show that such ordering is weak and proves to be inefficient with any increase in the complexity of the model or the number of recovered data items. In addition, we present a new improved sorting for finding consistent interpretations which can result in speed ups of over a thousand times for the test scenes considered.

We note that the general model matching procedure we describe is closely related to that of Murray and Cook and some of the terms and definitions are much the same as those described earlier by them.

5.2 A Serial Approach to Model Matching

Many algorithms have been developed in recent years to match scene data to object models held in memory. They have relied on dynamically growing and pruning an interpretation tree built up from information received from the scene. The problem in most cases has been to identify and retrieve the orientation and location of an object from a set of known objects. The objects are modelled as polyhedra with six degrees of freedom (3 rotational, 3 translational) relative to the sensor. Information about the scene can be recovered with the aid of various sensing devices and techniques. These include tactile and three dimensional range sensors. Structure from motion and structure from stereo algorithms can be used to recover the 3D geometry of the data. The matching can be performed using the face or edge information of the polyhedra. The algorithm is basically the same, the only noticeable difference being that there will be more edges than faces in a particular model, and the constraint look-up tables will be different. An accurate geometric representation of the model is stored in memory and recovered scene data is compared systematically to this.

The basic procedure is outlined:

1) Build an accurate geometric representation of the model in memory.
2) Recover the data edges of an object in the scene of interest using an appropriate reconstruction algorithm.
3) Put the data in the form of an interpretation tree (IT).
4) From the IT generate a set of feasible interpretations using local constraints.
5) Validate the feasible interpretations by testing the global validity of each match for compatibility with the model.

Step 1 is a simple procedure merely involving the calculation of the 3D positions of the vertices (endpoints) and determining the edges of the model. The recovery of the geometry of the data edges in step 2 is achieved using a structure from motion algorithm applied to a set of three time varying image sequences obtained from a single camera, as in the ISOR system [MCB89], or a structure from stereo algorithm applied to an image pair using the TINA toolset. In contrast to structure from motion algorithms, the TINA toolset may be used to recover absolute size. The edges are recovered to within a cone of uncertainty.

5.3 Generating Feasible Interpretations

A feasible interpretation is one which is possible but has not yet been tested for global consistency. After sensing an object with *e* real edges, we can recover up to *s* sensed data edges. This can be formed into an interpretation tree with each node bearing *e* children down to a level of *s*-plys.

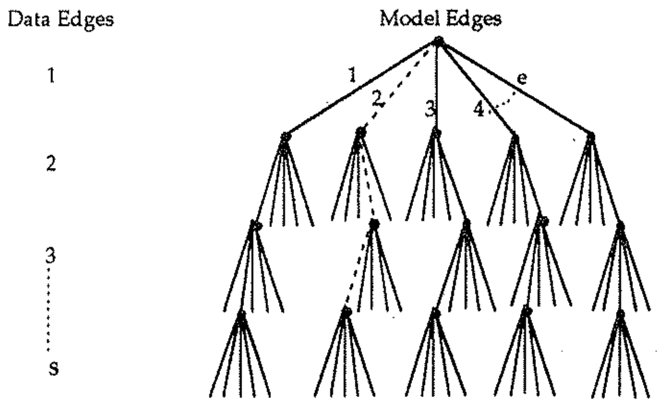


Figure 5.1: Interpretation Tree

Note that this is a bottom-up data based approach whereby the scene data is matched to the model primitives in the interpretation tree. This means that there may be multiple data edges on a single model edge and, therefore, the number of branches remains constant at all levels. The path to any leaf node in the IT completes an interpretation of the tree. Several interpretations are possible. Consider the path formed by the dashed lines in Figure 5.1. If there were 5 model edges, then the partial interpretation down to level 3, (where an interpretation is a path from the root node to any leaf node), concludes that it is possible for data edge 1 to be on model edge 2, whilst data edge 2 is on model edge 4, and data edge 3 is on model edge 1.

To put the problem of matching into perspective, let us consider an example. Given s number of sensed data edges to be matched against a polyhedral model with e model edges, the number of possible interpretations is e^s. Clearly this results in a combinatorial explosion of interpretations with an increase in the number of sensed edges. For a model possessing 12 edges and being tested with 5 sensed edges the number of possible interpretations stand at 248,832. If the number of recovered edges is increased to 8 there is a dramatic increase of

429,732,864 alternative interpretations to be considered. It is, therefore, not feasible to apply a brute-force search of the IT. The number of possible interpretations must be drastically reduced to a more manageable level.

The interpretations are limited by taking pairs of data edges and applying simple pairwise constraints to establish facts like:

```
if data edge a is matched to model edge i
  then
    data edge b
       can/cannot
    be matched to model edge j
```

These constraints maintain a consistency in the interpretation tree which becomes more powerful as we move to deeper levels. They are very effective in pruning whole sub-branches of the tree. As we move deeper into the tree, the work done in establishing a possible match initially increases but then falls rapidly as inconsistent matches are pruned out as shown below.

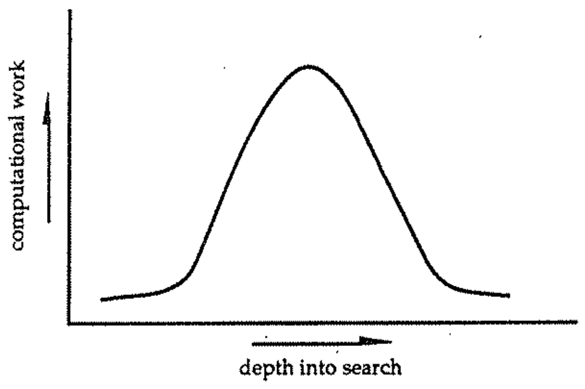


Figure 5.2: Work done during search

5.4 Sensing Errors and the Term 'Ideal Data'

It is obvious that during recovery of the 3D geometry of the data, errors will occur due to inaccuracies of the sensing equipment or algorithm. These must be taken into account if the matcher is to be applied to real data. The use of *ideal data* to test the matching algorithm cannot be guaranteed to give an accurate performance of the algorithm in the general case. By 'ideal data', we mean that if it were possible, data which could be recovered with no or, a negligible amount of error in the sensing equipment.

To account for the sensing errors we assume that the endpoints of the recovered data edge are located with uncertainty in an ellipsoid with *errpar* and *errper* sensing errors, as shown overleaf.

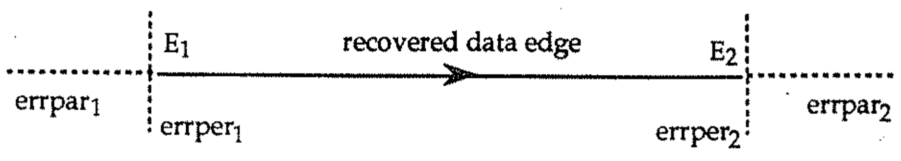


Figure 5.3: Sensing errors of recovered data edge

Errpar gives the uncertainty parallel to the data edge, and *errper* gives the perpendicular error. The uncertainty volume may easily be defined as a sphere where *errpar* and *errper* are equal.

The direction of the data edge (a vector) will now be uncertain with an error cone half angle α having a possible maximum value:

$$\alpha = \tan^{-1}\left(\frac{errper_1 + errper_2}{\max[\text{VERY\_SMALL}, (|E_2 - E_1| - errpar_1 - errpar_2)]}\right)$$

where VERY_SMALL is a very small number used to avoid an illegal division by zero.

These data edge error angles are now included in the constraint tests (detailed in the next section) to determine matched pairs. Also for the constraint tests, the data edges are reduced in length by an amount $errper_1 + errper_2$.

From the above equation, it should be noted that shorter edges are more sensitive to sensing error and produce a larger error cone half angle. This makes them free in matching to a larger proportion of model edges when the pruning constraints are applied. For example, a short data edge with error cone of 90° (which is not improbable) can have its direction oriented so significantly that it may pair with a large number of model edges which normally would not have been possible. In order to try and reduce the effects of this, a choice is given to limit the maximum error cone permissible for all edges.

We realise that the estimation of error from any sensing method is non-trivial especially 'real' stereo data. We feel, however, that although the quantification of error in two and three dimensional data is important, it is considered beyond the scope of this thesis. Experiments to check the performance of the algorithm with noisy or error-prone data have, however, been realised and may be performed by the addition of randomly distributed noise to the recovered data.

5.5 Pruning the Interpretation Tree

As only a few interpretations are consistent with the input data, we are able to establish this consistency by pruning the interpretation tree using local pairwise constraints. These constraints typically serve to prune out most of the nonsymmetric interpretations of the data.

The choice of constraints as suggested by [Grim84] and [Marr82] must be:

1) independent of any global coordinate frame;
2) simple and effective in reducing the size of the search tree;
3) made to degrade gracefully in the presence of noise;
4) independent of the particular sensing mode.

When matching is applied to structures obtained from visual motion, absolute size information is not available and, therefore, the constraints involve unit vectors or directions only. However, with the availability of absolute size information, full displacement vectors may be used. This simplifies extrema calculations required in determining the ranges of angles in the constraints tests. Also, an additional constraint may be included which utilises the length of each recovered edge fragment. The constraints used are referred to as the angle constraint, and the direction 1, 2, and 3 constraints. The additional constraint offered using absolute size information is termed as the distance constraint. They are defined thus:

- The Angle Constraint requires that if edge fragments a and b are assigned to model edges i and j respectively, then the range of possible angles between the sensed fragments must embrace the angle between the model edges.

- The Direction Constraint 1 requires that if edge fragments a and b are assigned to model edges i and j, then the range of angles between any vector P_{ab} from a point on a to a point on b, and the edge a itself must be wholly included in the angle between any vector q_{ij} between points on model edges i and j and the direction of model edge i itself.

- The Direction Constraint 2 requires that if edge fragments a and b are assigned to model edges i and j, then the range of angles between any vector P_{ab} from a point on a to a point on b, and the edge b itself must be wholly included in the angle between any vector q_{ij} between points on model edges *i* and *j* and the direction of model edge *j* itself.

- The Direction Constraint 3 requires that if edge fragments *a* and *b* are assigned to model edges *i* and *j*, then the range of angles between any vector P_{ab} from a point on *a* to a point on *b*, and a unit vector mutually perpendicular to both sensed edges must be wholly included in the angle between any vector q_{ij} between points on model edges *i* and *j* and the unit vector mutually perpendicular to the model edges.

- The Distance Constraint requires that if edge fragments *a* and *b* are assigned to model edges *i* and *j* respectively, then the range of possible distances between a point on *a* and a point on *b* must be wholly included in the range of distances between any two points on model edges *i* and *j* respectively.

It can be seen that if the labels *a*, *b* and *i*, *j* are exchanged but we maintain the actual data, direction constraint 2 is the same as direction constraint 1, but applied from the point of view of the other edge. This means that the direction constraint 2 information can be obtained from direction constraint 1. Also, an advantage of the size information is that it need not be applied in a pairwise manner. If data fragment *a* is assigned to model edge *i*, then the absolute size of fragment *a* **cannot** be greater than the length of model edge *i*. These constraints are similar to ones used by Grimson and Lozano-Pérez [GL-P84*b*] for the matching of faces and are very efficient in pruning out whole sub-branches of the interpretation tree. For the maximum efficiency they are used in order of effectiveness, i.e. distance, direction 3, angle, direction 1, and direction 2. A discussion of this can be found in the Experimental Results section.

One thing which is important to note when using edges in a matching algorithm is the ambiguity of direction. As the edge is merely a line in space, its sign and, thus, direction must be determined. A data edge fragment is termed '+' if the choice of start and stop end positions is consistent with the model edge. If different it is termed '-'. Thus, a data edge may have the same (+) or opposite (-) direction to a particular model edge. This sign information also serves to increase the strength of the local constraints. At the early stages of processing, it is an overhead but, as the matching algorithm proceeds, it becomes increasingly effective.

It is necessary to ascertain the direction of the data edges relative to their corresponding model edges for the validation of the interpretation. As we are working with vectors the directions must be known when calculating the rotation of the model.

The signs are determined during the search of the interpretation tree and, at any given time, five possibilities exist for a data edge *a*. These are (U), (+), (-), (+*b*), and (-*b*). The state (U) means that the sign is yet undetermined and all data fragments are initially in this state at the interpretation tree root. Again, (+) means that the choice of start and stop positions for the data edge corresponds with that chosen for the model edge and (-) means that they do not. The (+*b*) state denotes that data edge *a* has the same sign as another data fragment *b*, and (-*b*) the opposite. If, at a later, stage the sign of *b* is determined, for example, *b* is found to be (-) when *a* was labelled (-*b*), the sign of *a* may be updated as:

$$(a) = (-b) \;\Rightarrow\; (a) = (-)\,(-) = (+).$$

When the state is (+) or (-), we say that the sign is known absolutely. It must be noted that the scope of a sign is only within the active interpretation and if, at a later stage, that interpretation is found to be inconsistent, the signs must be re-evaluated.

As we are working with angles between lines in space, two values exist: the angle and its complement. To evaluate the signs of the data edges, the matcher must take this into account. We can derive satisfaction conditions *s* and *d* which determine if the fragments *a* and *b* have the same (*s*) or different (*d*) signs respectively. For example, the satisfaction conditions for the angle constraint where the angle between the unit vectors of *a* and *b* is α and the complement π-α is α^* are:

$$s = \mathrm{MAX}((\alpha - \mathrm{E}), 0) \le \mathrm{A}_{ab} \le \mathrm{MIN}((\alpha + \mathrm{E}), \pi))$$

or

$$d = \mathrm{MAX}((\alpha^* - \mathrm{E}), 0) \le \mathrm{A}_{ab} \le \mathrm{MIN}((\alpha^* + \mathrm{E}), \pi))$$

where E is the sensing error half cone angle, A_{ab} is the angle between model edges *a* and *b*. Note that the MAX and MIN are used to clip the angles at 0 and π. For a valid pairing, at least one of the two conditions must be true. However, because of measurement uncertainties it is possible for both to be true.

Each local constraint has its own satisfaction conditions and before entry into the constraints tests these must be determined. The tests are given below for each constraint.

<u>Angle Constraint</u>

1) If on entry the signs of *a* and *b* are uncertain (U): then if *s* is true and *d* false, the pairing is valid and *a* and *b* must have the same sign for that interpretation and can be relabelled (+*b*) and (+*a*) respectively. However if *d* is true and *s* is false the signs are different and the edges can be relabelled (-*b*) and (-*a*) respectively. If *s* and *d* are both true the pairing is still valid but nothing can be

learnt about the signs. If both are false the pairing is invalid and the search backtracks.

2) If on entry the sign of one of the edges is known: if for example the sign of *a* is known absolutely and *s* is true with *d* false, fragment *b* must have the same sign as fragment *a*. Conversely, if *d* is true and *s* false *b* must have an opposite sign to *a*. Again, if both conditions are true the sign remains undetermined and if both false the search backtracks.

3) If both signs are known on entry: if *a* and *b* have the same sign then *s* must be true else the search backtracks. Also, if the signs differ *d* must be true for a valid pairing.

Direction Constraint 1

1) If on entry the sign of *a* is not absolute: if *s* is true and *d* is false the pairing is valid and *a* can be signed (+). If however *d* is true and *s* false the sign of *a* is (-). If both are true nothing is learnt and if both false the search backtracks.

2) If on entry the sign of *a* is known absolutely (+) or (-): for a valid pairing the appropriate condition must be shown to be true, i.e. *s* or *d* respectively.

Direction Constraint 2

As Direction Constraint 1 but with references to edge *a* to be replaced by edge *b*.

Direction Constraint 3

As Angle Constraint.

Distance Constraint

Not Applicable

It can be seen that as Direction Constraints 1 and 2 involve only one edge vector they can determine edge signs absolutely. The Angle Constraint and Direction Constraint 3 merely propagate the signs as they involve the product of two edge vectors.

5.6 Validation of Interpretations

After a set of feasible interpretations have been obtained, it cannot be guaranteed that these are consistent with the model. This is because the constraints are applied successively to pairs of data and are therefore local. As it stands, it is not possible to know which interpretations are consistent with a single global transformation between model and sensor spaces.

Each interpretation is therefore tested to:

1) Determine the actual transformation parameters (scaling, rotation, and translation) from model space to sensor space.

2) Confirm that each data edge lies sufficiently close to its particular model edge.

The transformation is found as an average of *all* the data edges by assuming that a vector v_m in model space is transformed into sensor space by the following:

$$v_s = S[R]v_m + t$$

where: S is the scaling, R is the rotation, and t is the translation.

The transformation is applied to each datum in turn, requiring that each endpoint, after back transformation in model space, is sufficiently close to the model edge to which it is paired within a small degree of error.

From the matching description of data to model features, a geometrical description of a model M is composed of primitive descriptors P in a set $(P_1, P_2, \ldots P_n)$ with P_n being the final descriptor in the model set. The descriptors chosen may be vertices or edge vectors, or approximations of geometrical surfaces such as planes (faces) or quadrics. This set forms a rigid body which is able to undergo various affine transformations.

Assuming M' to be the description recovered from the data in the sensed world (be it by tactile, range, stereo, or structure from motion techniques), a transformation T must be found which will map each sensed primitive descriptor of M', i.e. $(P_1', P_2', \ldots P_n')$ to the corresponding model primitive descriptor. This transformation must be accurate enough in its mapping to within a predetermined scope of error for a valid pairing. It provides the translation and rotation (and possibly scaling) of the identified object of the scene.

The estimation of the best transformation is determined by the least-squares method. The algorithm was developed by Faugeras and Hébert [FH83] for the matching of plane primitive descriptors, but can equally be applied to edge vectors. It uses the idea of quaternions in the estimation of the rotation. An edge E is represented by two parameters w and d where w is the unit vector normal to the edge, and d is the perpendicular distance of the edge to the origin O.

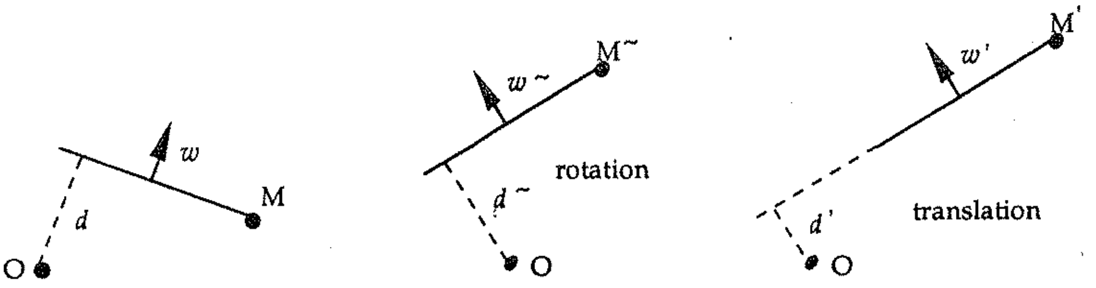


Figure 5.4: Rotation and translation of line in space

w determines the orientation of the edge and is directed parallel to d. By associating a point M of E, we can see that the distance d is equal to w.OM where '.' is the dot product of the two vectors (w being parallel to d, and perpendicular to the edge). When the transformation T is applied to the edge vector, a new edge is obtained. The rotation R is applied before the translation t. Assuming that the original edge was represented by E(w,d), the transformed edge vector is given by E'(w',d') where $w' = Rw$ and $d' = w'.t + d$. The derivation of this follows.

Referring to Figure 5.4, as the original distance is $d = w.\mathrm{OM}$ then after rotation and translation this becomes $d^\sim = w^\sim.\mathrm{OM}^\sim$ and $d' = w'.\mathrm{OM}'$ respectively. After rotation the orientation becomes $w^\sim$ and is equal to that after the subsequent translation. Also, the distance before rotation is the same as that after, thus the following may be written:

$$d = d^\sim$$

$$w' = w^\sim$$

$$\mathrm{OM}' = \mathrm{OM}^\sim + t \quad \text{where} \quad \mathrm{OM}^\sim = \mathrm{OM}$$

thus

$$d' = w'.(\mathrm{OM} + t) \;=\; w'.\mathrm{OM} + w'.t$$

Substituting $w'.\mathrm{OM}$ we have

$$d' = w'.t + d$$

Assuming a match Ma to be composed of pairs of model to corresponding data edges, then Ma = (E(w_i,d_i), E'(w_i',d_i')) i=1..n where n is the number of matched pairs, the rotation and translation sought is one which will minimize the sum

$$\sum_i \| w_i - Rw_i' \|^2 + \mathrm{W}.\,| d_i - d_i' |^2$$

i.e.

$$\sum_i \| w_i - Rw_i' \|^2 + \mathrm{W}.\,| d_i - d_i' - w_i.t |^2$$

This sum decomposes to two terms; a sum $\| w_i - Rw_i' \|^2$ to determine the best rotation, and a sum $| d_i - d_i' - w_i.t |^2$ for the best translation. Note that W is a weighting factor. The two sums are minimized using the notion of quaternions for the rotation. A quaternion q may be defined as a pair (v,s) where v is a vector of E3 and s is a scalar [FH83]. It may be regarded as a generalization of complex numbers with the vector v corresponding to the complex part of $a + ib$. Using this notation, the rotation problem reduces to a quadratic criterion:

$$F = \sum_i q.A_i.A_i^t.q^t = q.B.q^t$$

where $B = \sum_i A_i^t.A_i$ is a symmetric 4×4 matrix, and q is of dimension 1×4.

F is minimized with the criterion $\|q\| = 1$ where F_{min} is the smallest eigenvalue of B and q_{min} is the eigenvector of unit norm and of positive fourth coordinate corresponding to the eigenvalue of F_{min}.

Estimation of the best translation reduces to a vector which will minimize the sum

$$S = \sum_i | d_i' - d_i + w_i.t |^2 .$$

This is a simple least-squares problem which can be solved using the pseudo-inverse method as follows:

> If n is the number of match pairs and A is an $n \times 3$ matrix of $[w_1, w_2, ...w_n]^t$, and Z is an n-vector of $[d_1' - d_1, d_2' - d_2, ...d_n' - d_n]^t$ then $S = \| Z - At \|^2$. The estimation of the best translation then becomes: $t_{min} = (A^tA)^{-1} A^tZ$ and the error $S_{min} = Z^t(Z - At_{min})$.

A problem with these two estimation algorithms is that indeterminations may arise due to an insufficient number of matched pairs. The determination of the transformation requires at least two pairs of non-parallel edges for the rotation and three pairs of independent edges for the translation.

The scaling is achieved by firstly determining the scale of each available data edge and then averaging over the total number. Given the rotation $[R]$ and translation t, the scale of a data edge S may be derived from:

$$S = \frac{(\mathrm{Ea}_1 - t).\,[R]\,(\mathrm{M_i} \times (\mathrm{T_{i1}} \times \mathrm{T_{i2}}))}{[R]\mathrm{T_{i1}}.\,[R]\,\,(\mathrm{M_i} \times (\mathrm{T_{i1}} \times \mathrm{T_{i2}}))}$$

where Ea_1 is the start position of data edge fragment a.

M_i represents model edge vector i.

T_{i1} and T_{i2} represent the start and stop positions of model edge i respectively.

We note that with absolute size knowledge, the scale factor will be unity.

After determination of the rotation, translation, and scale, it is then necessary to test the actual validity of each edge. For this we need to determine how closely each data edge maps onto its respective model edge. This is achieved by transforming the sensed data back into model space where the endpoints are required to fit the model edge to within a deviation. This deviation is computed from a user supplied fraction *fracdev*, and the length of the longest vector between any two model vertices *maxdist*, as fracdev × maxdist. Fracdev must however be tuned for the best results. For a valid interpretation all edges must pass the deviation test.

5.7 Parallel Model Matching

In order to reduce the processing time, two main procedures may be adopted. The first is to heavily prune the interpretation tree, thus leaving a fraction of the original to be investigated. The second is the use of a faster machine and thus, parallel processing. The parallel matching algorithm devised by Holder and Buxton (H&B) [HB89*a*] and based on work by Grimson and Lozano-Pérez [GL-P84*b*] uses SIMD parallelism on the DAP in order to obtain a much reduced program run time. Parallel algorithms have also been used by other authors, namely, Flynn and Harris (F&H) [FH85], and Shankar, Ramamoorthy, and Suk (S,R&S) [SRS90]. Both have been adopted on the Connection Machine. Unlike H&B, the method of S,R&S uses vertex-pairs, not edges for matching.

The parallelism of H&B when compared with F&H is very effective. They found that with n model faces and m data points, n^m processing elements are required to achieve a processing time of the same order of magnitude. Also, for 250 objects, each with 10 faces and 3 data measurements, F&H required 250,000 of the 256,000 processors on the MIT Connection Machine [HB89*a*]. This will rise by a factor of 10 with each additional data point.

The effectiveness in the algorithm is achieved by initially determining matches of data pairs to model pairs in parallel and then testing these for local consistency. This means that in one instruction *all* the possible pairings of a data pair to model edges can be found. The order in which the constraints are applied to the data pairs is unimportant here because we only require the result obtained from application of all the constraints. These are stored in a logical matrix as in Figure 5.5. Note, however, that this is in contrast to the serial algorithm where the order of application of the pruning constraints may be significant. This is because in many serial implementations the constraints are applied in order of effectiveness so as to obtain cutoffs as quickly as possible. The serial algorithm, therefore, implements non-uniform termination, whereas in the parallel method, we perform an exhaustive search of the heavily pruned tree.

		data edge 2						
		1	2	3	4	5	6	7
	1		T			T		T
	2	T		T				
	3		T	T	T			
data edge 1	4		T		T	T		T
	5	T		T				
	6							
	7				T		T	

Figure 5.5: Logical look-up table of matches

From Figure 5.5 we can see that for the data edges 1 and 2, taken as a pair, possible matches are achieved where 'true' values 'T' are present in the matrix. Thus, data edge 1 may be associated with model edge 2 when data edge 2 is associated with model edge 1 or 3, and data edge 1 may be matched to model edge 4 when data edge 2 is on 2, 4, 5, or 7. The effectiveness of the algorithm is achieved by maintaining a large amount of data in the array store. For n data edges the number of data pairs is given by

$$\sum_{i=1}^{(n-1)} i \, .$$

This means that for 64 data items 2016 n × n matrices are required, where n is the number of model edges. It may be considered that a model containing 64 faces is sufficiently complex but this is certainly not so with edge matching. A model of 200-300 edges is required to show an equivalent complexity. This is very demanding on the memory requirements.

As well as generating and storing the model look-up tables, the parallel matcher is also made more effective by maintaining a one dimensional list of matches between data edges (a data look-up table) for each of the constraints. The length of this list is given by the number of data pairs. Matches between data edges are therefore calculated once only, and stored. This significantly improves the performance of the matching process.

Feasible interpretations are achieved by making trial assignments of alternative data pairs to model edges. These assignments are checked for consistency to assignments made higher up in the interpretation tree. This must be done because it is not physically possible for a data edge to be assigned to more than one model edge using bottom-up matching.

To illustrate this serial search for consistency of assignments we consider a list of possible matches of data pairs to model faces, as shown in Figure 5.7, for five data items recovered from an L-shape model of 8 faces (Figure 5.6). Note that for the same model, 18 model edges must be considered for edge matching.

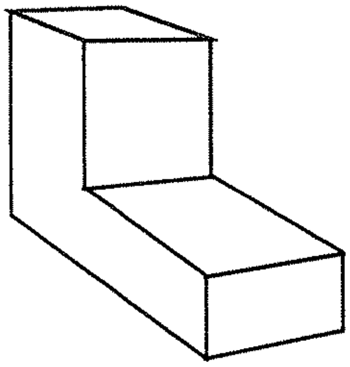


Figure 5.6: L-shape model

```
      data 2
    12345678
  1 .T...TTT
d 2 ........
a 3 ........
t 4 ....T.TT
a 5 .T...TTT
  6 ..T.T.TT
1 7 ........
  8 ........
     (L1)
```

```
      data 3
    12345678
  1 ..T.....
d 2 ........
a 3 ........
t 4 ........
a 5 ........
  6 ........
1 7 ........
  8 ........
     (L2)
```

```
      data 4
    12345678
  1 ...T....
d 2 ........
a 3 ........
t 4 ........
a 5 .....T..
  6 ........
1 7 ........
  8 ........
     (L3)
```

```
      data 5
    12345678
  1 .....TTT
d 2 ........
a 3 ........
t 4 ........
a 5 .T...TTT
  6 ........
1 7 ........
  8 ........
     (L4)
```

```
      data 3
    12345678
  1 ........
d 2 ..T.....
a 3 .T......
t 4 ........
a 5 ........
  6 ........
2 7 ........
  8 ........
     (L5)
```

```
      data 4
    12345678
  1 ........
d 2 ...T....
a 3 ........
t 4 ........
a 5 ........
  6 ........
2 7 ........
  8 ........
     (L6)
```

```
      data 5
    12345678
  1 .T...TTT
d 2 T.....TT
a 3 ...T..TT
t 4 ..T.T.TT
a 5 ...T.TTT
  6 T...T.TT
2 7 TTTTTT..
  8 TTTTTT..
     (L7)
```

```
      data 4
    12345678
  1 .T...TTT
d 2 T.....TT
a 3 ...T..TT
t 4 ..T.T.TT
a 5 ...T.TTT
  6 T...T.TT
3 7 TTTTTT..
  8 TTTTTT..
     (L8)
```

```
      data 5
    12345678
  1 .T...TTT
d 2 T.....TT
a 3 ...T..TT
t 4 ..T.T.TT
a 5 ...T.TTT
  6 T...T.TT
3 7 TTTTTT..
  8 TTTTTT..
     (L9)
```

```
      data 5
    12345678
  1 .T...TTT
d 2 ........
a 3 ........
t 4 ....T.TT
a 5 .T...TTT
  6 ..T.T.TT
4 7 ........
  8 ........
     (L10)
```

Figure 5.7: List of matches

The serial search algorithm requires that if a data edge is not currently assigned then it remains consistent for it to be assigned to any model edge along the appropriate row or column of the match look-up table. That is, for data pair 1-2 in the state Unassigned-Unassigned, all the 15 matches of the look-up table are consistent. Consider first the match of data edge 1 to model edge 1, and data edge 2 to model edge 2. For the level 2 look-up table (data pair 1-3), we can see that to maintain consistency with the above assignments, data edge 1 must be matched onto model edge 1. Data edge 3, being currently unassigned can match onto any model edge consistent with data edge 1, i.e. model edge 3. We now have matches of data edges 1, 2, and 3 to model edges 1, 2, and 3 respectively. For data pair 1-4, the consistency is again enforced by data edge 1 ensuring that it is only possible for data edge 4 to match with model edge 4. This consistency checking becomes more effective as the search deepens with maximal efficiency when both data edges are assigned, and minimal effectiveness with an Unassigned-Unassigned pair. It is therefore important that the data edges are sorted in a manner which will increase the power of the consistency cutoffs, i.e. the search backtracks when assignment of a data pair is not consistent with earlier assignments.

An interpretation is an assignment of each of the data items to a model face. The model faces can be repeated in an interpretation but each data item only appears once and must be consistent with all the previous assignments of that data item in the interpretation. From the above example, two feasible interpretations result:

Data	Model
1	1
2	2
3	3
4	4
5	7

Interpretation 1

Data	Model
1	1
2	2
3	3
4	4
5	8

Interpretation 2

After an interpretation has been found, it is stored and the search backtracks in an attempt to find more complete pairings. This means that the time taken to find a number of interpretations is highly unequal. After the first interpretation has been found, the work needed for the following interpretations is much reduced as the search does not restart from the root node. A DAP implementation of the algorithm is illustrated in Appendix C.

5.7.1 The Importance of Match Ordering

Unlike the serial version of Murray and Cook [MC88] when considering edges, the data edges need not be sorted in order of length prior to the matching algorithm to achieve matching efficiency. This is because all data edges are checked against model edges in parallel. Sorting of the data edges will therefore make no difference to the possible assignments. However, according to Holder and Buxton, the traversal of the interpretation tree is made more efficient by considering the data pairs in order of the least number of matches. The data pairs are thus sorted in a list accordingly. From Figure 5.7 it can be seen that if the assignments of data pairs 1-3 and 2-4 are considered first and 2-5, 3-4, 3-5 considered last, the consistency checking would be more effective and result in a faster traversal of the interpretation tree than if the data pairs were considered in the reverse order.

We show that, in general, sorting of the data pairs in order of least number of geometrical matches is inefficient and, in fact, proves to weaken the search as the number of data edges increases. However, the benefits gained by this type of ordering cannot be overlooked. A tree with one branch at the root node (data pairs 1-3 or 2-4) will, in general, be traversed faster than one with 33 (data pairs 2-5, 3-4, or 3-5). The problem, however, begins to occur when the ordering causes the generation of Unassigned-Unassigned data pairs during tree search. If the match list was sorted and traversed according to the least number of matches, e.g.

[1-3, 2-4, 1-4, 2-3, 1-5, 1-2, 4-5, 2-5, 3-4, 3-5]

as in Holder and Buxton, then immediately, an Unassigned-Unassigned pair is produced at level 2. Fortunately, in this example, the look-up table contains one possibility, but it could have easily been more. Consistency can be maintained so that only one Unassigned-Unassigned data pair (at the top level) is present in the match ordering. This is obtained by ensuring that a data pair is encountered only after at least one of its data edges has been earlier assigned. Thus, for our example, the ordering could be:

[1-2, 1-3, 2-3, 1-4, 2-4, 3-4, 1-5, 2-5, 3-5, 4-5].

This ensures effective consistency cutoffs as the list now reads

[U-U, A-U, A-A, A-U, A-A, A-A, A-U, A-A, A-A, A-A]

where U = Unassigned data edge, and A = Assigned data edge.

Considering the cases of A-U, it can be seen that the consistency maintainance can be made more effective by ensuring that pairs at this position have the minimum number of matches for each group. Here, a group is defined as data pairs whose second data edge are the same, thus, 1-4, 2-4, and 3-4 belong to group 4. Such sorting introduces the additional power obtained by reducing the number of matched pairs in a data look-up table. It should be noted that the number of matches at positions A-A is unimportant since only one location in the look-up table can be consistent.

The initial data pair is also important in the effectiveness of the tree traversal as was seen in the Holder and Buxton sorting. It can be included here by a further requirement that the initial data pair contains the least number of matches. This has the disadvantage that it is now possible to have two cases of U-U during tree traversal but the additional power of reducing the initial branches of the interpretation tree greatly outweighs this. In the remaining text this will be termed as the 'Combined sort' and its implementation is shown in Appendix C. Further investigations of this are also presented in the Experimental Results (Section 5.11).

5.7.2 Direction Sign Management

As in the serial version of the model matcher, the direction signs of the data edges are determined during the search of the interpretation tree. Along with the look-up table for matches between data pairs, individual matches at the local constraint level are also stored. This is also performed in parallel. For the five constraints; angle, distance, and direction constraints 1, 2, and 3, nine look-up tables are required, one for the distance and, two for each of the other constraints. Of the two, the first contains matches where the data edges both have the same or both have opposite directions to their corresponding model edges. When this is so, both data edges are said to be *same*. The second table holds those matches where one data edge has an opposite direction to its corresponding model edge and the data edges are then said to be *different*. Therefore, (+)(+) and (-)(-) are 'same', and (+)(-) and (-)(+) are 'different'. It must be noted that for a match to be possible a corresponding element in one of these tables must be true. If both are true, however, the pair still match but are unable to give further information in the determination of sign. To be more precise, a match is only possible if, for each constraint (excluding distance), a corresponding element in at least one of the two tables is true.

When traversing the interpretation tree, the local tests are performed at each level in order to determine data edge signs. Ambiguities, however, may arise on reaching a leaf node depending on the number of data pairs with directions being termed both as 'same' and 'different'. During the search, information corresponding to which data edge is signed and which is dependent on another is recursively updated in a list at each level. This updating of the list is performed in one single parallel step where the list is a vector data structure. During backtracking to a higher level the list is returned to the environment of that level.

5.8 Subgraph Matching of Large Models

A problem with the method of H&B is the large demand on memory for complex models. Although this is not as great as that of F&H, it does not allow for matching of models of greater than 32 edges for a 32×32 DAP, and 64 edges for a 64×64 DAP. However, the use of the new Fortran* (pron. Fortran Star) language makes it possible to exceed the limit of 32 and 64 edges respectively for the DAP without the need of special treatment of the data. The user is no longer limited to DAP_SIZED parallel structures but may work in arbitrary numbers or even non-uniform matrices. This does not mean that a computation on a 64×64 sized matrix is wholly parallel on a 32×32 DAP as the processing is still dependent on the edge size. It simply means that a user defined loop is not necessary and the execution can be invoked by one instruction. Fortran* will therefore maintain the parallelism in each DAP_SIZE × DAP_SIZE matrix but not globally in the MODEL_EDGE_SIZE × MODEL_EDGE_SIZE matrix. This pseudo-parallelism works well but is very demanding on memory especially with temporary variables. To attempt matching 512 data fragments, 130,816 data pairs must be considered. With each data pair requiring an $n \times n$ table, for a large model, this is not feasible.

We have devised a method, 'subgraph matching', for the parallel matching of large models which can be more memory conservative. This is achieved by attempting to systematically match to portions of the model. Consider the data from Figure 5.7. If each table was subdivided into quadrants, then these may be tested for possible matches and the quadrants checked for consistency. This may be presented in the form of a tree as shown in Figure 5.8 overleaf.

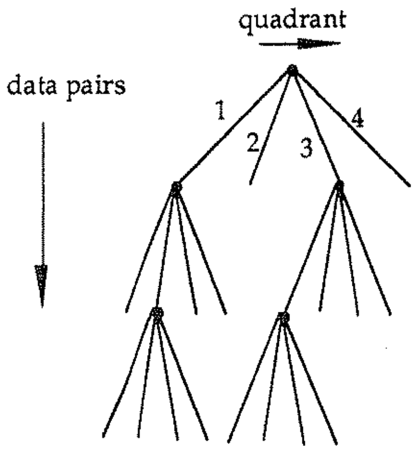
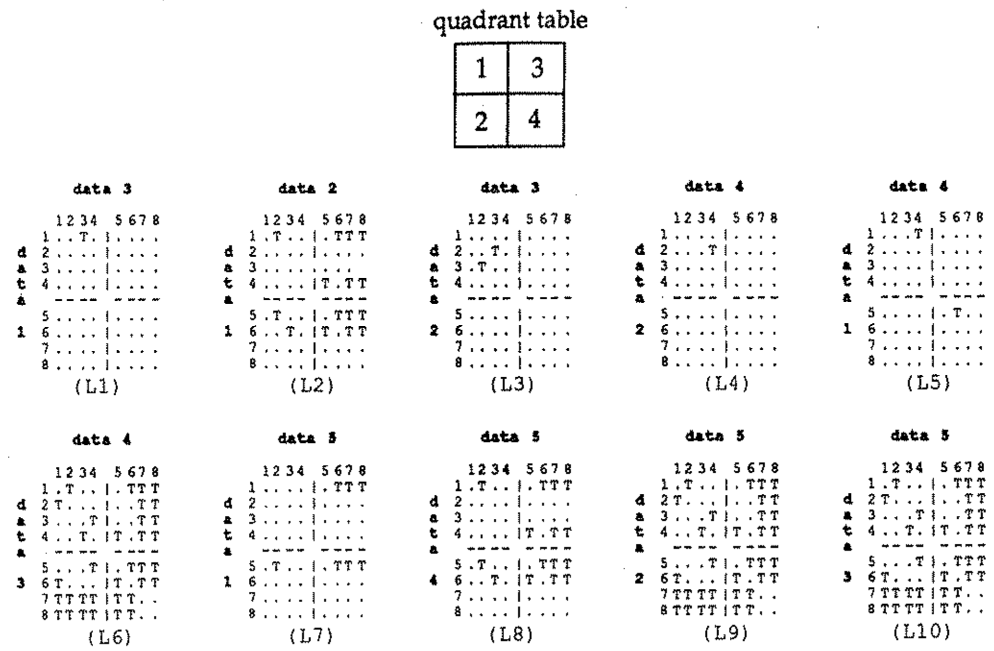


Figure 5.8: Quadrant tree

The size of each quadrant must be DAP_SIZE × DAP_SIZE for greatest efficiency. However, for the sake of simplicity we shall use quadrants of 4×4 in this example. If we consider Figure 5.7 it can be seen that a number of quadrants have no matches and others contain greater than 50%. By maintaining a list of the number of matches in each quadrant we may sort the 'quadrant interpretation tree' accordingly so that quadrants with less matches are encountered first, Figure 5.9.


quadrant table
1
3
2
4

```
      data 3

   1234 5678
   1..T.|....
d  2....|....
a  3....|....
t  4....|....
a  ---- ----
   5....|....
1  6....|....
   7....|....
   8....|....
     (L1)
```

```
      data 2

   1234 5678
   1.T..|.TTT
d  2....|....
a  3........
t  4....|T.TT
a  ---- ----
   5.T..|.TTT
1  6..T.|T.TT
   7....|....
   8....|....
     (L2)
```

```
      data 3

   1234 5678
   1....|....
d  2..T.|....
a  3.T..|....
t  4....|....
a  ---- ----
   5....|....
2  6....|....
   7....|....
   8....|....
     (L3)
```

```
      data 4

   1234 5678
   1....|....
d  2...T|....
a  3....|....
t  4....|....
a  ---- ----
   5....|....
2  6....|....
   7....|....
   8....|....
     (L4)
```

```
      data 4

   1234 5678
   1...T|....
d  2....|....
a  3....|....
t  4....|....
a  ---- ----
   5....|.T..
1  6....|....
   7....|....
   8....|....
     (L5)
```

```
      data 4

   1234 5678
   1.T..|.TTT
d  2T...|..TT
a  3...T|..TT
t  4..T.|T.TT
a  ---- ----
   5...T|.TTT
3  6T...|T.TT
   7TTTT|TT..
   8TTTT|TT..
     (L6)
```

```
      data 5

   1234 5678
   1....|.TTT
d  2....|....
a  3....|....
t  4....|....
a  ---- ----
   5.T..|.TTT
1  6....|....
   7....|....
   8....|....
     (L7)
```

```
      data 5

   1234 5678
   1.T..|.TTT
d  2....|....
a  3....|....
t  4....|T.TT
a  ---- ----
   5.T..|.TTT
4  6..T.|T.TT
   7....|....
   8....|....
     (L8)
```

```
      data 5

   1234 5678
   1.T..|.TTT
d  2T...|..TT
a  3...T|..TT
t  4..T.|T.TT
a  ---- ----
   5...T|.TTT
2  6T...|T.TT
   7TTTT|TT..
   8TTTT|TT..
     (L9)
```

```
      data 5

   1234 5678
   1.T..|.TTT
d  2T...|..TT
a  3...T|..TT
t  4..T.|T.TT
a  ---- ----
   5...T|.TTT
3  6T...|T.TT
   7TTTT|TT..
   8TTTT|TT..
     (L10)
```

Figure 5.9: List of quadrant tables

As discussed in Section 5.7.1, assignments are made starting from the data pair with the least total number of matches and maintaining consistency between the remaining number of data pairs. For the example, this means that the interpretation tree will only have a single branch at level 1 (data pair 1-3).

It is not necessary to enter each quadrant at every level. A number of checks are performed to determine the probability of consistent matches within a quadrant before matching is attempted. This greatly reduces unnecessary CPU computations. There are at present four conditions to determine if the model edges within a quadrant are significant for the particular interpretation.

1) If the quadrant contains no matches, abandon and move to next available quadrant.

2) If data edge 1 has been earlier assigned to model edge x and data edge 2 is unassigned, model edge x must be contained in the edge 1 side of the quadrant.

3) If data edge 2 has been earlier assigned to model edge y and data edge 1 is unassigned, model edge y must be contained in the edge 2 side of the quadrant.

4) If data edges 1&2 have been assigned to model edges x and y, the edge 1 side of the quadrant must contain model edge x, and the edge 2 side of the quadrant must contain model edge y.

It must be noted that the sorting of the quadrants in order of ascending number of matches, and the sorting of data pairs to maintain consistency, is very important to the efficiency of the method. The memory limitations of the hardware are now not as drastic as feasible interpretations are achieved using the DAP_SIZE × DAP_SIZE matrices of H&B for a model with n number of edges.

5.9 Parallel Data Validation

Using SIMD parallelism, it is easy to maximize the efficiency of calculations in the validation of feasible interpretations. The array structure of the DAP makes it possible to map the rotation matrix directly onto the processors. This would give 4 × 4 parallelism in the estimation of the best rotation for the sum $\sum_i A_i^t . A_i$ (see Section 5.6).

However, a greater efficiency is obtained by mapping several transformation matrices onto a single DAP array. For a 32 × 32 DAP the rotation calculations for up to 64 feasible interpretations are possible at each step. The larger 64 × 64 DAP would allow up to 256 interpretations.

Using Fortran* two possibilities exist when determining the best rotation. As mentioned above, sheet mapping subsequent 4 × 4 matrices onto a DAP array makes it possible to determine the rotation of several interpretations in one stage. The second method which was adopted was to map the 4 × 4 matrices onto an n × 4 parallel array where n is 4 times the number of feasible interpretations. Although this proved more taxing on memory, the numerous vector and matrix calculations were made more efficient. For example, to multiply and sum the columns in a series of 4 × 4 matrices in the array A(1..64) of 4 × 4 matrices, the following would be necessary:

```
initialise   result = 0
for count = 1 to 64
 for i = 1 to 4
  for j = 1 to 4
  {
    result(i,j) = 0
    for k = 1 to 4
     result(i,j) = result(i,j) + A(count)(k,i) * A(count)(k,j)
  }
```

and would prove messy. Also, it does not make the most of parallelism when applied to 4 × 4 matrices sheet mapped onto a 32 × 32 array. The problem collapses into a large number of shift processing operations in an attempt to place matrix components in their correct positions.

However, by mapping the matrices as an $n \times 4$ array, the parallelism can be maintained as shown in the following piece of Fortran* pseudo-code:

```
result = 0
for i = 1 to 4
 for j = 1 to 4
   result(i,j) = SUM(A(,i) * A(,j))
```

The operation is taken out on each column component of all the matrices and summed in one single parallel step.

The first method also allowed the estimation of up to 64 rotations to be determined in parallel for a 32×32 DAP but failed to maintain an effective parallelism in the calculations of the translation or determinations of scale or 'best fit'. As in the above example, the parallelism in the estimation of the rotation was made less effective by the need to perform a large number of matrix shifts in preparation for particular matrix operations. As the translation, scale, and best fit was determined for each data edge, and the overall result obtained by averaging, method number two was able to deliver the results effectively in parallel. As a large number of vector and matrix operations were required during these estimations, it proved wiser to map the 1×3, 3×3, and 4×4 arrays of each data edge onto a DAP array and determine the translation, scale, and best fit of all the data edges in a single step. This also applied to calculations in the estimation of the best rotation. By using this structure, an increase in the number of data edges shows only a fragmentary increase in the run time. This occurs due to the need to set up the extra edges and when the number of data edges becomes greater than the edge size of the DAP.

It has been assumed that during the estimation of the best rotation that the absolute signs of all data edges are known. This is not always so. Under such circumstances, the data edge cannot yield any helpful information. It is, however, possible to pull in this potentially useful data for later refinement of the rotation calculation. After estimation of the translation and scale, the unsigned data edge is fitted to its corresponding model edge. If the fitting is successful, the start and stop positions of the data edge may then be deduced and used for refinement of the rotation estimation.

5.10 The Bin and its Implementation

It was assumed above that all the data edges or data components will be matched to the model. In practice this is not so. Junk and clutter, noise in the image, or parts of the object being occluded can all lead to more data components than there are model components. Spurious models and model primitives that do not show up in the image tend to encourage the number of model features to be greater than the number of image features. Thus, in general, the number of model features does not equal the number of image features. The effect of spurious data will be to make quite legal interpretations invalid. A way of overcoming this is to introduce the concept of null-pairing or 'bin'. This means that if a particular data edge cannot be matched, then it is assigned to a bin and will be unpaired (hence the name). This requires an extra branch in the interpretation tree for the bin at each level. Thus, for a model of e edges with s sensed edges, the introduction of the bin will result in $(e + 1)^s$ possible interpretations.

A scene with three data and two model features is shown below. A possible interpretation may be difficult to obtain without a bin (the third branch).

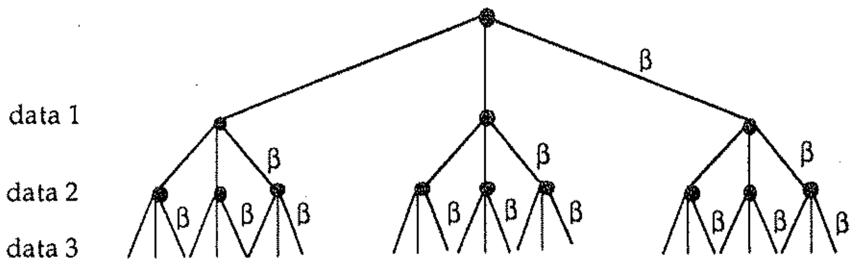


Figure 5.10: Interpretation tree with bin β

At each level, it is possible for that particular data feature to be paired with the bin β. This means that interpretations can have various numbers of bin pairings. Obviously, the interpretation with the minimum number of bin assignments is the most consistent with the data and, therefore, likely to be the correct one. Suppose that we have an interpretation of (1, 1) (2, 2) (3, 2). With the bin we can have further interpretations from this:

1 bin assignment	2 bin assignments	3 bin assignments
(1, 1) (2, 2) (3, β)	(1, 1) (2, β) (3, β)	(1, β) (2,β) (3, β)
(1, 1) (2, β) (3, 2)	(1, β) (2, 2) (3, β)	
(1, β) (2, 2) (3, 2)	(1, β) (2, β) (3, 2)	

These may be sorted in order of minimum number of bin assignments. For a particular interpretation without the bin, seven more are available with the bin in this case! The original interpretation is preferable than those of 1 bin assignment, which are better than 2 bin assignment, which in turn are better than 3.

In implementation, the matcher may discard an interpretation depending on the number of bin assignments in that interpretation. In tests, assignments of 10 - 20% (depending on the accuracy of the sensor) of the total number of data edges was found to be quite effective for bin content. The parallel implementation mapped well onto the processors of the DAP. It simply consisted of the increase of the n × n look-up tables to (n+1) × (n+1). The last dimension is made to contain the matches of the bin. Obviously, these will all be set to true in each table so that any data edge may be paired with the bin. Thus, a look-up table for a model with 6 edges must contain the assignments below.

data edge 2

data edge 1	1	2	3	4	5	6	7
1							T
2							T
3							T
4							T
5							T
6							T
7	T	T	T	T	T	T	T

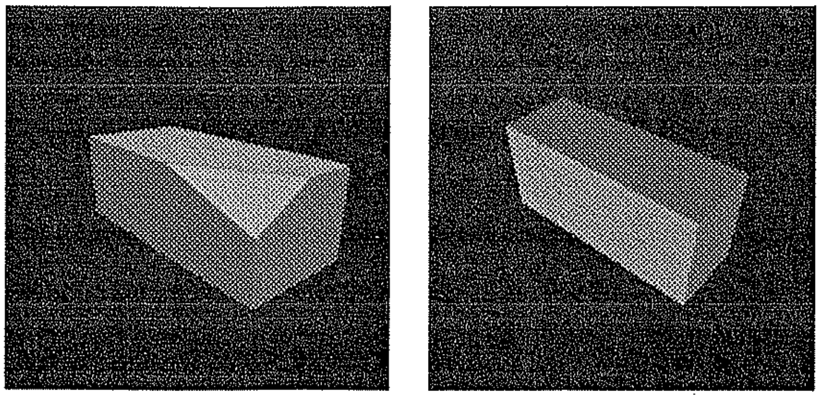
Figure 5.11: Logical look-up table of bin assignments

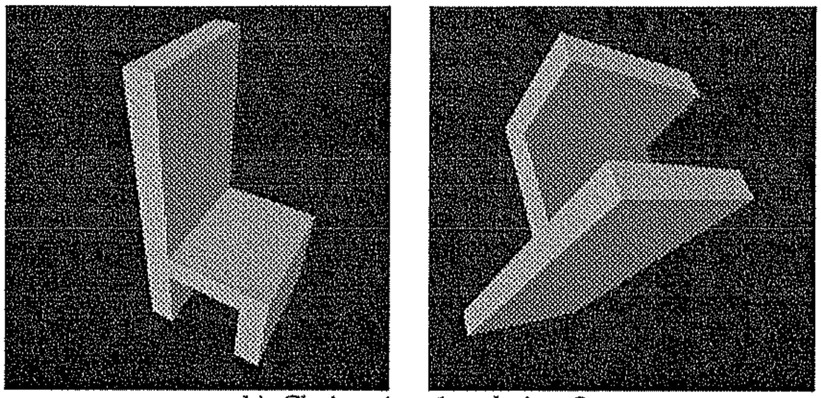
The addition of the bin makes little or no difference to calculating matched pairs as this step is a parallel one. However, an increase in time occurs during the serial search of the heavily pruned tree. Cut-offs and backtracking can be made to occur after the number of bin assignments exceed that of a user set value. In addition, the matrix locations not composing the bin are encountered first in order to deliver only non-bin interpretations early in the search.

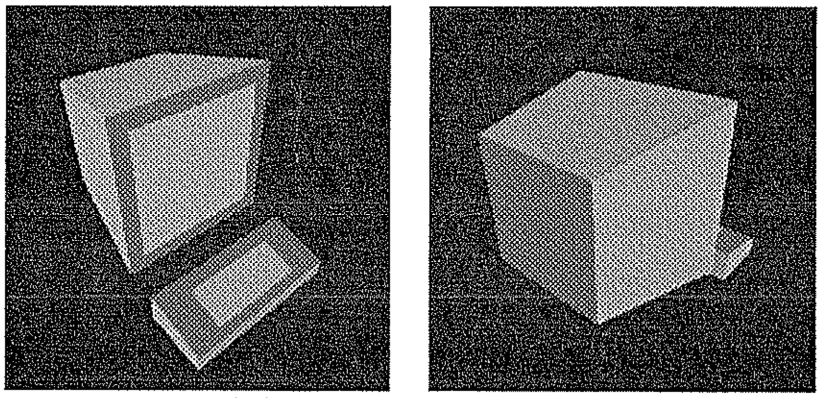
5.11 Experimental Results

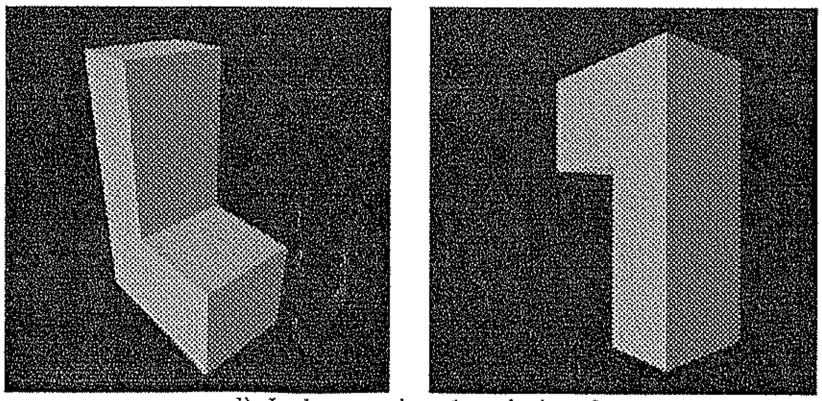
We present in this section experimental results obtained using the parallel model matcher for scenes of models and data generated using the parallel ray tracer described in the previous chapter. Experiments conducted on five CSG models are described initially, with a sixth larger model reserved for subgraph matching.

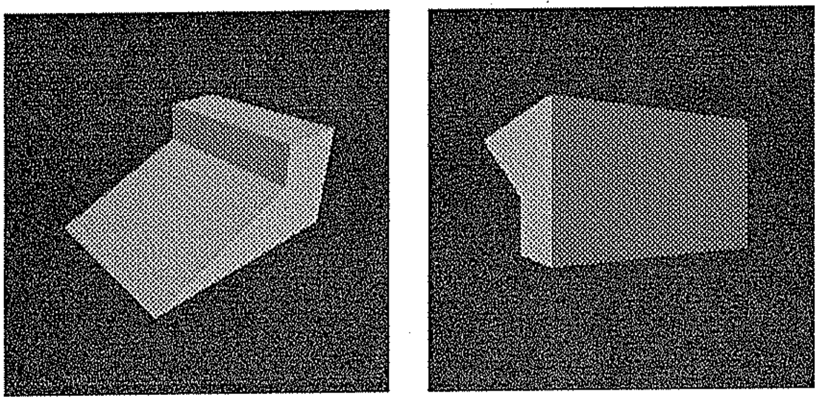
Each model is viewed from two positions. The first is termed 'interesting' and presents an accurate view of the model. The second is less so and aims to be from a more ambiguous viewpoint. Figure 5.12 illustrates the two views of the models. Each was generated as a 512 × 512 pixel image in a single level trace and is illuminated using a single light source. Their complexity in terms of CSG primitive content and halfspaces is shown in Table 5.1.


a) Chipped Block: view 1 and view 2


b) Chair: view 1 and view 2


c) Computer: view 1 and view 2


d) L-shape: view 1 and view 2



e) Wedge: view 1 and view 2

Figure 5.12: The test scenes

Model	No. of Primitives	No. of Halfspaces
Chipped Block	2	12
Chair	3	18
Computer	9	54
L-shape	2	12
Wedge	3	18

Table 5.1: Complexity of test models in terms of CSG primitives

As the data fragments in these results were obtained using stereopsis, it was helpful to the stereo processing algorithm that a sharp contrast at edge boundaries was present. This was achieved by 'painting' each face of the model a slightly different colour from its neighbours. Also, to improve the robustness of the stereo matching algorithm, the ratio of the interocular separation of the two pseudo-cameras to the viewing distance was kept to about 1:5 [TINA]. The stereo image pairs for each model was produced using the Tiling version of the parallel ray tracer. Figure 5.13 shows the stereo pair for view 1 of the Computer model under these conditions.

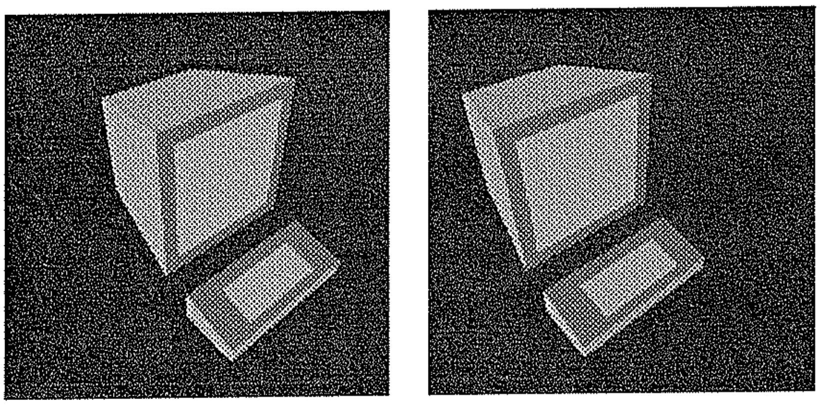


Figure 5.13: Stereo pair for view 1 of Computer model

The generation times of the stereo image pairs was extremely fast, considering the resolution, and can be seen in Table 5.2.

Model	Time to generate stereo pair (seconds)
Chipped Block view 1	12.6
Chipped Block view 2	10.3
Chair view 1	12.1
Chair view 2	9.6
Computer view 1	32.8
Computer view 2	26.0
L-shape view 1	14.5
L-shape view 2	11.9
Wedge view 1	25.5
Wedge view 2	11.0

Table 5.2: Generation time of stereo pairs using parallel tracer

Each model is defined in a right-handed coordinate system with a vertex at the origin (except in the case of the Computer model) and the model placed in the -x, +y, -z quadrant. They are of varying complexity in terms of faces and edges (Table 5.3).

Model	No. of faces	No. of edges
Chipped Block	7	15
Chair	12	30
Computer	14	32
L-shape	8	18
Wedge	7	15

Table 5.3: Complexity of models in terms of matching features

In the following sections we shall investigate various aspects of the parallel matching algorithm. Comparisons of the geometrical pruning constraints are made along with tests for robustness for different sensing errors. A serial implementation of the matching and validation phases, in the C programming language, on a Sun 4/260 workstation running under UNIX is compared to the parallel implementations. We show that the sorting algorithm of Holder and Buxton weakens as the number of data pairs and/or the sensing error increases. Also, we show that the new 'Combined' sorting of the match list (discussed in Section 5.7.1), in most cases, shows significant speed ups over the Holder and Buxton sort.

We note here that the serial implementation for the matching algorithm is coded 'efficiently' and that timings obtained may be faster than in the general case.

5.11.1 The Pruning Constraints - A Comparison of Effectiveness

In this section we compare the geometrical constraints; distance, angle, direction 1, direction 2, and direction 3, to try and establish which are most effective in collapsing large sub-branches of the interpretation tree. Comparisons are made using 'non-ideal' view 1 data recovered from the Computer and L-shape models. These contain 26 and 18 data fragments respectively. By 'non-ideal' we mean that the recovery of the data edges are subject to sensing errors. 'Ideal' data suffers from no, or very little sensing error.

Table 5.4 details the power of the constraints for the two data scenes. In both cases the angle constraint appears to be the weakest with the largest number of matched pairs. Next are the direction 2 and direction 1 constraints. However, in the case of the Computer, the distance constraint appears most effective, whereas in the case of the L-shape it is the direction 3 constraint.

Data scene	Model edges ME	Data edges	No. of data pairs DP	Potential matches ($ME^2 \times DP$)	Pruning Constraints				
					Dist	Angle	D1	D2	D3
Computer	32	26	325	332800	73032	181276	139694	166804	131026
L-shape	18	18	153	49572	20870	27864	24148	25366	13600

Table 5.4: The pruning effects of the constraints on the match list

Theoretically, if all constraints were applied and then each is switched off individually, the constraint which would produce the largest number for the total matched pairs by this action, i.e. the remaining constraints are applied, will in general be the most effective constraint. The result for this is shown below.

Data scene	Total matched pairs for constraints					
	All on	Dist off	Angle off	D1 off	D2 off	D3 off
Computer	21069	56709	27630	26685	23842	25732
L-shape	5846	8432	7876	6794	6344	8184

Table 5.5: Pruning power of constraints

We can see that in both cases the distance constraint appears most effective and the direction 2 constraint appears weakest. However, establishing the order of the remaining constraints, and in which manner to apply them is not so clear. Fortunately, this does not affect the parallel matcher. As all constraints are applied at the top level, and the results stored, the order in which they are applied is not significant, but can be so for the serial method which implements non-uniform termination.

We note that as the angle constraint involves an absolute measurement, it should be more effective than the direction constraints which involve calculating a range of angles. This is evident for the Computer scene whereas for the L-shape the angle constraint is second to the direction 3 constraint. We may, therefore, conclude that for matching with absolute size knowledge (structure from stereo), the distance constraint is the most effective constraint, whereas without absolute size (structure from motion), the angle constraint (or the direction 3 constraint) may be applied first.

5.11.2 The Effects of Sorting

Effective sorting of the match look-up tables is the key factor in the efficient execution of the serial search during the matching process. Here, we compare several sorting methods in an aim to determine their effectiveness (or non-effectiveness) for three data scenes - the Chair (view 1), the Computer (view 1), and the L-shape (view 1). These are fairly large models with 30, 32, and 18 model edges, and 20, 26, and 15 visible data edges respectively. In each case, the pruning constraints; angle, direction 1, direction 2, and direction 3 are applied.

The sorting methodologies compared here are the Holder and Buxton (H&B) ascending order sort, the consistency sort, consistency sort with ordering within groups, and consistency sort with ordering within groups and minimum matched data pair at level 1 of the match list - the Combined sort. Also included is the 'worst case' sort whereby data pairs are arranged in descending order of matches.

From the results obtained, all passing tests were able to return the single correct interpretation. As expected, the Combined sort outperformed the H&B sort in respect of total matching time (Figure 5.14). For the Chair scene it produces a speed up of 4.25 of the total run time, and for the L-shape model this reduces to 1.59. The introduction of least number of matches in the consistency sort produces a more effective ordering in all cases. For the 'worst case' sorting, both the Chair and the Computer scenes failed to terminate or return an interpretation after 36,000 seconds (10 hours!). The L-shape interpretation was returned after 1,360 seconds (22.7 minutes).

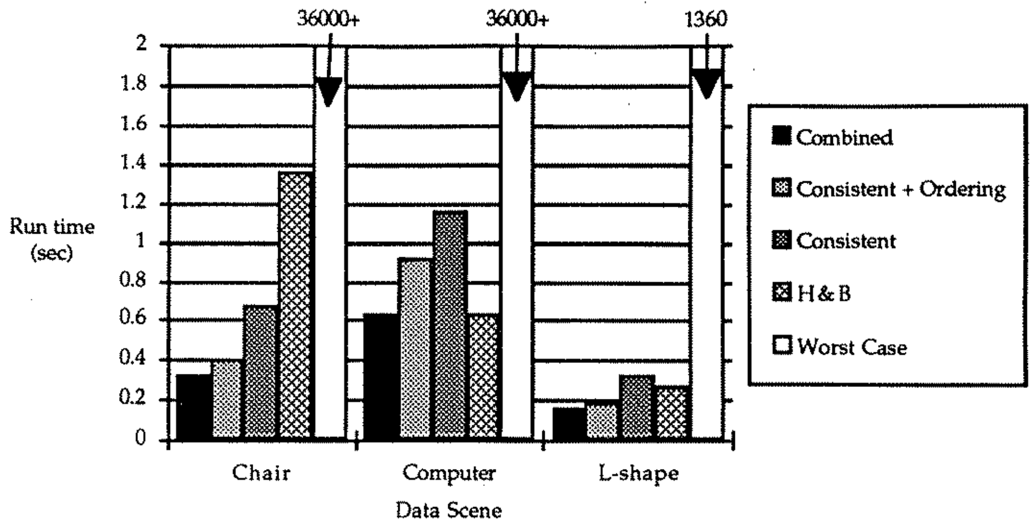


Figure 5.14: Comparison of run times for various sorts

In the case of the Computer scene, for the Combined and H&B sort, no significant difference in run time was observed. This may be explained by realising that although for the H&B sort two U-U data pairs are encountered during tree search, the average number of matched pairs for the first ten levels is 2. For the Combined sort the values are 1 and 44.6 respectively. Figures 5.15 and 5.16 illustrate the number of U-U data pairs, and the mean number of matched pairs for the first ten levels as relating to the search effort made (levels traversed) during tree search, Figure 5.17.

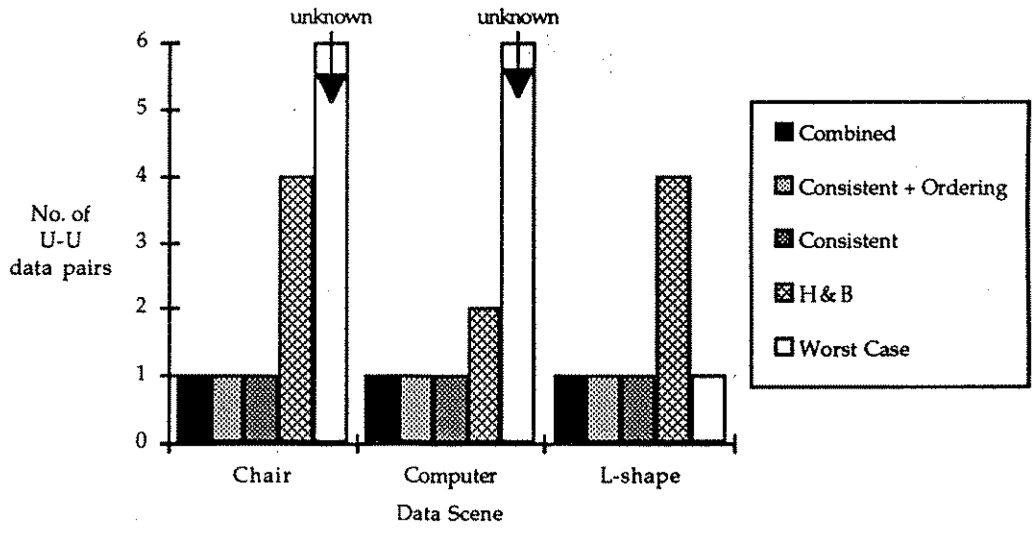


Figure 5.15: Number of U-U data pairs encountered during search

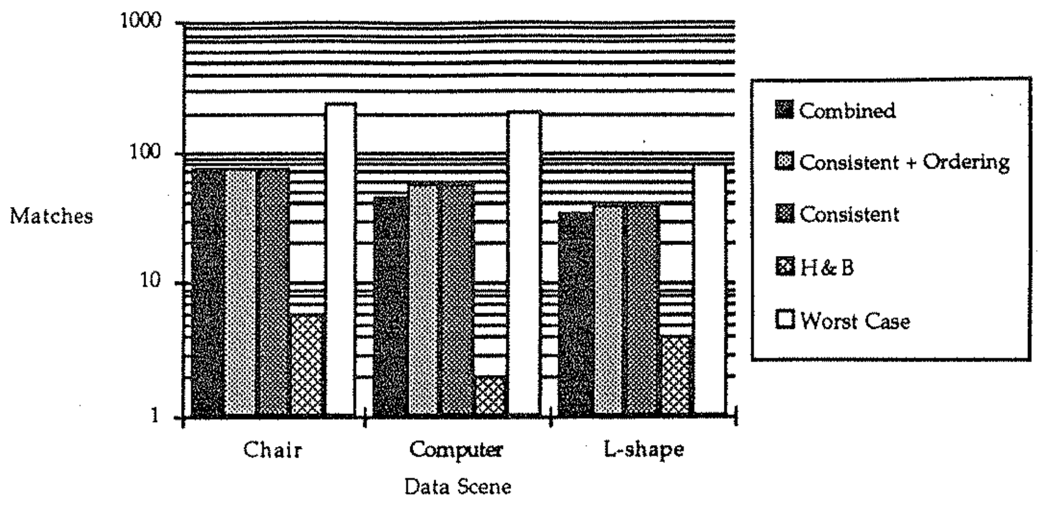


Figure 5.16: Number of matches for 1st ten levels of sorted match list

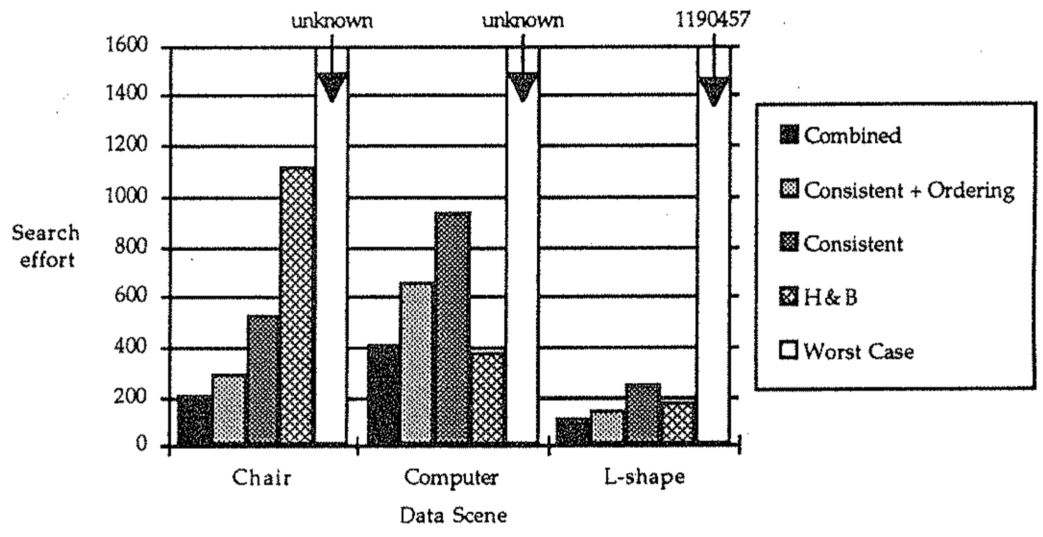


Figure 5.17: Search effort (levels traversed)

From the above Figures it can be deduced that a most important factor in the efficient traversal of the interpretation tree is the number of U-U data pairs encountered during the search. It can be seen that even though the H&B sort has far less matched pairs at the top levels of the search, it does little to aid it when compared to the Combined sort. A prime example of this can be seen in the Chair data scene where the mean number of matched pairs for the H&B sort is 12.5 times less, but the search effort is 5.3 times more! A similar case exists for the L-shape where the mean number of top level matches is 8.6. times less, and the search effort is 1.5 times more.

5.11.3 The Matching Process

The results to be discussed are those obtained from the matching process for the five test scenes (view 1 and 2). We compare the performances of the serial and parallel implementations of the Grimson and Lozano-Pérez matching algorithm when applied to ideal data. Also undertaken are tests of the parallel algorithm for data recovered by non-ideal means. This can generally produce more fragmentary data edges. The cost of determining the direction signs of the data edges is also discussed for the H&B and Combined sorts. This can give a comparative idea of the performance of the matching algorithm between edge and face features. We also profile the parallel algorithm in an effort to establish which procedures make the greatest demand on the CPU. This information may, in future, be used to improve the performance of the algorithm by parallelising (or improving the parallelism of) such procedures.

5.11.3.1 Matching Features - The Case for Ideal Data

We present here the data edges recovered from the various views of the test models. These data edges are 'ideal' in that they do not suffer from sensing errors because they are generated artificially. Table 5.6 demonstrates the number of model edges and how many of them are visible from each viewpoint.

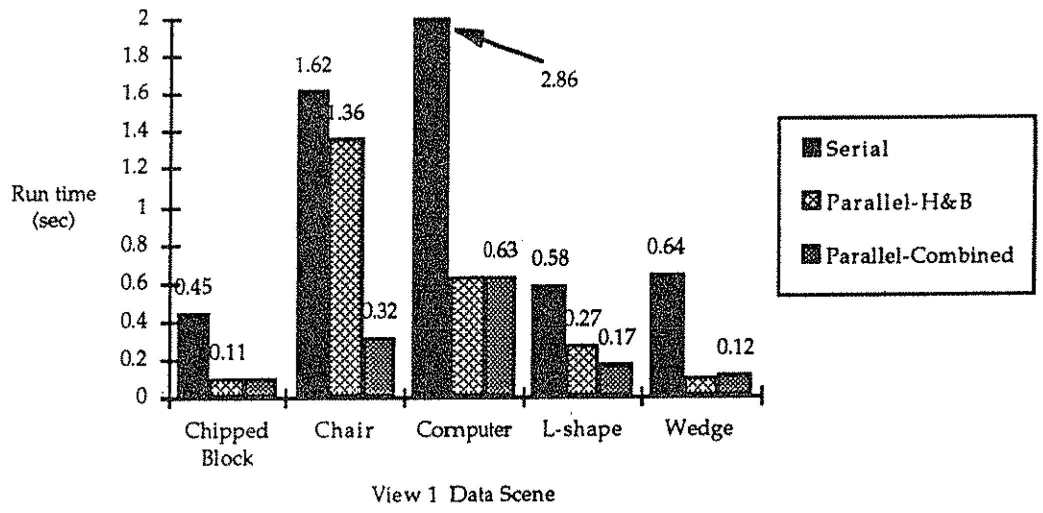
Scene	Model edges (e)	Data edges (s)	Data pairs	Potential Interpretations (e^s)
Chipped Block view 1	15	12	66	1.30×10^{14}
Chipped Block view 2		9	36	3.84×10^{10}
Chair view 1	30	20	190	3.49×10^{29}
Chair view 2		21	210	1.05×10^{31}
Computer view 1	32	26	325	1.36×10^{39}
Computer view 2		17	136	3.87×10^{25}
L-shape view 1	18	15	105	6.75×10^{18}
L-shape view 2		9	36	1.98×10^{11}
Wedge view 1	15	12	66	1.30×10^{14}
Wedge view 2		8	28	2.56×10^{9}

Table 5.6: Model and data edges for test scenes

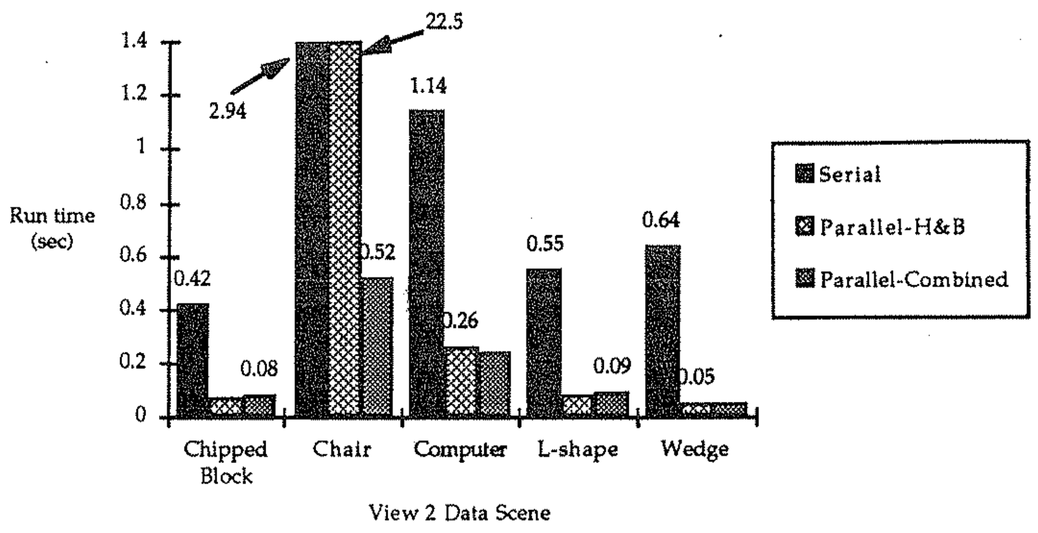
5.11.3.2 Serial vs Parallel - A Comparison of CPU Expenditure

The utilisation of ideal data can best be used to determine the efficiency of a matching algorithm. Here, we make comparisons between a Sun 4 implementation of the Grimson and Lozano-Pérez matching algorithm and a parallel 510 DAP equivalent using Combined sorting of the match list. In an attempt to show the importance of match list sorting, we also include results obtained for the H&B sort. In both cases the matching features are recovered ideally with *errpar* and *errper* as unity. The maximum error half cone angle is set to 1° and the angle, direction 1, 2, and 3 pruning constraints are applied. In each case both implementations were able to terminate and return a single feasible interpretation (correct of course), along with the sign of the data edge.

For both implementations, the initialisation and sorting overheads are included in the timings presented although they appear negligible when compared to the total matching time. For the sequential version, the sort time is that required to order the data edges by decreasing length. The time required to apply the pruning constraints to data edge pairs is not taken into account for the serial version as this is calculated off-line in the parallel implementation.



a) View 1 data scenes



b) View 2 data scenes

Figure 5.18: Total CPU matching time for test scenes

From Figure 5.18, the parallel Combined sort implementation is found to outperform the serial, more brute force, version in all cases. This was as expected. However, the H&B sort did not perform so well. For simpler scenes its performance was equivalent (or marginally better - Chipped Block view 2, L-shape view 1, and Wedge view 1) than the Combined sort. However, for the Chair view 2 scene, it is found to be 7.6 times **slower** than the serial version!

In general, the Combined sort was found to be 5.7 times faster than an equivalent serial version for all scenes. The greatest speed up is exhibited by the Wedge view 2 scene with 12.8 times, and the minimum of 3.4 times is shown by the L-shape view 1 scene.

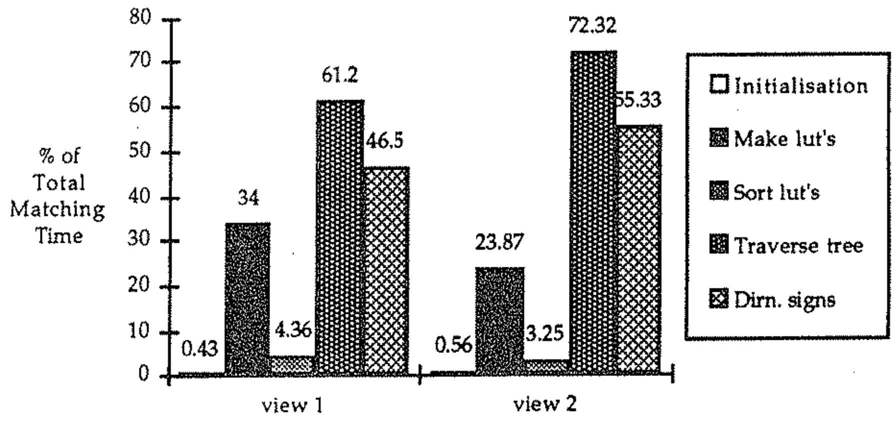
As with the human visual system, a crucial factor in the recognition of a model may be a single feature. For the Chipped Block this is obviously the 'chip'. This is found to be so in the two implementations of the matching process. From Table 5.7 we can see the relationship between the number of data pairs and the time required to traverse the interpretation tree for the Combined sort implementation.

Scene	No. of data pairs (% increase)	Speed of traversal (% increase)
Chipped Block	view 1 over view 2 45.5	view 2 over view 1 10.4
Chair	view 2 over view 1 9.5	view 1 over view 2 47.1
Computer	view 1 over view 2 58.2	view 2 over view 1 65.1
L-shape	view 1 over view 2 65.7	view 2 over view 1 39.0
Wedge	view 1 over view 2 57.6	view 2 over view 1 57.8

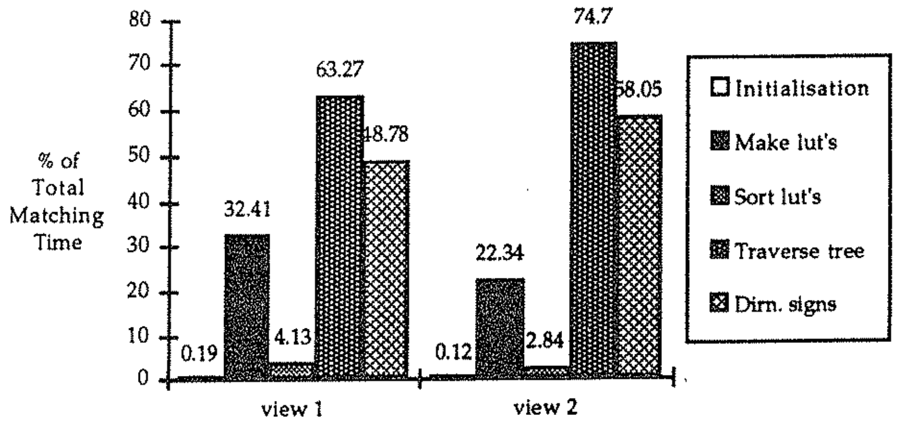
Table 5.7: Relation between traversal of IT and number of data pairs

For the Chipped Block it can be seen that even though the view 1 scene has over 45% more data pairs than the second view, the time of traversal for the second view is only 10.4% faster than the first. This is in contrast with the other scenes where a greater speed up is observed for tree traversal even when there is little increase in the number of data pairs. The serial implementation shows this 'crucial feature' clearly where the total matching time for view 1, possessing the 'chip', is 0.45 seconds, and for view 2, without this feature, the cost is hardly different at 0.42 seconds - a speed up of 6.7%. The explanation of this is in terms of the 'crucial feature' on the block, that is, the use of the 'chip' allows the constraints to prune more heavily in the first view.

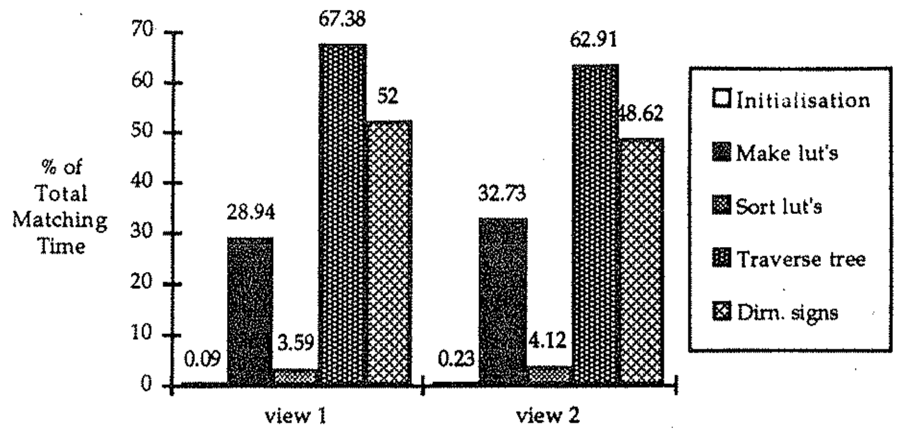
By profiling the parallel implementation (Combined sort) we can see how the matching process performs at various phases. The time spent in each procedure is taken as a percentage of the total matching time. The general trend is found to be the same as can be seen in Figure 5.19. Note that the Total Matching Time is the sum of the *Initialisation*, *Make lut's*, *Sort lut's*, and *Traverse tree*. The time spent in the *Traverse tree* module is partly attributed to the time required to determine the *Direction signs*.



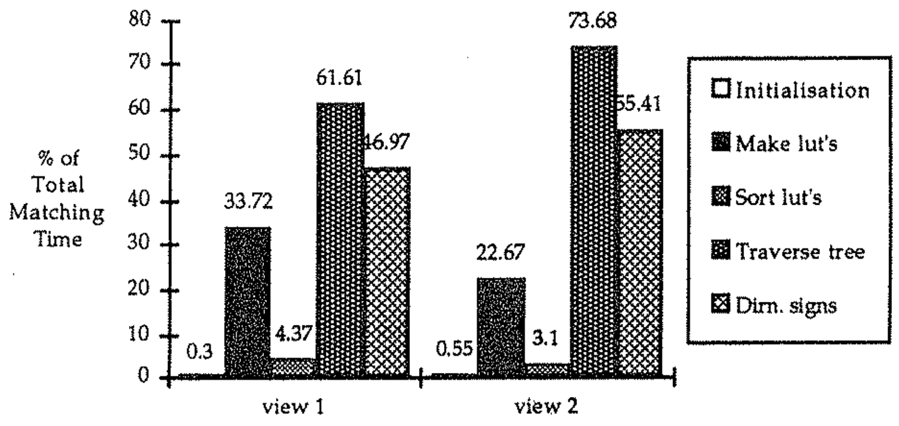
a) Profile of Chipped Block



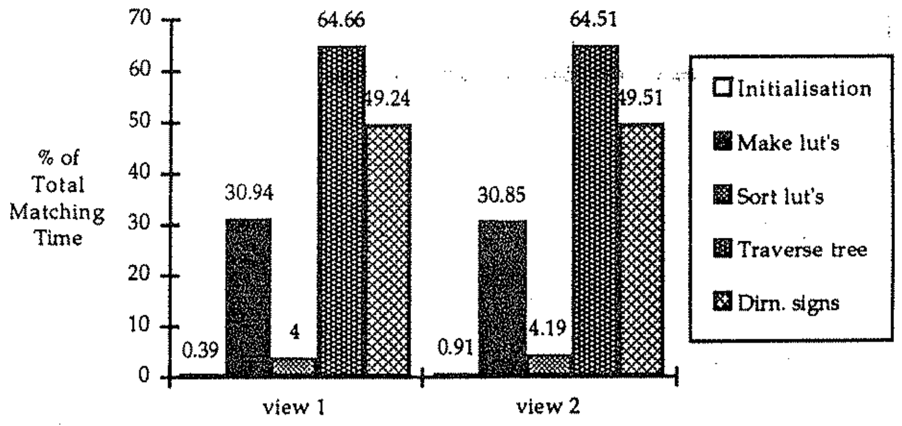
b) Profile of Chair



c) Profile of Computer



d) Profile of L-shape



e) Profile of Wedge

Figure 5.19: Profiles of matching process for various scenes

In all cases, the CPU time expenditure for initialisation of the DAP arrays and sorting the match lists are negligible. Computing the match look-up tables is, however, a more demanding process requiring, on average 26.4% of the total matching CPU time expenditure. We can see, as expected, that the greatest demands on the CPU is made by the serial traversal process. This has a mean demand of 66.6% of the total matching time. However, a large percentage of this is the cost of determining the direction signs of the data edges. This does not mean that by not depending on the signs of the data edges the algorithm will perform more efficiently. It simply means that the tree search, in the general case, will be faster. A discussion of this observation follows in the next section.

5.11.3.3 The Cost of Determining Direction Signs

Determining the direction signs of data edges is a factor necessary when using edges as a matching feature. The cost of this, in terms of CPU expenditure, is noticeable during the serial search process. It does not mean, however, that the search is any weaker. On the contrary, the direction signs, in most cases, produce cutoffs which reduce the search effort. They, therefore, act as an additional constraint with the disadvantage that generating them is somewhat expensive. However, we note that for the H&B sort the CPU time expenditure for the search is, in most cases, more without direction sign determination than with! This can only be accounted for by the assumption that this non-consistency ordering is highly dependent on backtracking by means of sign cutoffs. For scenes with little or no sign cutoffs, the unsigned search is more effective, but for scenes with a large amount of cutoffs, the unsigned search becomes very weak.

For the Combined sort, the unsigned search was able to terminate faster than the signed search for view 1 of the five test models (with exception to the Computer).

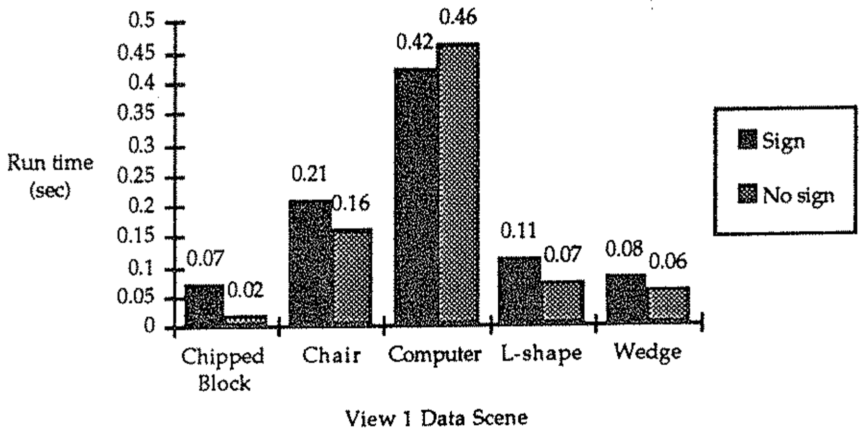


Figure 5.20: Run times with and without sign determination

However, the unsigned search was unable to determine between the correct interpretation and its symmetry for all but the Chipped Block and returned with two interpretations.

The additional pruning power of the direction signs can be seen from Figure 5.21. With exception to the Chipped Block, the signed implementation, possessing greater sign cutoffs (- the unsigned value is zero), is able to terminate with a search effort less than its unsigned equivalent.

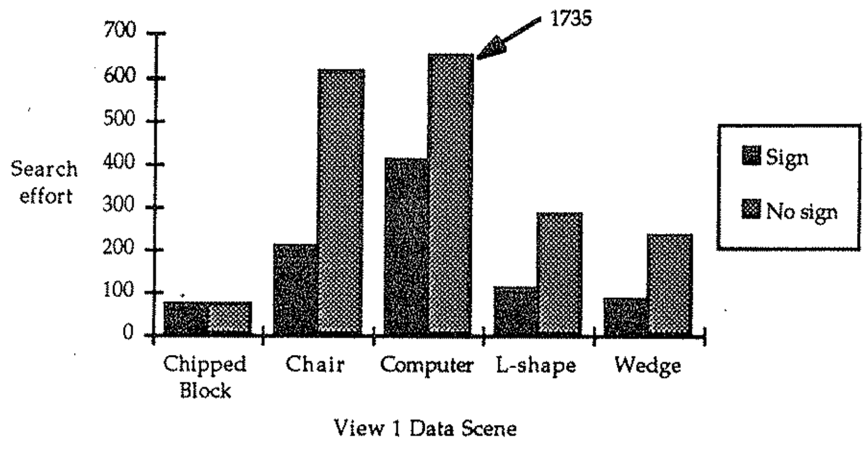


Figure 5.21: Search effort with and without sign determination

Thus, although the determination of data edge direction signs in real terms slows down the performance of the matching process, it can in fact behave as an additional constraint. A means of improving this operation will render the search phase more effective than at present.

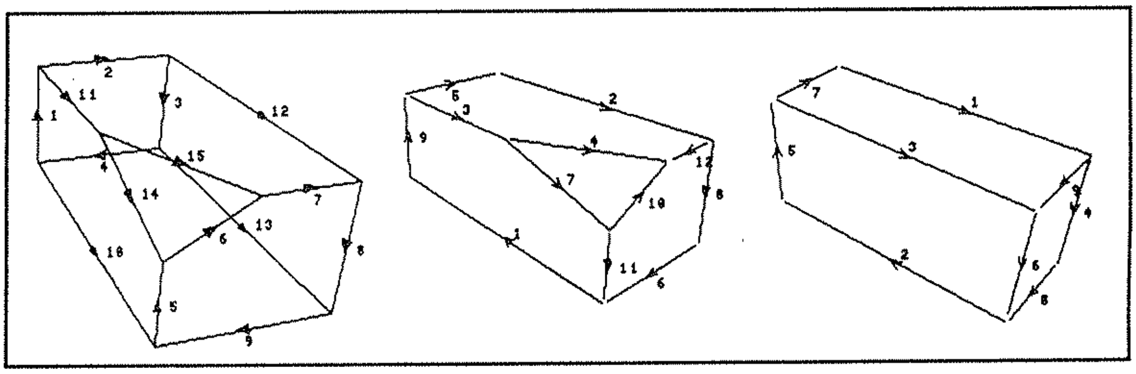
5.11.3.4 Matching Features - The Case for Non-Ideal Data

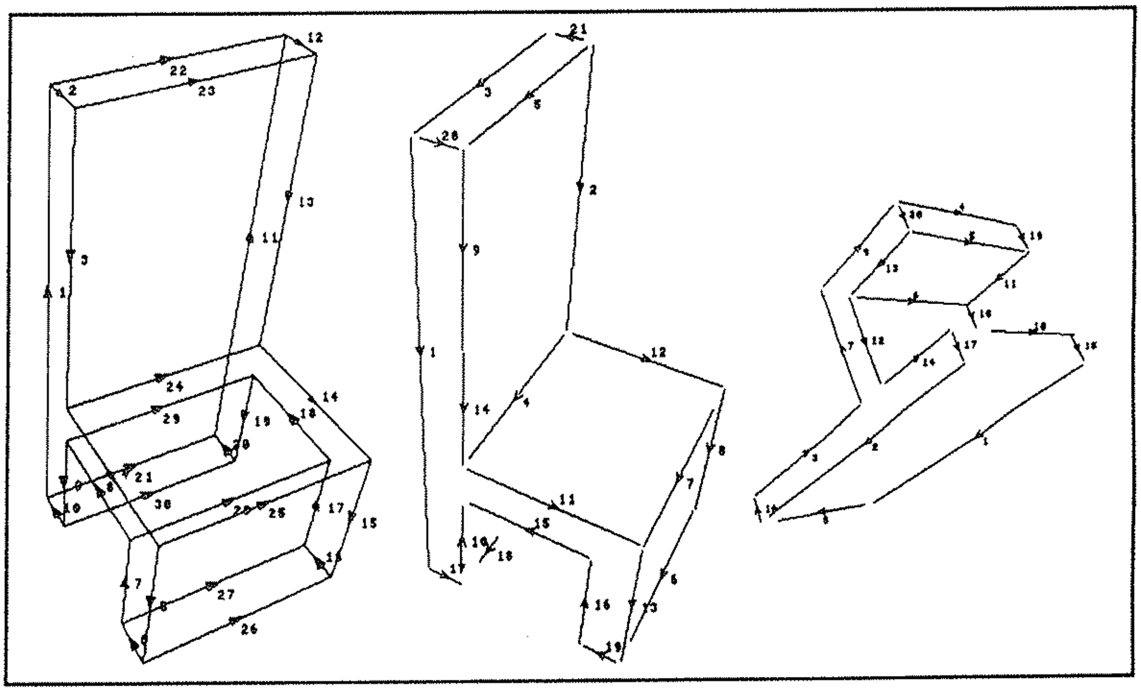
In order to try to determine the performance of the parallel matching algorithm, it is necessary to apply it to 'real' data. The data presented here are those recovered for the various views of the test models using the TINA tools visualisation system. These are prone to sensing errors and fragmentary data edges. Table 5.8 illustrates the number of data fragments recovered for each data scene. Values for the sensing errors are also presented.

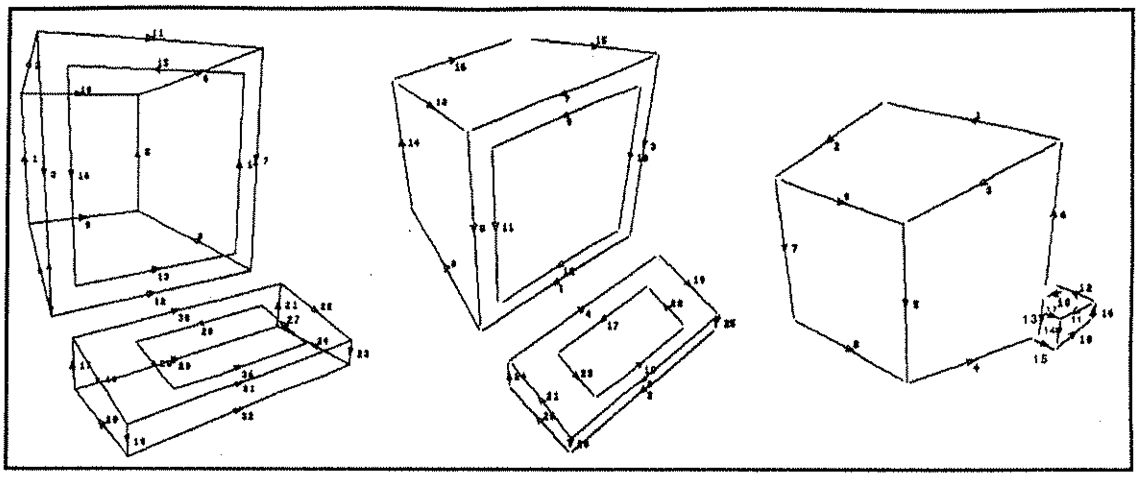
It is important to note that a scene may have more than one data edge composing the model edge. Under such circumstances, as is the case for bottom-up matching, more than one data fragment may be paired onto a single model edge. Such examples may be seen in Figure 5.22 for the recovered data fragments of the various test scenes.

Scene	Model edges (e)	Data fragments (s)	Potential Interpretations (e^s)	Errpar	Errper	Max. half cone error (°)
Chipped Block v1	15	12	1.30×10^{14}	5	5	5
Chipped Block v2		9	3.84×10^{10}	5	5	10
Chair v1	30	21	1.05×10^{31}	5	5	10
Chair v2		20	3.49×10^{29}	5	5	10
Computer v1	32	26	1.36×10^{39}	5	5	15
Computer v2		18	1.24×10^{27}	5	5	15
L-shape v1	18	18	3.93×10^{22}	5	5	5
L-shape v2		9	1.98×10^{11}	5	5	5
Wedge v1	15	12	1.30×10^{14}	5	5	5
Wedge v2		8	2.56×10^{9}	10	10	15

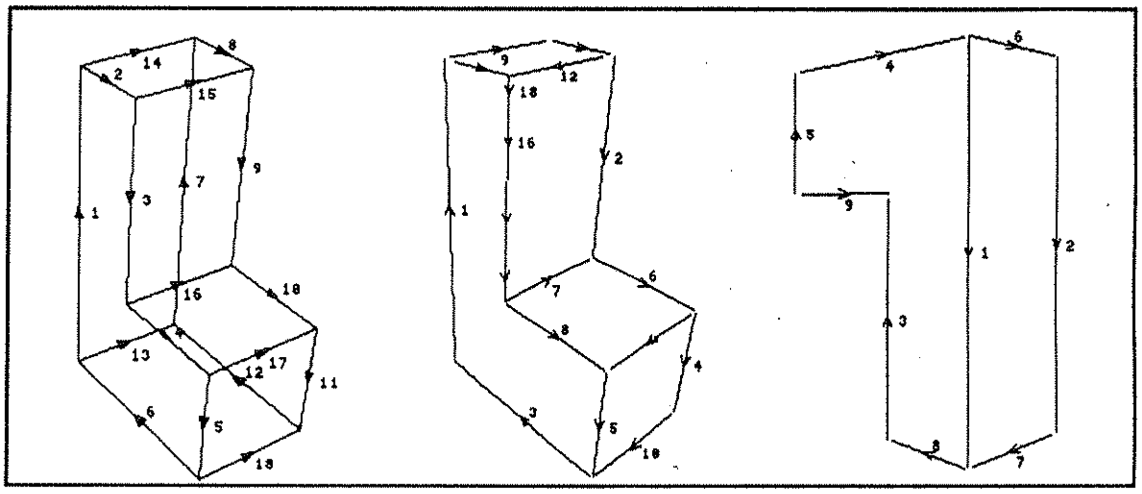
Table 5.8: Non-ideal data information


a) Chipped Block: Model - View 1 - View 2

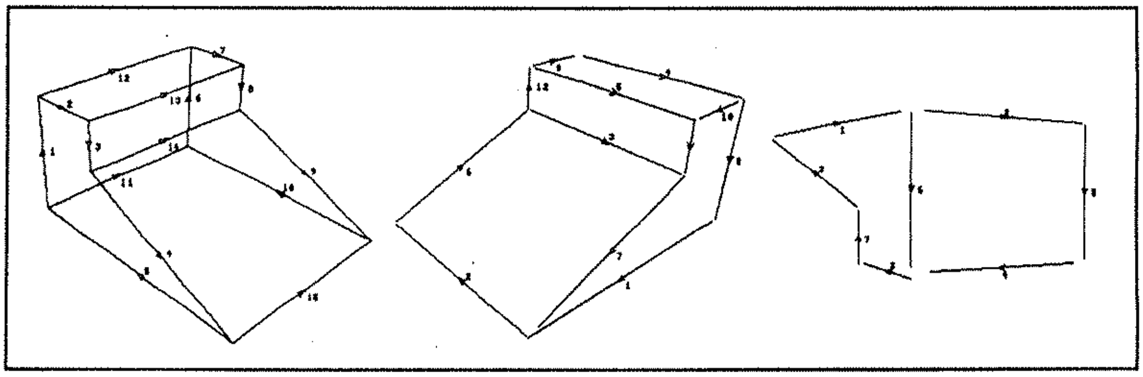

b) Chair: Model - View 1 - View 2



c) Computer: Model - View 1 - View 2



d) L-shape: Model - View 1 - View 2



e) Wedge: Model - View 1 - View 2

Figure 5.22: Recovered data edges for test scenes

5.11.3.5 Matching Using Realistic Data

The use of non-ideal data can demonstrate the effectiveness of the matching process in more realistic terms. The data recovered using the TINA stereo processing toolkit is examined here with all five pruning constraints switched on. This means that the timings discussed cannot be compared directly to those obtained in the ideal data case. It does, however, show the potential of the matcher in real terms.

In all tests, the parallel matcher (with Combined sort) was able to return the correct interpretation along with the direction signs (making multiple assignments where necessary). However, for both views of the Chair and the Computer view 1, an extra interpretation was found. These interpretations were composed of a single mispaired edge:

Scene	Data Edge	Model Edge	Sign
Chair view 1	12	18	-
Chair view 2	18	14	-
Computer view 1	18	31	+

We note that for both views of the Chair using the H&B sort, the parallel matcher was unable to return an interpretation for varying sensing errors in any reasonable length of time (i.e. there was a problem with obtaining a termination).

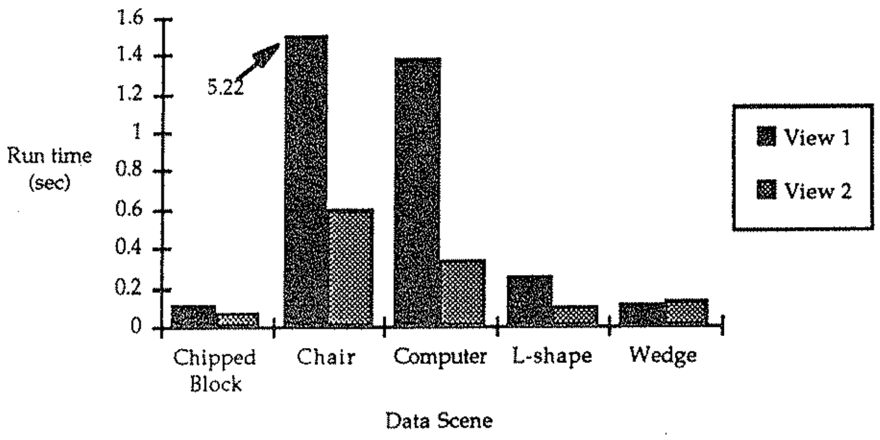


Figure 5.23: Matching time using non-ideal data

Figure 5.23 above shows the total matching times for the various views of the test scenes. Termination times are reasonably fast although some irregularities occur for view 1 of the Chair when compared to view 2. In both cases we should expect the angle constraint to have a weak effect - this is because all model edges are either parallel or perpendicular, and thus, only three angles are possible. Therefore, a possible explanation for the difference in matching times could be the fact that the view 1 match list has on average 34.4 matches for the first ten look-up tables compared to 15.0 for view 2.

5.11.4 The Validation Process

Determining the global validity of a feasible interpretation is an essential final process for the matching algorithm. Interpretations which may have before been passed as feasible can, in this phase, be filtered out as not possible. Determination of pose of the data can be used to eliminate incorrect matching features. For faces, it can be deduced whether each data face is actually visible given the pose and, for edges, whether each matched edge accurately maps onto its corresponding model edge after back-projection.

In this section we compare a Sun 4 serial implementation of the validation process to a parallel DAP 510 near equivalent. The parallel version implements the method of quaternions whereas the serial version uses a more simple, less effective, averaging system. Differences in this approach makes comparison of run times less objective. The power of the Sun 4 CPU along with the reduced parallelism of this phase (when compared with the matching process) also adds to make the parallel implementation appear somewhat weak. We also note that the serial version implements early cut-out when detecting a failed data edge. The DAP version terminates only when all data edges have been validated - this therefore gives a view of all failed data fragments.

Feasible interpretations for non-ideal data generated in the previous section are also validated and the results compared with the actual transformation from model to sensor space as obtained from the CSG solid modeller.

5.11.4.1 Veridical Transformations from Model to Sensor Space

The various transformations from model to sensor space for the two views of the five test scenes are given below. In each case the scale factor is unity.

Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Translation (x, y, z)	Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Translation (x, y, z)
Chipped Block v1	YROT(40) XROT(-20) $\begin{pmatrix} 0.7660 & -0.2198 & -0.6040 \\ 0.0000 & 0.9397 & -0.3420 \\ 0.6428 & 0.2620 & 0.7198 \end{pmatrix}$	(60.0, -40.0, 350.0)	Chipped Block v2	ZROT(-90) YROT(-130) XROT(-40) $\begin{pmatrix} 0.0000 & -0.7660 & 0.6428 \\ -0.6428 & 0.4924 & 0.5868 \\ -0.7660 & -0.4132 & -0.4924 \end{pmatrix}$	(-80.0, -50.0, 300.0)
Chair v1	YROT(60) XROT(-40) $\begin{pmatrix} 0.5000 & -0.5567 & -0.6634 \\ 0.0000 & 0.7660 & -0.6428 \\ 0.8660 & 0.3214 & 0.3830 \end{pmatrix}$	(10.0, -50.0, 200.0)	Chair v2	XROT(120) YROT(40) $\begin{pmatrix} 0.7660 & 0.0000 & -0.6428 \\ 0.5567 & -0.5000 & 0.6634 \\ -0.3214 & -0.8660 & -0.3830 \end{pmatrix}$	(-20.0, 0.0, 150.0)
Computer v1	YROT(40) XROT(-40) $\begin{pmatrix} 0.7660 & -0.4132 & -0.4924 \\ 0.0000 & 0.7660 & -0.6428 \\ 0.6428 & 0.4924 & 0.5868 \end{pmatrix}$	(70.0, -20.0, 300.0)	Computer v2	YROT(140) XROT(-30) $\begin{pmatrix} -0.7660 & -0.3214 & -0.5567 \\ 0.0000 & 0.8660 & -0.5000 \\ 0.6428 & -0.3830 & -0.6634 \end{pmatrix}$	(0.0, -80.0, 250.0)

a) Transformations for Chipped Block, Chair, and Computer

Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Translation (x, y, z)	Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Translation (x, y, z)
L-shape v1	YROT(40) XROT(-30) $\begin{pmatrix} 0.7660 & -0.3214 & -0.5567 \\ 0.0000 & 0.8660 & -0.5000 \\ 0.6428 & 0.3830 & 0.6634 \end{pmatrix}$	(10.0, -30.0, 150.0)	L-shape v2	XROT(180) YROT(40) $\begin{pmatrix} 0.7660 & 0.0000 & -0.6428 \\ 0.0000 & -1.0000 & 0.0000 \\ -0.6428 & 0.0000 & -0.7660 \end{pmatrix}$	(-30.0, 30.0, 120.0)
Wedge v1	YROT(-50) XROT(-30) $\begin{pmatrix} 0.6428 & 0.3830 & 0.6634 \\ 0.0000 & 0.8660 & -0.5000 \\ -0.7660 & 0.3214 & 0.5567 \end{pmatrix}$	(-30.0, 20.0, 250.0)	Wedge v2	XROT(180) YROT(30) $\begin{pmatrix} 0.8660 & 0.0000 & -0.5000 \\ 0.0000 & -1.0000 & 0.0000 \\ -0.5000 & 0.0000 & -0.8660 \end{pmatrix}$	(-30.0, 30.0, 200.0)

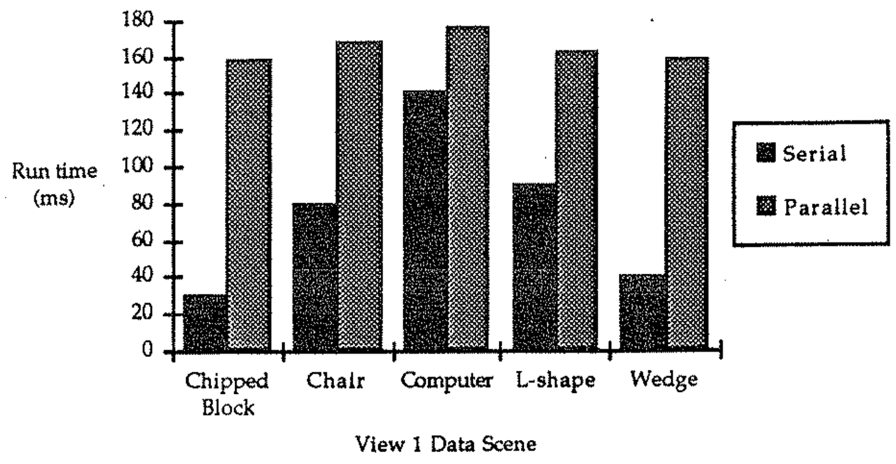
b) Transformations for L-shape and Wedge

Table 5.9: Veridical transformations from model to sensor space

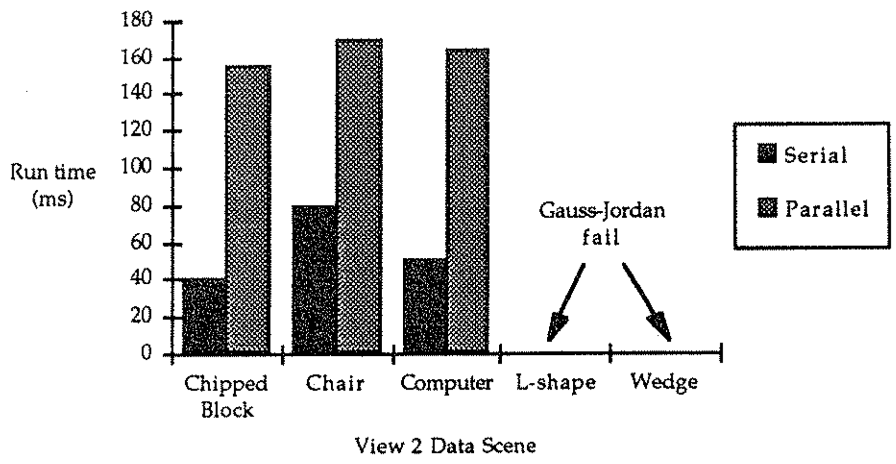
5.11.4.2 Serial vs Parallel - Validation of Ideal Matches

Being ideal data, the parallel validator was able to recover the 3D transformation from model to sensor space for most interpretations precisely! This was as expected, including the recovery of the scale factor as 1.0. However, for the L-shape and Wedge scenes (both view 2), the validator was able to recover the model to sensor space rotation accurately but failed for the translation and scale. This was due to the failure of the Gauss-Jordan process used in the determination of the translation for solving the system of linear equations $Ax = b$ for x, where A is a non sparse matrix. Being singular, this process was unable to operate on such rotation matrices.

A small word of caution must be expressed when examining timings obtained from the serial Sun 4 implementation. Because of the nature of the clock tick - 1 beat every 100th of a second - timings for very fast processes can be found to be inaccurate and vary for different runs using the same input data. Several of the processes (translations, scalings, and best fits) were found to have been executed in zero microseconds! On the DAP architecture, timings are measured as machine cycles - 10MHz. Figure 5.24 shows the CPU time expenditures for the validations of the ideal data interpretations with *fracdev* set to 0.1. It is important to note that the Jacobi process, implemented on the parallel version, for the determination of eigenvalues and eigenvectors, on average, was found to require 78.4% of the rotation estimation time.



a) Validation for view 1



b) Validation for view 2

Figure 5.24: Run times for serial and parallel validation

An advantage of the parallel implementation of the validation process over the serial version is the lack of effect of the number of data edges on the total validation time. This is because all data edges are validated in parallel. We can see from Figure 5.24 for the parallel implementation that although view 1 of the Computer contains 26 data edges, it only takes 1.13 times as long to validate than the Chipped Block (view 2) data scene containing 9 data edges. This is in contrast with the serial version where a difference of 3.5 times is observed for the same scenes.

Profiling the parallel validator reveals that the majority of the CPU processing time is given over to the rotation estimation. Of this, as mentioned earlier, a large percentage is taken up by the eigenvalue evaluation process. Figure 5.25 shows a profile for three data scenes. The trends seen are similar between the scenes. Set up of the DAP processors averages to 3.3% of the total validation time with estimation of the rotation requiring about 69.3% of the total CPU usage.

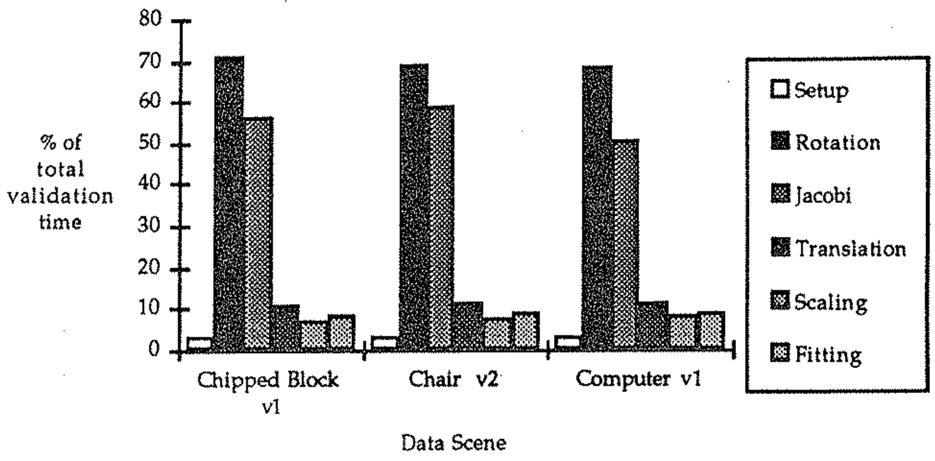


Figure 5.25: Validation profile for 3 data scenes

5.11.4.3 Validation of Non-Ideal Data Matches

Unlike the ideal data matches, interpretations using non-ideal data cannot be validated to the same precision accuracy. However, results obtained were very similar to those using artificially generated ideal data when we consider the sensitivity of the sensing process. Using a *fracdev* value of 0.1, each feasible interpretation obtained was passed but again with the L-shape and Wedge (both view 2) failing owing to a Gauss-Jordan failure. For the Chair (both views) and the Computer (view 1), only the correct interpretation was passed with the offending data edges failing when values for *fracdev* of 0.05, 0.05, 0.037 respectively were chosen. Results for the various data scenes are presented below.

Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Trans. (x, y, z) Scale (S)	Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Trans. (x, y, z) Scale (S)
Chipped Block v1	$\begin{pmatrix} 0.7608 & -0.2171 & -0.6116 \\ -0.0058 & 0.9401 & -0.3409 \\ 0.6490 & 0.2629 & 0.7140 \end{pmatrix}$	(61.14, -39.86, 350.51) S = 0.9991	Chipped Block v2	$\begin{pmatrix} -0.0013 & -0.7638 & 0.6455 \\ -0.6426 & 0.4952 & 0.5847 \\ -0.7662 & -0.4140 & -0.4915 \end{pmatrix}$	(-80.52, -48.79, 300.80) S = 0.9890
Chair v1	$\begin{pmatrix} 0.5062 & -0.5588 & -0.6569 \\ 0.0063 & 0.7641 & -0.6451 \\ 0.8624 & 0.3224 & 0.3903 \end{pmatrix}$	(9.59, -49.97, 201.93) S = 1.0010	Chair v2	$\begin{pmatrix} 0.7731 & -0.0050 & -0.6342 \\ 0.5486 & -0.4967 & 0.6726 \\ -0.3184 & -0.8679 & -0.3812 \end{pmatrix}$	(-19.07, -0.47, 151.04) S = 1.0051
Computer v1	$\begin{pmatrix} 0.7650 & -0.4026 & -0.5026 \\ 0.0043 & 0.7836 & -0.6212 \\ 0.6440 & 0.4731 & 0.6012 \end{pmatrix}$	(69.62, -22.50, 300.91) S = 0.9900	Computer v2	$\begin{pmatrix} -0.7611 & -0.3301 & -0.5583 \\ 0.0041 & 0.8584 & -0.5130 \\ 0.6486 & -0.3928 & -0.6520 \end{pmatrix}$	(0.11, -80.62, 253.24) S = 0.9869
L-shape v1	$\begin{pmatrix} 0.7665 & -0.3187 & -0.5576 \\ 0.0036 & 0.8704 & -0.4924 \\ 0.6422 & 0.3754 & 0.6683 \end{pmatrix}$	(9.72, -30.45, 150.84) S = 0.9966	L-shape v2	$\begin{pmatrix} 0.7631 & -0.0003 & -0.6463 \\ 0.0012 & -1.0000 & 0.0019 \\ -0.6463 & -0.0023 & -0.7631 \end{pmatrix}$	Gauss-Jordan Fail S = G-J Fail
Wedge v1	$\begin{pmatrix} 0.6441 & 0.3848 & 0.6611 \\ -0.0016 & 0.8649 & -0.5019 \\ -0.7649 & 0.3222 & 0.5577 \end{pmatrix}$	(-29.78, 20.17, 251.33) S = 0.9996	Wedge v2	$\begin{pmatrix} 0.8643 & 0.0002 & -0.5030 \\ 0.0066 & -0.9999 & 0.0109 \\ -0.5030 & -0.0127 & -0.8642 \end{pmatrix}$	Gauss-Jordan Fail S = G-J Fail

Table 5.10: Transformations for non-ideal data scenes

5.11.5 Subgraph Matching of a 3-Pin Plug

Hitherto, it has been shown that the parallel matching algorithm works well on models of low to medium range complexity. In this section we investigate the performance of the matcher on a more complex model - a 3-pin electric plug. We compare the performance of the subgraph matcher to a Fortran* equivalent, the prime aim being to show that a large model can be divided into a set of smaller sections to which the match process may be applied separately. We must note that the subgraph matching of large models was undertaken before the introduction of the Fortran* language which makes it possible to use unconstrained matrices. Matching times for the Fortran* version show a mean 2.4 times speed up over the subgraph matcher. Again, speed of execution was not the prime aim of this section but merely to show that the look-up table structure of the parallel matcher may be divided into sectors to cope with large model scenes which cannot be mapped directly onto the DAP processor arrays.

Two views of the Plug are presented. The first is unambiguous with the pins clearly visible. The second is less so and is taken from the back of the Plug with no pins visible.

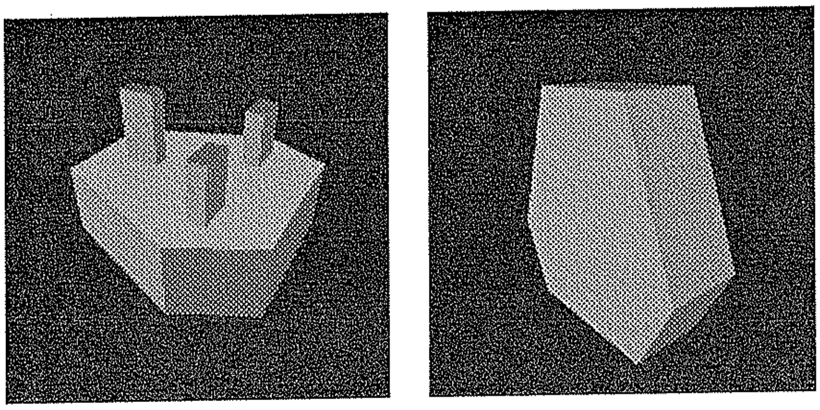
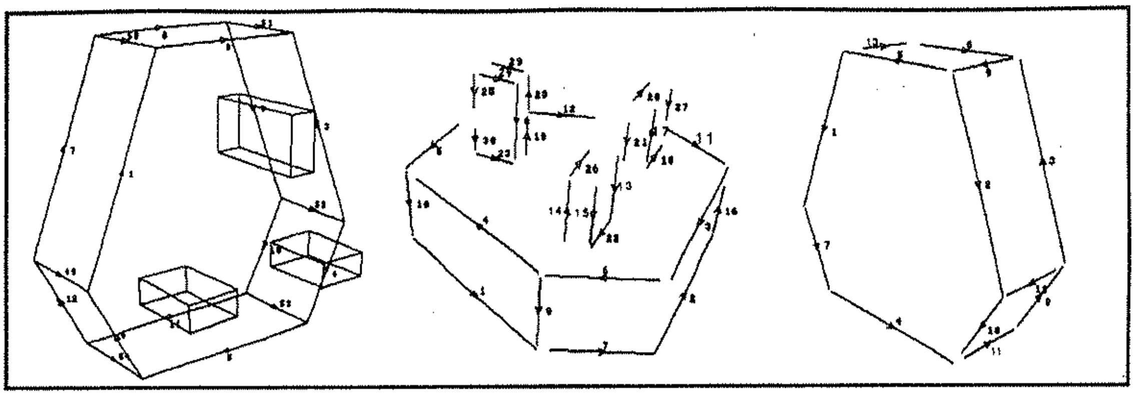


Figure 5.26: Views 1 and 2 of the 3-pin Plug

The model is constructed with 8 CSG primitives using 48 halfspaces and the stereo pairs were rendered in 21.7 and 18.4 seconds for views 1 and 2 respectively. It is composed of 54 edges with 40 being visible in view 1 and 13 in view 2.

Figure 5.27 illustrates the data fragments obtained from the two views using the TINA system.



Note: The pins of the Model are labelled thus - looking at Plug pin end-on with earth pin at top - edge directions go clockwise and outwards.
Key F=Face, N=North, E=East, S=South, W=West.
Top pin: W=21, N=22, E=23, S=24, FW=25, FN=26, FE=27, FS=28, SW=41, NW=42, NE=43, SE=44.
Left pin: W=13, N=14, E=15, S=16, FW=17, FN=18, FE=19, FS=20, SW=37, NW=38, NE=39, SE=40.
Right pin: W=29, N=30, E=31, S=32, FW=33, FN=34, FE=35, FS=36, SW=45, NW=46, NE=47, SE=48.

Figure 5.27: Recovered data edges for Plug :- Model - View 1 - View 2

For both ideal and non-ideal data matches, the five pruning constraints were applied. Table 5.11 shows the total matching time for the two data forms with comparisons between the Fortran* and the subgraph implementations. Note that using the H&B sort, both implementations of the matcher failed to terminate for the non-ideal view 1 data scene in a reasonable period of time.

Scene	Implementation	Run time using ideal data (sec)	Run time using non-ideal data (sec)
Plug view 1	Fortran*	1.80	15.24
	Subgraph	4.54	31.48
Plug view 2	Fortran*	0.28	0.28
	Subgraph	0.61	0.63

Table 5.11: Run times for Fortran* and subgraph implementations

The correct interpretation was returned in all cases. Not surprisingly, though, both returned an additional interpretation - the rotation of the Plug for view 2:

Data edge	1	2	3	4	5	6	7	8	9	10	11	12	13
Model edge	1	3	9	5	2	8	6	10	51	4	53	52	50
Sign	-	+	-	-	-	+	-	-	+	+	-	+	-

The veridical transformations from model to sensor space are given in Table 5.12.

Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Translation (x, y, z)	Scale
Plug v1	XROT(90) YROT(60) XROT(-35) $\begin{pmatrix} 0.5000 & -0.4967 & -0.7094 \\ 0.8660 & 0.2868 & 0.4096 \\ 0.0000 & -0.8192 & 0.5736 \end{pmatrix}$	(-25.0, -40.0, 110.0)	1.0000
Plug v2	YROT(130) XROT(-20) $\begin{pmatrix} -0.6428 & -0.2620 & -0.7198 \\ 0.0000 & 0.9397 & -0.3420 \\ 0.7660 & -0.2198 & -0.6040 \end{pmatrix}$	(-30.0, -40.0, 120.0)	1.0000

Table 5.12: Veridical transformations for Plug

As expected, the ideal interpretations were accurately validated. However, for view 2 of the Plug, the parallel validator was unable to distinguish between the two interpretations and passed them with equal values. The additional interpretation of above was validated thus:

Rotation $\begin{pmatrix} 0.6428 & 0.2620 & 0.7199 \\ 0.0000 & 0.9397 & -0.3420 \\ -0.7660 & 0.2198 & 0.6040 \end{pmatrix}$

Translation (-6.7534 -19.8829 175.2710)

Scale 1.0000

Using non-ideal data, however, the validation process was able to pass both interpretations of view 2 but with the genuine interpretation receiving a greater value. Results of the validation can be seen in Table 5.13.

Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Translation (x, y, z)	Scale
Plug v1	$\begin{pmatrix} 0.5053 & -0.4949 & -0.7069 \\ 0.8630 & 0.2859 & 0.4166 \\ -0.0041 & -0.8206 & 0.5715 \end{pmatrix}$	(-24.79, -39.91, 111.03)	0.9991
Plug v2	$\begin{pmatrix} -0.6556 & -0.2588 & -0.7093 \\ -0.0020 & 0.9400 & -0.3411 \\ 0.7551 & -0.2222 & -0.6168 \end{pmatrix}$	(-30.41, -39.48, 121.43)	0.9877

Table 5.13: Validated transformations for Plug

5.11.6 Applications of the Bin to Matching

The null-pairing or bin theory is investigated in this section. Tests are conducted using the non-ideal data recovered from the Chipped Block and Wedge data scenes (both view 1). For each data scene, one and two spurious data edges are included, and the five pruning constraints are applied. Inclusion of these spurious data edges would normally result in non-interpretations without the bin implementation. However, here, the correct interpretation was returned for all cases with the offending data edges correctly matched to the bin. Figure 5.28 shows the total matching times for the determination of the correct interpretation using Combined sort for the two data scenes. It should be noted that for the H&B sort, all attempts to include spurious data edges failed to terminate after 2000 seconds.

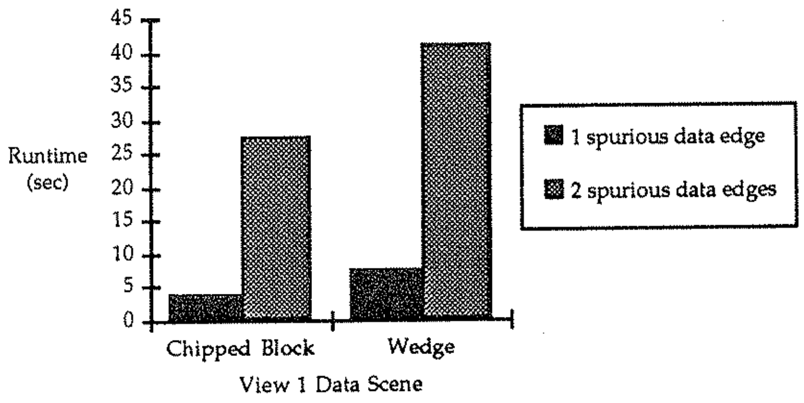


Figure 5.28: Run times with spurious data

The returned interpretations were correctly validated with the transformations equal to those seen for the Chipped Block (view 1) and Wedge (view 1) non-ideal validations in Section 5.11.4.3.

5.12 Conclusion

It has been shown that the matching algorithm of Grimson and Lozano-Pérez can be mapped efficiently onto the array processors of the DAP. The parallel algorithm has been demonstrated using scenes of varying complexity. Comparison to a serial implementation on a Sun 4 workstation showed a marked improvement in run time. A problem with the serial matching process is knowing how best to order the pruning constraints in order to produce effective cutoffs. The distance constraint has been shown to be the most effective constraint and can therefore be applied first. However, establishing the order of the remaining constraints is not so clear-cut. This is not a problem for the parallel matcher as all constraints are applied at the top level and the results stored. On the subject of constraints, although the determination of direction signs is an overhead which will not be found in a face matching equivalent of the algorithm, it has been shown to act as an additional constraint. Experimental evidence has shown that with the determination of direction signs in the matching algorithm, the search effort in the serial tree traversal is reduced. However, the processing time required to determine the edge signs is much larger than that saved from the reduced search effort.

In addition, we have demonstrated the need for effective sorting of the match look-up tables and presented a sort technique which in most cases shows a major improvement over the ascending order sort of Holder and Buxton. The time to do this Combined sort process was also found to be less, so we have achieved two speed ups, the first being to sort the match list, and the second being the subsequent tree traversal.

Using the array structure of the parallel algorithm, it is possible to match models with features larger than the edge size of the DAP. This has been shown both with subgraph matching and the unconstrained matrices of the Fortran* language. Although the Fortran* implementation performed better than the subgraph matcher, the subgraph method was not considered a failure as the prime objective was to demonstrate that large models can be sub-divided and the parallelism of the algorithm maintained.

We have also shown the H&B sort technique can fail to produce an interpretation in a reasonable period of time using an implementation of the bin. The Combined sort technique presented (with the bin) was able to return the correct interpretation with the addition of spurious data edges. These, however, were recovered after a longer period of time when compared to ones with no inclusion of spurious data and goes to show how essential it is for the match list to be effectively ordered.

Some limited experiments with the confusibility of models was also undertaken for completeness but the results are not presented as they were straightforward. The models were not confused in any of the cases with and without the distance constraint. Also, ordering the constraint sign tests so that constraints which can determine direction signs absolutely (i.e. direction constraints 1 and 2) were encountered first was found to not have any significant effect on the tree traversal time.

Using the parallel ray tracer for closed-form testing proved very effective in producing stereo image pairs in about half a minute. However, one remaining problem is that it is currently rather tedious to calculate edge positions by hand for the model look-up tables of the matcher. We know that automated calculation is possible for CAD models and, thus, such a retrieval of model edges directly from the CSG tree would be very convenient and is an avenue for future work.

Chapter 6

From Polyhedra to Planar Curves

6.1 Introduction

Objects consisting of planar surfaces with their straight line edges are easier to model than those comprising of less uniform features. The simplicity of the planar faces or linear edges of polyhedra makes the task of recognition more feasible but not trivial. It is for this reason that many authors have reported work on the analysis of polyhedral objects. Using edge and surface normal matching features, the complexity of a model and the matching process may still be high. Another reason why the recognition of planar objects has been popular is related directly to current sensing algorithms and equipment. Segmentation and fitting of straight lines to produce planes is, on the whole, easier and more robust than for curved surfaces. However, in a world full of curves, a dedicated polyhedral recognition system is very limited. It is therefore important for such systems to be extended to curved objects, or objects containing curved features, especially if the system is to operate in a real-world environment, e.g. a robot visual system.

An obvious problem with the recognition of curved surfaces when thinking in terms of edges is that they do not necessarily possess real, creased edges. Very often, a simple curved geometric model has no discontinuities. An obvious example is a sphere. An edge-based approach would therefore result in occluding edges being found as opposed to 'real' polyhedral edges. However, in the case of other primitives, such as cylinders and cones, it is possible to recover true edges in the form of their planar ends. Using such planar features, it is possible to apply a polyhedral based recognition algorithm, such as the Grimson and Lozano-Pérez matching technique, to the primitive. It is, however, still not possible to incorporate the developable surfaces of such primitives directly.

In the following sections we will present an extension of the polyhedral edge-based matcher to incorporate curved, creased edges. This will be able to deal with objects containing planar curves, such as the class of objects defined by a developable surface and, may easily be adapted for face matching. The approach we adopt uses the surface normal of the planar ends and the axis of rotation of the primitive. These are treated as 'real' edges by the matcher and processing proceeds using the Grimson and Lozano-Pérez matching paradigm of the parallel model matcher described in the previous chapter. We also present results of this approach from tests conducted on various scenes including a real world environment.

In general, we can only consider those primitives which can be recovered accurately from the scene. This is very much dependent on the sensing technique. Using the TINA system, it is possible to recover the circular ends that define a cylinder. However, recovery of the axis of rotation is not at present possible using the current version. Therefore, we must determine the axis of rotation by hand. A simple automation of this step could be to fit a quadric surface between consecutive planar curves using a set of rules defined by the class of primitives. If the fitting is possible, the two curves can be taken as belonging to the same primitive and the axis of rotation may be determined. Other sensing techniques, for example, a range-finder, may be use to determine this feature more easily.

6.2 Curved Object Recognition Systems - A Brief Overview

In an early 1983 paper, Oshima and Shirai [OS83] describe work on a system which is able to recognise a set of stacked objects containing planar and curved surfaces. Using scene data obtained from a range-finder, the system is able to describe the scene in terms of regions and relations between these regions.

Unlike many reported approaches, the system is autonomous, generating its own model of the scene. This is achieved in the first of two phases. Here, in the 'learning phase', the system is presented with a series of scenes containing a single known object, one at a time. With each, it builds a description of the object. Several views of the object may be taken. In the second phase, the system is again presented with a scene, this time containing unknown objects. In this 'recognition phase', the scene is described in the same way as in the learning phase. Using these descriptions the data features are matched to the model features.

Description of the model and data scenes proceeds in five main stages:

1) The depth map recovered using a range-finder is grouped into small surface elements.

2) These are merged together into regions termed *elementary regions.*

3) The elementary regions are classified either as planar or curved.

4) The curved regions are extended by merging to adjacent curved surfaces and quadric surfaces are fitted to them.

5) Finally, the scene is described in terms of these regions. These include the type of surface, the number of adjacent regions, the area of the surface, and the mean, minimum, and maximum radius.

The matching then proceeds by determining suitable pairings of data to model regions. Experimental results have shown the method to be successful with scenes containing objects with smoothly curved surfaces such as cylinders and cones. Curved machine parts and spheres have also been successfully matched.

A system, able to automatically generate an object recognition strategy from a 3D model, and then recognise the object using this strategy, is described in [KOO90]. As in [OS83] the system is divided into two stages: the strategy generation stage and the recognition stage. Objects are described from various viewpoints using 2D features such as parallel lines and ellipses. For example, two ellipses and a pair of parallel lines are the intermediate features of a cylinder. These features are ranked as to their visibility from various viewpoints to generate a strategy graph which is searched for similarities between the model and image features. Tests using valves and industrial tools containing cylinders have been successfully reported. However, the system was sometimes found to detect false objects, i.e. objects which do not exist in the scene, when the background is complex.

Unlike the systems described hitherto, BONSAI [FJ90] uses an inherently Grimson and Lozano-Pérez algorithm with constrained search of an interpretation tree. Unary and binary constraints are applied to data obtained from a range image to extend matching to curved surface primitives. It compares relational graphs constructed from the data to that obtained from the model. The use of NULL-matching means that spurious data entities may be present.

The models are composed of planar, cylindrical, or spherical surfaces and unary and binary features are calculated using over 300 synthetic viewpoints. Data features are obtained using regression and curvature-based techniques [Flyn90]. Matching commences with curved surface patches, initially starting with the largest patch, and then planar patches if necessary. Pruning of the interpretation tree proceeds as the tree is traversed. This is achieved by the application of *predicates* to the interpretations as they are generated. At present three unary and four binary predicates have been implemented. These return TRUE or FALSE depending on satisfaction or failure by the interpretations. The predicates are similar to the description of scenes adopted by Oshima and Shirai [OS83] and are applied to the 'interpretation so far'. The unary predicates determine area, type, and radii of the surface patches. The binary predicates are applied to pairs of associations in the interpretation. These adopt global consistency checks in order to determine correct pair associations. The *rotation validity* and *orientation* predicates take all pairs of associations in the interpretation and compare the rotation and orientation parameters of the data to the model pairs. The visibility of the surfaces are checked by another predicate which is satisfied only if, for all non-NULL paired associations, the model patches can be simultaneously visible from some view. Finally, a *parallel plane* predicate is satisfied if, for a parallel data plane pair, the distance between the parallel model plane pair corresponds within a small tolerance value. Surviving interpretations are verified by synthesizing a range image with subsequent segmentation of the generated object. Pixel-to-pixel comparisons of synthesized to input range images are made to determine the correct interpretation from an area-based matching value.

The systems described thus far have all been restricted to data obtained via a range detector and are, therefore, sensor dependent. However, the problem with the use of stereoscopic or motion techniques is concerned with the robustness of curved surface recovery. Approaches have been documented which use optical flow [BB88, BC89, BCZ90], and stereoscopic vision [Wein90] in order to recover surface curvature including along the extremal boundary. These, however, have relied on specular highlights and feature-rich surfaces to determine the degree of curvature of surface contours.

6.3 Extension of the Edge-Based Matcher for Planar Curves

Using stereoscopic or motion techniques, it is possible to recover the creased, curved edges at surface boundaries. The recovery does not rely on specularities or a feature-rich surface, but on contrasts between surfaces. The curved edges may be reconstructed to form a plane which may be described by the direction of its surface normal and the perpendicular distance of the plane from the origin of the coordinate system of the environment. As is seen with polyhedral face matching, the surface normal is an important matching feature in model-based object recognition. It may be recovered using several image processing algorithms, e.g. structure from stereo, or structure from motion, or sensing apparatus such as laser range-finders or tactile sensors. Also important for the use of the planar surface normal of a primitive as a matching feature is that it can be recovered with a reasonable amount of accuracy. When considering the recognition of curved objects, it is at present difficult to recover robust matching features from say, stereopsis, due to the lack of true edges often seen (or shall we say, not seen) in primitives such as spheres, ellipsoids, and toruses. Using trinocular vision, however, it may be possible to recover the curvatures of such objects and, possibly, use these as matching components. Other curved objects, which may also be described by a quadric function, e.g. cylinders and cones, are not devoid of true edge boundaries. The ends of these primitives define a plane which, obviously, can be defined by a surface normal. It is this surface normal which we propose to use in an edge-based planar curve matcher. The approach is essentially the same as in the polyhedral matcher of Chapter 5 with the surface normal of the primitive being treated as a true edge. Using this approach, the polyhedral algorithm may easily be extended to a face-based system which will use the planar faces of the cylinder and cone primitives. Another matching feature relating to the planar ends of primitives is the axis of rotation. This can be viewed as the line joining the centres of the planar curves in the case of the cylinder and cone.

The extension of the polyhedral algorithm is explicitly directed at the class of primitives which can be defined by a developable or 'ruled' surface, i.e. cylinders and cones, although the method still holds for a more general class of planar curved objects. A developable surface is defined here as belonging to the group of surfaces which has one of its principal curvatures equal to zero. Using such a surface ensures that an axis of rotation will exist.

6.3.1 The Planar Curve Normal as a Recognition Aid

The outward surface normal of a planar face is an important feature in recognition as its direction relative to the model remains constant irrespective of orientation, translation, and scale. Also, it can be recovered with a reasonable amount of accuracy using standard visualisation techniques. The use of the planar curve normals in an edge-based fashion means that on entry to the tree traversal phase, the direction signs of the normals will be known absolutely and can be used to determine the signs of 'true' edges more effectively.

Treating the normal as an edge means that it has a unit length of 1.0 and its 3D start point at the centre of the circular or elliptical planar curve. With the addition of the planar curvature, objects containing curved primitives can therefore be represented with a reasonable amount of accuracy when compared to the simplicity of the method, i.e. the primitives are defined by surface normals, centres of curvatures, and radii for regular cylinders and cones. Figure 6.1 shows a number of primitives which can be represented in this way. We can see that the normals need not necessarily be anti-parallel.

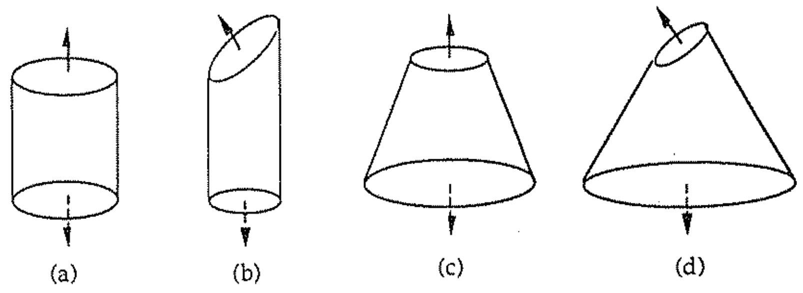


Figure 6.1: Normal 'edges' for various primitives

Using this methodology, the elliptical ends of sliced cylinders and cones may also be represented. In order to improve the representation of the primitives of the above Figure, each surface normal is defined along with the size of the planar curve. This is merely the radius of the circle in the case of circular ends, and the major and minor axes in the case of elliptical sliced ends. Using this, it is now possible to distinguish between the regular cylinder (a), and the regular cone (c), and also between the cylinder (b) and cone (d), of Figure 6.1. It is, however, not essential that a pair of normals be recovered during the segmentation process. A circle drawn on top of a box should give a sufficient visual cue to determine the orientation of the box.

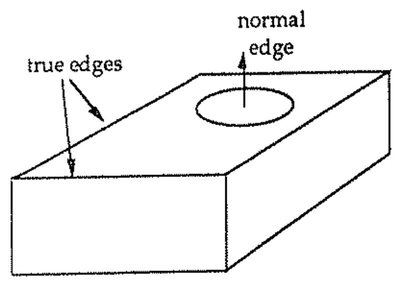


Figure 6.2: The distinguishing feature of a box

6.3.2 Addition of Axis of Rotation

If the segmentation process allows, the axis of rotation of the curved primitive may be recovered and used in the matching process. Here, the axis of rotation is taken as the line joining the centres of the planar curves of the primitives. This can be used to impose additional constraints in the recognition process when we treat it as an edge. Unlike the surface normal, treating this line as a vector does not produce a unique direction as we see for true edges. In the case of regular cylinders and cones, this new 'edge' can often coincide with the planar end normals and can act as the length of the primitive.

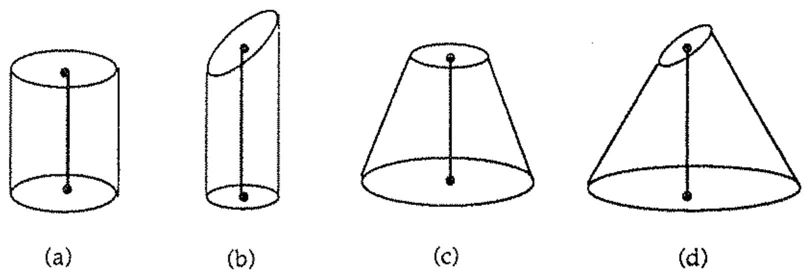


Figure 6.3: Axis 'edges' for various primitives

We note, however, that this matching feature does not take into account the size of the planar curves as in the case of the surface normal to the planar ends. Therefore, on its own, the regular cylinder (a) may not be immediately distinguishable from the regular cone (c) of Figure 6.3. This, however, does not pose a problem, as in order to obtain the axis of rotation, some data relating to the curved surface must be available.

It is not strictly necessary that a 'real' planar curve must be present for the described method to work. The planar curve which is inferred by a cylindrical cut-out may be recovered and used in the algorithm. This is possible using the TINA system. Also, the deduced axis of rotation of the cylindrical cut-out may also be used (depending on whether recovery is possible) in this approach. Objects of the type shown below may now be capable of more accurate determination.

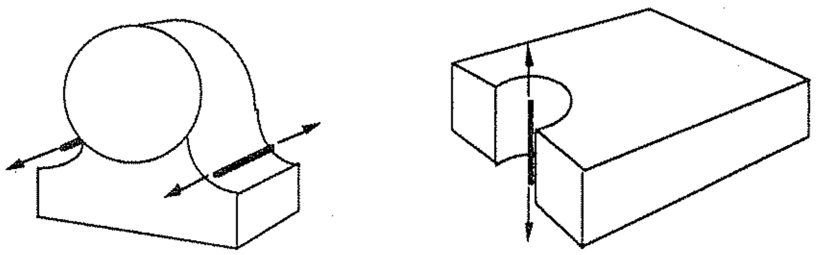


Figure 6.4: Extended range of planar curves

6.4 Recovering the Pseudo-Edge Match Features

We understand that it is possible to recover the outward surface normal of a plane and the curvature of a cylindrical type primitive by a number of means. Immediate examples are tactile sensors and laser range-finders. In our work, we use the TINA visualisation system. This is capable of recovering the 3D planar circle from a stereo image pair. This is achieved via a segmentation tool within the system which is able to produce sections of a curve using a number of segmentation strategies. Initially, a linear segmentation algorithm amalgamates adjacent edge fragments into curves when their combined edge strings can be sufficiently closely approximated by a circle. After this initial 2D segmentation, the curved edge strings may be fitted by higher level primitives which at present are interpolating cubic splines and conic sections. We choose to fit a conic using a Bias Corrected Kalman Filter (BCKF) after an initial fitting which passes through five well spaced points on the curve. This fitted 2D curve may be interpreted in 3D in several ways of which we adopt the planar method. This involves projecting it onto the 'best fit' plane in 3D.

The current version of the TINA system is limited to the recovery of circles and does not, at present, deal with the unique determination of ellipses. In our experiments, we will therefore be restricted to deal with uniform cylinders and cones which have circular planar ends. Also, it is not always possible to recover the direction of the outward surface normal uniquely as the planar face may not necessarily be visible for the curve to be segmented. This can be seen below in Figure 6.5.

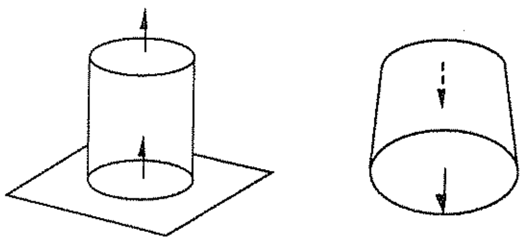


Figure 6.5: Ambiguities of surface normal directions

By treating the surface normals as edges, this is not a problem. It simply means that the normal can not be used uniquely to determine the direction signs of the axis and true edges absolutely, i.e. if we know the direction of the normal, we can determine the direction of subsequent edges taken as a pair with it (see Section 5.5 in the previous chapter). The axis of rotation is at present recovered by hand as it is not incorporated in the system. However, a simple modification of the TINA system may be made to determine this for the more regularised cylinders and cones.

6.5 Sensing Error of Pseudo-Edges

As was seen in Section 5.4, equipment and algorithmic sensing errors must be taken into account when considering data recovered by non-ideal means. The error handling used here for the pseudo-edges is similar to that used for the true edge. We assume that the centre of the planar curve is recovered within a sphere of error, *errsphere*, which is determined by by the user in accordance with the accuracy of the sensing methodology. The error half cone angle for the pseudo-edge normal is also user set. This is obviously because being so short (a length of unity), it is very easy for the calculated half cone angle to approach 90° (see Section

5.4). For the axis of rotation, we calculate its half cone error angle in the same way as for true edges by simply substituting values of *rpar* and *rper* with *errsphere*. In our experience, however, using the TINA system, we have found these error values, i.e. for curves and edges, to be very similar.

Also, to maintain consistency with the edge approach and to ensure that the normal length is contained within that of the model, we 'pinch' its end points inwards by 0.1. This value is used rather than *errsphere* (for true edges *rpar*) due to the value of the latter being so large (usually a value of 5.0 has been used) that the direction of the normal will obviously become negated. For the axis pseudo-edges *errsphere* is used.

6.6 Pruning and Match Look-Up Tables

A problem with an edge-based matcher is that we do not know absolutely the direction of the data edges with respect to the model edge initially. This required during validation and must be determined during the tree traversal phase of the matching process. Using the surface normal of a curve emanating from the centre of the circle we have two options on how we apply some of the pruning constraints to the data. These constraints are those described in the previous chapter, i.e. angle, distance, direction 1, direction 2, and direction 3. The choice comes when we consider the distance, and directions 1, 2, and 3 constraints. These involve running up and down the length of the edges and determining a range of distances or angles. We can either apply the constraints to the normals, treating them as an edge, or explicitly target the centre of the planar curve. By the latter, we need only find the range of distances, or angles from the centre of the planar curve to any point on a real or axis edge in order to produce the match look-up tables. This means, however, that we need not be concerned with running along the normal edge and that calculations are simplified, but it still involves a range of measurements (with the exception of the curve-curve pair). We therefore adopt the former with the advantage that we can recover the direction of the planar surface normal depending on the sensing apparatus. This can then be used to recover the direction signs of subsequent edges more efficiently.

By treating the new matching features as edges, we find we only require one type of model look-up table (edge-based), not two; 1) edge-based for true and axis edges and 2) face-based for planar curves, as is also possible. The parallel implementation is equivalent to that described in Chapter 5 with the exception that given *a priori* knowledge of match feature type, a curve edge is only permitted to pair with another curve edge. This also applies to true edges and axes. The edges are therefore ordered by type and a list identifying the type of each feature is maintained in the order

[(true edge) 1...i, (curve edge) 1+i...1+i+j, (axis edge) 1+i+j+1...1+i+j+n]

where i, j, and n are the number of true, curve, and axis edges respectively.

Six logical masks are used to ensure that it is only possible for similar match features to pair. These are illustrated in Figure 6.6 for a six edge feature model with two of each data type recovered.

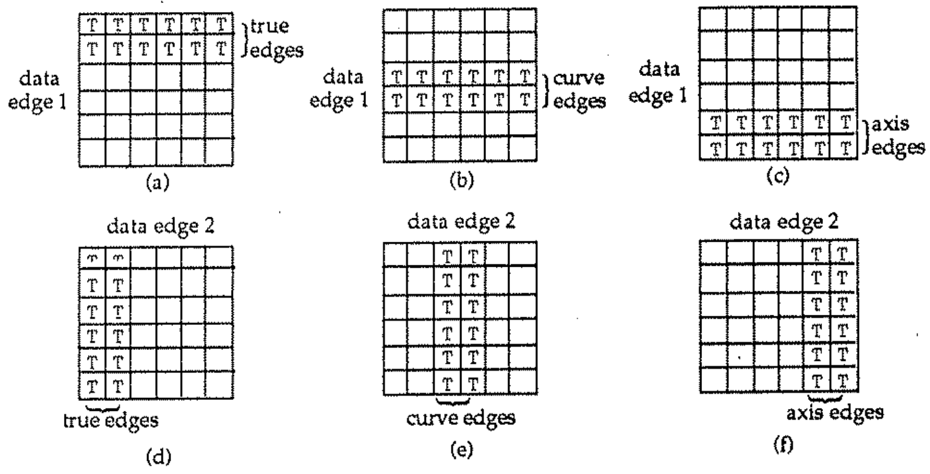


Figure 6.6: Logical match feature tables

After the pruning constraints have been applied to the data pair, appropriate match feature masks are applied to the match look-up table depending on the types of the data pair. For example, after applying the constraints we find that several locations in the match look-up table have passed the tests even though it is not valid (given *a priori* knowledge) for say, a true edge to pair with a curve. If data edge 1 is say, a true edge, and data edge 2 is a curve edge, we can eliminate these mispairings by applying masks (a) and (e) of Figure 6.6 to the match look-up table. This is a parallel operation where the three logical matrices are simply AND-ed. Equally, the appropriate masks may be AND-ed and stored before determining the match look-up tables. This will, however, require the storage of nine matrix masks. Generation of these masks, including those of Figure 6.6, is also parallel and is executed efficiently on the DAP.

After this removal of mispaired edges, the match look-up tables (the match list) may be searched in the tree traversal phase of the matching process. This is equivalent to the method described in Chapter 5, Section 5.7.

6.7 Validation of Interpretations

The validation of feasible interpretations recovered here is essentially the same as the parallel implementation described in Chapter 5, Section 5.9. The only difference is that we must take into account the size of the planar curves. No modifications need be made to the determination of the best rotation. This involves unit position vectors of the edge features being rotated to correspond to their matched unit model position vectors. The best rotation is determined by averaging these over all data features. Determination of the best translation and scale may also remain unmodified.

Fitting of the data edges to the model is again essentially the same as in the edge validation of Chapter 5. However, we are faced with a minor decision about how to treat the edge normals. If we apply the deviation computed using *fracdev* and *maxdist* (described in Section 5.6) we will find that the normal edges will very often pass the fitting test as the deviation is likely to be greater than the length of the normal edge. After consideration, however, we see that if the data curve edge is badly oriented to its corresponding model curve edge, say, by 90°(!), it would not have passed the pairwise pruning constraints during the matching phase. We, therefore, fit curve and axis edges in the same manner as for true edges, that is, ensure that their endpoints fit sufficiently close to their respective model edges (see Section 5.6).

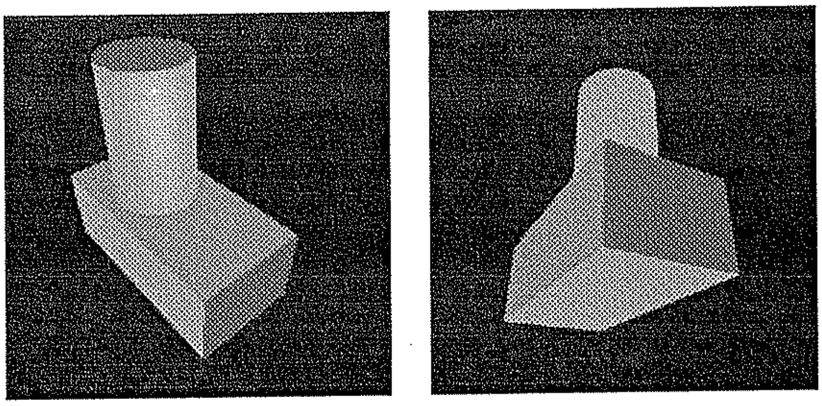
After this initial fitting of curve edges, we must ensure that the data curve radii of matches that pass the local constraints for the look-up tables fit closely to their respective model curves. This is determined using a deviation calculated as $curvedev \times maxradius$, where *curvedev* is a user supplied curve deviation factor, and *maxradius* is the largest curve radius determined from the model curves. To pass this test, all data curves must fit their respective model curves to within the deviation, that is:

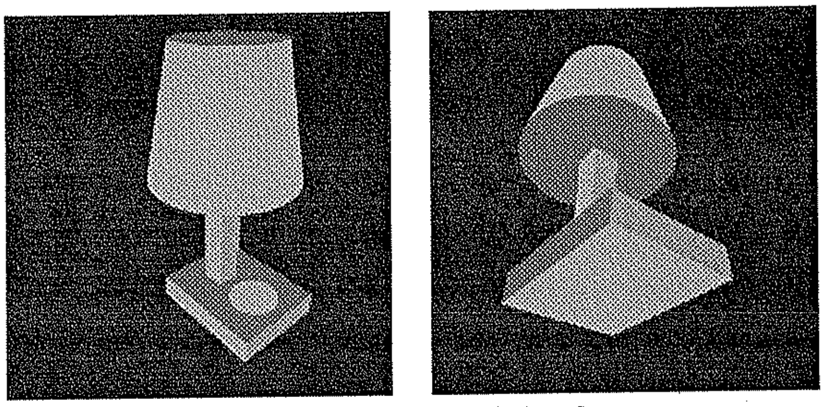
$$\text{pass} = \text{MAX}(\text{DataRadius} - \text{deviation}, 0) \leq \text{ModelRadius} \leq \text{DataRadius} + \text{deviation}.$$

In the case of elliptical ends, we shall use the length of the major axis when establishing the value of *maxradius*. Both major and minor axis must fit the ellipse to within the calculated deviation.

6.8 Experimental Results

In experiments conducted in this section, models composed of, or containing circular planar curves, were investigated. Three models were used; a CylinderBox, a Lamp, and a set of Tubes. As in the previous chapter, two views of the models were taken differing in ambiguity with exception to the Tubes which, using the current recovery technique, was difficult to obtain enough data features for accurate determination of the correct interpretation and, more important, the global consistency. We are reminded from Section 5.6 that the determination of the rotation requires at least two pairs of non-parallel edges and, for the translation, three pairs of independent edges. The views are presented below.


a) CylinderBox: view 1 and view 2


b) Lamp: view 1 and view 2

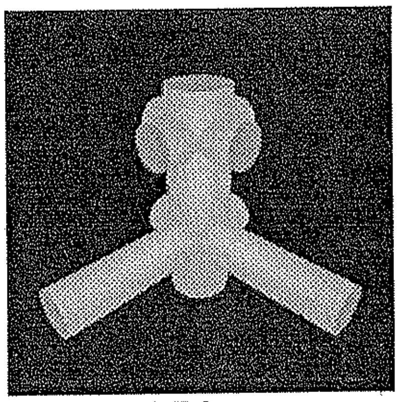

c) Tubes

Figure 6.7: Test scenes containing planar curves

The matching curve features are derived from a regular cone and a planar circle in the Lamp scene, and regular cylinder(s) in the CylinderBox and Tubes model scenes. The range of matching features are expressed in Table 6.1.

Model	Planar faces	True edges	Curve edges	Axis edges	Total edge features
CylinderBox	7	12	2	1	15
Lamp	13	24	3	1	28
Tubes	10	0	10	5	15

Table 6.1: Matching features of test models

The models are defined in a right-handed coordinate system with the CylinderBox placed in the -x, +y, -z quadrant, and the Lamp and Tubes models centred along the +y axis. They are defined using (where available) true, axis, and curve edges. The data is non-ideal, recovered using the TINA system. The planar curve normals were found to be recovered with a maximum half cone error angle of a reasonable degree (±15°) when we realise that the normals are derived from an estimation of the planar surface. Tables 6.2 and 6.3 illustrate the data features recovered for each scene and the sensing errors used in the matching process.

Model	Model edge features (e)	Data edge features				No. of data pairs	Potential interpretations (e^s)
		True	Curve	Axis	Total (s)		
CylinderBox v1	15	9	2	1	12	66	1.30×10^{14}
CylinderBox v2		11	1	0	12	66	1.30×10^{14}
Lamp v1	28	16	3	1	20	190	8.77×10^{28}
Lamp v2		15	2	1	18	153	1.12×10^{26}
Tubes	15	0	8	3	11	55	8.65×10^{12}

Table 6.2: Complexity of scenes in terms of data features

Model	Errpar	Errper	Errsphere	-edges- Max. half cone error (°)	-curves- Max. half cone error (°)
CylinderBox v1	3	3	3	10	10
CylinderBox v2	3	3	3	10	10
Lamp v1	3	3	3	10	10
Lamp v2	3	3	3	10	10
Tubes	5	5	5	15	15

Table 6.3: Match sensing errors

6.8.1 Matching With Pseudo-Edge Features

In recovery of the data features, we are able to obtain 'edges' representing the extremal boundaries of the curved surfaces. However, the stereoscopic 3D reconstruction of these 'edges' cannot be relied upon as an accurate means of recovering such non-creased edges. The ability to include this information in the recognition process will be very useful in the early determination of data to model interpretations. Figure 6.8 illustrates the 'edge' information recovered from the Tubes data scene. The matching features are shown in Figure 6.9 along with those recovered from both views of the CylinderBox and Lamp.

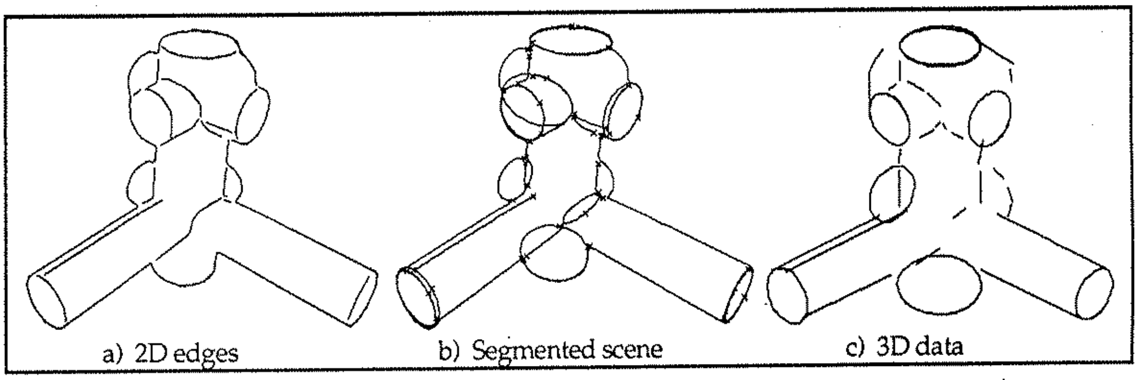
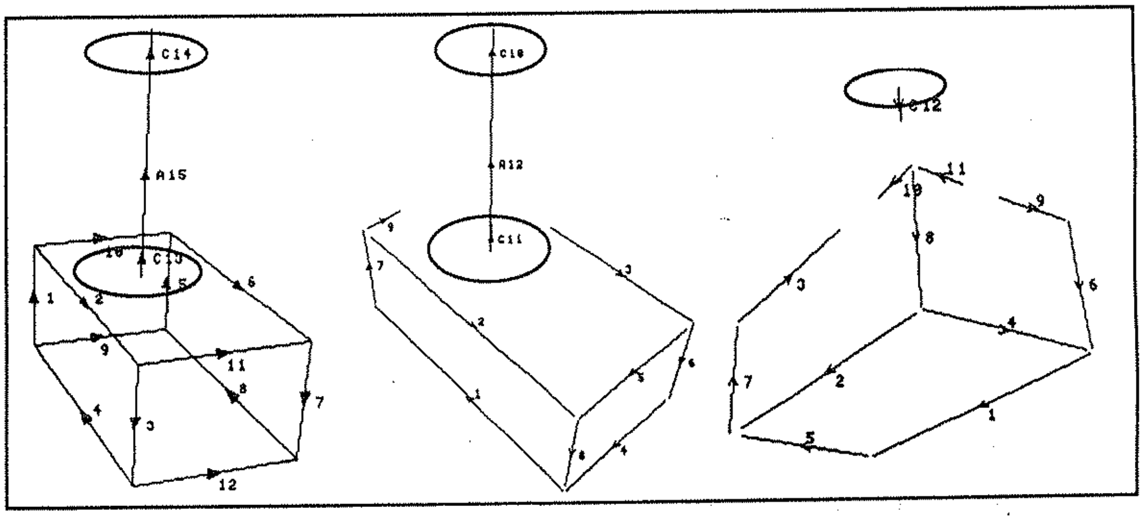
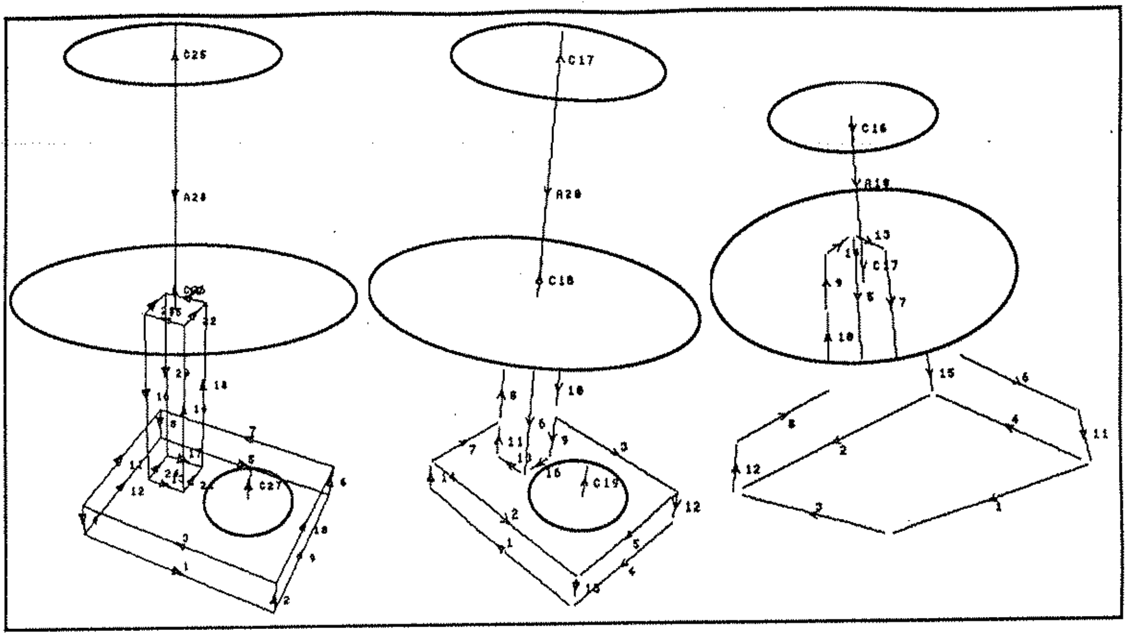


Figure 6.8: Potential edge information available from cylinder primitives



a) CylinderBox: Model - View 1 - View 2


b) Lamp: Model - View 1 - View 2

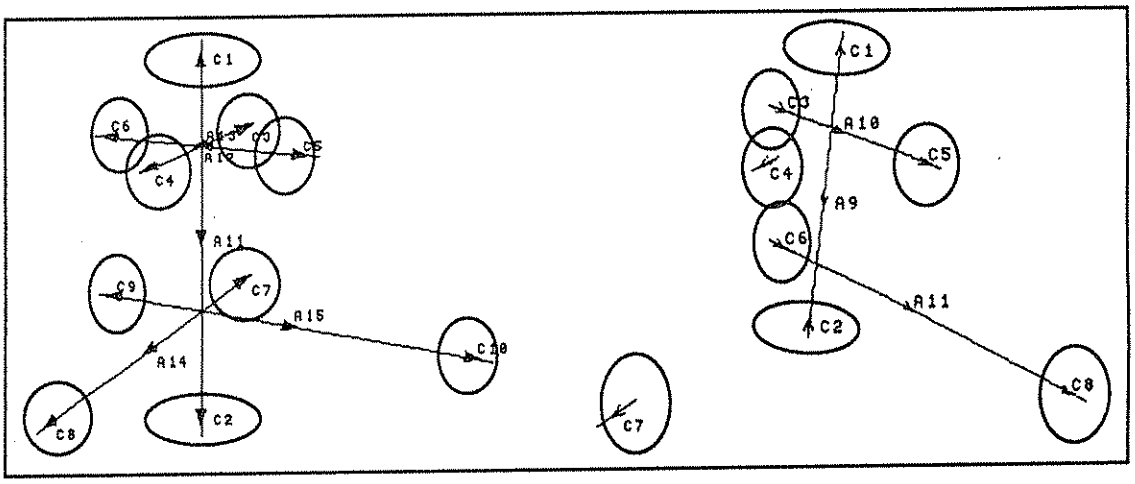

c) Tubes: Model - View

Figure 6.9: Recovered edge features for test scenes

The five pruning constraints described earlier were applied during processing of the CylinderBox and Lamp scenes. However, for the Tubes scene, the direction constraint 3 was switched off as it caused the failure of feasible interpretation recovery. This was thought to be due to the intersection of the model axes. Data pairs were ordered using the Combined sort. In all tests, the parallel matcher, with planar curve extension, was able to recover the correct interpretation of the data scenes along with the edge directions. However, for the Tubes scene, an extra interpretation was returned. This is shown overleaf.

Correct Interpretation:

Data	1	2	3	4	5	6	7	8	9	10	11
Model	1	2	6	4	5	9	8	10	11	12	15
Sign	+	-	-	+	+	-	+	+	+	+	+

Failing Interpretation:

Data	1	2	3	4	5	6	7	8	9	10	11
Model	1	2	3	5	4	7	10	8	11	12	14
Sign	+	-	-	+	+	-	+	+	+	+	+

Figure 6.10 below shows the matching times for the various views of the test scenes.

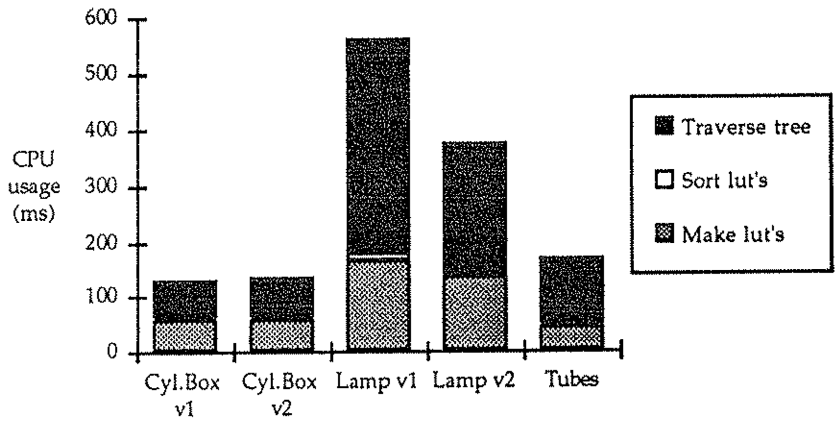


Figure 6.10: Matching times for test scenes

As expected, the matching times are comparable with those obtained for true edges alone in the previous chapter. This is impressive when we realise that the planar curve extension has not resulted in a total matching time penalty.

6.8.2 Validation of Interpretations

The recovered interpretations obtained in the previous section were correctly validated for global consistency. However, for the Tubes scene both interpretations failed to terminate due to a Gauss-Jordan arithmetic failure. The correct interpretation (number 2), however, returned a more accurate estimation of the rotation.

Table 6.4 illustrates the veridical and computed transformations from model to sensor spaces. Values for *fracdev* of 0.01, 0.015, 0.01, and 0.02 were chosen for the CylinderBox view 1, CylinderBox view 2, Lamp view 1, and Lamp view 2, respectively. *Curvedev* values of 0.02, 0.04, 0.01, and 0.05 were also chosen.

Veridical Transformations			Computed Transformations		
Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Trans. (x, y, z) Scale (S)	Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Trans. (x, y, z) Scale (S)
Cylinder Box v1	YROT(40) XROT(-40) $\begin{pmatrix} 0.7600 & -0.4132 & -0.4924 \\ 0.0000 & 0.7660 & -0.6428 \\ 0.6428 & 0.4924 & 0.5868 \end{pmatrix}$	(30.0, -20.0, 180.0) S = 1.0000	Cylinder Box v1	$\begin{pmatrix} 0.7637 & -0.4154 & -0.4942 \\ -0.007 & 0.7650 & -0.6441 \\ 0.6456 & 0.4922 & 0.5839 \end{pmatrix}$	(30.30, -20.01, 181.17) S = 1.0032
Cylinder Box v2	YROT(-150) XROT(30) $\begin{pmatrix} -0.8660 & -0.2500 & 0.4330 \\ 0.0000 & 0.8660 & 0.5000 \\ -0.5000 & 0.4330 & -0.7500 \end{pmatrix}$	(-50.0, -20.0, 150.0) S = 1.0000	Cylinder Box v2	$\begin{pmatrix} -0.8624 & -0.2502 & 0.4401 \\ 0.0055 & 0.8646 & 0.5024 \\ -0.5063 & 0.4357 & -0.7442 \end{pmatrix}$	(-50.44, -19.86, 151.87) S = 1.0004
Lamp v1	YROT(40) XROT(-30) $\begin{pmatrix} 0.7660 & -0.3214 & -0.5567 \\ 0.0000 & 0.8660 & -0.5000 \\ 0.6428 & 0.3830 & 0.6634 \end{pmatrix}$	(-30.0, -50.0, 200.0) S = 1.0000	Lamp v1	$\begin{pmatrix} 0.7732 & -0.3208 & -0.5470 \\ 0.0058 & 0.8662 & -0.4998 \\ 0.6341 & 0.3832 & 0.6716 \end{pmatrix}$	(-30.46, -50.01, 201.49) S = 1.0024
Lamp v2	YROT(40) XROT(40) $\begin{pmatrix} 0.7660 & 0.4132 & -0.4924 \\ 0.0000 & 0.7660 & 0.6428 \\ 0.6428 & -0.4924 & 0.5868 \end{pmatrix}$	(-10.0, -50.0, 130.0) S = 1.0000	Lamp v2	$\begin{pmatrix} 0.7676 & 0.4071 & -0.4951 \\ -0.0012 & 0.7733 & 0.6340 \\ 0.6410 & -0.4861 & 0.5940 \end{pmatrix}$	(-9.95, -50.16, 131.76) S = 1.0046
Tubes	YROT(45) XROT(-30) $\begin{pmatrix} 0.7071 & -0.3536 & -0.6124 \\ 0.0000 & 0.8660 & -0.5000 \\ 0.7071 & 0.3536 & 0.6124 \end{pmatrix}$	(-30.0, -50.0, 250.0) S = 1.0000	Tubes	Interpretation 1 $\begin{pmatrix} -0.7099 & -0.3529 & -0.6096 \\ -0.0648 & 0.8945 & -0.4424 \\ 0.7014 & -0.2746 & -0.6578 \end{pmatrix}$ Interpretation 2 $\begin{pmatrix} 0.7068 & -0.3343 & -0.6234 \\ -0.0161 & 0.8734 & -0.4867 \\ 0.7072 & 0.3540 & 0.6120 \end{pmatrix}$	Gauss-Jordan Fail S = G-J Fail

Table 6.4: Veridical and computed transformations

The results obtained for the computed transformations are very accurate considering that 1) the data was non-ideal, and 2) we are using data which is derived, i.e. the surface normals and axes. With the mean translation error of (±0.31, ±0.31, ±1.57) for x, y, z, this shows that matching using the pseudo-edge features maintains the robustness observed when using true edges alone. Note, also, that the error along the z-axis (the most error prone axis using 2D image data) is surprisingly low.

In order to appreciate the accuracy of the method, however, we must consider the Tubes scene containing only pseudo-edges. Here, the translational result is not available but, considering the rotation (from which it is derived) and interpretation 2 being the correct one, we can see that the rotational values for this comply closely to the veridical rotation. A mean rotational half cone error angle of 0.5° is observed for the CylinderBox and Lamp scenes, and 1.1° for the Tubes. The failing interpretation of the Tubes scene was found to have a half cone error angle of 70.9°.

6.9 Real World Determination

The robustness of any system can only be determined by application to a large number of testing conditions. Initially, however, these conditions are ideal in order to determine the effectiveness of the algorithm or system pathway. It is therefore helpful to be able to obtain a large amount of test data, quickly and efficiently. We have described a solid modeller which is able to render stereo image pairs using CSG combination operators applied to a number of geometric primitives to realise polygonal and planar curved objects. This is implemented in parallel for added efficiency in terms of speed. We have, subsequently, used these images as input to a parallel model matcher with the successful identification of the objects contained. The images, however, are ideal in that they lack noise and, therefore, result in much more accurate scene segmentation and 3D primitive recovery than can be the case for 'real' images. In testing the accuracy and robustness of the matching process, we cannot rely solely on such closed-form tests. They serve, merely, as an indication that the approach holds under synthetic conditions. We must demonstrate the effectiveness of the process under 'real world' conditions in order to explore its robustness.

6.9.1 Experiments in Real World

Hitherto, we have shown that the parallel implementation of the Grimson and Lozano-Pérez matching paradigm is 'robust' when applied to synthetic data. In order to determine the accuracy of the system to a greater degree, we must, inevitably, apply it to the real world. Here, we present results of experiments conducted using a real world scene containing a number of instances of the same object - 3 *widgets* obtained from Sheffield University's AIVRU laboratory. The scene contains planar curves and exhibits shadowing and regions of darkness equivalent to occlusions. Again, all data recovery processes are performed using the TINA toolset. The stereopair of the widgets is shown in Figure 6.11. Detailed tests, however, could only be performed on the left and right widgets as the transformation for the middle widget was unavailable. We shall therefore refer to the left and right widgets as *widget1* and *widget2* respectively.

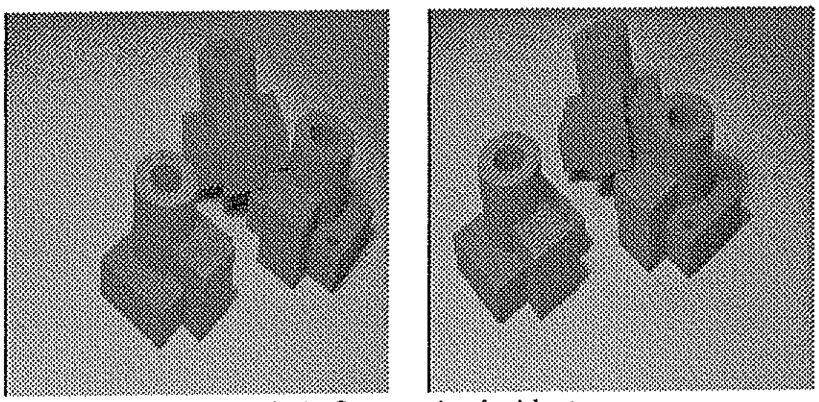


Figure 6.11: Stereopair of widget scene

The model is defined in a right handed coordinate system and is placed in the +x, +y, -z quadrant. It is composed of 32 matching features expressed as 28 true edges, 3 curve edges, and 1 axis edge.

The presence of spurious data edges in the scene can lead to non-interpretations or interpretations being delivered after an extended period of time when implementing the bin (see Section 5.13.6). We, therefore, select edges belonging to each widget to avoid the inclusion of spurious features. This, however, does not mean that we are side-stepping the real world tests. Indeed, the selected data features are not altered in any way, the process simply ensures that matching will terminate much faster. By including all recovered data features the matcher should *eventually* terminate with a set of interpretations for each widget and the non-matching edge features paired to the bin! Tables 6.5 and 6.6 show the edge features recovered for the two test widgets and the sensing errors used in the matching process.

Model	Data edge features				No. of data pairs	Potential interpretations
	True	Curve	Axis	Total		
Widget1	15	3	1	19	171	2.22×10^{22}
Widget2	15	2	0	17	136	9.85×10^{19}

Table 6.5: Complexity of widget scenes in terms of features

Model	Errpar	Errper	Errsphere	-edges- Max. half cone error (°)	-curves- Max. half cone error (°)
Widget1	5	5	5	8	12
Widget2	6	6	4	8	5

Table 6.6: Widget match sensing errors

From Table 6.6 we can see that the data features have been recovered with some degree of accuracy when compared to the sensing errors used for synthetic data in Tables 5.8 and 6.3. Figure 6.12, below, illustrates the model features along with the selected data. It can be seen that the selected data features are reasonably accurate in representing the structure of the widgets.

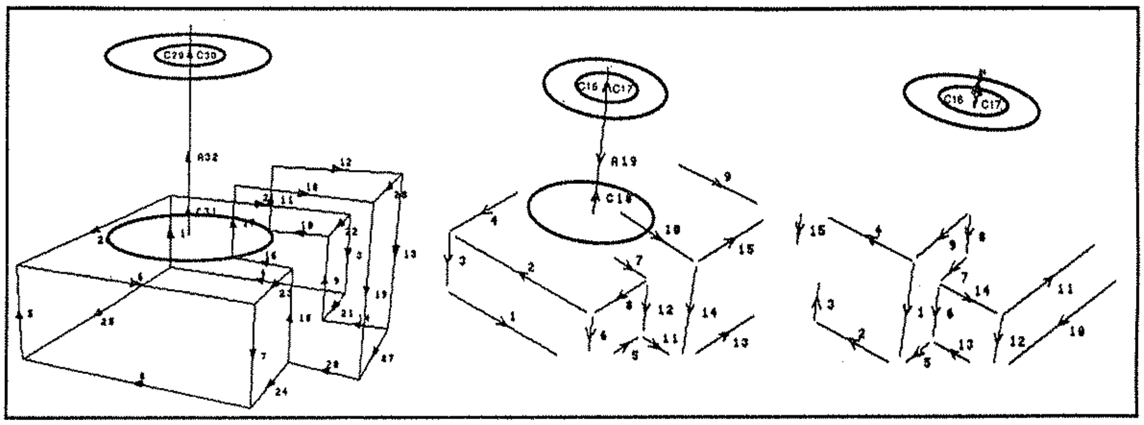


Figure 6.12: Model - Widget1 - Widget2 features

6.9.2 The Matching Results

The five pairwise pruning constraints were applied for both widgets and the Combined sorting technique was utilised. In both cases, the correct interpretation was returned along with three additional hypotheses. The extra interpretations were a result of the permutation of the data curve edges matched to the planar curve at the top end of the widget cylinder. Here, two curve edges coincide as their centres and surface normals coincide. We can see from Figure 6.13 that the matching times are comparable to tests conducted using synthetic data. As it would be expected that synthetic data would produce less geometric matches in the match look-up tables, this also shows the accuracy of the data recovery process.

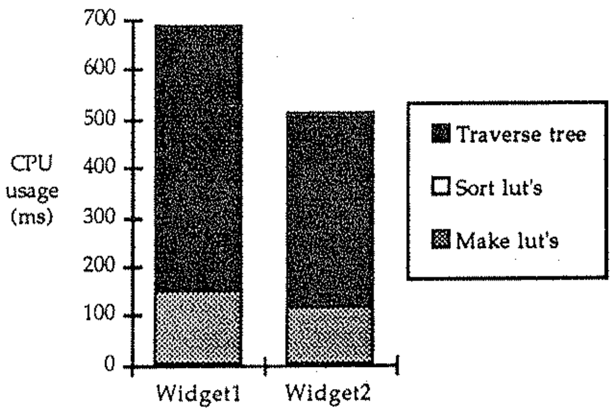


Figure 6.13: Match times for widgets

6.9.3 Validation to Determine Pose

The correct interpretations returned for the two widgets were successfully validated for global consistency. Although the additional interpretations all delivered the correct transformation, they were failed on the basis that after backprojecting, the data curves were unable to fit to within a threshold degree to their corresponding model curves.

Veridical Transformations			Computed Transformations		
Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Trans. (x, y, z) Scale (S)	Scene	Rotation Matrix $\begin{pmatrix} r11 & r12 & r13 \\ r21 & r22 & r23 \\ r31 & r32 & r33 \end{pmatrix}$	Trans. (x, y, z) Scale (S)
Widget1	$\begin{pmatrix} 0.7085 & 0.4570 & -0.5379 \\ -0.0162 & 0.7724 & 0.6349 \\ 0.7056 & -0.4411 & 0.5546 \end{pmatrix}$	(-13.49, 35.05, 489.81) S = 1.0000	Widget1	$\begin{pmatrix} 0.7246 & 0.4350 & -0.5346 \\ -0.0295 & 0.7945 & 0.6066 \\ 0.6886 & -0.4238 & 0.5885 \end{pmatrix}$	(-14.06, 35.44, 487.79) S = 0.9828
Widget2	$\begin{pmatrix} -0.6651 & 0.4117 & -0.6230 \\ -0.0621 & 0.8009 & 0.5956 \\ 0.7442 & 0.4348 & -0.5071 \end{pmatrix}$	(102.59, -6.68, 553.63) S = 1.0000	Widget2	$\begin{pmatrix} -0.6653 & 0.4258 & -0.6132 \\ -0.0492 & 0.7946 & 0.6052 \\ 0.7449 & 0.4328 & -0.5077 \end{pmatrix}$	(104.03, -7.40, 553.13) S = 0.9940

Table 6.7: True and computed widget transformations

Table 6.7 demonstrates the transformations obtained for the widgets when values for *fracdev* of 0.03 and *curvedev* of 0.1 were chosen for widget1, and 0.17 for *fracdev* and 0.09 for *curvedev* were chosen for widget2. The translational error may be viewed as (±0.57, ±0.39, ±2.01) and (±1.45, ±0.72, ±0.50) along x, y, and z for widget1 and widget2 respectively. The rotational errors, however, are very small considering that the data is **not** synthetic, with a mean rotational half cone error angle of 0.6° for widget1 and 0.9° for widget2.

6.10 Conclusion

It has been shown that the surface normal of the circular planar ends of regular cylinders and cones can be accurately recovered using stereopsis. By treating these normals as edges along with the axis of rotation of the primitives, we were able to include them in an edge-based parallel matcher using the Grimson and Lozano-Pérez matching algorithm. If the direction of the outward planar curve surface normal can be recovered uniquely, it may be used during the tree traversal phase to determine the edge direction signs more effectively.

From the experiments conducted, the pseudo-edges have been shown to be robust matching features and have been used to accurately determine the model to scene space transformations. However, in the experimental tests, we have been limited by the data recovery technique to primitives containing circular planar ends. The principle is easily applicable to elliptical sliced ends. During validation, however, we must ensure that both major and minor data axes are closely fitting to the model elliptical axes. Further extensions may be made to include non-developable surfaces such as in generalised cylinders, but again we are constrained by the sensing and segmentation approaches. The generalised cylinder of Figure 6.14 may be included in the range of primitives if we can accurately segment its image scene such that a 'characteristic' pseudo-axis of rotation may be obtained. If this is not possible, however, we may still include the primitive by merely using the surface normal of the planar ends.

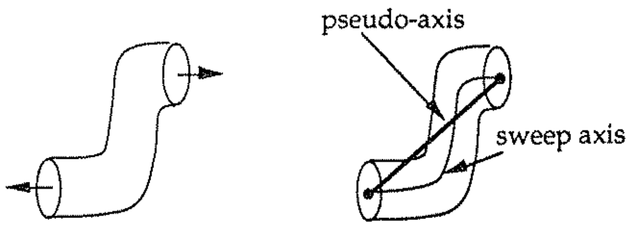


Figure 6.14: Extension to non-developable surface primitive

Practical applications of the planar curve extension are varied. An immediate example is for use in a robot's visual system mounted alongside a conveyer belt where industrial components, often possessing regular planar curves, can be found. Objects possessing more creased edges than planar curves are also suited to the described method. They may be defined with the curve features as the main match feature which may be determined first during the matching process. The match list may be sorted such that curve look-up tables are encountered first. However, to maintain efficiency of tree traversal, consistency must be retained between the data pairs (see Section 5.7.1).

Although the extension we have presented is simple, it has been shown to be attractive for the recognition of planar curves in terms of matching time. It also retains the robustness of the polyhedral matcher with very accurate computed model to sensor space transformations. Obviously, matching termination will be much faster if information pertaining to the developable surface could be included directly into the matching process. Nevertheless, the system is fast in the determination of scenes containing polyhedral and curved planar surfaces and, thus, has potential in real-time applications.

Before such applications, however, it is important to realise the accuracy of the technique under real world conditions since this will be the eventual operating environment. In the experiments conducted to realise real world applicability, we have attempted to show that the matching process holds and that transformation parameters may be retrieved with reasonable accuracy. From the results obtained, we can conclude that the parallel implemented matcher, can not only cope with data generated synthetically, but also with matching features obtained from the real world. Its speed of execution is preserved using the two data sources (synthetic and real) and, therefore, leads to the algorithm being extremely capable for real world tasks given presegmented data.

Chapter 7

Contributions and Conclusions

7.1 Introduction

During the past three decades powerful methods for image analysis have emerged owing to supercomputer production on a cheaper and more powerful scale. Recent advances in VLSI technology have made it possible for the design and construction of massively parallel machines. Such massively parallel fine-grain architectures have included the DAP and Connection Machine. These may utilise several thousand processors. These processors, however, are simple, making it essential to combine them on a large scale to perform complex operations efficiently in terms of execution time. Another type of architectural parallelism has also emerged but, unlike the fine-grain SIMD architectures, uses more powerful processors in a coarse-grain independent manner. These MIMD architectures include the BBN Butterfly and Intel Cube which have individual processors executing their own instructions in local memory. Advantages and disadvantages exist in both architectural design. Tasks exhibiting a regular structure of data are usually more suited to the SIMD architecture and those which are more control-dependent are found to be suited to the MIMD design.

A problem with many image and computer vision tasks is that operations must be carried out on a large amount of data items. Consider a 1024 × 1024 pixel image. If a set of computations is performed on each pixel in unit time then over one million units are required to process the whole image. Such image data, however, being regular, has inherent parallelism which can be exploited with the right choice of parallel architecture. Given the SIMD DAP with 1024 processing elements, the image data may be mapped onto the processor array in blocks of 32 × 32 pixels or as a 1024 pixel raster-line to achieve execution over the whole image in a fraction of the time. The introduction of parallelism and, thus, the reduction in processing time, is desirable in many tasks especially those requiring interactive or real-time operations. It is primarily for this reason why we chose to investigate the parallel nature of computer recognition operations which are, essentially, destined to operate under real-world, real-time conditions.

In the following concluding sections, we present the contributions offered by this project as demonstrated in the preceding chapters along with a conclusion of our results. This is essentially the realisation that uniform parallel processing offers efficient traversal of the interpretation tree and reduction of the search space during Grimson and Lozano-Pérez style matching. In Section 7.2 we discuss the advantages and disadvantages of the SIMD architecture and outline the power of parallelism in relation to the work described. This is followed in Section 7.3 by the contributions offered in both the field of graphics and vision. Section 7.5 concludes our work preceding which we suggest future directions and enhancements in Section 7.4.

7.2 Why Use SIMD Parallelism?

The question as to whether to use an SIMD or MIMD architecture in the parallel solution of a serial problem is largely dependent on the type of problem. The SIMD method of parallel processing can be advantageous over MIMD in a number of cases. For example, a single control unit is required to drive the processors which means that there is a minimal instruction decoder cost compared to MIMD where a decoder is present in each PE. Also, the MIMD paradigm must support expensive irregular non-local processor communications. However, MIMD architectures are, in general, more flexible in that there are no constraints on operations that can be performed concurrently. In addition, conditional statements are more efficient. For example, in the MIMD paradigm each processing element executes as if it were a uniprocessor whereas this must be serialized on an SIMD machine. The processor execution time is also more favourable on an MIMD architecture since after execution of a block of instructions a processor may move onto the next block. From Figure 7.1 the difference in variable-time instructions is shown between SIMD and MIMD. In SIMD, processors must wait until the slowest processor has executed the instruction before moving onto the next set of instructions. We note, however, that this difference in variable-time instructions between SIMD and MIMD is such only when the data is independent.

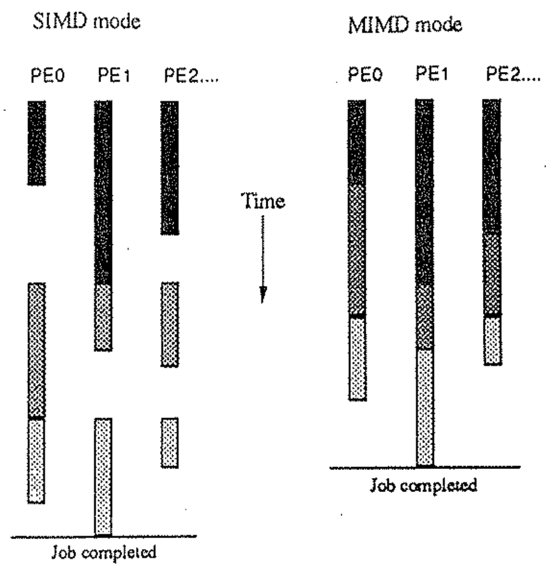


Figure 7.1: Time required to complete a job in SIMD and MIMD processing modes'

After consideration of these points however, an SIMD architecture was used in the work described. The main reason for this was to investigate the applicability of graphics and vision problems on such a machine architecture. Also, since the problems undertaken contained regularly structured data, the solution could be mapped onto the processors in a very efficient manner. In addition, the inherent SIMD parallelism in such data made the solution efficient in terms of program execution time. The advantages of implementing such regular problems in an SIMD manner therefore seems to outweigh the disadvantage of processors being dependent on the slowest PE.

In the work conducted on parallel ray tracing, the power of the SIMD processing is most noticeable for a single-level deep search. This is partly due to the inefficiency of recursion in the DAP along with the difficulty in performing an independent tree-search on an SIMD architecture. Again, the bottleneck in the reformulation of the Grimson and Lozano-Pérez serial algorithm to map onto the DAP processors can be seen during the serial tree search. The effectiveness of the algorithm therefore comes about from the parallel computation of matches of a data pair to *all* possible model pairs. This determination is generated in an optimal manner since the procedure exhibits regularly structured data which can be operated on efficiently on the SIMD architecture. We note that a certain degree of parallelism is still possible during the serial-search in maintaining consistency of an assignment of a data item with assignments made higher up the interpretation tree.

Although the Holder and Buxton algorithm is effective in its implementation on the DAP, it would still be desirable to further reduce the total matching time of scenes by removing the bottleneck of the serial-search. Implementation onto hybrid architectures such as that described in Section 3.4 may be considered. The SIMD portion of the hybrid can be used to generate the match look-up tables whilst the MIMD section is responsible for traversing the heavily pruned interpretation tree in a more efficient parallel manner.

7.3 Contributions

The principal direction of this project was in the area of parallel processing for the efficient determination of models from non-sensor dependent data. In preparation for the investigation, we required an effective means of data generation for robust testing of derived processes. Since many vision systems determine matches using camera images, we implemented such a system for the production of synthetic camera images for closed-form testing in the investigation of parallel matching techniques. The system is based on the SIMD parallel methodology of Williams *et al.* [WBB87] and is able to generate realistic image sequences or stereopairs. This, in effect, means that it may also be used by other systems as a fast tool for the generation of synthetic data. The current implementation differs from that of Williams *et al.* in that it includes enhancements such as anti-aliasing and various surface and lighting properties such as transparency and reflection, together with specular, diffuse, and an approximation of ambient lighting. These contribute to more 'natural' image properties which can be used in detailed recognition process tests. However, in the experiments described thus far, anti-aliasing of images was not necessary for accurate 3D feature recovery.

We have also described an implementation of a 2D spatial subdivision algorithm. This was shown to be extremely effective in reducing the total CPU expenditure time for scene rendering. A mean speed up of 18 times over the more naive, brute force, approach was reported for the test scenes used. An even greater speed up is observed when the parallel implementation was compared to an equivalent serial implementation thus making the system an effective first tool in the production of model matching strategies. We did not, however, feel it necessary to extend the 2D Tiling algorithm to 3D as in doing so we digress from the major aims of the project. Realistic images illustrating properties of reflection and transparency can still be produced using the brute force approach.

We have demonstrated from a number of experiments that the matching paradigm of Grimson and Lozano-Pérez can be mapped with reasonable efficiency onto the processors of the DAP. In doing so, we have shown that the parallelism in the algorithm may be exploited on an SIMD architecture. The interpretation tree data structure is reformulated to map onto the SIMD processors. We have not, however, undertaken any investigations to determine the most efficient transposition of the algorithm onto an MIMD architecture. In reformulating the interpretation tree to map onto the DAP processors we have adopted the approach described by Holder and Buxton [HB89*a*]. Using this, large models, too big to map directly onto the processors, may be subdivided by a novel subgraph method. We have also shown, more importantly, the significance of ordering the data pairs and have derived a new sorting technique. This is an intelligent sort which maintains consistency between the data pairs in combination with ordering by the least number of geometric matches. Tests of this Combined sort technique have proved that it is always superior. Speed ups of 2-4 times over the less consistent H&B sort have been reported. This, however, was found to increase with increasing complexity of the scene.

In anticipation of a real world application, we have investigated the null pairing hypothesis by an implementation of the bin. This is able to cope with spurious data features in the scene. Obviously, the addition of an extra branch at each level of the interpretation tree results in a much larger search space. We, therefore, attempt to recover the most probabilistic hypothesis by initially selecting interpretations with no bin matches. The importance of match list ordering reveals itself here. In all experiments to include spurious data edges, the H&B sort failed to terminated after 2000 seconds.

In confirming the validity of a hypothesized interpretation we have investigated the task of extending the polyhedral matching algorithm to include curves, namely, in a planar fashion. This has been achieved by treating the planar curve surface normal, taken at the centre of cylindrical ends, along with the axis of rotation, and treating them as physical edges. Inclusion into the parallel edge matcher has then been possible with little modification of the approach, i.e. face and edge based look-up tables have been amalgamated into a single true/pseudo edge look-up table. The results have shown a surprising accuracy and maintains the robustness observed when using true edges alone.

Thus, in summary, we have produced a set of tools which have exploited the parallelism inherent in many model-based vision tasks to greatly reduce the time required to solve them. On their own, the tools may be included in a system requiring that a particular job be performed more efficiently. Together, however, they compose a powerful vision system which is able to exploit full SIMD parallelism. More importantly, the system can incorporate recognition of objects possessing planar curves without resulting in any additional matching time penalty.

7.4 Future Work

As a discipline, Computer Vision is still in its early days, a far cry from the visual systems created by science fiction writers and movie producers. In this research, we have only managed to explore the tip of a machine vision iceberg. A number of research avenues into the model-based recognition of generalised objects are still open for investigation.

It appears that the major hindering factor in application to the real world is in ignoring junk and clutter in the scene, i.e. being able to segment individual objects and concentrating the recognition process on these. The formulation of the problem using uncertified scene elements attempts to resolve the problem of spurious data items by systematically associating non-matching items to a 'bin'. This, however, is slow as we immediately get an extra branch at each level of the interpretation tree. Investigations have been reported on the combinatorics of the search space using constrained search [Grim88, Grim90]. If all the data is known to come from a particular object, the search effort is quadratic and is of order

$$SE(m^2 + ams)$$

where m and s are the number of model and sensed features respectively, and a is a small constant. However, if spurious data is present, the search becomes exponential and is bounded above with search effort of order

$$SE(ms2^c + m^2s^2[1 + \alpha]^c + bm^6 + m[1 + \gamma]^s),$$

and below with order

$$SE(m2^c + ms),$$

where m is the number of model features, s is the number of sensed features, c is the number of sensed features correct in the model, and $\alpha, \gamma \leq 1$ are small constants. The introduction of parallelism in the implementation of the bin will greatly increase the ability for recognition to be performed on more realistic data. In our work, the exhaustive serial search of the heavily pruned interpretation tree is the limiting factor in the parallel implementation. A possibility may be to try to eliminate any failing items by testing all the matches of the data pairs in parallel during a preprocessing phase. Data items continuously failing the pairwise constraint tests may be considered to not belong to the model and, therefore, be excluded from the tree traversal phase. Consider Figure 7.2.

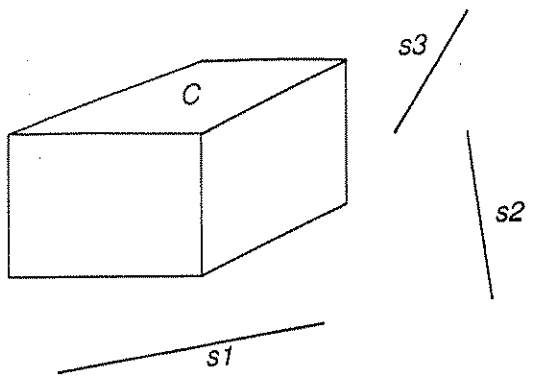


Figure 7.2: True and spurious scene data

Given that the data items *s1*, *s2*, and *s3* are spurious to the model, it is likely that they will fail more of the constraint tests than the data items of *C*. This, however, is more reliable when the sensed features corresponding to the model are greater than the spurious items. Therefore, in order to outweigh the overhead of the preprocessing and for greater efficiency, this is envisaged for larger models.

Using data obtained from motion, we realise that it is possible to segment such scenes to obtain features belonging to individual objects [MW85]. A possibility then exists for the use of motion algorithms to recover locally segmented image features. Each group of features may then be matched to the model separately.

To further increase the scope of the matcher, curvature information may be incorporated to extend to the model class of generalized cylinders. This can then include primitives possessing more general cross-sectional planar ends. In dealing with such classes of curved objects, it will be very helpful to be able to pull out the information contained in the extremal boundaries. Again, we are limited by the segmentation technique which must be required to recover accurate curvature information. This, we know is possible using range finding techniques or trinocular stereo.

These extensions, however, do not make any greater use of the parallelism. As we suggested earlier, the hindering factor in the Grimson and Lozano-Pérez model-based approach is the combinatorics experienced when additional model and data features are included. We feel, therefore, that in order to produce a significant improvement to matching systems, this is the area of investigation on which to concentrate and, is therefore, the direction which much of our future work will follow.

7.5 Conclusions

We have investigated parallel techniques in visualisation and model-based recognition. The main thrust of the work was towards real-time model-based vision up to the object recognition stage using massive data parallelism.

We have produced a parallel visualisation tool for the generation of camera image data. Output from this has been used in the testing of an extension of the SIMD parallel algorithm of Holder and Buxton. The object recognition scope has been extended to deal with objects containing developable surfaces by incorporating the surface normal and axis of cone and cylinder primitives in CSG models into the matching scheme. There is no time penalty for this extension. In addition, an improved sorting for finding consistent interpretations has been presented which results in matching times of up to 1000 times faster over the earlier sorting technique to give total matching times of 0.12-0.68 seconds for the example scenes considered. The geometrical transformations recovered in the validation phase are also found to be extremely accurate in the closed form tests which promises that the technique should find practical application.

In combining the parallel visualisation tool with the extension of the parallel model matcher, we conclude a complete system which can be considered as state of the art. We note that the choice of an SIMD architecture has meant that tree search routines cannot be solved effectively in parallel. However, implementation may be possible on a hybrid SIMD/MIMD architecture which will be able to exploit both data and task parallelism. We also note that although the methods discussed and implemented have been specific to the DAP, the solutions are general to SIMD arrays and may be ported across such architectures.

Appendix A

Ray Tracing Code for the DAP

The ray tracing and anti-aliasing procedures described in Chapter 4 are illustrated below for implementation on the DAP. Note that this is a skeletal sample of the Fortran* code.

```
C *** main ray tracing loop
  do 1 tile_y = 1, 16, 1
    do 2 tile_x = 1, 16, 1

C  *** anti-aliasing loop ... x9
      do 3 pixel_tile_y = 1, 3, 1
        do 4 pixel_tile_x = 1, 3, 1

C *** get directions of ray from eye and normalize
        ray_x = GET_RAY_X
        ray_y = GET_RAY_Y
        ray_z = GET_RAY_Z

        call NORMALIZE(ray_x, ray_y, ray_z)

C *** invoke appropriate module for the data-type
        if (data_type.EQ.1)
           call TRACE(eyep_x, eyep_y, eyep_z, ray_x, ray_y, ray_z,
                      color_r, color_g, color_b, dist, object_bound_hit)
        if (data_type.EQ.2)
           call CSG_TRACE(eyep_x, eyep_y, eyep_z, ray_x, ray_y, ray_z,
                          color_r, color_g, color_b, dist, object_bound_hit)
        if (data_type.EQ.3)
           call DTM_TRACE(eyep_x, eyep_y, eyep_z, ray_x, ray_y, ray_z,
                          color_r, color_g, color_b, dist, object_bound_hit)


C *** If no objects hit set to background colour
        color_r(dist.LE.0) = float(background_r)
        color_g(dist.LE.0) = float(background_g)
        color_b(dist.LE.0) = float(background_b)
```

```
C *** jump out of anti_aliasing loop if not anti-aliasing
          if (.NOT.anti_alias) goto 111

          i = i + 1
           store_color_r = store_color_r + color_r * alias_val(i)
           store_color_g = store_color_g + color_g * alias_val(i)
           store_color_b = store_color_b + color_b * alias_val(i)

   4    continue
   3   continue

C *** get average intensity for each pixel
    color_r = store_color_r/16.0
    color_g = store_color_g/16.0
    color_b = store_color_b/16.0

111  continue

C *** limit rgb to 0-255
    color_r(color_r.GT.255.0)   = 255.0
    color_r(color_r.LT.0.0)     = 0.0
    color_g(color_g.GT.255.0)   = 255.0
    color_g(color_g.LT.0.0)     = 0.0
    color_b(color_b.GT.255.0)   = 255.0
    color_b(color_b.LT.0.0)     = 0.0

C ***
C ***  OUTPUT TO SCREEN
C ***

2  continue
1  continue
```

Appendix B

Ray Tracing Combination Procedure

The ray tracing combination procedure described in Chapter 4 is illustrated below for implementation on the DAP. Note that this is a skeletal sample of the Fortran* code.

```
C *** run along the lists and combine the
C *** objects according to the operators
  stack_pointer = 0

  do 100 i = 1, n, n_of_nodes

    if (operator_list(i).NE.'0') goto 10
    stack_pointer = stack_pointer + 1

C *** get the entry and exit points of the ray with the
C *** object and return the sides which are hit
    call GET_IN_OUT(source_x, source_y, source_z, ray_x, ray_y, ray_z,
                    object_list(i), sidein, sideout, in, out)

C *** store the entry and exit values
    entry(,, 1, stack_pointer) = in
    entry(,, 2, stack_pointer) = 0
    exit(,, 1, stack_pointer)  = out
    exit(,, 2, stack_pointer)  = 0

C *** store object hit and side hit as one
    in_object(,, 1, stack_pointer) = object_list(i) * 100 + sidein
    out_object(,, 1, stack_pointer) = object_list(i) * 100 + sideout

    goto 100

10   continue
     left_pointer = stack_pointer - 1

     if (operator_list(i).NE.'U') goto 20

C *** combine the two lists according to UNION operator

     call COMBINE(entry, exit, in_object, out_object, left_pointer, stack_pointer, 'U')

     stack_pointer = left_pointer

    goto 100

20   continue
```

```
C *** combine the two lists according to INTERSECT operator
if (operator_list(i).NE.'I') goto 30
      call COMBINE(entry, exit, in_object, out_object, left_pointer, stack_pointer, 'I')
      stack_pointer = left_pointer
      goto 100
30    continue
      if (operator_list(i).NE.'D') goto 100
C *** combine the two lists according to DIFFERENCE operator
      call COMBINE(entry, exit, in_object, out_object, left_pointer, stack_pointer, 'D')
      stack_pointer = left_pointer
100   continue
```

After traversing the tree, the t-value of the composite object at the root node will be: t-value = entry(, , 1, 1), and the first object encountered will be in_object(, , 1, 1).

Appendix C

DAP Implementation for Model Matching

The parallel algorithm described in Chapter 5 is illustrated here. This is a fragmentary Fortran* code of the matching paradigm.

```
C *** define MatchList to be
C *** (NumberOfModelFeatures x NumberOfModelFeatures) x
C *** NumberOfDatapairs    e.g. MatchList(32,32,496)

C *** determine MatchList by computing data pairs passing the constraints
      DO 10 i = 1, NumberOfDatapairs
         MatchList(,,i) = apply pruning constraints
10    CONTINUE

C *** sum the number of geometric matches for each data pair
      DO 20 i = 1, NumberOfDatapairs
        MatchSum(i) = SUM(MatchList(,,i))
20    CONTINUE

C ***
C *** sort the match list for most efficient traversal using MatchSum to
C *** produce TraverseOrder - see SORTING TECHNIQUES below .
C ***

C *** traverse the heavily prune interpretation tree
      level = 1
      count = 1
      CALL TraverseTree(level)
      count = count - 1

C *** count is the number of interpretations found



C ***
C *** define TraverseTree
C ***
       subroutine TraverseTree(level)

C *** declare variables

       i = TraverseOrder(level)
       dataedge1 = Datum1(i)
       dataedge2 = Datum2(i)
```

```
C *** Mask for match consistent with interpretation so far
        ConsistentMatch = (Match(,,i)
     +  .and.((edge1.EQ.edge(dataedge1,count)).OR.(edge(dataedge1,count).EQ.0))
     +  .and.((edge2.EQ.edge(dataedge2,count)).OR.(edge(dataedge2,count).EQ.0)))

C *** early backtrack ***
      IF (ELN(ConsistentMatch).EQ.0) GOTO 20


C *** determine if the data edges are already paired
      edge1_clear = (edge(dataedge1,count).EQ.0)
      edge2_clear = (edge(dataedge2,count).EQ.0)


C *** look for successive interpretations with regard to data pair i

30    k = ELN(ConsistentMatch)
      IF (k.EQ.0) GOTO 20
       ConsistentMatch(k) = .FALSE.

C ***
C *** do edge direction tests and store result
C ***

C *** Place object model edge labels in the interpretation array
      edge(dataedge1,count) = edge1(k)
      edge(dataedge2,count) = edge2(k)


C *** if an interpretation is complete, make a copy before
C *** backtracking to look for up to NOI interpretations

      IF (level.LT.NumberOfDatapairs) GOTO 40
      IF (count.GT.NOI) GOTO 20

C ***
C *** store interpretation
C ***

      goto 50

C *** move to next level in MatchList
40    CALL TraverseTree (level + 1)

50    continue

      IF (count.GT.NOI) GOTO 20

C *** prepare to scan for alternative interpretations
C *** reset interpretation array if edges previously unassigned

      IF (edge1_clear) edge(dataedge1, count) = 0
      IF (edge2_clear) edge(dataedge2, count) = 0

C ***
C *** reset direction signs
C ***

      goto 30


20    continue

C *** Backtrack to previous data pair
      return
      end
```

Sorting Techniques:

In the above code, the ordering of the match list may be performed in several ways. Below is a demonstration of the new Combined sort technique along with the Holder and Buxton ascending order sort.

```
C *** example of Holder and Buxton ascending order sorting
      DO 100 i = 1, NumberOfDatapairs
        position = ELN(MINP(MatchSum))
        TraverseOrder(i) = position
        MatchSum(position) = 9999
100   CONTINUE


C *** example of the Combined sort
      firstpair = ELN(MINP(MatchSum))
      firstpairgroup = Datum2(firstpair)
      TraverseOrder(1) = firstpair
      IF (firstpair.eq.1) THEN
        k = 2
      ELSE
        TraverseOrder(2) = 1
        k = 3
      ENDIF

      DO 200 i = 3, NumberOfDataItems
        ii = i
        grouppairs = Datum2.EQ.i
        IF (i.EQ.firstpairgroup) THEN
          ii = i-1
          grouppairs(firstpair) = .FALSE.
        ENDIF
        mingrouppairs = MERGE(MatchSum, 9999, grouppairs)
        position = ELN(MINP(mingrouppairs))
        TraverseOrder(k) = position
        grouppairs(position) = .FALSE.
        k = k + 1

      DO 200 j = 2, ii-1
        position = ELN(grouppairs)
        TraverseOrder(k) = position
        grouppairs(position) = .FALSE.
        k = k + 1
200   CONTINUE
```

Note that ELN is a DAP Fortran function which returns the location of the first 'true' value from a logical matrix or vector. MINP returns a logical matrix or vector (depending on its argument) with the positions of the minimum values of its argument marked 'true' and all other positions marked 'false'.

Appendix D

Published Papers and Technical Reports

The work described in this thesis has been used to form several technical reports and published papers. They are summarized here.

Report 1
Usoh, M., "A Generalised Ray Tracer Using SIMD Parallel Processing", *Technical Report no. 499*, Dept. of Computer Science, Queen Mary and Westfield College, Feb. 1990.

- This report outlines the work described in Chapter 4.

Report 2
Usoh, M., "Parallel Techniques in 3D Model Matching", *Technical Report no. 527*, Dept. of Computer Science, Queen Mary and Westfield College, Feb. 1991.

- This report outlines the work described in Chapter 5.

Report 3
Usoh, M., Buxton, H., "Geometric Matching with Curved Surfaces: Parallel Visual Evaluation and Display", *Technical Report no. 587*, Dept. of Computer Science, Queen Mary and Westfield College, March 1992.

- This report outlines the work described in Chapters 4 and 6 with real world based experiments. It has also been submitted to the Visual Computer as an extension and enhancement to Paper 1.

Paper 1

Usoh, M., Buxton, H., "Using Visualisation as a Tool for Model-Based Recognition", *The Visual Computer (special edition on parallelism)*, to appear, 1992.

Also as Technical Report no. 553, Dept. of Computer Science, Queen Mary and Westfield College, March 1991.

- This paper is a combination of the reports of 1 and 2 with enhancements and more detailed results. It describes a parallel visualisation system which, using SIMD processing, is able to efficiently generate image data for closed-form testing of an SIMD parallel edge-based model matcher.

Paper 2

Usoh, M., Buxton, H., "SIMD Geometric Matching: From Polyhedra to Planar Curves", *Proceedings of Vision Interface '92, Vancouver*, pp. 157-162, May 1992.

Also as Technical Report no. 554, Dept. of Computer Science, Queen Mary and Westfield College, Sep. 1991.

- This report describes the work performed in Chapter 6.

References

[AKMS85] Arnould, E., Kung, H.T., Mensilcioglu, O., Sarocky, K., "A Systolic Array Computer", *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, vol. 1, pp. 232-235, 1985.

[Aman84] Amanatides, J., "Ray Tracing with Cones", *Computer Graphics (Proc. SIGGRAPH84)*, 18(3), pp. 129-135, July 1984.

[AMT90] AMT Customer Newsletter, May 1990.

[Appe68] Appel, A., "Some Techniques for Shading Machine Renderings of Solids", *SJCC*, pp. 37-45, 1968.

[BA84] Brady, M., Asada, H., "Smoothed Local Symmetries and Their Implementation", *Int. J. of Robotics Research*, 3(3), Fall 1984.

[BB88] Blake, A., Brelstaff, G., "Geometry from Specularities", *Proc. 2nd. Int. Conf. on Computer Vision*, pp. 394-403, 1988.

[BB90] Blake, A., Bülthoff, H., "Does the Brain Know the Physics of Specular Reflection?", *Nature* 343, pp. 165-168, 1990.

[BC89] Blake, A., Cipolla, R., "Robust Estimation of Surface Curvature from Deformation of Apparent Contours", *Technical Report No. OUEL 1797/89*, University of Oxford, October 1989.

[BCZ90] Blake, A., Cipolla, R., Zisserman, A., "Towards Qualitative Vision: Motion Parallax", *Proc. of BMVC90*, pp. 115-120, Sep. 1990.

[Bied87] Biederman, I., "Recognition-by-Components: A Theory of Human Image Understanding", *Psychological Review*, 94(2), pp. 115-147, 1987.

[BJ85] Besl, P.J., Jain, R.C., "Three-Dimensional Object Recognition", *Computing Surveys*, 17(1), pp. 75-145, March 1985.

[BM90] Bourdon, O., Medioni, G., "Object Recognition Using Geometric Hashing on the Connection Machine", *Proc. Tenth Int. Conf. Patt. Recog.*, pp. 596-600, 1990.

[Bouv85] Bouville, C., "Bounding Ellipsoids for Ray-Fractal Intersection", *Computer Graphics (Proc. SIGGRAPH85)*, pp. 45-52, 1985.

[Bray90] Bray, A.J., "Object Recognition Using Local Geometric Constraints: A Robust Alternative to Tree-Search", *First Euro. Conf. on Comp. Vision*, pp. 499-515, April 1990.

[Broi79] Broit, M., "The Utilization of an 'Artificial Skin' Sensor for the Identification of Solid Objects", *Proc. Ninth Int. Symp. Industrial Robots*, pp. 529-548, 1979.

[Broo83] Brooks, R.A., "Model-Based Three-Dimensional Interpretations of Two-Dimensional Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, pp. 140-150, 1983.

[BS88] Brady, M., Scott, G., "Parallel Algorithms for Shape Representation", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 97-118, 1988.

[BWR90] Burns, J.B., Weiss, R., Riseman, E.M., "View Variation of a Point Set and Line Segment Features", *Proc. Image Understanding Workshop*, pp. 650-659, Sep. 1990.

[Cai89] Cai, L.D., "Spline Smoothing: A Special Case of Diffusion Smoothing", *Proc. fifth AVC Conf.*, Reading, pp. 273-276, July 1989.

[Cann86] Canny, J.F., "A Computational Approach to Edge Detection", *IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI-8(6)*, pp. 679-698, 1986.

[CB90] Cipolla, R., Blake, A., "The Dynamic Analysis of Apparent Contours", *Third Int. Conf. on Computer Vision*, Osaka, pp. 616-623, Dec. 1990.

[CCOD87] Cyganski, D., Cott, T.A., Orr, J.A., Dodson, R.J., "Development, Implementation, Testing and Application of an Affine Transform Invariant Curvature Function", *Proc. ICCV Conf.*, pp. 496-500, 1987.

[CD86] Chin, R.T., Dyer, C.R., "Model-Based Recognition in Robot Vision", *ACM Computing Surveys*, 18(1), pp. 67-108, March 1986.

[CDHMS89] Crow, F.C., Demos, G., Hardy, J., McLaughlin, J., Sims, K., "3D Image Synthesis on the Connection Machine", In: Parallel Processing for Computer Vision and Display, ed. P.M. Dew, R.A. Earnshaw, and T.R. Heywood, Addison-Wesley Publishers Ltd, England, pp. 254-269, 1989.

[CJ90] Clemens, D.J., Jacobs, D.W., "Model-Group Indexing for Recognition", *Proc. Image Understanding Workshop*, pp. 604-613, Sep. 1990.

[CJCM86] Cannon, S.R., Jones, G.W., Campbell, R., Morgan, N.W., "A Computer Vision System for Identification of Individuals", *Proc. IECON 86 0, WI.*, vol. 1, pp. 347-351, 1986.

[CMSB87] Castelow, D.A., Murray, D.W., Scott, G.L., Buxton, B.F., "Matching Canny Edgels to Compute the Principal Components of Optic Flow", *Proc. of the Alvey Vision Conference*, pp. 193-200, Sep. 1987.

[CPPMF90] Cai, L.D., Porrill, J., Pollard, S.B., Mayhew, J.E.W., Frisby, J.P., "Segmentation of Planar Curves Using Local and Global Behaviour Analysis", *Proc. of BMVC90*, pp. 247-251, Sep. 1990.

[Crow81] Crow, F.C., "A Comparison of Antialiasing Techniques", *IEEE Computer Graphics and Applications*, 1(1), pp.40-48, 1981.

[DC71] Dieudonné, J., Carrel, J.B., "Invariant Theory Old and New", Acedemic Press, 1971.

[DS84] Dippe, M., Swensen, J., "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis", *Computer Graphics (Proc. SIGGRAPH84)*, 18(3), pp. 149-158, July 1984.

[Faug90] Faugeras, O., "On the Motion of 3D Curves and its Relationship to Optical Flow", *Technical Report No. 1183*, INRIA, March 1990.

[FH83] Faugeras, O.D., Hébert, M., "A 3D Recognition and Positioning Algorithm Using Geometric Matching Between Primitive Surfaces", *Proc. Int. J. Conf. on AI*, Karlsruhe, pp. 996-1002, 1983.

[FH85] Flynn, A.M., Harris, J.G., "Recognition Algorithms for the Connection Machine", *Proc. Ninth International Joint Conf. on AI*, Morgan and Kaufman, pp. 57-60, 1985.

[FH86] Faugeras, O.D., Hébert, M., "The Representation, Recognition, and Locating of 3D Objects", *Int. J. Robotics Research*, 5(3), pp. 27-52, 1986.

[FHPP83] Faugeras, O.D., Hébert, M., Pauchon, E., Ponce, J., "Object Representation, Identification, and Positioning from Range Data", In Brady, J.M., and Paul, R., editors, *Proc. 1st Int. Symp. on Robotics Research*, MIT Press, Cambridge, MA, 1983.

[FJ90] Flynn, P.J., Jain, A.K., "BONSAI: 3D Object Recognition Using Constrained Search", *Proc. Third Int. Conf. on Computer Vision*, Osaka, pp. 263-267, Dec. 1990.

[Flyn90] Flynn, P.J., "CAD-Based Computer Vision: Modeling and Recognition Strategies", *PhD Thesis*, Department of Computer Science, Michigan State University, 1990.

[FMZ90] Forsyth, D., Mundy, J., Zisserman, A., "Transformational Invariance - A Primer", *Proc. of BMVC90*, pp. 1-5, Sep. 1990.

[FMZB90] Forsyth, D., Mundy, J.L., Zisserman, A., Brown, C.M., "Invariance - A New Framework for Vision", *Proc. Third Int. Conf. on Computer Vision*, Osaka, pp. 598-605, Dec. 1990.

[Forr88] Forrest, B., "Restoration of Binary Images Using Networks of Analogue Neurons", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 19-31, 1988.

[GG84] Geman, S., Geman, D., "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images", *IEEE Trans. PAMI*, vol. 5, pp. 721-741, 1984.

[GH90] Grimson, W.E.L., Huttenlocher, D.P., "On the Verification of Hypothesized Matches in Model-Based Recognition", *First European Conf. on Computer Vision*, pp. 489-498, April 1990.

[GL-P84*a*] Gaston, P.C., Lozano-Pérez, T., "Tactile Recognition and Localization Using Object Models: The Case of Polyhedra on a Plane", *IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI-6(3)*, pp. 257-266, May 1984.

[GL-P84*b*] Grimson, W.E.L., Lozano-Pérez, T., "Model-Based Recognition and Localization from Sparse Range or Tactile Data", *Int. J. Robotics Research*, 3(3), pp. 382-414, Fall 1984.

[GL-P84*c*] Grimson, W.E.L., Lozano-Pérez, T., "Parallel Algorithms for Computer Vision", In Winston, P.H., editor, *A Research Proposal to DARPA*, Chapter Parallel Recognition Algorithms, MIT, 1984.

[GL-P87] Grimson, W.E.L., Lozano-Pérez, T., "Localizing Overlapping Parts by Searching the Interpretation Tree", *IEEE Trans. PAMI-9(4)*, pp. 469-482, 1987.

[Goad83] Goad, C., "Special Purpose Automatic Programming for 3D Model-Based Vision", *Proc. Image Understanding Workshop*, Virginia, USA, pp. 94-104, 1983.

[Gree89] Greenshields, I.R., "A Dynamically Reconfigurable Multimodal Architecture for Image Processing", In: Parallel Processing for Computer Vision and Display, ed. P.M. Dew, R.A. Earnshaw, and T.R. Heywood, Addison-Wesley Publishers Ltd, England, pp. 153-165, 1989.

[Greg78] Gregory, R.L., "Eye and Brain", McGraw Hill, New York, 1978.

[Grim84] Grimson, W.E.L., "The Combinatorics of Local Constraints in Model-Based Recognition and Localization from Sparse Data", *Technical Report AI Memo 763*, MIT AI Laboratory, Cambridge, M.A., April 1984.

[Grim88] Grimson, W.E.L., "The Combinatorics of Object Recognition in Cluttered Environments Using Constrained Search", *Technical Report 1019*, MIT AI Laboratory, Cambridge, M.A., 1988.

[Grim90] Grimson, W.E.L., "The Effect of Indexing on the Complexity of Object Recognition", *Proc. Third International Conf. on Computer Vision*, pp. 644-651, 1990.

[GTGB84] Goral, C.M., Torrance, K.E., Greenberg, D.P., Battaile, B., "Modeling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics (Proc. SIGGRAPH84)*, pp. 213-222, 1984.

[GY03] Grace, J.H., Young, A., "The Algebra of Invariants", Cambridge University Press, Cambridge, 1903.

[Harm82] Harmon, L.D., "Automated Tactile Sensing", *Robotics Res.*, vol. 1, pp. 3-32, Summer 1982.

[HB89*a*] Holder, D., Buxton, H., "Polyhedral Object Recognition with Sparse Data in SIMD Processing Mode", *Image and Vision Computing*, 7(1), pp. 71-78, Feb. 1989.

[HB89*b*] Holder, D., Buxton, H., "Polyhedral Object Recognition with Sparse Data - Validation of Interpretations", *Proc. of the Fifth Alvey Vision Conf.*, Reading, pp. 19-24, Sep. 1989.

[Heck84] Heckbert, P., "The Mathematics of Quadric Surface Rendering and SOID", *New York Inst. of Tech.*, Computer Graphics Lab TM4, July 1984.

[HH84] Heckbert, P., Hanrahan, P., "Beam Tracing Polygonal Objects", *Computer Graphics (Proc. SIGGRAPH 84)*, 18(3), pp. 119-127, July 1984.

[Hill82] Hillis, W.D., "A High-Resolution Image Touch Sensor", *Robotics Res.*, vol. 1, pp. 33-44, Summer 1982.

[Hill85] Hillis, W.D., "The Connection Machine", The MIT Press, Cambridge, Massachusetts, 1985.

[Hogg83] Hogg, D., "Model-Based Vision: A Program to see a Walking Person", *Image and Vision Computing*, 1(1), pp. 5-20, Feb. 1983.

[HU90] Huttenlocher, D.P., Ullman, S., "Recognising Solid Objects by Alignment with an Image", *Int. J. of Computer Vision*, 5(2), pp. 195-212, Nov. 1990.

[Jarv83] Jarvis, R.A., "A Perspective on Range Finding Techniques for Computer Vision", *IEEE Trans. Pattern Anal. Machine Intell. PAMI-5(2)*, pp. 122-193, 1983.

[JL87] Jalowiecki, I.P., Lea, R.M., "A 256-Element Associative Parallel Processor", *Proc. Int. Solid State Circuits Conf.*, New York, pp. 196-197, Feb. 1987.

[Kana77] Kanade, T., "Computer Recognition of Human Faces", Birkhauser Verlag, Basel and Suttgart, 1977.

[KD75] Koenderink, J.J., Doorn, A.J. van, "Invariant Properties of the Motion Parallax Field due to the Movement of Rigid Bodies Relative to an Observer", *Optica Acta*, 22(9), pp. 773-791, 1975.

[KD76] Koenderink, J.J., Doorn, A.J. van, "Geometry of Binocular Vision and Model for Stereopsis", *Biological Cybernetics*, vol. 21, pp. 29-35, 1976.

[KK86] Kay, T.L., Kajiya, J.T., "Ray Tracing Complex Scenes", *Computer Graphics (Proc. SIGGRAPH86)*, 20(4), pp. 269-278, August 1986.

[KL89] Krikelis, A., Lea, R.M., "Low-Level Vision Tasks Using Parallel String Architectures", In: Parallel Processing for Computer Vision and Display, ed. P.M. Dew, R.A. Earnshaw, and T.R. Heywood, Addison-Wesley Publishers Ltd, England, pp. 103-118, 1989.

[Koen84] Koenderink, J.J., "What Does the Occluding Contour Tell us About Solid Shape?", *Perception*, vol. 13, pp. 321-330, 1984.

[KOO90] Kuno, Y., Okamoto, Y., Okada, S., "Object Recognition Using a Feature Search Strategy Generated from a 3-D Model", *Proc. Third International Conf. on Computer Vision*, pp. 626-635, 1990.

[Kuch88] Kuchkuda, R., "An Introduction to Ray Tracing", *NATO ASI Series, vol. F40, Theoretical Foundations of Computer Graphics and CAD*, edited by R.A. Earnshaw, Springer-Verlag Berlin, Heidelberg, 1988.

[KV-H84] Kajiya, J.T., Von Herzen, B.P., "Ray Tracing Volume Densities", *Computer Graphics (Proc. SIGGRAPH84)*, 18(3), pp. 165-174, July 1984.

[Lea86*a*] Lea, R.M., "SCAPE: A Single-Chip Array Processing Element for Signal and Image Processing", *IEE Proc. Pt. E.*, 133(3), pp. 145-151, 1986.

[Lea86*b*] Lea, R.M., "VLSI and WSI Associative String Processors for Cost-Effective Parallel Processing", *The Computer Journal*, 29(6), pp. 486-494, 1986.

[Lea86*c*] Lea, R.M., "VLSI and WSI Associative String Processors for Structured Data Processing", *IEE Proc. Pt. E.*, 133(3), pp. 153-162, 1986.

[L-HP80] Longuet-Higgins, H.C., Pradzny, K., "The Interpretation of a Moving Retinal Image", *Proc. Royal Soc. London*, B208:385-397, 1980.

[Lin87] Lin, C., "New Forms of Shape Invariants From Elliptical Fourier Descriptions", *Pattern Recognition*, 20(5), pp. 535-545, 1987.

[LS88] Lin, T., Slater, M., "Parallel Ray Tracing Using an Integrated Space Subdivision Scheme", *Proc. SURF 2nd Ann. Meeting*, 1988.

[LSW88] Lamdan, Y., Schwartz, J.T., Wolfson, H.J., "Object Recognition by Affine Invariant Matching", *IEEE Conf. Computer Vision and Pattern Recognition*, pp. 335-344, 1988.

[LW88] Lamdan, Y., Wolfson, H.J., "Geometric Hashing: A General and Efficient Model-Based Recognition Scheme", *Proc. of ICCV*, Tampa, Florida, pp. 238-249, Dec. 1988.

[Malh90*a*] Malhi, R.K., "Parallel Tiling Algorithms for Visualising Solid Models", *PhD Thesis*, Dept. of Computer Science, QMW College, London, pp. 73-76, June 1990.

[Malh90*b*] As [Malh90*a*] - p. 104, June 1990.

[Mari81] Marik, V., "Algorithms of the Complex Tactile Information Processing", *Proc. 7th Int. Joint Conf. on AI*, pp. 773-774, 1981.

[Marr82] Marr, D., "Vision", W.H. Freeman, San Francisco, C.A., 1982.

[MB90] Murray, D.W., Buxton, B.F., "Experiments in the Machine Interpretation of Visual Motion", The MIT Press, Cambridge, Massachusetts, 1990.

[MBLSB92] Marsh, B., Brown, C., LeBlance, T., Scott, M., Becker, T., Quiroz, C., Das, P., Karlsson, J., "The Rochester Checkers Player: Multimodel Parallel Programming for Animate Vision", *COMPUTER*, 25(2), pp. 12-19, Feb. 1992.

[MC88] Murray, D.W., Cook, D.B., "Using the Orientation of Fragmentary 3D Edge Segments for Polyhedral Object Recognition", *Int. J. Comput. Vision*, vol. 2, pp. 147-163, 1988.

[MCB89] Murray, D.W., Castelow, D.A., Buxton, B.F., "From Image Sequences to Recognized Moving Polyhedral Objects", *Int. J. of Computer Vision*, vol. 3, pp. 181-208, 1989.

[MDW88] Manning, L., Dew, P.M., Wang, H., "Design and Analysis of Image Processing Algorithms for Programmable VLSI Array Processors", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 217-242, 1988.

[MF81] Mayhew, J.E.W., Frisby, J.P., "Psychophysical and Computational Studies Towards a Theory of Human Stereopsis", *Artificial Intelligence*, vol. 17, pp. 349-385, 1981.

[MH80] Marr, D., Hildreth, E., "Theory of Edge Detection", *Proc. Royal Soc.*, London, B207, pp. 187-217, 1980.

[MKB86] Murray, D.W., Kashko, A., Buxton, H., "A Parallel Approach to the Picture Restoration Algorithm of Geman and Geman", Image and Vision Computing, vol. 3, pp. 133-142, 1986.

[MP88] Morrow, P., Perrott, R.H., "The Design and Implementation of Low-Level Image Processing Algorithms on a Transputer Network", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 243-259, 1988.

[MT87] Mundy, J.L., Thompson, D.W., "Three-Dimensional Model Matching From an Unconstrained Viewpoint", *Proc. IEEE Robotics and Automation*, pp. 208-220, 1987.

[MU91] Moses, Y., Ullman, S., "Limitations of Non Model-Based Recognition Schemes", *ECCV-92*, to appear.

[Murr87] Murray, D.W., "Model-Based Recognition Using 3D Shape Alone", *Computer Vision Graphics and Image Processing*, vol. 40, pp. 250-266, 1987.

[MW85] Murray, D.W., Williams, N.S., "Detecting the Image Boundaries Between Flow Fields for Several Moving Planar Facets", *Pattern Recognition Letters*, 1985.

[NB86] Nalwa, V.S., Binford, T.O., "On Detecting Edges", *IEEE Trans. on Pattern Analysis and Machine Intelligence PAMI-8(6)*, pp. 699-714, 1986.

[NLC90] Nasrabadi, N.M., Li, W., Choo, C.Y., "Object Recognition by a Hopfield Neural Network", *Third Int. Conf. on Computer Vision*, Osaka, pp. 325-328, Dec. 1990.

[NN85] Nishita, T., Nakamae, E., "Continuous Tone Representation of Three Dimensional Objects Taking Account of Shadows and Interreflection", *Computer Graphics (Proc. SIGGRAPH85)*, pp. 23-30, 1985.

[NS89] Naqri, A.A, Sandler, M.B., "Performance of the OSMMA Image Processing System", In: Parallel Processing for Computer Vision and Display, ed. P.M. Dew, R.A. Earnshaw, and T.R. Heywood, Addison-Wesley Publishers Ltd, England, pp. 145-152, 1989.

[OS83] Oshima, M., Shirai, Y., "Object Recognition Using Three-Dimensional Information", *PAMI*, 5(4), pp. 353-361, July 1983.

[OT77] Okada, T., Tsuchiya, S., "Object Recognition by Grasping", *Pattern Recognition*, 9(3), pp. 111-119, 1977.

[Page88*a*] Page, I., "Parallel Architectures and Computer Vision", Clarendon Press, Oxford, 1988.

[Page88*b*] Page, I., "The Disputer: A Dual-Paradigm Parallel Processor for Graphics and Vision", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 201-216, 1988.

[Page89] Page, I., "Graphics + Vision = SIMD + MIMD (A Novel Dual-Paradigm Approach)", In: Parallel Processing for Computer Vision and Display, ed. P.M. Dew, R.A. Earnshaw, and T.R. Heywood, Addison-Wesley Publishers Ltd, England, pp. 89-102, 1989.

[PMF85] Pollard, S.B., Mayhew, J.E.W., Frisby, J.P., "PMF: A Stereo Correspondence Algorithm Using a Disparity Gradient Limit", *Perception*, vol. 14, pp. 449-470, 1985.

[Poll88] Pollard, S., "The PMF Stereo Algorithm: Theory and Implementation", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 61-79, 1988.

[PPM87] Pridmore, T.P., Porrill, J., Mayhew, J.E.W., "Segmentation and Description of Binocularly Viewed Contours", *Image and Vision Computing*, 5(2), pp. 132-138, 1987.

[Quar84] Quarendon, P., "WINSOM Users Guide", *Doc. no. UKSC 123*, Aug. 30, 1984.

[Redd88] Reddaway, S., "Mapping Images onto Processor Array Hardware", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 299-314, 1988.

[RH91] Rigoutsos, I., Hummel, R., "A Bayesian Approach to Model Matching with Geometric Hashing", *ECCV-92*, to appear.

[RH92] Rigoutsos, I., Hummel, R., "Massively Parallel Model Matching: Geometric Hashing on the Connection Machine", *COMPUTER*, 25(2), pp. 33-42, Feb. 1992.

[RL85] Reiger, J.H., Lawton, D.L., "Processing Differential Image Motion", *J. Optical Soc. of America*, A2(2), pp. 354-359, 1985.

[Roth82] Roth, S.D., "Ray Casting for Modeling Solids", *Computer Graphics and Image Processing*, vol. 18, pp. 109-144, 1982.

[Ruff88] Ruff, B., "A Pipelined Architecture for a Video-Rate Canny Operator Used at the Initial Stage of a Stereo Image Analysis System", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 171-185, 1988.

[Slat86] Slater, M., "Segments on Bit Mapped Graphic Displays", *Software-Practice and Experience*, 16(11), pp. 965-980, Nov. 1986.

[Spri77] Springer, T.A., "Invariant Theory", Springer-Verlag Lecture Note in Mathematics, 585, 1977.

[SRH88] Sleigh, A., Radford, J., Harp, G., "RSRE Experience Implementing Computer Vision Algorithms on Transputers, DAP and DIPOD Parallel Processors", In: Parallel Architectures and Computer Vision, ed. I. Page, Clarendon Press, Oxford, pp. 133-155, 1988.

[SRS90] Shankar, R.V., Ramamoorthy, G., Suk, M., "Three Dimensional Object Recognition on the Connection Machine", *Pattern Recognition Letters*, 11(7), pp. 485-492, July 1990.

[SS87] Salmon, R., Slater, M., "Computer Graphics - Systems and Concepts", Addison-Wesley Publishing Company Inc., pp.420-423, 1987.

[STN87] Shinya, M., Takahashi, T., Naito, S., "Principles and Applications of Pencil Tracing", *Computer Graphics (Proc. SIGGRAPH87)*, pp. 45-54, 1987.

[Stoc87] Stockman, G., "Object Recognition and Localization via Pose Clustering", *Comp. Vision, Graphics, Image Proc.*, vol. 40, pp. 361-387, 1987.

[TINA] Pollard, S.B., Porrill, J., Thacker, B.A., "Tina.1 Tinatool User Documentation", *AIVRU Ref. No. 50*, University of Sheffield.

[TL90] Tanaka, H.T., Lee, D.T.L., "Representing Surface Curvature Discontinuities on Curved Surfaces", *Proc. Third Int. Conf. on Computer Vision*, Osaka, pp. 304-308, Dec. 1990.

[TM89] Trotter, J.A., Moore, W.R., "MESH: An Architecture for Image Processing", In: Parallel Processing for Computer Vision and Display, ed. P.M. Dew, R.A. Earnshaw, and T.R. Heywood, Addison-Wesley Publishers Ltd, England, pp. 136-144, 1989.

[WBB87] Williams, N.S., Buxton, B.F., Buxton, H., "Distributed Ray Tracing Using an SIMD Processor Array", *NATO Advanced Study Institute*, Il Ciocco, Italy, 1987.

[Webb92] Webb, J., "Steps Towards Architecture-Independent Image Processing", *COMPUTER*, 25(2), pp. 21-31, Feb. 1992.

[Wein89] Weinshall, D., "Direct Computation of 3D Shape and Motion Invariants", *Technical Report A.I. Memo No. 1131*, MIT AI Lab, May 1989.

[Whit80] Whitted, T., "An Improved Illumination Model for Shaded Display", *Comm. ACM*, 23(6), 1980.

[WLT89] Wong, K.H., Law, H.H.M, Tsang, P.W.M, "A System for Recognizing Human Faces", *Proc. ICASSP*, pp. 1638-1642, 1989.

[WMMW92] Wallace, A.M., Michaelson, G.J., McAndrew, P., Waugh, K.G., Austin, W.J., "Dynamic Control and Prototyping of Parallel Algorithms for Intermediate and High-Level Vision", *COMPUTER*, 25(2), pp. 54-63, Feb. 1992.

[Wolf90] Wolfson, H.J., "Model-Based Object Recongition by Geometric Hashing", *First Euro. Conf. on Comp. Vision*, pp. 526-536, 1990.

[Wyso89] Wysocki, J., "A Parallel Implementation of the Canny Operator for the DAP", Technical Report no. 491, Dept. of Computer Science, Queen Mary College, London, August 1989.

[ZMFMR90] Zisserman, A., Marinos, C., Forsyth, D.A., Mundy, J.L., Rothwell, C.A., "Relative Motion and Pose From Invariants", *Proc. of BMVC90*, pp. 7-12, Sep. 1990.