A Three Dimensional Image Processing Program For A Parallel Computer

Michael G Norman

MSc in Information Technology,
Department of Artificial Intelligence
University of Edinburgh
1987

Abstract

A program has been developed to manipulate and display three dimensional NMR bodyscan data on a Multiple Instruction Multiple Data parallel computer: the Meiko Computing Surface.

A similar program had been implemented on a SUN 2 serial machine by previous MSc students.

Innovative aspects of the work have been in dividing the three dimensional dataset between processors, in arranging for secure communications between processors and in implementing a generalisation of the Zucker-Hummel surface detection operator to non-cubic voxels

The main features of the are that it will display data in two dimensions, calculate the derived T1 dataset, calculate the Zucker Hummel coefficients, threshold data and manipulate binary flagset maps of the dataset.

Acknowledgements

Undertaking this project has been an extremely rewarding experience for me, and I would doubtless have floundered without the constant help of many people of many different affiliations. The fact that it happened at all is a remarkable tribute to the collaboration between departments in the university, and between the university and industry. I should like to explicitly thank the following people, although in truth they are the tip of the iceberg.

Dr Robert Fisher of the Department of Artificial Intelligence.

Greg Wilson of the Department of Physics.
For their help as day to day, (and day to night) supervisors.

Duncan Roweth of Meiko ltd.
For his invaluable help with the inscrutable Meiko M40.

Adam Zentner of the Department of Computer Science.
For developing the user interface through which the program communicates, and for his constant help and companionship.

Professor David Wallace of the Department of Physics.
For allowing access to the Edinburgh Concurrent Supercomputer facilities.

Contents

Chapter 1

Introduction

1.1. Background

This dissertation describes a study to show how three-dimensional bodyscan data can be manipulated within a Multiple Instruction / Multiple Data (MIMD) parallel architecture.

Bodyscan data comes in three main types: Computerised Tomography or Computerised Axial Tomography (CT or CAT) is an X-Ray based three dimensional imaging system. The pictures may be produced in real time and are rarely subjected to significant image processing; Positron Emission Tomography (PET) is a relatively new imaging technique whose clinical usefulness is currently under investigation; Nuclear Magnetic Resonance or Magnetic Resonance Imaging (NMR or MRI) is a well known and useful imaging technique, currently in use in most Area Health Authorities in the UK.

The program is set up to analyse MRI data, although with a little modification CT, and PET could be used. Indeed there is no reason why any type of three dimensional data such as sequential embryonic sections or gamma scan images of industrial parts might not be analysed by the system.

MRI data is of the form of sections through the body or part of the body. The spatial resolution of the data is lower in the axial direction (normal to the plane of the section) than in the plane of the

section. Typically datasets have 128*128 or 256*256 pixels in the section, and there are 10 - 20 sections.

Over the past two years there have been two MSc projects in the Department of Artificial Intelligence which have addressed the problems of applying A.I. techniques to medical images. (Nelson [85] and Reeve [86]). The work described here was an attempt to implement a similar program within a multiple instruction multiple data (MIMD) parallel architecture so that it would run at an acceptable speed.

This work had two main goals. On the one hand, it aimed to duplicate and enhance many of the image processing functions which Nelson and Reeve had previously implemented. On the other hand it aimed to show how three-dimensional image processing could be mapped efficiently onto MIMD parallel hardware. The structure of this document reflect the two aspects. Chapter two describes the way tasks and data were distributed between processors and chapter three describes how communications were arranged between processors. Chapter four deals with the image processing aspects of the work. It describes the enhancements and other changes made to the image processing algorithms of Nelson and Reeve.

Chapter five gives an assessment of the program produced, both in terms of the efficacy and speed of the implementation and also in terms of the general feasibility of medical image processing on MIMD hardware. This chapter also discusses possible extensions and improvements to the system.

Appendices include a user guide (which may be read separately from this document); a description of the protocols used for inter-processor

communication signals; and a description of the hardware on which it was implemented.

1.2. The Previous Program

Over the past two years, between the two MSc students involved, a large program was developed for bodyscan manipulation.

Nelson originated the program, and developed a three-dimensional surface detection algorithm based upon the Zucker-Hummel edge detection operator. Boundary detection was achieved in the following way:

The user specified a point which was judged to be on the surface. The boundary was tracked away from this point using values of edge contrast and edge orientation to guide the formation of the boundary. The detected boundary was made up of voxels, and the surface of the boundary was determined by marking those faces of the voxel which were on the boundary. A surface display was produced by displaying the marked faces, and using the standard computer graphics techniques of viewing transformation, surface shading and hidden surface removal, as supplied by the Suncore graphics interface. He also developed a surface editor to allow the operations of union intersection and difference to be performed on surfaces, and surfaces to clipped in the X or Y planes.

Reeve's contribution was to improve the techniques of surface detection by developing three-dimensional generalisations of the Canny and Walsh edge detection operators. He also improved the quality of the user interface.

1.3. A Description Of The Problem

Although Nelson and Reeve have together produced a program which is capable of manipulating and presenting bodyscan data, it must be remembered that the program was implemented on conventional hardware: a SUN 2 workstation, in a conventional language: C.

The move to parallel hardware was necessary to improve the speed of the program. From a commercial point of view it is important that a medical image handling system should be powerful, fast and not excessively expensive. The parallel architecture of the Meiko Computing Surface allows considerable computing power to be attained at a fraction of the cost of comparable machines.

It has been traditional for A.I. workers to abstract their programs from the hardware on which they were to run, but in the case of the move to parallel hardware this is at present difficult to achieve. It is apparent that for a program to run efficiently and for the benefits of parallelisation to be seen, the structure of the program must map very closely onto the architecture of the machine on which it is to run. The task of producing a parallel implementation of the bodyscanner program is not best solved by parallelising the existing code, but by re-assessing the problem and writing a program which solves it in parallel.

The goals of this research were to implement the following operations which Nelson and Reeve had previously implemented.

1. Display two dimensional sections of medical NMR image data.
2. Calculate the T1 dataset, given Proton Density and Difference datasets.

3. Apply conservative smoothing to the data.

4. Apply a three-dimensional version of the Zucker-Hummel edge detection operator to the data.

5. Detect surfaces in the three-dimensional data by tracking points of high edge magnitude away from a user specified seed point.

6. Store surfaces and manipulate them with intersection, union and other operators.

7. Display the surfaces in three dimensions

In addition it was possible to implement the following operations:

8. Display of the magnitude of the surface normal found by the Zucker Hummel operator.

9. Display only of data lying within user-defined thresholds:

(a) Thresholding the intensity of the data at each point.

(b) Thresholding the intensity of the edge at each point.

1.4. Constraints On The Program

There are constraints on the way that the program was written. Some of these were due to the specific medical requirements of the program, others with the data, and still others were related to the hardware being used.

As explained earlier in section 1.1, the volume of data to be manipulated is very large, typically there will be 128*128 pixels in a section, each pixel taking up 8 bits, and 10 to 20 sections will be

required to make up a whole three dimensional image. It is possible that the data will be of higher resolution than this, perhaps 256*256. In addition it will be necessary to store up to three such datasets.

The new program was written for a Meiko Computing Surface, a multi transputer machine, and developed on the machine belonging to the Edinburgh Concurrent supercomputer facility. The current configuration of that machine is given in Appendix 1.

Twenty five processors were available for image processing and one for controlling the program. Each of these processors was an INMOS IMS T414 microprocessor with 256K Bytes of memory. The program was written in the language occam which is the native language of the Transputer. The input and output of the program were supplied by interfaces written by Zentner [87] and Meiko [87], [85].

Some difficulties were encountered during the work which related directly to problems with the hardware and system software being used. The Edinburgh Concurrent Supercomputer facility is still under development and there remain several bugs and inconsistencies in it. I do not intend to discuss system problems in detail, although chapter five does contains some assessment of the Computing Surface's suitability for this type of program.

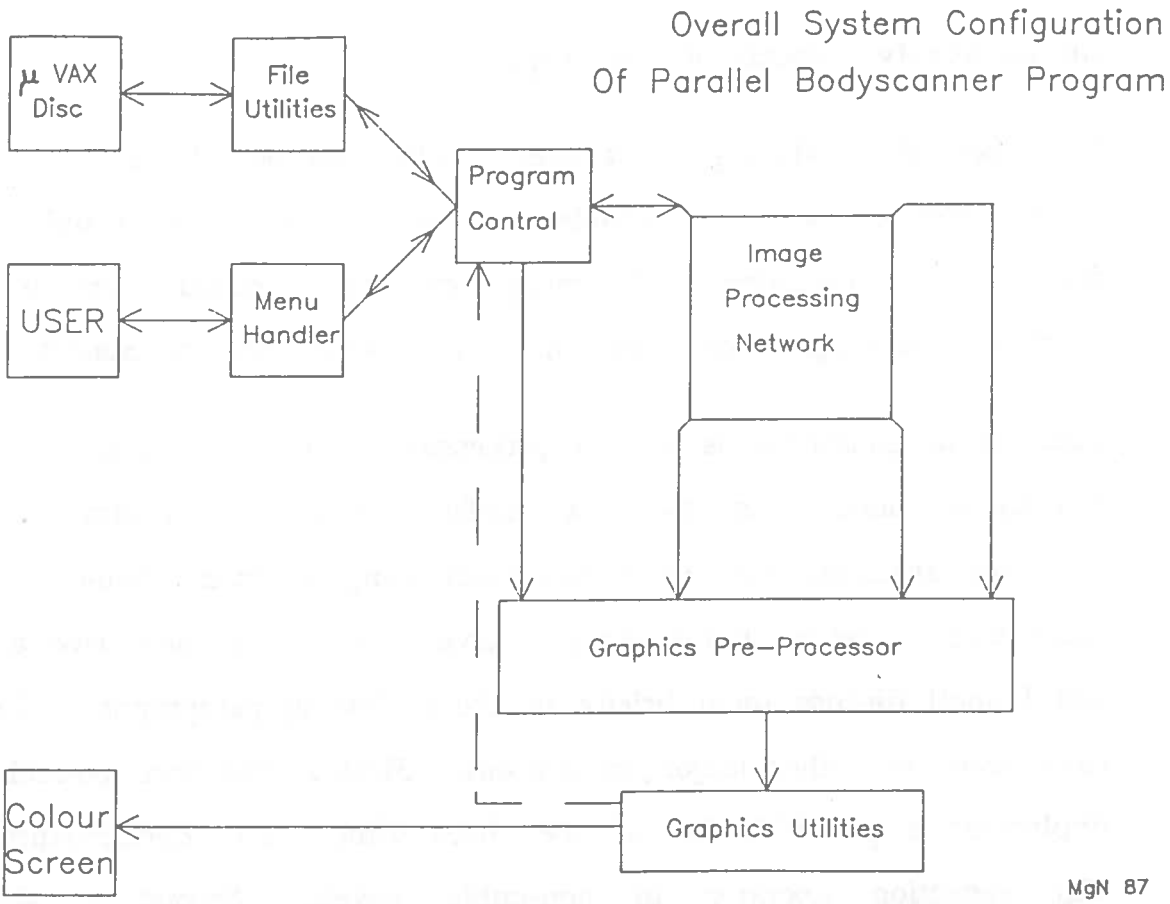
Next come the medical constraints. The move to a parallel architecture was made primarily to increase the speed of image processing. This speed is directly related to the size of the dataset, and to its tomographic nature. Many slow image processing operations work on the neighbourhood of a voxel. Examples include the boundary tracking and Zucker-Hummel operations mentioned earlier as goals for the program. In three dimensions each image element has

27 neighbours as compared to only 9 neighbours for elements of a two dimensional image. Moreover, the three dimensional datasets typically have an order of magnitude more data than the typical two dimensional image.

The image processing operations need to be performed at a speed which medical practitioners would consider as real time. Some assessment of the speed of the program is given in Chapter Five. It is beyond the scope of this project to discuss the medical usefulness of the operations that the program performs.

1.5. An Overview Of The Program

Below is a block diagram of the modules into which the program divides.



The conceptual program as discussed previously only refers to the program control and image processing parts of the diagram. The other modules are separate processes, many of which run on separate transputers, but in truth they may be regarded as interfaces similar in essence to the menu, window and 3D graphics facilities of the Suncore Library on the SUN 2.

Graphics and file utilities are Meiko system code, but the Menu Handler and the graphics pre-processor were written specifically for the bodyscan project by Adam Zentner (Zentner [87]).

Arrowed lines in the above diagram represent directional connections between program modules. The dotted line between the graphics and the program control shows the intended connections for a mouse interface to the program.

1.6. Innovative aspects of the program

The task of producing a parallel implementation of any program centres around two main problems. First there is the problem of dividing the processing load among processors, second there is the problem of arranging for communication between these processors.

There is no consensus as to how processing should be divided between processors, neither is there a useful library of communication harnesses available for the Meiko Computing Surface. Some of the innovative elements of the work I have done lie in these two areas, and I shall discuss them briefly in the following paragraphs. There have been two other major innovations. First it has been possible to implement a generalisation of the three dimensional Zucker-Hummel edge detection operator to non-cubic voxels. Second a simple

modification of the datastructures used for storing surface data has allowed an increase in speed for surface manipulation. Nelson and Reeve's approach, where two lists were intersected took of the order of fifteen minutes for a decent sized surface. The version I have is a straight bitwise comparison of two integer vectors on each processor, and takes of the order of fifty milliseconds for a surface of any size.

The distribution of tasks among processors is done in a way that is essentially data determined. The data is divided into blocks which are assigned to each of the different processors. The dataset division is done so as to approximately equalise the data which is assigned to each processor in both a given section and the whole dataset. It is complicated by aiming to optimise the speed of the surface tracking operations. The details of the block assignment process are given in chapter 2.

The protocols for communication between the processors are performed by separate processes from those which are performing image processing, and in parallel. The communications are packeted, and each processor redirects packets so as best to spread the load of communication across the available inter-processor bandwidth. The communications are actively buffered, and elaborate mechanisms are in place which aim to stop the network from deadlocking, and also to allow for slight variations in the inter-processor communication rates.

Chapter 2

Distribution of Data and Processing

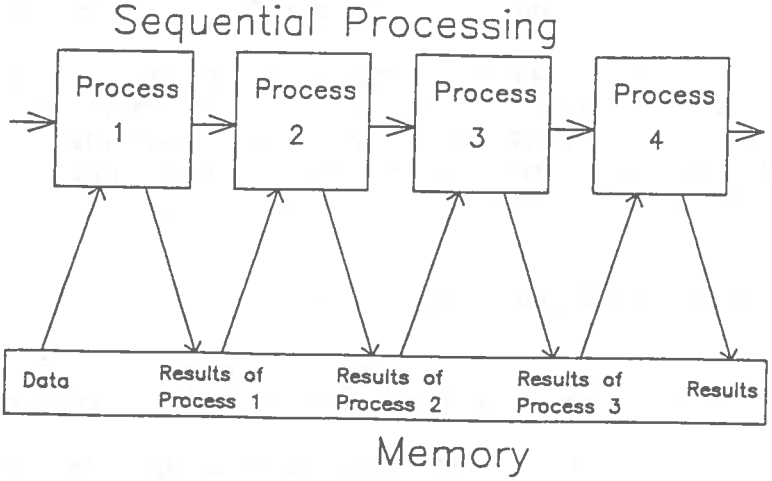
2.1. Programming with Communicating Processes

In order to perform a task within a conventional serial computer architecture it must be broken down into subtasks to be executed in sequence. Each of these subtasks may be further broken down into smaller subtasks until each is a machine instruction. It is the execution of the machine instructions in sequence which performs the computation.

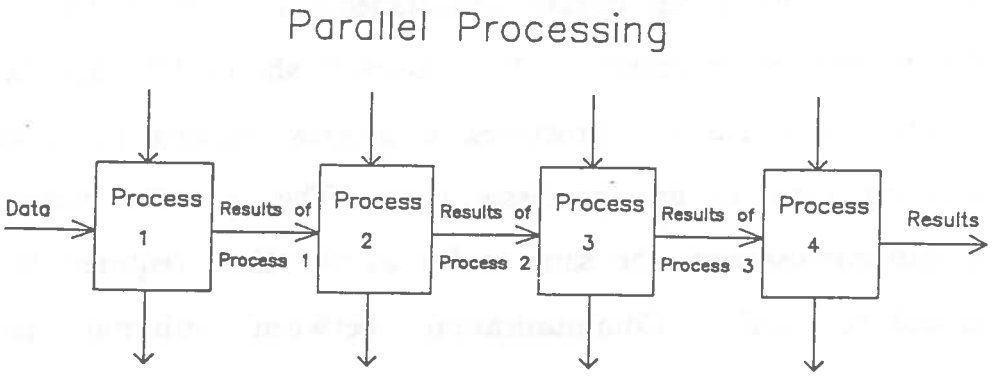
When decomposing a task for a program which is to execute on MIMD parallel hardware, at each level of subdivision there is a choice: whether to execute the subtasks in sequence or in parallel. Subtasks whether executed in sequence or in parallel are usually referred to as processes.

In practice the choice of sequential or parallel subdivision is not arbitrary. It is possible to perform any computation in either mode, but the efficiency of particular mappings of code onto hardware, and the conceptual ease of programming vary markedly according to the choices which are made. It is most important that the division of the problem allows each processor within the MIMD machine to run a different parallel process, since only then may all the processors in the machine be in use.

In understanding the MIMD parallel approach to programming, as exemplified by the language occam it is useful to think of programs as sets of processes which communicate. The description applies as well to sequential programs as to parallel ones. Consider a generalised sequential program constructed of subroutines.



The subroutines may be thought of as communicating via the processor's associated memory. A parallel version of the program is shown below. It is composed of processes arranged in a pipeline which communicate their results to each other via channels. (A channel is a conceptual entity which allows secure communication between processors. The implementation of a channel may vary considerably.) The input of the pipeline is the data to be manipulated, and the output is the result of the processing. Each of the processes may be allowed to execute in parallel, and indeed be separated in space from the other processes since communication occurs via channels.



A Model of Parallel Processing Showing
Communication Between Processes via Channels

The language occam embodies the concepts of communicating processes within its structure . The following is given as an introduction to the language by May [87]

> A process starts, performs a number of actions, and then either stops or terminates. Each action may be either an assignment, an input or an output. An assignment changes the value of a variable, an input receives a value from a channel, and an output sends a value to a channel.
>
> At any time between its start and termination, a process may be ready to communicate on one or more of its channels. Each channel provides a one way connection between two concurrent processes; one of the processes may only output to the channel, and the other may only input from it.

2.2. Hardware and Data Constraints

At this point it is useful to describe the hardware for which the program was written. The image processing was distributed between 25 INMOS IMS T414 processors (transputers), part of the Meiko M40 Computing Surface of the Edinburgh Concurrent Supercomputer Project, at the University of Edinburgh A description of the current system configuration is given as Appendix 1

The INMOS IMS T414 transputer (INMOS [87]) is a high performance 32 bit microprocessor with 2K bytes of on chip random access memory and on chip communications hardware. It has four bidirectional hardware buffered communications links which allow point to point connection between transputers, and along each of which it may communicate simultaneously. The native language of the transputer is occam. The processor shares its time between any number of concurrent processes, a process waiting for communication does not consume any processor time. The time for process switching is sub-microsecond, the same order as the time required to generate a procedure call. Communication between internal processes is

implemented by memory to memory block move operations. Communication to external processes is available via the communications links.

The Meiko M40 (Meiko [87]) is one of a range of multiprocessor machines built by Meiko and based around the INMOS transputer. The machine allows dynamic reconfiguration of networks of transputers. It provides a programming environment and some utilities. These are supplemented by interfaces specially written for the bodyscan program. Screen output and keyboard input are available via a menu shell developed by Zentner [87]. Graphics output is available via a graphics pre-processor (Zentner [87], Theoharis [86]),

The test datasets to be manipulated by the program are two 128*128*12 arrays of 8 bit data. (Proton Density and Difference) and a further 128*128*12 array (T1 data) to be calculated from the other two. The total size of the datasets is therefore 128*128*12*3 Bytes; 576 K Bytes. The inhomogeneity of the data precludes any great reduction in the size of the datasets by such techniques as run length encoding or octree representations.

2.3. Approaches to Parallelisation

The division of a task into separate subtasks to be executed by different processes in parallel is the essence of parallel programming. Optimal solutions to the problem of task division maximise the processing load at each processor, whilst minimising the proportion of the processing which results directly from inter-process communication.

There are three opposed approaches.

The algebraic parallelism, or task parallelism approach, as exemplified by the pipeline of processors (see Theoharis [86] for a graphics pipeline example) divides the given task among processes in a similar way to the way sequential programmers tend to modularise programs. The program is divided into as many processes as there are modules to be executed.

Task parallelism is often simple to program. In particular the signals to be communicated between processes tend to be stereotyped so that communication protocols are simplified. The efficiency of parallelism is often low in comparison to that which may be attained with data parallelism, and it is often difficult to extend the approach to large number of processors. (Morrow & Perrot [87])

The data parallelism approach, as exemplified by the usual SIMD programming method, Reddaway [87] is to divide the data, and to assign a portion of the dataset to each of the processes. It is then possible to arrange that all operations on a given portion of the dataset are performed by a single process, and that each process runs the same program.

Data parallelism can be shown to be efficient only if operations are to be performed globally on the dataset or if the flow of computation is known in advance and the data can be assigned so as to maximise processor usage.

The task farm approach as exemplified by the ALICE is to assign both data and processes to processors dynamically at runtime according to the overall processing strategy. It is common in task farming approaches for the task to be sent out in the form of the actual code which must be executed.

It is also possible to implement a distributed task farming control structure where every processor is able to generate tasks to be performed by other processors.

Task-farming is efficient only if the overheads inherent in the farming process are insignificant as a proportion of the processing involved in performing the tasks. Since the data and the task are both sent out in the farming process it is clearly less suitable for data intensive tasks such as low level image processing.

2.4. Mapping tasks onto Processors

2.4.1. The Type of Mapping Used

The major problem that was faced in determining an efficient problem decomposition was the large size of the dataset. It was not possible for the whole of the data to be placed on a single processor nor was it practicable for it to be overlaid to disk: the rate of data transfer from disk to processors was so slow as to make this unworkable. A form of data parallelism was required simply to allow the whole of the data set to be available to the set of processors. Given that the task mapping had to include an element of data parallelism, there was no necessity for it to be exclusively data determined. The possibility remained of a distributed task-farming approach where although the data was distributed between processors, each processor could generate tasks to be performed by other processors. It was also possible to allow a degree of processor task specialisation within the essentially data-parallel framework.

The task decomposition that was used included a large degree of data

parallelism, and I should like to describe it initially in these terms. The specialisations which allow a limited form of task-farming will be described along with the reasons for my choice of task mapping in section 2.4.4. .

In mapping 2D image arrays onto 4-connected processor arrays, such as arrays of transputers, and indeed the ICL DAP there are two broad approaches (Reddaway [87]). In both cases the image is divided into regions at the edges of which there are discontinuities in the way the data is assigned. The sheet mapping assigns adjacent pixels within given regions of the image to adjacent processors, and the crinkled mapping assigns all of the pixels within an region to the same processor. In its pure form there are as many pixels as there are processors.

The approach I have used is similar to the crinkled mapping in that the data is broken down into physically contiguous regions each of which is assigned to a processor. It differs from a pure crinkled mapping in that more than one box in the image is assigned to each processor. Another difference is that the bodyscan data is tomographic and so the regions have a third dimension, they are boxes of data rather than flat image segments.

2.4.2. Dataset Partitioning

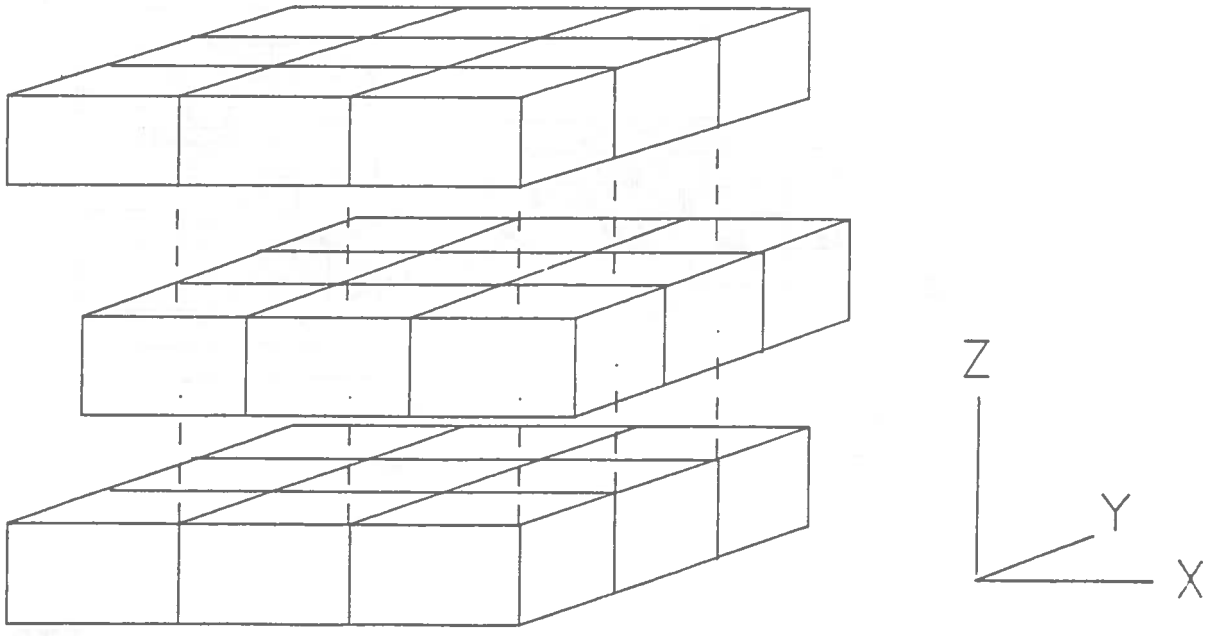
The dataset is divided in the following way

1. The data is divided into boxes in X,Y and Z dimensions. The X and Y dimensions are defined as being in the plane of the data section, the Z dimension being normal to the plane of the section.

2. The box size allows a whole number of voxel widths in each dimension, and the size of the box in a given dimension is invariant across the dataset. Typical box sizes will be 8 by 8 in the X and Y dimensions, and 4 in the Z dimension, reflecting the undersampling of the data in the Z dimension.

3. If the data is viewed as slabs of boxes then the X,Y box origins in each slab are offset from those in the box above by one half a box size in both the X and the Y dimensions. Each box is thus face-connected to four boxes within the slab, four boxes in the slab above and four boxes in the slab below. In general a slab will contain more than one section, typically four.

The Boxes of Voxels Within the Data With the Z-dimension Exploded to Show the Inter Slab Stagger.



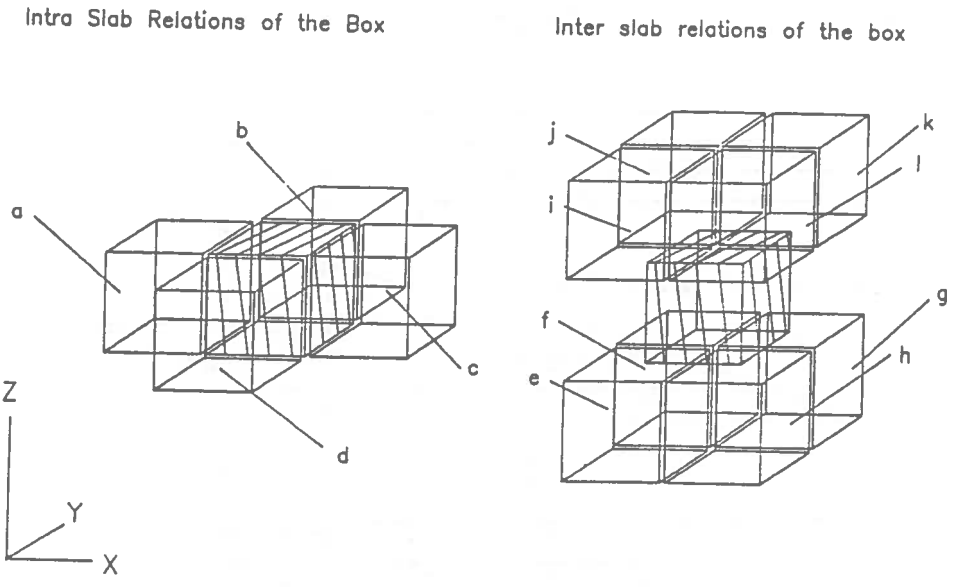
2.4.3. Assignment of Data to Processors

The assignment of processors to boxes within the slab is done so that if any cell of 5 boxes (one central and its 4 face adjacent neighbours) is considered all the boxes are assigned to different processors.

There are two further constraints. Processors which are face-connected in their box assignments within the slab are never neighbours in the physical arrangement of processors (see chapter 3) and the number of boxes assigned to each processor within the slab should be about the same.

Each of the slab of assignments is identical except that there is a one and a half box size stagger in the X and Y dimensions between slabs which are adjacent in the Z dimensions.

The 12 Neighbours of a Box



P.T.O. for key.

a,b,c and d are all assigned to different processors from each other and from the central box.

e,f,g and h are all assigned to different processors from each other and from the central box.

i,j,k and 1 are all assigned to different processors from each other and from the central box.

The net result is that each assignment box is face-adjacent in its own slab to 4 boxes which are neither assigned to the same processor as the box nor to any of that processor's immediate neighbour processors. The box has four neighbours in the slab below each of which is assigned to a different processor, and four neighbours in the slab above each of which is also assigned to a different processor.

The box dimensions are fixed at compile-time (in order to allow array dimensioning in occam) but the overall dimensions of the dataset are input by the user. Once these values are available the assignment of boxes to processors is performed (in parallel to either generation of test data or loading of real NMR data). A list of processor assignments which have not been used is kept. The assignment process randomly cycles through this list, attempting to find an assignment for each box which meets all the constraints. If no suitable assignment can be found then the list of processors available to be assigned is re-filled and the cycling process is re-started. The assignment process does not allow backtracking.

There are at most 20 processors that a assigned box may not be assigned to: the four that its in-slab neighbours are assigned to and each of those processors' four neighbours in the physical array of transputers. The above assignment process is therefore guaranteed to succeed if more than 20 processors are available for assignment. The program was implemented on twenty five processors. In practice the

assignment process tends to succeed unmodified if fewer processors are available. To make sure that it does so the condition of physically connected processors never being assigned to face-connected boxes may be waived if after cycling through the refilled list of available processors, no suitable processor is found.

I would have liked to have implemented a more conventional search process to find the optimal assignment of data to processors given the above constraints and to have attached heuristic forfeits to breaking them. Indeed there is a spare moment during data load up when all 25 image processing transputers are available to perform the search. It is, however, difficult to implement search in occam, especially in parallel so this task was left for future work.

2.4.4. The reasons behind the mapping

The mapping aims to optimise three classes of computation.

1. An operation is to be applied globally across a section within the dataset. (eg a section is to be drawn)

   Each processor is assigned approximately the same number of boxes within each slab, and within each slab it is assigned all of the sections. For each section every processor will therefore hold approximately the same amount of data for the section, and the processing load will be approximately evenly distributed.

2. An operation is to be applied globally across the dataset. (eg the data is to be smoothed)

   Each processor is assigned approximately the same number of boxes within a slab, and the dataset is composed of a set of slabs

therefore each processor will hold approximately the same proportion of the datasets, and again the processing load will be approximately evenly distributed.

3. An operation is to be performed which spreads from a single arbitrary point in the data across any arbitrary surface in the data. (e.g the boundary detection and the surface tracking operations of Nelson [85].)

   Many of the features of the mapping relate to this particular class of operation. Consider a boundary being detected by tracking away from a single point. Initially only one processor, the one assigned to the data at that point is active in detecting the boundary. As soon, however, as a voxel on the edge of the box is detected as being on the boundary the processor is in a position to seed the processor to which the neighbouring box is assigned. It is important at this point that the box of data into which the seed is to be sent is assigned to a different processor from the one that has generated the seed voxel. The processing will then spread to another processors

   The processor that is initially active can seed any of its four neighbouring boxes in the slab, all of which are assigned to different processors. It can also seed the eight boxes which it is face-connected to in the sections above and below it.

In describing the data mapping described above I have for simplicity omitted to explain task-farming component of the task to processor mapping. It was envisaged that space on the transputers would allow five copies of the dataset to be kept: one on each of the neighbours of a nominally "in charge" transputer for each box of the data. Every

transputer could then farm out tasks for boundary detection to its physically connected neighbours in the network of processors without sending the data associated with the task. The returned datastructure including only the boundary detected voxels within that box would be returning only from the physically adjacent processor and thus a large communication overhead would be avoided. The transputers available for the project were only supplied with 256K bytes of memory, so lack of space did not allow the necessary duplication of the data.

2.4.5. A Lisp Model To Assess Parallelisation Efficiency

In the early stages of the project (whilst parallelisation approaches were being developed) a model of the spread of processing across a surface was used in order to test the efficiency of proposed task to processor mappings.

The model considered the spread of processing across the surface of a sphere within a box of data which was assigned to a set of processors according to various assignment methods.

The model considered concentric rings of surface tracking activity spreading out from a point on the surface of the sphere. The rate of movement of the ring of activity was judged to be proportional to the number of processors active in processing: the number whose assignments of data were intersected by the ring. It was also judged to be inversely proportional to the length of the ring. The mean number of processors active in tracking the ring was approximated by averaging the number of processors active in tracking a large set of rings. Each ring was weighted according to a value inversely proportional to the rate of movement of the ring.

The assumptions in the model were numerous, and are discussed in chapter 4. It remains to be seen whether the model does indeed usefully describe the spread of processing.

Two investigations were carried out with the model.

1. In order to show the inadequacy of a simple mapping of processors to data a comparison between

(a) assigning to each of 27 processors 1/27 of the image in a single box.

And

(b) Assigning to each of the processors 8 spatially separated boxes of data, together comprising 1/27 of the image.

In each case the processing was to track the surface of the largest sphere which could be wholly included in the box of data.

The results showed that when a single box of data was assigned to each processor on average only 29% of processors were active in tracking the sphere. When each processor's assignment of data was split into 8 sites on average 49% were active in tracking the sphere.

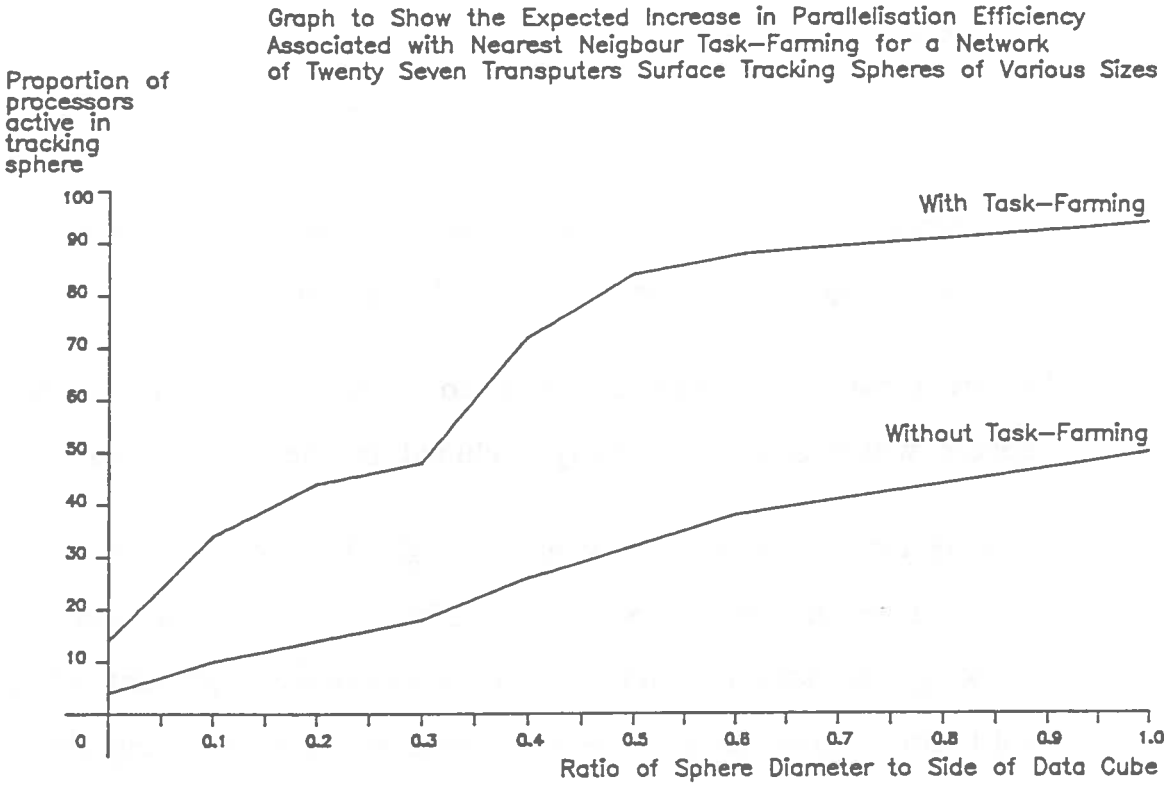
2. Once the image to processor assignment as described above had been implemented in occam, a sample assignment was tested within the Lisp model.

Again 27 processors were assumed, each of which was assigned 8 spatially separated boxes. Two cases were compared.

(a) Each processor was only able to process data which had been explicitly assigned to it.

(b) Each processor was allowed to process data which had been explicitly assigned either to it or to one of its immediate neighbours in a physical arrangement of processors devised according to the rules given in chapter 3. This corresponded to the task-farming described above.

The average processing load of the network was determined for tracking the surface of spheres of various sizes centred within a cubic dataset.



The task-farming can be seen to increase the processor loading by a factor of two to three for a wide range of sizes of spheres. For large spheres the loading can approach 100%.

2.5. Accessing the Distributed Dataset

The distribution of the data into blocks and its assignment to different

processors means that the dataset is distributed, both notionally and also in spatial terms. The division of the dataset causes programming problems since a given processor may only directly access the data which it has in its own memory.

When the Zucker Hummel surface detection operator is applied to a voxel it is necessary to reference all of that voxel's twenty six neighbours. To perform the Zucker Hummel operation on voxels which are at the edge of a block of data as defined by the assignment process requires access to voxels which are over the edge of the block. There are two solutions to the problem: either the processor performing the Zucker Hummel operation must request the value of voxels from that processor to which the face-connected block of data has been assigned, or each block must be sent out with an extra voxel-width layer of data all the way around it. The former solution requires communication, the latter solution requires more data storage on each processor since data is duplicated. In the case of blocks of data which are 8 by 8 by 4 voxels (as was used) a block of 10 by 10 by 6 must be stored. 600 voxels must be stored in place of 256. Despite the increased memory requirement, the latter solution was used.

Another problem is that after the assignment process has occurred, and the data has been distributed it is necessary for each processor to know something about how blocks other than its own are distributed. One solution would be for each processor to have a record of the assignment of each block of data. Given the above block sizes, and 128 by 128 by 12 datasets each processor would need to store 768 integer values for the addresses of the blocks, or 3K bytes of data. The approach I used was to send out with each block of data the

addresses of the processors to which the six blocks face-connected to it had been assigned.

2.6. Conclusions on Dividing the Dataset

As the above discussion would suggest, the programming overheads involved in dividing the datasets between processors are large. The task would be simplified if all of the data could be stored on each processor. An assignment of tasks to processors would still be required, along the lines of the current assignment of data to processors, but the efficiency of the local task-farming algorithm might be increased if all the processors in the network, rather than merely 4 immediate neighbours, were available for the task to be farmed to. There would, however, be a problem in assuring that the flags which made up the surface were consistent across the network of processors.

The problem of devising an efficient distribution of tasks relates mainly to the surface tracking operators. It might be possible to implement a similar process which was less sequential in nature, and which therefore mapped easier onto the parallel architecture.

Chapter 3

Processor Connection and Communication

3.1. Goals

Chapter 2 discussed the communicating process model of computation, and the way in which computation was mapped onto processors. Little stress was put on the need for inter-processor communication, but clearly each processor requires such communications to receive data and tasks to process, and also to transmit its results. In this Section I discuss the way in which the processors were connected in a network so as to allow them to communicate.

The goals set for the network were:

1. to connect the n processors in such away as to make it possible for any processor to send a signal to any other that it may require to communicate with.

2. to minimise the average distance such a signal has to travel.

In addition it was necessary for the network EITHER

3. a) to arrange for the signals to be transmitted and interpreted with optimal speed given that any of the processors en route may be busy performing their own processing.

   b) to handle the received signal given that the recipient itself may be busy.

OR

4. to arrange for synchrony within the network so that signals are passed during allotted parts of a processing cycle.

1 and 2 will be discussed in section 3.2 and then 3, and 4 in section 3.3

3.2. Connecting Processors

3.2.1. Literature Review

The problem is stated by Yao-Nan Lien [86] in the following way

> There is a problem of partitioning the problem to be solved into many subtasks and mapping these tasks to the real processors. The major concern is to balance the processor utilisation and to minimise the possible communication among processors.

In order to minimise the communication within a network of processors a connection scheme where each processor is connected to each other processor by as few links as possible is often advocated. Such networks are often viewed as graphs and the network problem is then the converse of a famous mathematical puzzle first set by Moore (circa 1958) and stated by J.Holm [85] in the following way (I have changed his symbols to conform to those which it is more usual to employ)

> Consider undirected, connected graphs where the degree of any vertex does not exceed Δ and there is a path traversing at most D vertices. For fixed constants Δ what is the maximum

number of vertices contained by such graphs ?

Moore's original bound to the number of vertices is given below

D (diameter of graph)	Moore's Bound	Greatest Number of Vertices Found
1	5	5
2	17	15
3	53	40
4	161	95

Optimal Delta D Graph Sizes for 4—Connected Nodes
And the Best Graphs that Have Been Found
After Bermonde & Delorme (87)

Unfortunately it may be seen from the above that graphs with as many nodes as Moore's Bound predicts have not been found even for small D. Furthermore it may be shown by higher algebra (Biggs [74]) that such graphs are not achievable for $\Delta = 4$ and $D > 1$.

The architecture of the Meiko Computing Surface allows arbitrary interconnection of processors. Within the constraint that $\Delta = 4$ The hardware does not force the user's network to conform to any particular family of connected graphs, so in theory any of the best-found-so-far graphs that are collected by Bermonde, Delorme and Quisquater [86] may be mapped onto the hardware, providing sufficient wiring resource is available (Meiko [86]).

There are many reasons for not using any of those arrangements. First it is not clear how to generalise such a graph to n processors. The obvious approach is to take an optimal network with a larger number of nodes and to remove nodes from it. The removal of a node from an optimal graph containing n processors leaves eight 'sticky' links (to use a biochemical analogy). Any four pairs of links can be connected together to complete the network, however it is not clear that the network resulting is the best that can be achieved for n-1 processors.

Indeed in regarding the design of networks as equivalent to Moore's problem we are merely asking how many processors may be connected within D links of each other. We are not addressing the problem of connecting arbitrary numbers of processors within the shortest distance of each other.

Second, if one is intending to minimise interprocessor link distances, it is the average interprocessor distance that should be minimised, not the maximum interprocessor distance. Moreover the network design problem is more complex than this simplistic argument would suggest since it is how the network behaves in a given program that counts. As a result of the structure of a program each processor in a network may spend much of its time communicating to a small set of other processes and so the graph should be optimised for the minimum average signal transfer distance, rather than the minimum average inter-processor distance.

Third, the graphs of Moore et al are fully connected. A system based on such a graph could never receive user input or indeed input or output to/from any external agent since it has no spare links.

Inclusion of the user, for example, as a node in the network is not possible since users are rarely 4-connected.

Finally, it makes sense to modularise the program so that some processors are running one part and others are running another part. In terms of the approaches to task mapping described in Section 2 this corresponds to task parallelism at a modular level. In the context of the program it was envisaged that image processing and graphics would be separated into modules and that the image processing part of the program would have to communicate to a graphics pre-processor with a fixed small number of input channels.

If it can be arranged that the communication among modules in the program is bandwidth limited than there is no visible speed increase to be gained by ultra-efficient networks within those modules.

The following constraints are put upon the network arrangement for the image processing part of the program.

1. There is one link in and out of the network to and from the user.

2. There is one link out of the network to the filing system.

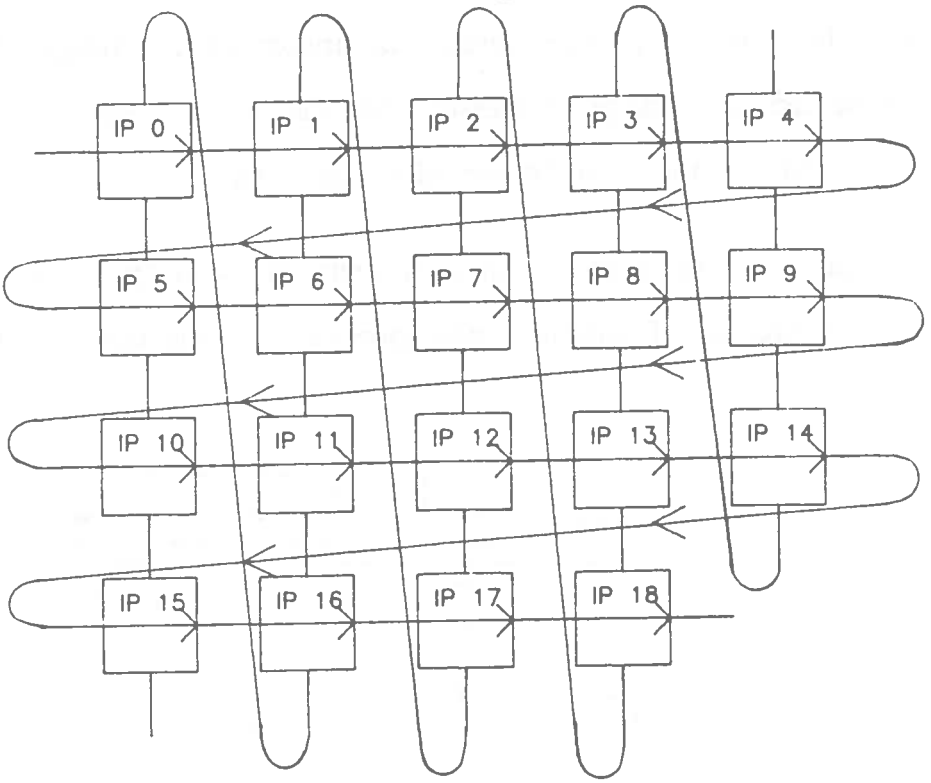
3. There are links out to the graphics pre-processor (the number chosen was 4).

4. It is most important that the network should generalise according to some relatively simple algorithm to n available image processing transputers.

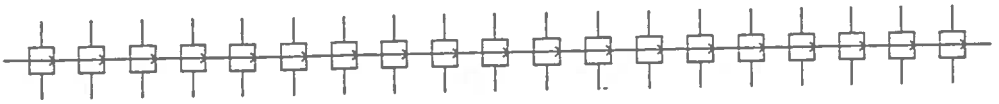
3.2.2. The Chosen Arrangement

For a network of n processors the arrangement is a helical toroid of 1 by m rectangularly connected nodes. Alternatively it may be looked at as a pair of interlaced chains of connections The values of m are chosen in the following way: they are the smallest pair of integers such that $1 \times m \geq n$ and $|1 - m| \leq 1$.

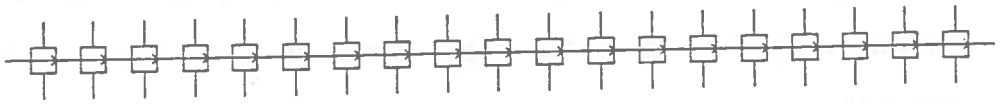
At the corners of the network (or the ends of the chains) the links are left free to communicate with other modules of the program. If $1 \times m > n$ then the last row of the minor axis of the toroid is left incomplete.



Chain Along Long Axis



Chain Along Short Axis



MgN 87

The diagram shows the network for nineteen processors numbered from 0 to 18 The processors may both receive and transmit on each of their four connections. The arrows outline a chain of inter-processor links which is drawn above as the "chain along the long axis". The "chain along the short axis" of the toroid is also shown. Using the notation as above, for n=19, l=4 and m=5. l*m=20 thus

the last row of the toroid is incomplete. Image Processor 3 is connected to Image processor 14 instead of the non-existent Image Processor 19 in order to complete the loop.

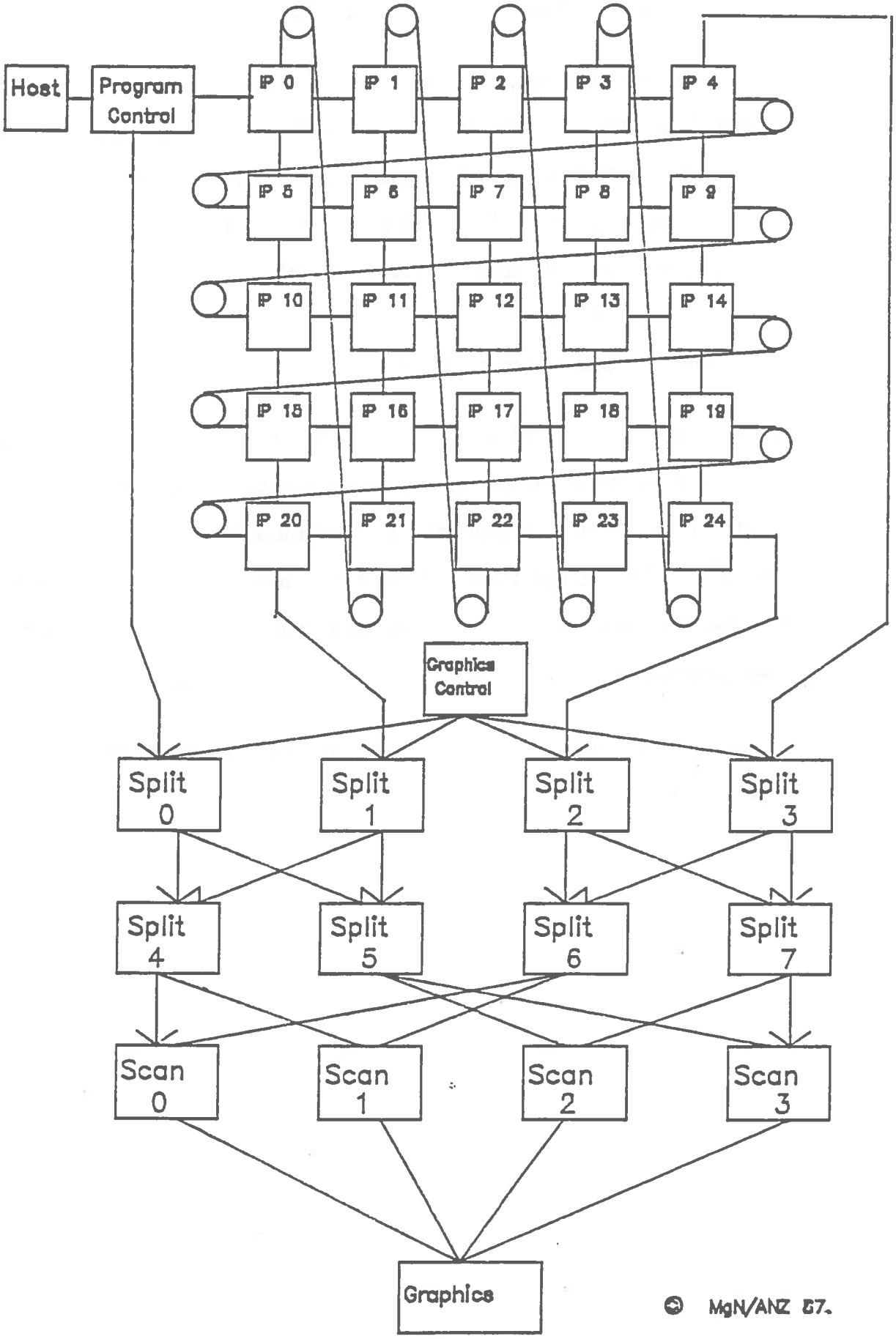
It may be seen that the arrangement has 4 output connections for any n. It has the following inter-processor connection lengths for various n.

Processors	Major Axis	Minor Axis	Maximum Inter-Processor Distance	Average Inter-Processor Distance
49	7	7	8	3.5
25	5	5	5	2.4
20	5	4	4	2.2
19	5	4	4	2.1
12	4	3	3	1.7
9	3	3	3	1.4

Inter-Processor Connection Statistics
For Various Numbers Of Processors

To complete the description of the arrangement of processors it remains only to add in the other parts of the program: The program controller and the graphics processing transputers (Zentner [87]).

The Arrangement of Processors for the Bodyscan Program



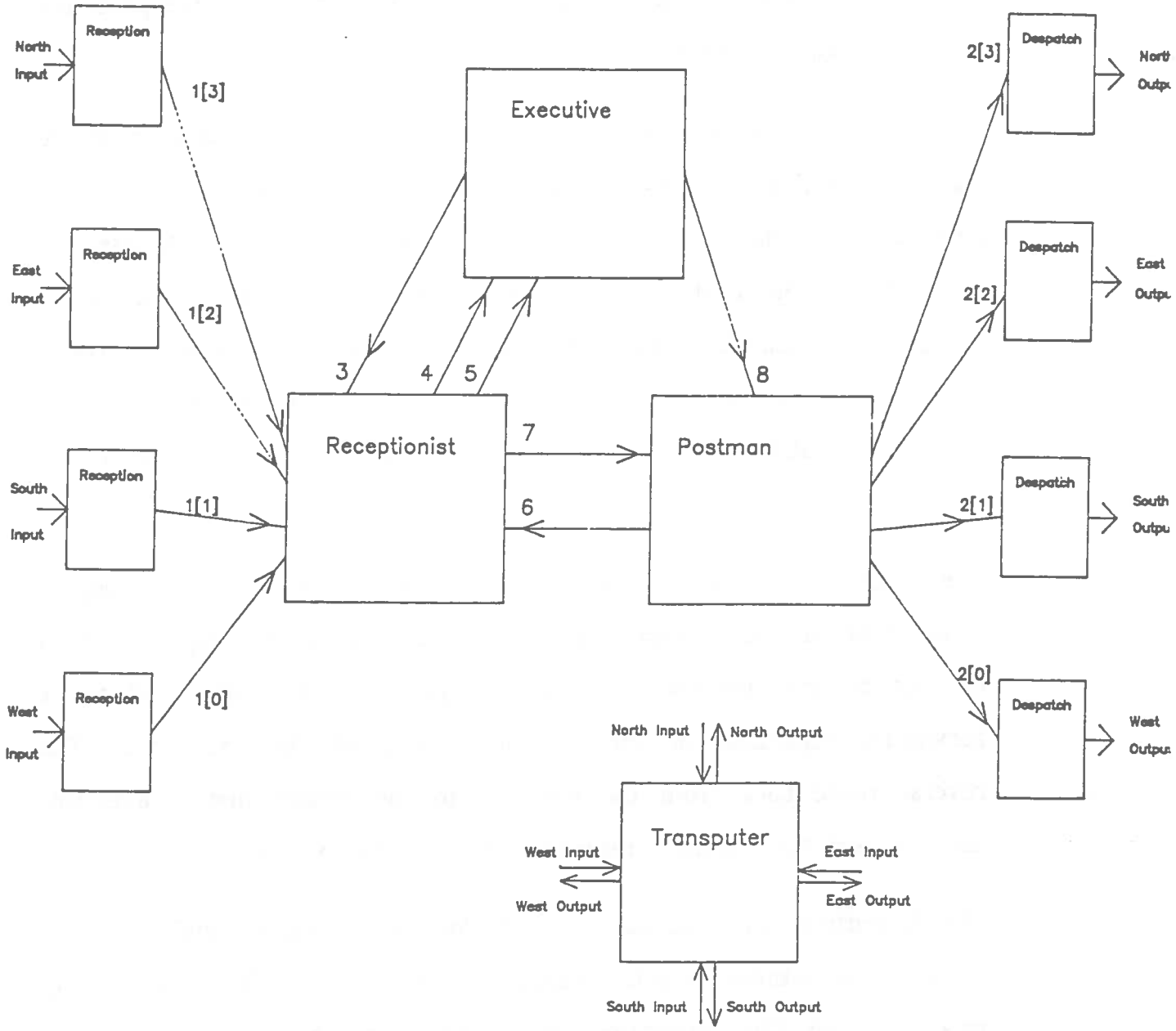
3.3. Signals and Signal Handling

3.3.1. Processes Internal to Each Image Processor.

Occam has the facility to allow many processes to run in parallel on a single physical processor. The transputer invisibly time-slices between them, and sets up the secure communication between processes along virtual channels. In addition to easing prototyping, the use of parallel processes to handle communications and image processing allows an abstraction from the detailed mechanics of the order in which communication events happen.

To separate communication from image processing, each image processor runs three major processes in parallel. The unit is analogous to an office, and consists of a Receptionist, a Postman and an Executive. The Receptionist processes of incoming signals, the Postman forwards signals to other transputers and the Executive performs image processing.

Virtual channels are set up to allow communication between the three major processes.



Key:

```
1)  [4] CHAN reception.to.receptionist:
2)  [4] CHAN postman.to.despatch:
3)  CHAN executive.to.receptionist:
4)  CHAN receptionist.to.executive.DATA:
5)  CHAN receptionist.to.executive.COMMAND:
6)  CHAN postman.to.receptionist:
7)  CHAN receptionist.to.postman:
8)  CHAN executive.to.postman:
```

The Receptionist communicates to the Executive through two channels: the receptionist.to.executive.DATA channel which is used to transmit signals containing data, and the receptionist.to.executive.COMMAND

channel which is used to transmit signals containing tasks for the Executive to perform and is also used to interrupt Executive processes which do not terminate.

The Executive communicates back to the Receptionist through a single channel, executive.to.receptionist which is only used under circumstances when the Receptionist has directly requested its use. If both the Receptionist to Executive channel, and the Executive to Receptionist channels were freely used then deadlock situations could occur where the Receptionist was attempting to communicate with the Executive whilst the Executive was attempting to communicate to the Receptionist.

The Receptionist communicates directly with the Postman along a single channel, receptionist.to.postman. This route allows signals which are not destined for the processor to bypass the Executive and to be forwarded regardless of the current activity of the Executive. The reverse route back from the Postman to the Receptionist is available but unused for deadlock reasons (see immediately above).

The Executive communicates to the Postman along a single channel, executive.to.postman which constitutes the output from the image processing on the transputer to all other processors. There is no channel from the Postman back to the Executive.

Finally there are eight processes - four Receptions and four Despatches - which run in parallel to the main three and act as software buffers between the communications processes and the channels. They allow the transmission and receipt on all input channels simultaneously since transfers along virtual channels can be executed orders of magnitude faster than transfers along physical inter-processor links, and the

physical channels are hardware buffered.

3.3.2. Packets and Addresses

All processors within the image processing network may generate signals to be transmitted to any other processor. In addition signals must be passed in and out of the network. To accomplish this without the aid of global control, the signal must contain within it the address of the processor to which it is to be sent, and some indicator of the way in which it is to be interpreted upon arrival.

In the solution adopted, each processor is given a unique address and each type of signal is given a unique flag code. Flags are listed in an appendix. The signals are packeted, the length of the packet being variable and transmitted in advance of the packet. The first two bytes of the packet contain a 16 bit value, the address of the packet. The next two bytes contain two flags indicating how the packet is to be interpreted.

Each of the image processors is given a number along the major axis of the helical toroid, processor 0 being at the notional top corner, and connected to the program controller. Destinations outside the network are given negative identifiers, -1 for the controller and -2 for the nearest link to the graphics pre-processor. The links into the graphics pre-processor may also be individually addressed, although this has not been found to be necessary.

The routing of the packets is carried out locally at each image processor which decides

(a) If the packet has reached its destination

(b) If not, which of the links to send it out of so as to get it closer to its destination.

To get from any processor to any other on the network may be achieved in many ways, some of which will be shorter than others. For many inter-processor transfers (particularly over long distances in large networks) there are more than one different equally short ways of accomplishing the transfer. In an attempt to utilise the full signal bandwidth of the network the following routing procedures are used

1. At start up each processor searches from its position in the network using a limited depth-first branch and bound search algorithm until it has found and stored the initial send-out direction(s) for the shortest route(s) to every processor in the network.

2. Packets which have addresses inside the network are sent to their destinations by sending out of the physical channel in one of the directions which will allow a shortest route. Subsequent packets to that destination are sent via any alternative output channel which allows a shortest route, and so on. Once all possible shortest-route output channels have been used the next packet is sent out through the first in the list again.

3. Packets which have addresses outside the network are treated as destined for the corners of the network at which output channels are situated. At these corners they are sent out through the appropriate output channel.

3.3.3. Queues and Packet Handling

The packet handling process is similar in some ways to that suggested and implemented (on SIMD hardware) by Jesshope [87]. The essential components are the separation of processing and communication, and the queuing of packets received by the communication process for subsequent processing.

Jesshope's system requires a global control of every processor. the Meiko hardware has no such facility, and so an asynchronous communications network has been built.

Details are given below.

Each processor maintains three queues of packets. First there is the command queue: a list of commands that the executive is to execute in sequence. Second the interesting queue: The executive may request the receptionist to pass packets out of the queue when it requires them.

It was envisaged that this queue would be required during stages of the program where every processor was generating large numbers of data packets whose destinations were inside the network. The interesting queue would then work as a sponge soaking up packets and removing any possibility of deadlock. The particular stage of the program for which the interesting queue was intended was the farm-out of data to neighbouring processors at initialisation. This has not been implemented so the interesting queue is unused.

Finally the strange data queue, which is an attempt to allow for variations in the exact data transmission rates in the network. If the data associated with a command is received before the command itself (as is possible given that although they may be coming from

the same processor they may follow different routes), it is important that the data is stored until its associated command arrives.

The sequence of events when a packet is received by the receptionist is as follows.

1. The packet is received by the Receptionist via one of its receptions.

2. The status of the packet is analysed:

   (a) If the packet is an interrupt, it is sent to the executive via the COMMAND channel.

   (b) If the packet flag is in the list of those packets currently required as soon as possible by the executive, it is sent directly to the executive via the DATA channel.

   (c) If the packet flag is in the list of those packets currently considered to be interesting by the executive, it is placed in the interesting queue.

   (d) If the packet is a command to be executed by the Executive, it is placed in the command queue.

   (e) If the packet is in none of the above categories it is held in a queue for strange packets.

3. The packet is received by the Executive from the Receptionist.

   The events occurring at this point depend upon the nature of the packet, and the current state of the executive.

   (a) If a termination signal is received while a procedure which requires continuous data input is running on the Executive (such as the initial load-up process) then that procedure is terminated.

(b) If a packet is received which is required as soon as possible by the Executive then the processing of that packet is determined by the procedure which has requested it.

(c) Similarly to the above, procedures running in the executive may request packets from the receptionist's interesting queue, and the processing of them will be procedure specific.

(d) If the packet is a command then that command is executed.

The receipt of a command by the Executive can only occur after a handshake between the Receptionist and the Executive. The Executive notifies the Receptionist that it is no-longer busy and the Receptionist sends on to the Executive the next packet in the command queue (if any). The Executive then sends back the lists of those packets which it now considers interesting, and those packets it requires as soon as possible so as to be able to perform the command it has received. The Receptionist then shuffles its queues as described below.

4. If a command has been transmitted on to the Executive the Receptionist shuffles its queues.

(a) The interesting queue is searched for those packets which are now required as soon as possible by the Executive which are forwarded. Those packets in the interesting queue which are still interesting are retained in the queue, others are discarded.

(b) The queue of strange packets is checked similarly, those which are required being sent to the executive, those which are interesting being passed into the interesting queue, and those which are still strange being discarded.

3.4. Assessment of the Network

The original goals of the network have, at least in part been satisfied. The following list corresponds to that given at the beginning of the chapter.

1. The routing of signals both within the network and to processors outside is accomplished by packet headers and re-direction at intervening processors.

2. The network is not optimised so as to minimise the distance signals have to travel, but it is not clear as to how such an optimisation might be accomplished. The network is certainly not the worst possible, and satisfies the constraints put upon it by the rest of the program.

3. (a) Passing of signals between processors which are actively image processing is accomplished by separating the processes of communication from those of calculation, and communicating between them by virtual channels.

   (b) The packet queueing and storing procedures performed by the receptionist allow the received packet to be processed even though the Executive may be busy image processing.

4. Network synchrony has been rejected for reasons given above.

Chapter 4

Image Processing

4.1. The Image

Techniques such as nuclear magnetic resonance, (NMR) or X-ray computerised tomography produce two-dimensional image sections through the body. Each of these sections is made up of rectangularly prismatic volumes or voxels arranged in a planar array. The intensity of the image at a given voxel is calculated from the intensity of projections taken from different angles around the body. The intensity represents the calculated value of some property of the tissue at the volume of space corresponding to the volume of the voxel in the image.

For NMR data there are two measured sets of intensity values: the proton density data, a rough measure of the amount of water present in the tissue, and the Difference data, a value related to the rate of spin re-reversal of the nucleus of the proton after being subjected to an initial reversal by an electromagnetic pulse. A third dataset may be calculated from these two, for example the T1 dataset used for cardiac data.

The particular data used for testing the program was that obtained by Nelson in 1985. The formula used to calculated the T1 data was as given in his thesis (Nelson [85]).

$$T1 = \frac{200.0}{\ln(\text{Proton Density} / \text{Difference})}$$

4.2. A non-interpolated version of the Zucker-Hummel operator

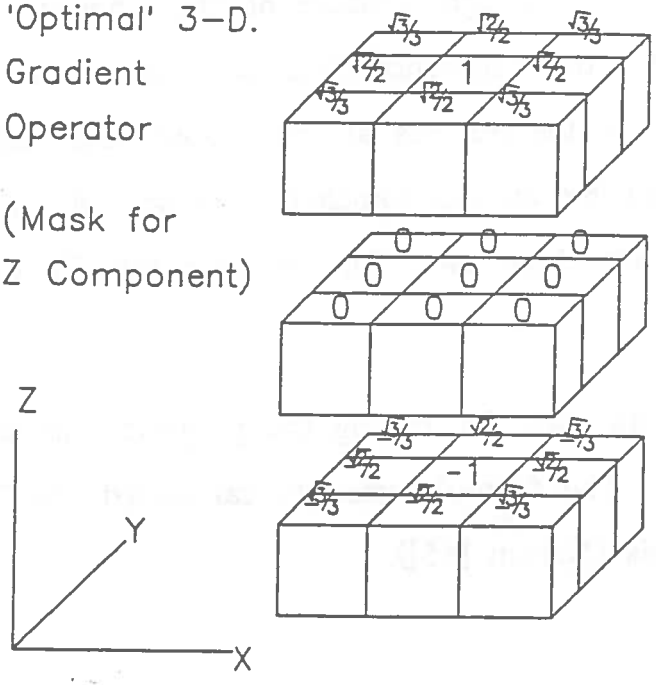
Datasets generated by NMR imaging techniques have voxels which are non-cubical. The length of the side of the voxel in the Z dimension is considerably longer than the corresponding length in the other two dimensions. The voxel may be considered to be a cubic prism.

The Zucker-Hummel 'optimal' gradient operator, (Zucker and Hummel [81]) as used by Nelson is a three dimensional generalisation of the two dimensional Sobel operator. It defines for every voxel the best orientation of surface plane to match the data. It also gives the intensity of the edge at the point in the data.

The data is convoluted with three operators, one in each dimension. The operator is given below for the X/Y plane in the case of perfectly cubic voxels.

The Zucker Hummel 'Optimal' 3–D. Gradient Operator

(Mask for Z Component)



Nelson's approach to the problem of non-cubic voxels was to define a set of cubic voxels such that every section was divided into five interpolated sections each of which was assigned the same value as the original section of which it was a part.

It was impracticable for this approach to be used on the parallelised version because of the memory limits of the system. Furthermore it is not a justifiable approach since the contribution of the Z dimension to the overall surface gradient calculated by the Zucker-Hummel operator varies greatly between the interpolated sections within the original section. The central interpolated sections within the original section have identical interpolated sections above and below them. The Z component of the surface gradient is therefore non-existent. Interpolated sections which are at the edges of the original section have an identical interpolated section on one side, and a different one on the other side, and so the of the Z component of the gradient for these sections is roughly half the value of the Y and X components.

As a result, the overall magnitude of the surface gradient, and indeed the calculated direction of the surface gradient vary greatly between adjacent interpolated sections. The criteria used for surface tracking are dependent on the direction of the surface normal calculated from the components to the gradient, and so the interpolation approach used by Nelson is clearly inapplicable.

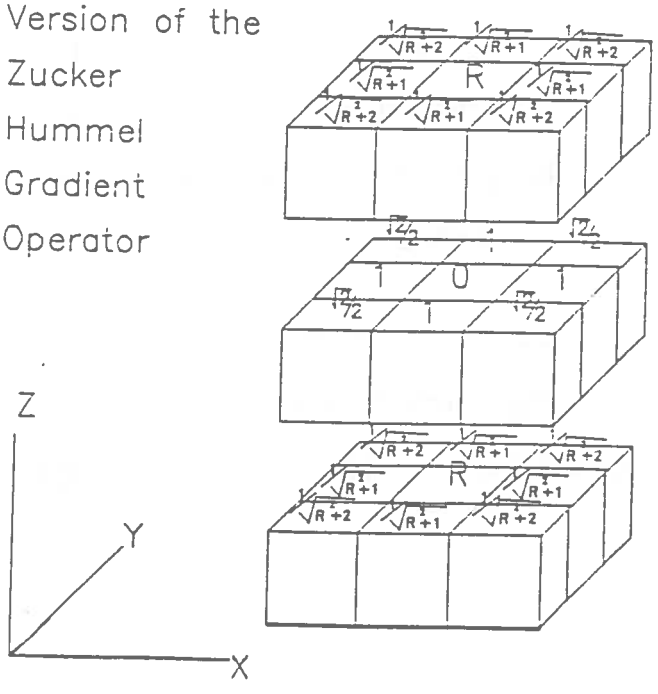
As a response to the problems of interpolated sections, and to the impossibility of storing them in the way that Nelson did, an attempt was made to generalise the Zucker-Hummel operator to non-cubic voxels.

The Zucker-Hummel operator is a local gradient operator, the values of the weighting factors of the voxel's neighbours which are used in the convolution may be seen to vary as the distance of the neighbour from the voxel. A face-connected voxel is weighted at 1, an edge connected voxel at $1/\sqrt{2}$ and a corner connected voxel at $1/\sqrt{3}$

The weighting may be seen to be exactly dependent on the distance of the centre of the neighbour from the centre of the voxel. One approach to generalising the operator to a non-square metric in all dimensions is to correct the weighting factors so as to take into account the variable distance between voxel centres.

Whenever voxels in the section are used in the convolution their weightings are as originally suggested by Zucker and Hummel. When voxels are in sections above and below, their weightings are dependent on the value of the ratio between the X/Y side-length of the voxel, and the Z side-length. This value is set as a parameter by the user at initial data load up time, and is referred to in the diagram below as R. The block of weightings that is used is shown below.

A Non-Square Metric Version of the Zucker Hummel Gradient Operator



The convolution masks for each of the X,Y and Z dimensions are made using the weights as above. For a dimension D, the voxels of the plane which shares the same D coordinate as the voxel are given zero weighting. The voxels of the plane which has a D coordinate one greater than the D coordinate of the voxel are given positive weightings as in the above diagram, and those whose D coordinate is one less than that of the voxel are given weightings which are the negative of the values in the above diagram.

One of the reasons that Nelson gives for using the interpolated data is that it allows the display of cubic section polygons for three dimensional projection. In fact it is quite possible to draw polygons which are sections through non-cubic rectangular prisms, although I have not been able to complete the three dimensional display for this project.

A more usual interpolation method than the one that Nelson used is merely a linear interpolation between sections. Nelson suggests that this would blur edges in the third dimension. I would suggest that the edge detection operator I have used is equivalent to unmodified edge detection with linearly interpolated sections. The local-ness of the Zucker-Hummel operator requires that data which comes from a large distance away is considered as less important. The contribution of the Z dimension to the value and direction of the local gradient is therefore decreased, which is equivalent to the blurring of the linear interpolation.

The linearly interpolated data is often used by radiologists and it would not be hard for it to be generated as an extension to the system.

4.3. An improved surface tracking algorithm

The surface tracking operation aims to generate a representation of the surfaces of objects within the body to allow three dimensional projection of those objects. The user specifies a point which is considered to be on the surface of the object and the surface is tracked away from that point according to various criteria.

According to Nelson [85] the properties used to qualify a voxel as belonging to the boundary are:

(a) Connectivity with previous boundary elements.

(b) High boundary contrasts.

(c) Similarity of contrast across neighbouring elements.

(d) Consistency of the surface normal.

The boundary detection algorithm of Nelson is a sequential process driven by a queue of boundary voxels. Voxels which are neighbours of a voxel which is already found to be on the boundary are tested for inclusion on the boundary according to the above criteria. The initial boundary voxel is specified by the user.

The exact method used by Nelson is as follows. The first boundary voxel on the queue is de-queued, its six face-adjacent neighbours are then considered for inclusion on the boundary. If a neighbour has previously been marked as visited then it is skipped, otherwise it is marked as visited. The surface normal of the voxel is then compared with the value of an average of surface normals of voxels previously found on the boundary. If the normal exceeds a given fraction of the average of surface normals found so far then it passes on to the

final test.

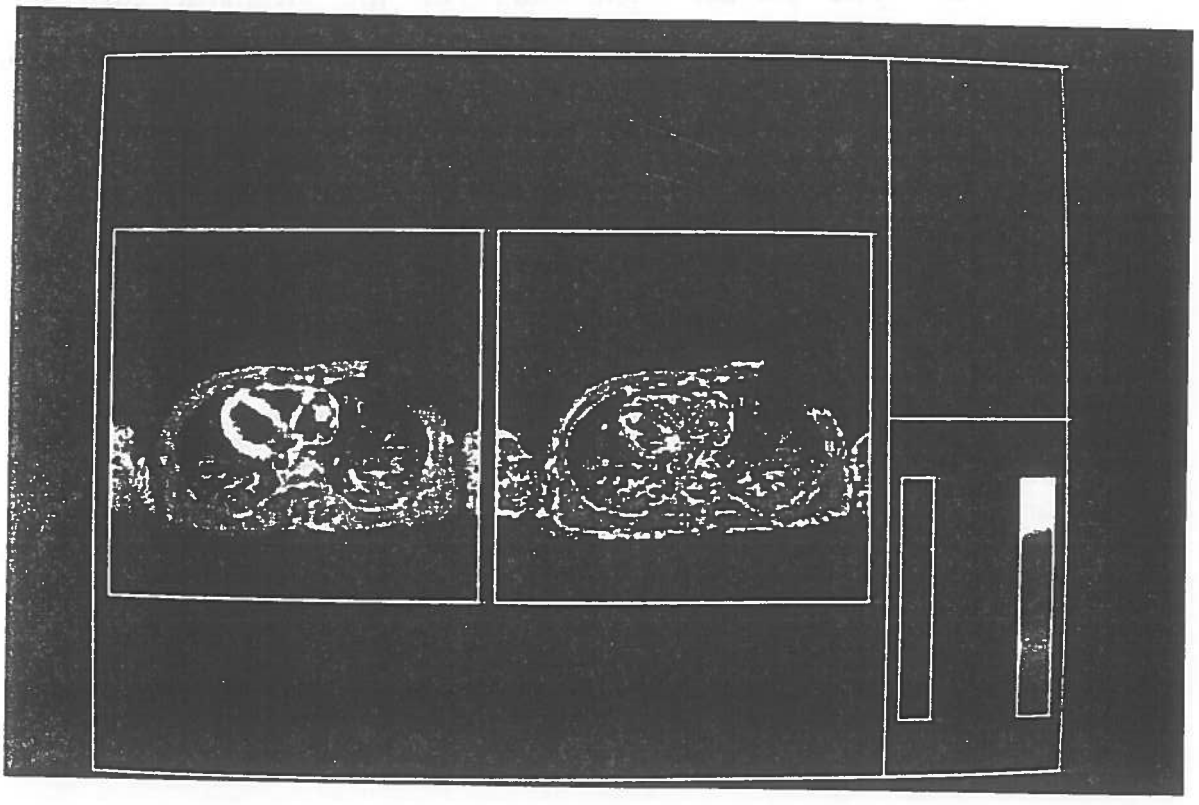
The angle between the surface normal of the neighbour and the surface normal of the voxel is compared (actually the cosine of this angle). Neighbours whose surface normals are wildly deviant from the surface normals of their neighbours are rejected. Only those voxels which pass all three tests are included in the boundary.

The method was modified by Reeve to include voxels which were edge-connected for consideration for inclusion in the boundary.

The modifications I made to the surface tracking algorithm fall into two categories: The parallelisation modifications and the fundamental modifications.

4.3.1. The Changes to the Algorithm

Consider the heart shown below.



A cross section through the thorax in Difference data. The edges within the data are shown displayed side by side with the image.

The surface tracking algorithm outlined above suffers from a fundamental weakness when tracking around surfaces of objects in complex surroundings. It may be seen that the neighbouring structures to the heart vary from lung through to body wall. The intensity of the surface of the heart as measured by the edge operator will vary depending upon the tissue with which the heart is contiguous. The strength of the edge is not a good indication of the continuity of the surface.

Given that the edge tracking attempts to find continuous closed surfaces, a more useful measure of surface continuity is a comparison between the value of the voxel internal to the voxel being considered, with a local average intensity value of the tissue internal to the surface. One problem associated with such a comparison is determining the voxel which is internal to the surface. This may be accomplished by asking the user to specify a voxel on the inside of the surface to start with. The surface then starts at the strongest edge in the vicinity of the voxel, and the value of the local average of internal intensity is initialised at the intensity of the initial voxel. The direction corresponding to internal is also initialised.

The continuity of the direction associated with internalness may be maintained by using a technique similar to that which assures that only face voxels are drawn by Nelson's program.

Unfortunately this enhancement has not been implemented.

4.3.2. The Parallelisation of the Algorithm

The division of the dataset into blocks has been discussed in chapter 2. The problem arises as to what to do at the edges of a block of data during surface tracking. If the surface tracking reaches a voxel which is over the edge of a block then this voxel must be seeded to another processor, the one on which that voxel is resident.

Nelson's algorithm was implemented as a single process communicating with a neighbour seeding process in parallel. Whenever the tracking algorithm reaches a voxel at the edge of the block of data that the processor has been assigned, the voxel is passed to the neighbour seeding process.

The seeding process holds lists of voxels for each of the six neighbouring blocks of data. The voxel is added to the list which relates to the block of data which it its primary home When each of these lists becomes full it is sent as a list of seed points to the processor associated with that block. The exceptions to this rule are as follows:

First the initial seed point for each of the six neighbouring blocks is sent out as a single element in a list of seed points. This is to activate as many processors as soon as possible. Second when the surface tracking process has finished tracking within the block of data, the seed point lists are sent out whether or not they are full.

4.3.3. The Lisp Model Versus The Implementation

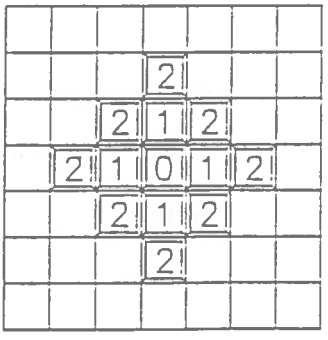
At this point I should like to demonstrate the correspondence between my model of processing and the surface tracking algorithm

implemented by Nelson and Reeve. The model gives an insight into the surface tracking algorithm and, importantly into how to parallelise it.

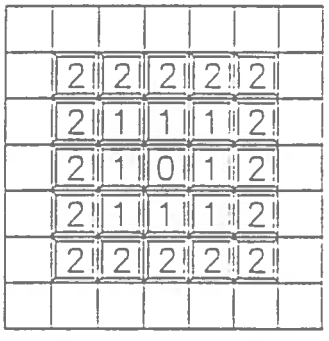
Consider a flat surface within the dataset. An original seed point is specified by the user in the centre of the surface. The surface is then tracked away from the seed point according to a) face connected rules and b) edge and face connected rules. For each voxel those neighbours in the surface which qualify are put into the queue of voxels to be processes. The queue is processed sequentially. The resulting spread of detected surface is shown below. The search may be seen to be breadth first and the generations of the search are as indicated on the diagrams.

Surface Tracking from a Point on a Flat Surface
Generations of a Breadth First Search Process

a) Face–Connected Rules



b) Face And Edge Connected Rules



The time taken to move between generations is as follows. Values are in terms of units of voxels to be processed.

	Time To Move Between Generations		
	0-1	1-2	2-3
Face Connected Rules:	1	4	8
Face and Edge Connected Rules:	1	8	16

For my model I assume the time to process a voxel is independent of whether its neighbours have been visited. This was not true of Nelson and Reeve's program, but is closer for mine since I pre-calculate the Zucker-Hummel coefficients of the neighbouring voxels.

If the planes shown above are spread across the surface of a sphere it may be seen that the generations approximate to circular rings on the surface of the sphere being tracked. This approximation was used in the model. The time taken to process a ring depends on how many voxels are in it, or on its circumference. Rings are processed sequentially, as in the model.

The model is parallelised in the following way. The number of processors which have been assigned to at least one block on the ring is calculated and the time taken to process the ring is determined by dividing the circumference of the ring by the number of processors active in processing it.

The major approximation of this model of parallelisation is that processing is assumed to be shared equally between those processors which are capable of processing. In fact where a processor is active at more than one place along the circumference of the ring it cannot

be as active at each location as a processor which is active at only one location along the ring. The result must be for the growth of the ring to be retarded at one or more places along the ring.

Counter-intuitively this will probably increase the parallelisation efficiency since the distorted ring is longer than a circle would be, and on average will pass through blocks assigned to more processors.

Task farming is modelled by allowing any of the physical neighbours of a processor to become active, if they are not already processing data along the ring. The grand total of processors active is then kept, and the speed of movement of the ring is dependent on it. Intuitively it seems possible that task farming may approximately normalise the effects of having a processor active at more than one location along the ring.

There remains the question of the time-resolution of task-farming. Are tasks farmed out on a block by block basis, a voxel by voxel basis, or what? The model assumes that task-farming is a process with continuous (rather than discrete) time resolution since processing is shared equally between processors which are able to be active on the ring. A continuous time-resolution is precluded by the division of the data into voxels. Since the aim of the task-farming is to keep as many processors active as possible, whilst not producing excessive communication overheads, the time-resolution of the farming process is important. I have not been able to come to any conclusions on how to do it since it has not been possible to implement task-farming in the program.

There is also the problem of maintaining the continuity of the flags which are held on each of the processors involved in the task

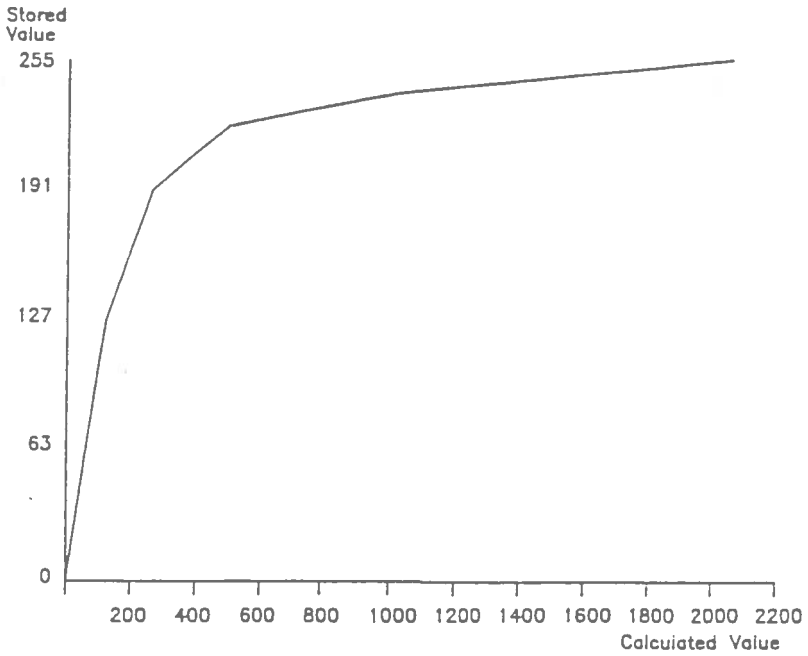
farming. The overheads involved in this might well negate the apparent advantage of task-farming outlined in chapter 2.

4.4. The Display of Edge-Images

The program calculates and stores the values of the Zucker-Hummel edge detection operator for the whole dataset. The calculation takes about ten seconds for a 128 by 128 by 12 image, and occurs at a stage when the user is unlikely to require any other processing to be performed. The values of the surface normals in each of the directions are normalised with respect to the overall magnitude of the edge vector, and stored scaled up to signed byte format. The magnitude of the surface normal vector is also stored, but it too is normalised, although in a different way.

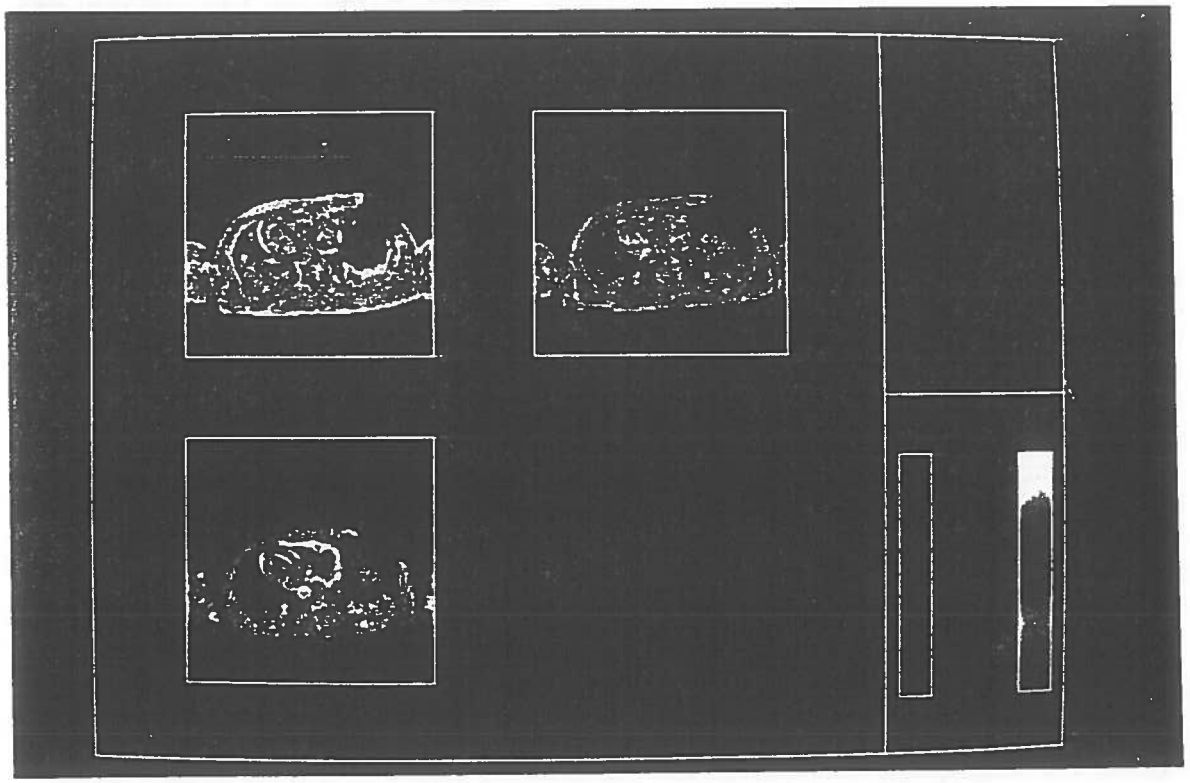
The global calculation of the Zucker-Hummel coefficients allows the display of the value of the magnitude of the surface normal for any point in the image, but the dynamic range of the value is such that a purely linear scaling does not show much detail in the edge-image.

The surface tracking algorithm requires access to the magnitude of the surface normal to decide whether candidate points are on the surface, but the discrimination threshold used in the test tends to be a reasonably low value. The value of the magnitude of the surface normal in real NMR images tends to vary between 0 and about 1000. It was decided to keep the scaling factor of the magnitude of the surface normal into its stored value unity over the range 0 - 128. This is the range over which the surface tracking algorithm requires to discriminate between surface normal values. The graph below shows the whole scaling.



The Scaling of Calculated Zucker Hummel Magnitude
Into the Stored 8 bitt Value

The following picture shows the sort of images which are produced by the display of the Zucker-Hummel normal. From top right to bottom right in a cyclic order they are Proton Density, Difference data and T1.



4.5. A new Datastructure for Boundaries

Nelson implemented the Boundary datastructure as a list of voxels which were on the surface. To intersect or union surfaces required set operations to be carried out on the two lists. Such operations require order of n squared calculations. I implement the Boundary datastructure as a vector of flags. The position of the flag associated with a voxel is directly related to the position of that voxel in the vector of data stored by the processor to which it has been assigned. To intersect or union two such datastructures requires a global bitwise compare and assignment on the integers which make up the bit vectors of the datastructure. The operation is simple, fast and works well in a parallel assignment of data to processors. It requires order of n operations. The achieved speed up for this operation has been about five orders of magnitude, as outlined in the introduction, section 1.6.

Chapter 5

Conclusion

5.1. A Summary of the Facilities Provided

At this point it is useful to give a summary of what the program actually does. The complete set of facilities available to the user are described in the user manual in Appendix 2, but below there is a summary. The program will display two dimensional sections of medical NMR data, it will calculate the T1 derived dataset, and apply conservative smoothing to the data. It will perform global calculation and normalisation of the Zucker Hummel edge detection operator coefficients. For all points in the image it will display the magnitude of the surface in a direction normal to the surface, scaled using a compressive non-linearity as described in chapter 4. The displays are in the same section based format as the two dimensional data displays, and may be displayed simultaneously with them.

The program will apply a simple thresholding operator to the intensity of the data, and also to the magnitude of the Zucker Hummel edge operator. Only voxels whose values lie between the upper and lower bounds of the threshold are displayed. The operations is useful for segmenting the image into regions.

Boundaries in the data may not be successfully tracked, nor may three dimensional projections of surfaces be drawn. Datastructures corresponding to surfaces, but generated by the thresholding procedures

may be manipulated with the bitwise set operations.

5.2. Some Conclusions about Parallel 3-D Image Processing

To efficiently parallelise low level global image processing operations given the constraints put upon the program which are described in Chapter two three it was only practicable to use a data determined division of processing. Task or algorithmic parallelism would have been inefficient since large numbers of processors were available, and task farming would have been inefficient because the processing would have required a large amount of data to be passed around in the farming process, with the associated overheads.

Surface tracking is an operation which is basically sequential in nature. (Nelson [85]) The efficient implementation of this operation on a parallel machine requires a division of data between processors which is more complex than the simple sheet or crinkled mappings generally used for SIMD low level image processing. (Reddaway [87]) The approached used was to divide the data into physically contiguous blocks and to assign each of the blocks to a processor. Each processor was given more than one block, and each block contained the voxels which nominally extended over the edge of the block to allow the Zucker Hummel operator to be calculated without requiring communication between the processor and those to which the neighbouring blocks of voxels were assigned.

The assignment of blocks to processors was done in such a way as to allow a high degree of efficiency of parallelisation of the surface track operation. It was intended to implement local task farming within the network of transputers used for image processing, and the

assignment algorithm reflected this. During early stages of the surface tracking operation the processing occurs in a small section of the image. The process is analogous to a breadth first search which spreads away from a point on the surface, finding all the other points which are on that surface. It is necessary for processing to spread as soon as possible to other processors if the algorithm is to be efficiently parallelised.

The algorithm which was used to generate the assignment of data to image processors aims to maximise the rate of spread of processing among the processors. It staggers the block assignments within the image so as to allow each of the blocks to have twelve neighbours. It was written under the assumption that processing of a block of data could be shared among five processors, each of which had a copy of the data, and four of which were connected via a physical channel around the fifth. The algorithm aims to keep blocks of data which are close together assigned to processors which are far apart in the network. A model of the processing showed high levels of processor utilisation for the network with such a local task-farming process in place. It was not possible to implement the task-farming because of lack of space on the transputers.

5.3. Some benchmarks

The Following timings were produced for twelve sections of 64 by 64 sections. The version of the graphics pre-processor (Zentner [87]) used for timings was only running on a single processor.

Action	Old Time (sec)	New Time (sec)
Surface voxel detection	130	***
Surface face tracking	40	***
2D section display	60	2
3D shaded surface display	150	***
Image editing (clip)	20	<< 1
Image editing (peel)	45	<< 1
Image editing (intersect)	10	<< 1

5.4. An Assessment of the Computing Surface

The facilities which was used for developing the program were: a Meiko M40 running the Occam Programming System OPS2; a Meiko multi-user machine running Multiops (the Meiko multi-user operating system currently under development); and various IBM PC compatible machines with INMOS B004 boards running the TDS2 transputer development system. The facilities belonged to the Edinburgh Concurrent Supercomputer Project. The problems with the system have been many:

1. The facilities were under development and continuously changing.

2. The Occam Programming System contains several inconsistencies, and bugs. In several parts the software is counter-intuitive. File input output is tortuous, and debugging facilities are poor and themselves bugged.

3. Occam is a low level language. The compiler for the transputer is bugged, some of these bugs are documented, but the most obscure and time-wasting ones are not.

4. There is no consensus as to how to produce efficient and effective occam programs. Indeed there are very limited subroutine libraries: an equivalent of the Executive Receptionist Postman facilities I developed and which are explained in chapter 3 should be available as system code.

On the other hand the potential for implementing three dimensional image processing on parallel hardware is very good, and I should like to list some positive aspects of a Computing Surface approach.

1. The system performs very fast, and there is possibility of considerable further improvement in processing speed by adding more processors. The efficiency must in part be attributed to the efficient implementation of occam code on the transputer.

2. Since there is no necessity in the occam language to specify which processor a process runs on, it is simple to develop a program on a small machine which will scale up easily on to one with many processors. As a suggestion for future users of the facilities, the PC AT compatibles provide a reliable, self-contained and stable environment for program development and it is advisable to continue developing code on them for as long as possible.

3. The staff of the Edinburgh Concurrent Supercomputer Project, and indeed Meiko employees are very helpful in solving system-related problems.

5.5. Suggestions for Improvement

The achievements of the project have been mainly in achieving efficient parallelisation, in the connection of processors and in the arrangement of communication. Unfortunately it has not been possible to achieve the image processing and displays which the previous program of Nelson & Reeve could.

Simply improving the system to incorporate these facilities is probably the first and simplest suggestion for improvement. To achieve boundary detection would have required merely a few days, assuming the bugs in the system code could be fixed to allow proper debugging. To get a program incorporating all the features available in the previous program, with suitable modifications, would require about

two months development on a reliable computing surface, assuming the existence of suitable three dimensional graphics utilities.

The next set of improvements I should like to suggest relate to the user interface. There has been increasing interest over the last few years in bodyscan data manipulation (New Scientist [87])

The consensus among the image processing community is that three dimensional surface display is a useful goal for bodyscan data manipulation programs. Whether this statement is justified is rather dependent on the attitude of the medical profession to the technique. It is important for a bodyscan data manipulation program to address the needs of three groups of clinical personnel: The Clincians, both medical and surgical, Radiologists, and those who have to use it, the Radiographers. The technology, in the long term, must gain the acceptance of the medical profession.

The structure of the present menu driven software is closely matched to the procedures being executed by the program. It would be useful to have a comprehensive study of the potential for human-computer interaction in the field of bodyscan data manipulation and presentation. This study should be carried out at a level which is abstracted from the details of what has been achieved in the programs that have been developed over the last three years. Once this has been performed a proper user interface might be written. Perhaps this might be suitable for another MSc project.

On a more specific level, there are a list of improvements which might be made to the new bodyscanner program:

1. Produce a shell of the existing program, incorporating only the

networking and communications, and benchmark the communications within it. Modify the shell so as to improve communication speeds and the secure-ness of the protocols. It is possible to deadlock the network during the data load-up phase by sending commands in from the program controller, and I have once managed it at another stage of the program.

2. Implement the task-farming for which the algorithm used to assign data to processors was written (given processors with more memory to do it with).

3. Improve the algorithm used to assign data to processors to allow a backtracking heuristic based search process to be performed during the data load up phase.

4. Give clinicians the option of displaying the linear interpolations between sections that they often use.

5. Draw two dimensional displays in planes other than the plane of the section.

In the tradition of my predecessors Nelson and Reeve I shall also suggest the general modification of the code I have written to improve its efficiency.

References

Bermonde, Delorme and Quisquater [86]
Bermond,J.-C, Delorme C , and Quisquater J.-J,
Stratgies for Interconnection Networks: Some Methods from Graph Theory,
Journal of Parallel and Distributed Computing 3, pp 433-449 1986.

Biggs [74] Biggs N.
Algebraic Graph Theory Cambridge Tracts in Math. No. 67.
Cambridge University Press, London, 1974.

Jesshope [87] Jesshope C.,
A Dynamic Load-Balanced, Active-Data model of Parallel Processing for Vision
British Computer Society Parallel Processing Specialist Group,
Parallel Architectures and Computer Vision Workshop. 1987.

Holm [85]
Holm J.,

INMOS [87]
Transputer Reference Manual
INMOS Limited 1987.

May [87]
May, D. "Occam 2 Language Definition" in
A Tutorial Introduction To Occam Programming (Pountain).
INMOS Limited 1987.

Meiko [86]
The Computing Surface Reference Manual
Meiko Limited 1986.

Meiko [87]
GFX user manual
Meiko Limited 1987.

Morrow and Perrot [87]
Morrow, P.J, Perrot, R.H,
An Investigation Of Low Level Image Processing Algorithms On A Transputer Network
British Computer Society Parallel Processing Specialist Group,
Parallel Architectures and Computer Vision Workshop. 1987.

Nelson [85]
Nelson, A,
Body Scan Data Surface Detection And Presentation
MSc Dissertation, Department of Artificial Intelligence, University of Edinburgh 1985.

New Scientist [87]

Reddaway [87]
Reddaway, S,
"Mapping Images onto Processor Array Hardware"
British Computer Society Parallel Processing Specialist Group,
Parallel Architectures and Computer Vision Workshop. 1987.

Reeve [86]
Reeve, D.P,
Walsh and Canny Based Surface Detection in Body Scan Data MSc Dissertation, Department of Artificial Intelligence, University of Edinburgh 1986.

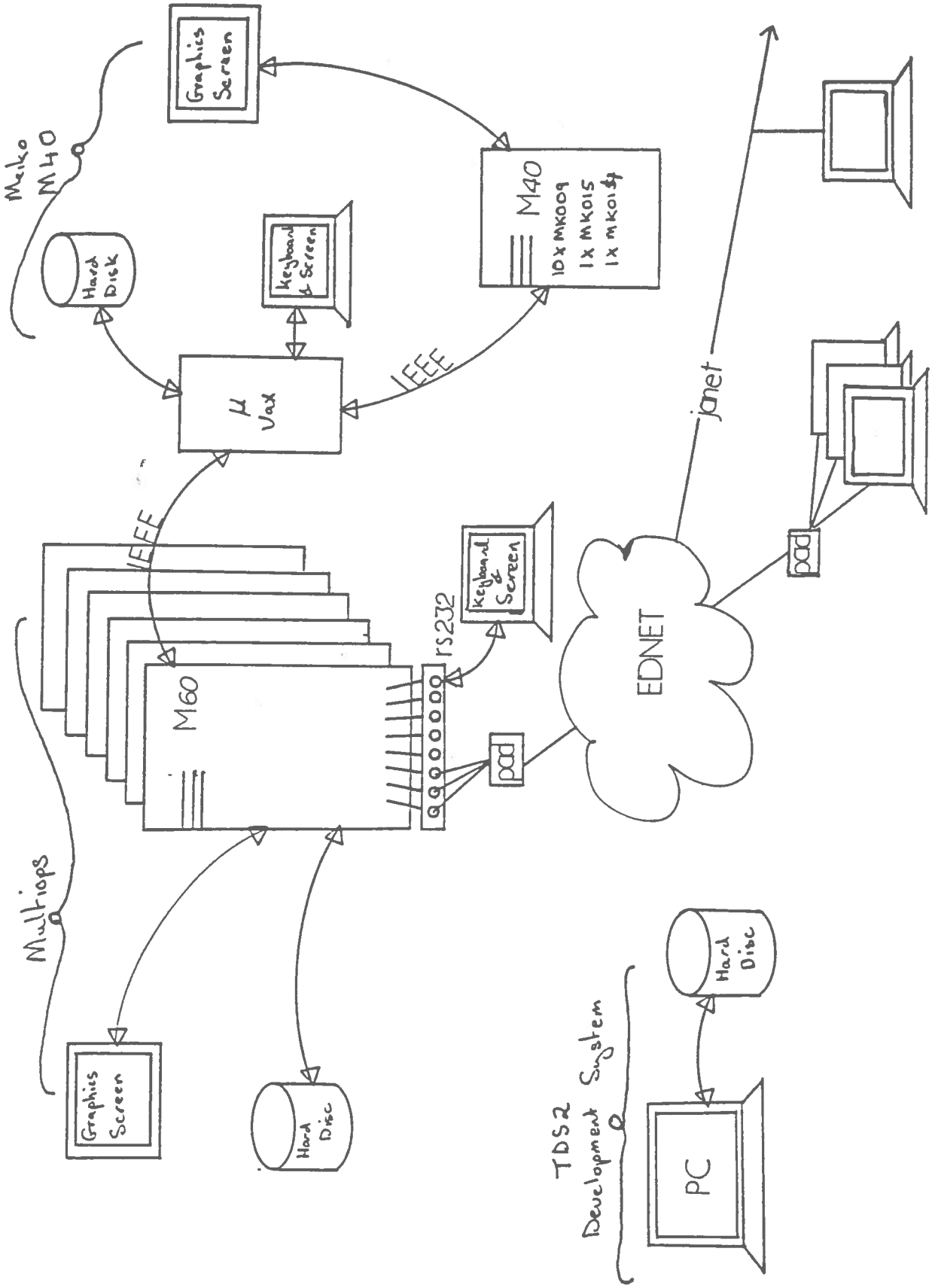
Theoharis [86]
Theoharis, T,
Exploiting Parallelism in the Graphics Pipeline Oxford University Press, 1986.

Yao-Nan Lien [86]

Zentner [87]
Zentner, A,
Parallel Front End for Processing 3D 'Bodyscan' Images on the Meiko Computing Surface.
MSc Dissertation, Department of Computer Science, University of Edinburgh 1987.

Zucker and Hummel [81]
Zucker, S.W. and Hummel, R.A. "An Optimal Three Dimensional Edge-Operator",
IEEE Transactions on Pattern Analysis and Machine Intelligence Vol PAMI-3 pp324-331. 1981.

Appendix 1: The current Edinburgh Concurrent Supercomputer Facility (30/9/87)



acknowledgements to
C. Foster (Meiko Ltd)

Contents of Appendix 2

APPENDIX 2:

User Manual For Bodyscan Data Manipulation Program.

This program was written as part of the research portion of of an MSc course in Information technology by:.

Michael Norman (Knowledge Based Systems)
Adam Zentner (Computer Systems Engineering)

Submission date 30/9/87

This version is currently under development and is available for demonstration use only under the condition that all bugs found in it and its documentation are notified to Mike Norman (mgn@ed.edai).

Any enquiries concerning the commercial application of this program should be addressed to Dr. Robert Fisher, Dept of Artificial Intelligence, Forrest Hill Edinburgh. (email: rbf@ed.edai).

References to Norman [87] are to the MSc thesis:

Implementation of Medical 3D Image Processing
on MIMD Parallel Hardware.

M. Norman.
Department of Artificial Intelligence.
University of Edinburgh.

1. Introduction

The program is an initial study to show how three dimensional bodyscan data data can be manipulated within the parallel architecture of the Meiko Computing Surface.

Over the last two years a bodyscan data manipulation program has been developed within the Department of Artificial Intelligence at the University of Edinburgh. It runs on conventional (SUN 2) hardware and suffers from the speed limitations of that machine. A new program has been written in order to harness the speed of the Computing Surface. The program uses the full power of data parallelism and is extendable to any number of processors. It is written in occam around the standard Meiko system utilities.

The present version of the system has only one Transputer for graphics pre-processing. It is envisaged that the number will be extended to thirteen. At present the system draws pictures fifty times as fast as the SUN 2 program, and performs data processing five hundred times as fast.

Bodyscanner data comes in three main types: Computerised Tomography or Computerised Axial Tomography (CT or CAT) is an X-Ray based three dimensional imaging system. The pictures may be produced in real time and are rarely subjected to significant image processing; Positron Emission Tomography (PET) is a relatively new imaging technique whose clinical usefulness is currently under investigation; Nuclear Magnetic Resonance or Magnetic Resonance Imaging (NMR or MRI) is a well known and useful imaging technique, currently in use in most Area Health Authorities in the U.K.

The program is set up to analyse MRI data, although with a little modification CT, and PET could be used. Indeed there is no reason why

any type of tomographic data might not be analysed by the system.

MRI data is of the form of sections through the body or part of the body. The spatial resolution of the data is lower in the axial direction (normal to the plane of the section) than in the plane of the section. Typically datasets have 128*128 or 256*256 pixels in the section, and there are 10 - 20 sections.

The MRI machines produce two datasets, Proton Density which is effectively a measure of the amount of water present in the tissue being imaged, and Difference, a rough measure of the relaxation time of the proton nucleus after reversal of its spin by an electromagnetic pulse. From the two datasets others may be calculated, for example T1 and t2 which are used for cardiac imaging.

An important feature of MRI, and one reason that it is preferable in many cases to X-ray based imaging techniques is that it images soft tissue. For example our demonstration images of thorax show heart, lungs and the mediastinal vessels clearly and without artefacts from the overlying bone. Images of thorax do show an artefactual 'shadow' asssociated with the wire used to synchronise the MRI pulses with the ECG of the patient. This will explain the rather alarming hole in the lung wall of the patient in the thorax images used for demonstrations.

The achievements of the program have been in showing the possibility of three dimensional data manipulation within a parallel processing environment. The operations which may be carried out on the data are not as wide-ranging as those which were implemented on the SUN, although it is hoped that in time they will be extended. The operations available include:

Drawing of the sections and calculation of the derived datasets.

Thresholding of the data.

A three dimensional Zucker Hummel edge detection operation and display of the value of the gradient calculated.

Tracking of edges detected by the Zucker Hummel operator.

AND-ing and OR-ing etc. of surfaces and thresholds.

2. Starting up the Program

The instructions below assume a basic knowledge of the occam Programming System (OPS) as implemented on the Computing Surface. For further details refer to the Meiko Computing Surface Reference Manual.

The program lives in the directory [import.users.ecru13.project] Enter OPS on the M40 and give import/users/ecru13/project as the OPS root directory.

Once inside the top level fold, get the electronic router , and run it. Reply n to the default and type r bodyscan.wir to read the wiring diagram used by the program. Type p to place the wiring, and q to exit the router.

Get the configurer and the terminal emulator, and load the program fold "bodyscan.tsr", or alternatively the cpr file associated with the program. Finally run the program.

3. A demonstration

Below is a sample run of the program in which a display is generated of the heart, separated from other structures in the thorax. The display clearly shows both the inner and the outer walls of the ventricles.

Options to be selected on menus are shown in the following format:

```
<option number to be selected> OPTION <accompanying text>.
```

The user must input only the option number to be selected.

Answers to be given to questions are shown in the following format:

```
<answer to question> <text of question>.
```

The user must input only the answer to the question.

The user input values are thus present in a column indented by one tab from the left hand margin of this document.

——————————oOo——————————

First load up 12 sections of real data from thorax.dat.

```
2   OPTION   Use real NMR data.
12           Pixels in Z direction.
6            Ratio of X/Y pixel size to thickness of section.
```

It may take several minutes to read in the data from the Microvax host. The exact time depends on the level of usage of the multiuser file server. Once the data is loaded, change the T1 calculation parameters.

```
4   OPTION   Change T1 Calculation Parameters.
60           Low Threshold For P.Density.
100          Low Threshold For T1.
700          High Threshold For T1.
```

Then calculate the T1 data from the two measured datasets already on display.

```
3   OPTION   Calculate And Display T1.
```

In the T1 dataset the blood in the left ventricle is yellow and clearly differentiated from the thick muscular wall. The contrast is good for cardiac data in T1, the resolution is however necessarily poorer than in the two directly measured datasets.

Now that T1 is calculated, move on to the next menu.

```
1   OPTION   Accept the calculated dataset.
```

Change section to show a section slightly lower down the thorax, and set a low threshold at 30 to isolate the thorax from artefactual noise in the background of the dataset.

```
4   OPTION   Draw Another Section.
```

```
6            Which Section.
5  OPTION    Threshold The Data.
30           Lower Threshold.
255          Upper Threshold.
```

Next draw the three datasets alongside their edge-detected versions. The Zucker Hummel may be looked upon as a three dimensional spatial frequency filter which lets through only the high spatial frequencies associated with edges. Edges are thus seen as yellow, high intensity, and areas where there are no edges are blue. The thresholding out of the background applies to the Zucker-Hummelled data too in as much as only those areas which have data values between the previously set thresholds are displayed as edge-images. The inner walls of the ventricles are easily seen as strong edges.

```
7  OPTION    Draw Both The Image And Its Edge Detected Version.
```

Now move on using Difference data, all the other data is wiped.

```
2  OPTION    Move on Using Difference Data.
```

Finally produce a line-drawing of the heart, showing the internal structure of the atria and ventricles by thresholding and dataset intersections.

```
6  OPTION    Threshold the data.
60           Lower Threshold.
255          Upper Threshold.
8  OPTION    Store.
13 OPTION    Threshold The Detected Edges.
80           Lower Threshold.
255          Upper Threshold.
8  OPTION    Store.
7  OPTION    Compare Or Combine.
2  OPTION    Intersection (And).
1            First Surface.
2            Second Surface.
8  OPTION    Store.
```

In the resulting line-drawing the heart is isolated from its background. Sections 4 to 9 show cross-sections from inferior to superior (bottom to top if the patient were standing upright). Inferior sections show left ventricle. Right ventricle becomes visible at about section 5, right atrium at about 6, and left atrium at about section 7. Section 7 also shows the entry of left atrium into left ventricle.

4. The Menus

4.1. Menu 1: Getting started

Option.	Action.
1. Use Fabricated Data	Generate a 3D 'egg' of data for Proton Density and Difference and display it.
2. Use real NMR data	Load up to 12 sections of Proton density and Difference data from the file import/users/ecru13/project.
3. Use test data	Generate a noisy surface to be tracked by the surface tracking algorithm, to test the of the tracking process. (Not Yet Implemented).

4.2. Calculating T1 or T2

Option	Action.
1. Accept the calculated dataset	Move on to the next menu, the acceptance of the dataset is not final – this menu may be revisited once left.
2. Draw another section.	Display another of the available sections.
3. Calculate and display T1.	Calculate the T1 data corresponding to the proton density and Difference data already on display, and display in the third window on the screen. Use the current values of T1 parameters, or default values if they have not been set.
4. Change T1 calculation parameters.	Change the parameters used to calculate T1. This does not perform the calculation.
5. Calculate and display T2.	Calculate T2 according to the current set of T2 calculation parameters, in a similar way to the T1 calculation as above. (Not Yet Implemented).
6. Change T2 calculation parameters.	Change the parameters used to calculate T2. This does not perform the calculation. (Not Yet Implemented).
7. Return to previous menu.	Return to initial data load–up menu. (Not Yet Implemented).

Note that the first picture drawn is either raw data as input from thorax.dat or raw calculated egg data. For subsequent pictures the datasets have been subjected to conservative smoothing.

4.3. Manipulation of 3 datasets & dataset selection

Option	Action.
1. Move on using Proton Density data.	Draw the Proton Density display, scaled up to fill the screen. This option also shuffles the data so as to remove data that is not Proton Density, and calculates the Zucker Hummel edge operator values for every point in every section of the Proton Density data, storing the values for later reference.
2. Move on using Difference data.	As for Proton Density above, using the Difference data.
3. Move on using T1 data.	As for Proton Density above, using the previously calculated T1 data.
4. Draw another section.	Display another of the available sections.
5. Threshold the data.	Mask out all data except that between upper and lower thresholds set by the user, Masked data appears as black.
6. Draw the image after edge-detection (Zucker Hummel) .	Show the magnitude of the Zucker Hummel edge operator at every point in the image instead of the actual value of the data at that point. The operator is a high spatial frequency filter, and thus edges are enhanced. The values are scaled into the available palette using a compressive non-linearity. (Norman [87] ch. 4)

7.	Draw both the image and its edge–detected version.	Show the edge magnitude and the original data side by side.
8.	Restore original un–thresholded data.	Return to showing only the he actual data if either of the two previous options have been selected.
9.	Return to the previous menu.	Return to the T1 menu

4.4. Manipulation of a single dataset

1. Draw a 3.D display.

   Display a surface as a three dimensional projection. (Not Yet Implemented)

2. Detect a surface.

   Perform surface tracking from a user specified point on the image. The surface will be displayed as a black line in the section on display.

3. Change the surface detection parameters.

   Change the parameters used by the surface tracking algorithm.

4. Draw another section.

   Display another section of the data set being used.

5. Change detected surface on display.

   The threshold and surface track operators produce notional surfaces. The data on display is ANDed with these surfaces at all times. By default the display is anded with the Working Surface which is initially set universally true. Surfaces may be stored by using option 8 on tis menu, and recalled using this option.

6. Threshold the data.

   Mask out all data except that between upper and lower thresholds set by the user, Masked data appears as black.

7. Compare or combine two stored surfaces.

   Surfaces which have been stored with the option below may be combined using the bitwise operators AND, OR, XOR, etc. In addition a single stored surface may be inverted. See the menu below for details.

8. Store the Working Surface	Store the working surface, only four surfaces may be stored at present, they are used sequentially until all have been used, then the user is prompted for one to overwrite.
9. Return to showing unprocessed data.	Return to showing unprocessed data if edge detection images are being displayed. If not, this option has no effect.
10. Draw the image after edge detection (Zucker Hummel)	Show the magnitude of the Zucker Hummel edge operator at every point in the image instead of the actual value of the data at that point. The operator is a high spatial frequency filter, and thus edges are enhanced. The values are scaled into the available palette using a compressive non–linearity. (Norman [87] ch. 4)
11. Draw both the image and its edge–detected version.	Show the edge magnitude and the original data side by side.
12. Wipe the Working Surface, restoring the original data.	Remove the thresholding or surface detection flags on the dataset and return to showing the original data.
13. Threshold the detected edges.	Mask out all data except those points where the magnitude of the Zucker Hummel edge operator lies between upper and lower thresholds set by the user, Masked data appears as black.

Note that initially the graphics screen may not respond to the menu commands. The program is calculating Zucker Hummel coefficients for the whole of the selected dataset for the first few seconds that the menu is active. All commands that the menu received will eventually be executed in sequence.

4.5. Bitwise Compare menu

Option

1. Invert a dataset (NOT)
2. Intersection of two datasets (AND)
3. Union of two datasets(OR)
4. Inverse intersection of two datasets (NAND)
5. Difference between two datasets (XOR)
6. I did not want to do this anyway

Inverting a dataset reverses it so that, for example, data which has been thresholded out becomes thresholded in. For surfaces, data in the surface becomes visible, while points not on the surface become black. The other operators allow logical point-by-point combination of two surfaces. In both cases surfaces must have been stored in registers. The result is stored in the Working Surface.

"I did not want to do this anyway" exits the menu without executing anything.

5. The Questions

5.1. Data setup questions

1. Pixels in x direction ?

2. Pixels in y direction ?

The dimensions of a section must be specified by the user. The maximum allowable value for the X and Y coordinates is 128 the minimum being 16. If small datasets are used not all 25 image processing Transputers are active. Large datasets require large initial data generation times. This is done sequentially on a single Transputer, so if egg data is required, it may be necessary to wait.

3. Ratio of X/Y pixel size to thickness of section (1 - 1000) ?

NMR data typically has much lower resolution in the axial (Z) dimension than in the other two (X and Y). The degree of this undersampling must be given by the user both for real NMR data and also for the generated egg. TRY 6 - it's about right for the real data.

4. Pixels in z direction ?

The number of sections, or pixels in the Z direction is given by the user. If using fabricated data only this number of sections are generated, and the egg is centralised in the sections. If using real data only this number of sections are read from the data file. If more sections are specified than are present in the data file an error message is output, but the program will continue, using as many sections as were present.

5.2. Section and surface questions

1. Which section, numbered from 1 ?

The above question is asked whenever the user attempts to change section. Only when the returned value is within the number of data sets read/generated is the user allowed to proceed. The section referred to is a cross section in the XY plane of the data set. The current section on display is changeable using options on menus 2 to 4. A section remains on display until changed. In particular although edge detections and thresholding occur on the whole dataset, their effects on the current section only are displayed.

2. Which surface numbered from 1, surface 0 is Working.Section ?

3. You have exhausted all the available surface buffers Please give a surface you are prepared to overwrite if you wish NOT to continue you may enter 0

For a description of what a surface is, see Norman [87] ch 4. These questions refer to storage and recall of surfaces. At present only three surfaces are available to the user in addition to thee Working Surface. Note that when a surface other than the Working Surface is being viewed that surface remains on display until a threshold, surface track or bitwise compare operation is performed, when the result (stored in the Working Surface) is displayed.

5.3. Bitwise operator questions

1. Give a surface you wish to be manipulated ?

2. Give a second surface you wish to be manipulated ?

The surface to be manipulated must previously have been stored. The result is put into the Working Surface. It too may then be stored. (The NOT operator only requires specification of one surface.)

5.4. Threshold Questions

1. Give lower threshold, 0 -> 255, (-1 to quit) ?

2. Give upper threshold, lower threshold -> 255, (-1 to quit) ?

Both the thresholding of data and the thresholding of edge require two values. The first value is the lower threshold, the value below which data/edge will be ignored. The second value is the upper threshold, the value above which data/edge will be ignored. When displaying real NMR data a lower threshold of about 30 will remove the noisy background of a dataset without removing the tissue being imaged. To remove all but interesting edges try thresholding the edge magnitude at 60.

5.5. T1 calculation questions

1. Low threshold for P.Density (20 - 100) default was 30 ?

2. Low threshold for T1 (50 - 200) default was 100 ?

3. High threshold for T1 (400 - 1000) default was 612 ?

The Proton Density threshold is a value below which T1 will not be calculated (to avoid spuriously high values of T1 where the percentage Difference between log Proton Density and log Difference is high merely because the values themselves are low) The result in these areas for T1 will be set to 0. For real NMR data a value of about 60 will remove most of the spuriously high T1 data.

The high and low thresholds for T1 are the highest and lowest values of T1 which are considered acceptable. Values calculated to be outside this range are set to the extremities of the range. The values are then scaled into the available dataset range: 0 at T1 high threshold and 255 at T1 low threshold. The scaling is linear.

5.6. Surface Tracking Questions

1. X position of voxel to be tracked from (origin lower left) ?

2. Y position of voxel to be tracked from (origin lower left) ?

In the absence of a mouse interface the user is required to specify the x and y coordinates of a point on the currently displayed section from which the surface tracking must start.

3. What is the new value for threshold gradient ?

4. What is the new value for threshold cosine ?

The threshold gradient is the lowest gradient along which the edge tracker will continue tracking. The threshold cosine is the cosine of the greatest voxel to voxel discrepancy in edge direction that the edge tracker will continue to track along.

Appendix 3:
The Packet Handling Protocols.

```
--  Variables used to input packets, or to hold status information.

--  longest packet array size constants

-- The longest packet is the notional packet length which is never
-- exceeded by the packet generating subroutines.
-- part of it is used to input all packets of length less than or equal
-- to its length.  It happens to be equal to the length of the DATA packet

VAL Header.Ints  IS 12:
VAL Header.Bytes IS Header.Ints * 2:

VAL Extended.X.B IS Voxels.Per.X.B + 2:
VAL Extended.Y.B IS Voxels.Per.Y.B + 2:

VAL Pkt.Data.Bytes   IS  ((Voxels.Per.X.B + 2) *
                         (Voxels.Per.Y.B + 2)) * (Voxels.Per.Z.B + 2):
-- the extra twos are for boundary conditions

VAL Data.Pkt.Length  IS Header.Bytes + Pkt.Data.Bytes:
VAL Longest.Pkt.Length IS Data.Pkt.Length:



--  longest packet declarations

-- The address of all packets is held in the first two bytes as an INT16
-- number.  The Flag of the packet is always the third byte, and the
-- auxiliary flag is the fourth.

-- Headers may, of course, contain more information than this but it
-- is left to packet receiving procedures to specify the location of
-- this information.

INT pkt.length:
[Longest.Pkt.Length] BYTE longest.packet:
[Header.Ints] INT16 header RETYPES
  [longest.packet FROM 0 FOR Header.Bytes]:

VAL Flag.Byte IS 2:
address    IS header[0]:

flag       IS longest.packet [Flag.Byte]: --  These two are BYTE values which
aux.flag   IS longest.packet [3]: -----    together make up header[1](INT 16)



-- The flags associated with packets
```

```
VAL Command                  IS 0   (BYTE):
VAL Terminate.Flag           IS 1   (BYTE):
VAL Initialisation.Stream    IS 5   (BYTE):
VAL Data.Stream              IS 10  (BYTE):
VAL Data.Termination         IS 11 (BYTE):
VAL Image.Stream             IS 20  (BYTE):

VAL Dummy.Flag               IS 255 (BYTE):


--  The codes associated with commands.

VAL Terminate.Signal           IS 0  (BYTE):
VAL Initialise.Command         IS 1  (BYTE):
VAL Take.In.Data.Command       IS 2  (BYTE):
VAL Draw.2.D.Image.Command     IS 3  (BYTE):
VAL Farm.Out.Data.Command      IS 4  (BYTE):
VAL Threshold.Command          IS 5  (BYTE):
VAL Store.Command              IS 6  (BYTE):
VAL Bitwise.Compare.Command    IS 7  (BYTE):
VAL Wipe.Working.Flag.Vector   IS 8  (BYTE):
VAL Clear.Scene                IS 9  (BYTE):
VAL Draw.Processed.2.D.Command IS 10 (BYTE):
VAL Calculate.T1.Command       IS 11 (BYTE):
VAL Cons.Smooth.Command        IS 12 (BYTE):
VAL Shuffle.Data.Command       IS 13 (BYTE):
VAL Global.ZH.Command          IS 14 (BYTE):
VAL Threshold.Gradient.Command IS 15 (BYTE):
VAL Track.Gradient.Command     IS 16 (BYTE):
VAL Continue.Tracking.Command  IS 17 (BYTE):


--  The notional addresses associated with channels out of the network

VAL Controller IS -1 (INT16):
VAL Nearest.Corner IS -2 (INT16):

VAL North.Neighbour IS -10 (INT16):
VAL South.Neighbour IS -11 (INT16):
VAL East.Neighbour IS -12 (INT16):
VAL West.Neighbour IS -13 (INT16):
```