# VISUALIZATION OF 3-D EMPIRICAL DATA: THE VOXEL PROCESSOR

ir. Peter L.J. van Lieshout, ir. Wim Huiskamp
TNO Physics and Electronics Laboratory, P.O. box 96864, 2509 JG The Hague, The Netherlands

Image processing is one of the areas which are known to be very computational intensive. To achieve interactive response of imageprocessing systems usually dedicated systems or (mini-) supercomputers are necessary. Imageprocessing is also an area in which the application of parallelism is very suitable, because of the large amounts of (similar) data involved. In this particular case, 3-D images (images in 'voxel space') have to be processed interactively. Operations on the voxel space involve 3-D image processing algorithms and visualization of the data from arbitrary viewing angles, and with several options for the type of rendering. This paper describes an alternative for special hardware or supercomputers for a voxel processor, based on a network of INMOS T800 Transputers.

## Introduction

The TNO Physics and Electronics Laboratory (TNO-FEL) in the Hague is a part of the TNO Division of National Defence Research (TNO-HDO).
The activities of TNO- FEL focus primarily on operational research, information processing, communication and sensor systems. To support the fast data-processing usually required in sensor systems applications, research was started into parallel processing techniques. This research has now resulted in three major application areas : radar data processing, real-time computer generated imagery and 3D image analysis, processing and visualization. 3D image processing and visualization is the subject of this paper. With the growing availability of 3D scanning devices, the need for high performance processing and display systems increased significantly. The development of an experimental parallel processing system is described for the visualization of three dimensional voxel based images. The aim is the visualization of the (unknown) object in such a way that its spatial structure can be understood. An additional demand is that the system is fast enough to be used interactively. Because of the large number of voxels involved, a considerable processing capacity is required. Processing the data in parallel on a network of Transputers provides the necessary computing power. The volume data may be visualized in several ways, involving operations like object transformation, hidden-surface removal, depth-shading and cross-sectioning. The main advantages of the proposed architecture over dedicated hardware solutions are:

- cost/performance ratio
- flexibility
- expandability

## The Voxel space

Volume images are normally represented as a series of parallel two dimensional slices. These slices may have been obtained from several possible sensor systems, examples are Computer
Tomographic- (CT), Nuclear Magnetic Resonance- (NMR), Ultra Sounding or Optical- (LASER) scanners. Voxel representations are very suitable for applications with 3D empirical data. However synthetic data may also be used, examples are: solid modeling and fluid dynamics simulations. Before volume rendering became feasible, experts had to interpret every single slice to deduce the 3D information. Until recently, computer assisted techniques to visualize the volumes were based on displaying contours only, because of the processing time involved. These contours often had to be traced manually from the actual data. Full use of the 3D data could only be made through off-line computing.

Several architectures based on dedicated hardware have been proposed to increase performance (Ref. 1, 2).Dedicated systems however have the disadvantage of inflexiblity to any change in rendering options or object sizes (also the costs are high). Other systems make use of high performance general purpose machines, which are always very expensive and not always suitable. This explains the reason for TNO-FEL to apply a system of programmable (low cost) processors operating in parallel. Prototype data for the voxel processor was obtained from an experimental Confocal LASER Scanning Microscope (CLSM). The CLSM can be focussed on several consecutive layers of the object, producing a slice of data for each layer. A slice typically consists of 256256 volume elements (voxels), with an intensity resolution of 8 bits per voxel (FIG. 4). The number of slices may vary, but a typical value is 32 to 256 (FIG. 2). This data structure is called the voxel space. Examples of application areas for the CLSM are medical-and biological-research and inspection (e.g. Integrated Circuits). Voxel space sizes depend largely on the sensor type, in CT scans for example it is possible to get resolutions of 512*512*128 with 12 bits per voxel, and these numbers still grow. The TNO-FEL voxel processor has the modularity to deal with varying size-and performance-demands.

## The 3-D Reconstruction

The Voxel space consists of a block in 3-D space (FIG. 3). Displaying this data under different angles on a 2D screen involves a 3-D transformation of the object space into the display space (FIG. 4). Basically such a transformation consists of a vector-matrix multiplication on each voxel coordinate (i.e. a vector). The matrices for a rotation around the X-, Y-, and Z - axis with rotation angles A,B and C respectively are :

$$R_x = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos A & -\sin A \\ 0 & \sin A & \cos A \end{vmatrix}$$

$$R_y = \begin{vmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{vmatrix}$$

$$R_z = \begin{vmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

These matrices are concatenated into a single matrix before the actual multiplication:

$R = R_z * R_y * R_x =$

$$\begin{vmatrix} (cB{*}cC) & (sA{*}sB{*}cC-cB{*}sC) & (cA{*}sB{*}cC+sA{*}sC) \\ (cB{*}cC) & (sA{*}sB{*}sC+cB{*}sC) & (cA{*}sB{*}sC-sA{*}cC) \\ -sB & (sA{*}cB) & (cA{*}cB) \end{vmatrix}$$

( 'c' for cos, 's' for sin ).

Suppose a resolution of 128 is used in X-,Y- and Z-direction with 1 byte per voxel. The voxel space then occupies 2 Megabytes. 2 Million vector-matrix multiplications of the kind described have to be performed to compute the correct orientation. Every vector-matrix multiplication consists of 9 multiplications and 6 additions. To compute a new projection therefore 18 million Mults and 12 million Adds would be needed. And this is just the pure computational load, without any overhead like instruction fetches etc. Fortunately, there are possibilities for a simplification. Since vector-matrix multiplication is a linear operation and basically all of the voxel coordinates have to be transformed, it is not necessary to perform this multiplication for each and every coordinate. We may instead use previously calculated results to compute the next. In that case three simple additions are needed to step from one transformed coordinate to the next. This method offers a considerable reduction in the computational load.

The first step is to transform the unit-vectors from object-space into display-space, by multiplication with the previously calculated rotation matrix R.

$$\begin{vmatrix} nx.x' \\ nx.y' \\ nx.z' \end{vmatrix} = \begin{vmatrix} 1 \\ 0 \\ 0 \end{vmatrix} * R$$

$$\begin{vmatrix} ny.x' \\ ny.y' \\ ny.z' \end{vmatrix} = \begin{vmatrix} 0 \\ 1 \\ 0 \end{vmatrix} * R$$

$$\begin{vmatrix} nz.x' \\ nz.y' \\ nz.z' \end{vmatrix} = \begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix} * R$$

The transformed coordinates (x',y',z') of voxel (x,y,z) are now found with :

$$\begin{vmatrix} x' \\ y' \\ z' \end{vmatrix} = x* \begin{vmatrix} nx.x' \\ nx.y' \\ nx.z' \end{vmatrix} + y* \begin{vmatrix} ny.x' \\ ny.y' \\ ny.z' \end{vmatrix} + z* \begin{vmatrix} nz.x' \\ nz.y' \\ nz.x' \end{vmatrix}$$

By simply changing only 1 dimension at the time (i.e. moving along the X-, Y-or Z-axis) a new coordinate is now generated by just three additions. The projection of the 3-D data onto a 2D surface (the screen) involves the hidden surface elimination: 'distant' voxels are obscured by 'closer' voxels if they are projected on the same location on the screen. Comparing the z-value of a new pixel with the z-value of the pixel already present on that screen location (Z-buffer algorithm), is avoided by traversing the voxel space in a back-to-front direction. When generating the screen this way, new pixels can simply overwrite any old value (Painters algorithm).

Several ways of rendering the transformed data on the screen are possible, the available options are:

a) Display the object's intensity, as seen from the selected orientation ('front view'). (Photo 1)

b) Display the object's 'distance' from the screen at each location, resulting in a realistic depth illusion. ('depth shading').

c) Display the object's density at each screen location, ('integrate function').

d) Display an intensity related to the layer from which the visible voxel originated. ('layer view'). (Photo 2)

e) Select a 'Volume-Of-Interest' within the available voxel space (this volume must be block-shaped). Through this option uninteresting or disturbing parts of the voxel image may be 'peeled away'. (Photo 3)

f) Select a cutting plane through the object; voxels in front of this plane will not be visualized. This option will create a cross-section through the object after rotation. (Photo 4)

g) Select a threshold; voxels with a value below this threshold will become transparent.

h) Edit and select different colour look-up tables. This feature enables the use of pseudo-colours or grey-scale transforms for certain intensity values, thereby increasing the visibility of interesting areas.

The original images from the scanning device tend to be noisy in many cases, so noise filters are needed. Further image analysis operations (edge detectors etc.) are also provided. Currently implemented 3D image processing algorithms are :

- Mean filter.
- Sobel and Roberts edge detectors.
- Laplace filter.
- Median filter.

These filters are based on their 2D counterparts and operate on a (3*3*3) space.

Parallel Processing.

Computer applications tend to need increasing amounts of processing capacities. Single processor systems are reaching the limits of performance improvements. It is obvious that using more processors running in parallel should provide (theoretically) unlimited power. Many existing multi-processor systems use a common communications channel (the bus) for interconnections. With a growing number of processors the bus capacity becomes a bottleneck for system performance. Communication bandwidth of the network should be increased also when processors are added. Providing processors with direct (point to point) connections for all data exchange will supply this increased bandwidth.

Several classes of multi-processor systems may be defined. A common way to distinguish classes is between SIMD (Single Instruction stream Multiple Data stream) and MIMD (Multiple Instruction stream Multiple Data stream ) type parallelism. In SIMD parallelism each processor in the network will execute the same instruction (synchronously) on different data. Array processors fall in this class. Examples are image processing applications where each processor performs the same filter operation on a different part of the image. When using MIMD parallelism processors can all be running different programs, possibly sending results to others when they are finished. Examples are pipelined systems or multi-user applications.

Many existing sequential programs could benefit from being able to perform more than one action at a time. It is however generally not trivial to implement a parallel program on a processor network. Problems arising are:

- Decomposing the problem in a number of processes running in parallel.
- Allocate processes to processors and select the network topology.
- Load-balancing the processors.
- Distributing data across the processors.
- Efficient inter-processor communication.
- Synchronization between processors.
- Debugging the software.

The INMOS T800 Transputer is a computer-on-a-chip, containing a 32 bit RISC ALU, a 64 bit Floating-Point Unit, memory and four high-speed (1.5 MByte/s) input/output links for point-to-point communication (Fig. 8, Ref. 3). The Transputer was specifically designed for efficient parallel processing : it is a high performance component (10MIPS, 1.5 MFLOPS), with an on-chip process scheduler and low-overhead communication facilities. A network of Transputers may be constructed by connecting them via links. Each Transputer in a network has (private) local memory to store program and data. Transputers
may be programmed in high level languages like PASCAL, FORTRAN or C. These languages must have facilities added to implement the special features of the Transputer (processes running in parallel, communication etc.). OCCAM is a language that was created by INMOS to describe parallel processing and communication via channels (Ref. 4). In fact the Transputer may be considered a hardware implementation of OCCAM.
Transputer versions without the floating-point unit are also available : the T414 (32 bit) and the T212 (16 bit).

Transputer networks belong to the MIMD class of parallel processing systems, all nodes in a network are basically independent units, communicating and synchronizing only
when necessary. A MIMD network is the most flexible solution to parallel processing, since part of the network may actually be operating in SIMD-mode . At TNO-FEL, research has concentrated on the Transputer as the computational element in parallel processing applications, because of its useful features, high performance and software support.

Parallelism may be accomplished in 3D image processing by splitting up the computations in either display space or in object space :
Display space parallelism implies that each processor is assigned to a certain area of the final image (e.g. a number of scanlines). Since views of the rotated voxel image will be generated, this solution means that each processor must have access to the complete voxel space.
Complete access is possible when a voxel image copy is stored in each processor (large memory requirement) or alternatively, processors could request voxel data elements when needed from a central store (communication overhead). Load-balancing may be a problem, since the most computation intensive parts in the display-space will shift according to the rotation angle. Ray-tracing is a typical example where parallel processing in display space is often used. The load-balancing can be tackled by implementing a processor farm construction. In this construction a controller process "farms out" a new piece of work (i.e. a part of the display) to each processor in the network as soon as this has finished work on a previous part. The controller does not need to know which processor will actually perform the job. Object space parallelism is based on access of a limited part of the original voxel image. This implies that each node is assigned to a section of the voxel image, which is stored locally. A node will produce the contribution of the local data to the result. The actual result will be available after combining (merging) all the contributions. The advantages of this method over the previous one are :

- Less memory requirement.
- Fast access to the (local) voxel data.
- Good load-balancing, all contributions will need the same computation time, when the voxel data sizes are equal.

Disadvantages are : the overhead of the merging operation and the fact that some data calculated by the nodes may not be needed in the final result.

The advantages of the second method where strong enough to use object space parallelism in the voxel processor system.

System Architecture

Figure 5 shows a schematic representation of the voxel processor architecture. Ellipses are used to represent modules running in parallel. Parallelism was achieved in several ways, the most important step is (as mentioned) to divide the object space into eight (equally sized) subspaces (FIG. 3), where each subspace has been assigned to one Transputer. The modules will be explained in more detail below.

The Controller

In the controller the user-interface software and the graphics control unit (sending commands to the Graphics subsystem) are combined. The operator communicates with the 'user-interface' part, which interprets and executes commands. The controller process is capable of sending commands to a Transputer based framegrabber, which may be used to acquire voxel data from some type of sensor. Alternatively, it is possible to read object data from subspace data, the object may be rotated and viewed interactively. The Voxel processor will produce a new image within 1 second after giving a command (for a 256*256*32 object).
Continuous rotation is possible, since the subspace transformation and the may run in parallel. The resulting images can be stored on disk, and read in again at a later time.

The Subspace Processor

This module performs the object transformation and generates a partial result for its subspace. A subspace result will be of size 256*256 for an object of 256*256*32 voxels, while the complete resulting image will be 512*512 pixels. The dimension (D) of a subspace result is easily derived from the Voxel data-base dimensions (DX, DY, DZ) :

$D = SQRT( (DX^2 + DY^2 + DZ^2) )$

A subspace partial result represents the intensity value for each pixel on the screen, except for the 'integrate' function, in which case a voxel count will be produced. The subspace data (the voxels) are loaded only once for each new object and will not be changed during the transformations. The Transputer will begin processing its data after receiving a command, which includes the transformed unit vectors and the selected type of rendering (e.g. front view, depth shade etc.). Besides performing the voxel image transformation, this module is also used for the 3D image processing operations. For this end, memory is reserved to store both the original voxel image and a filtered version.

The Merger

All partial results must be combined (merged) into the complete resulting image. This result is transferred to the controlling process where it will be stored and displayed. The 'merger' process receives eight 2D results from the subspace processors. In order to combine the partial results in a correct way, the merger needs some additional data. This data consists of :

a) A subspace offset. The offset is based on the x and y ccordinates of the subspace's transformed origin. This value is needed to place the subspace result on the correct position in the final image (FIG. 6).

b) A subspace priority. The priority is based on the z-value of the subspace's transformed origin. The lowest priority is for the subspace with the greatest distance from the viewer. The partial results of this subspace will be obscured by any subspace result of a higher priority (FIG. 7).

The merging operation is mainly performed with the 2D block-move instruction (DRAW2D) of the T800 Transputer, which can be used when the subspace results are combined into the final image in order of increasing priority (the final image is initialized to zero first). The DRAW2D can not be used for the 'integrate' function, in which case the numbers in the subspace results corresponding to the same pixel have to be added.

## The Graphic subsystem

This unit is used only for controlling a framebuffer in order to display the resulting images.

## Implementation Remarks

- Voxel coordinates are represented using an INT32 during transformation time. This INT32 is in fact a fixed-point real (16 bit integer part, 16 bit fraction). The accuracy is sufficient for this application and the performance is slightly better than when using REAL32.
- The object is translated to the centre of the screen, independent of the object's orientation.
- Whenever possible, special processes were assigned to communication in order to achieve maximum efficiency of the Transputer Links.
- A communication layer was integrated into the system. This layer provides data and command transport to all processes, and it is also capable of sending (debug) messages from each process to the operator screen.
- The system is very flexible in the dimensions of the objects that are to be transformed. Basically the only limitation is the memory size of each Transputer (currently 1MB). At the moment these dimensions (and a few derived values) are declared in a library. Changing this library and recompiling the software will automatically generate a new version. (N.B. The object dimensions do not have to be a power of two). Some other possible sizes which will give the same performance are : 128*128*128, 256*256*32 or 64*128*256.
- Because of the modular set-up, it is very easy to trade-off system performance against cost (a version with 4 subspace processors is also built).

## Hardware

The Voxel processor is built up entirely with off-the-shelf hardware. Each processor board offers two T800's with 1MByte of memory each. Other boards used, are the Display System with a T800 and 1MByte of video-ram and finally a framegrabber with an on-board T800. Physically the system consists of a 19" cabinet with 7 single euro-sized boards installed. The host system in the Voxel processor is an IBM-AT. All the program code was written in OCCAM. For this application, it is not strictly necessary to use T800's for all processing modules (few floating-point operations are needed and only the merger uses block-moves). However, their higher link-speed does increase over-all system performance.

## Current Activities

Work on the Voxel processor prototype is continued in a number of areas :

a) Addition of more 3D image-processing algorithms. An important feature will be the computer assisted image segmentation (region growing) to select interesting areas in the voxel image.

b) Implementation of 3D geometrical measurements. For medical- and biological- imaging geometrical data is very important. Surface computations, distances and volume measurements have to be applied to the objects in the voxel space.

c) Increase system performance by further code improvement and architecture optimization. For larger voxel space sizes a system with 16 Transputers will be developed. In this larger system the architecture will be changed to a tree structure. The advantage of a tree over a pipeline is in the shorter average length of the communication path between the PE's and the merger.

d) Implement the computation of statistical level information over (part of) the voxel data. Besides for user inspection, this has to be available to specific image processing functions like automatic thresholding, histogram equalization or edge detection.

e) Investigate (voxel) data-compression, determine effects on data transport times and implications on transform algorithms.

f) Feasibility study on stereoscopic display facilities.

## Conclusions

The Voxel processor is a successful demonstration of the performance improvement and flexibility that parallel processing can deliver. It is shown that at least in some applications transputer-type parallel processing may be a good alternative for either supercomputers or dedicated hardware. Transputers have proved to be a very powerful tool. The development of the system software was not trivial but the clear representation and support of parallelism that OCCAM offers helped a lot.

The key features of the developed Voxel processor are :

- Fast, interactive system. Typical rendering speeds are 1 sec. for 2 Mbyte voxel images. The speed may be increased by using more processors.
- Highly modular and easily adaptable software. The prototype is a general purpose framework for 3D image processing.
- Low-cost, small-sized off-the-shelf hardware. Transputers are general purpose processors and the system may also be used for other (computational intensive) applications.
- Flexible performance (linear cost/performance function).

## References

1) S. M. Goldwasser and R. A. Reynolds.
Real-Time Display and Manipulation of 3D Medical Objects :
The Voxel Processor Architecture. Computer Vision, Graphics and Image Processing 39.
1-27 (1987).

2) A. Kaufman and R. Bakalash.
Memory and Processing Architecture for 3-D Voxel Based Imagery.
IEEE Computer Graphics & Applications.
November 1988.

3) The Transputer Databook. INMOS publication, 1989.

4) C.A.R. Hoare (Ed). OCCAM 2 Reference Manual, Prentice Hall, 1988.

5) A. C. Tan, R. Richards, A.D. Linney.
3-D Medical Graphics - Using the T800 Transputer. Proceedings of the 8th technical meeting of the OCCAM User Group.
March 1988.

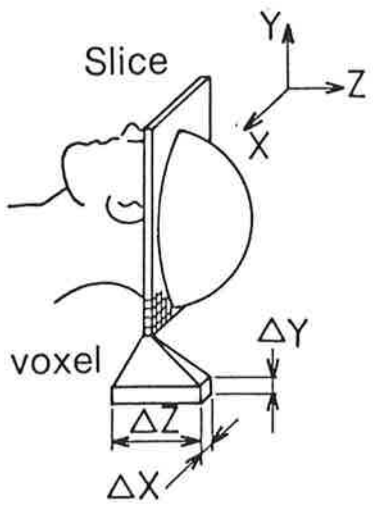


Fig. 1 Voxel Definition.

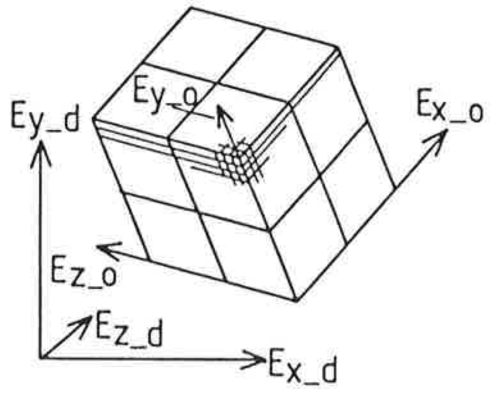


Fig. 4 Display Space.

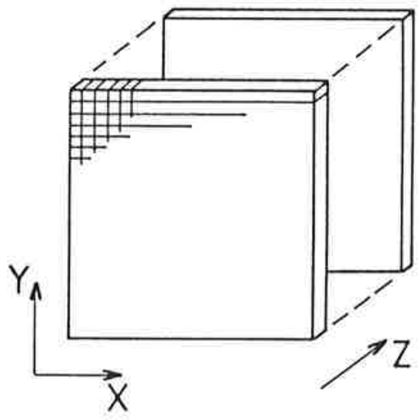
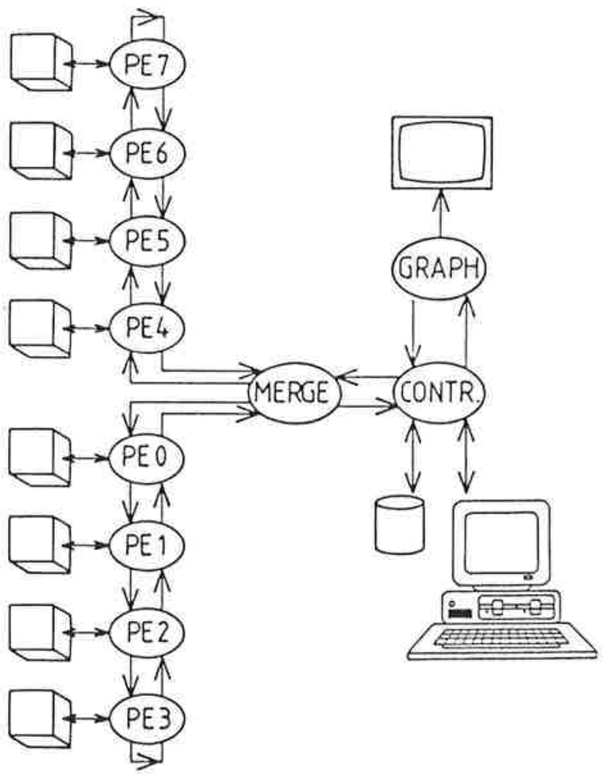


Fig. 2 Voxel Image.



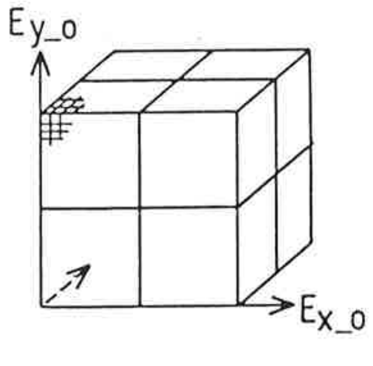


Fig. 3 Object Space.

Fig. 5 System Architecture.

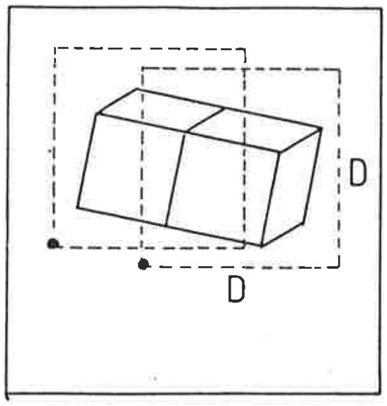


Fig. 6 Subspace Offset.

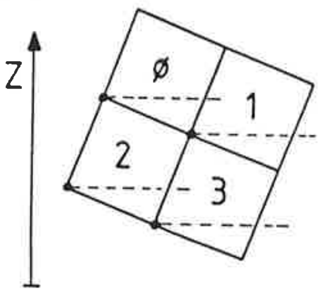


Fig. 7 Subspace Priorities.

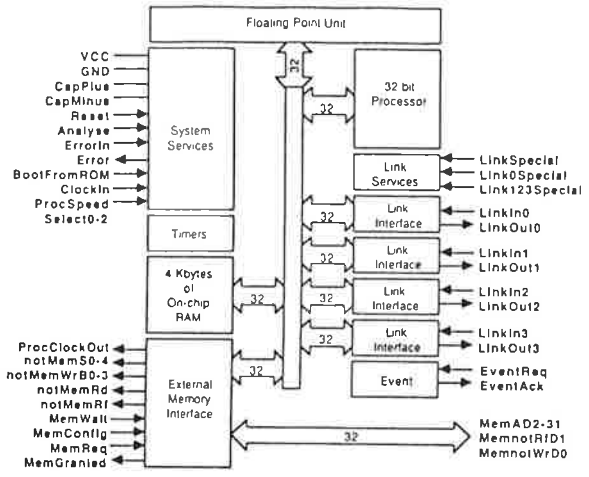


Fig. 8 T800 Block Diagram.

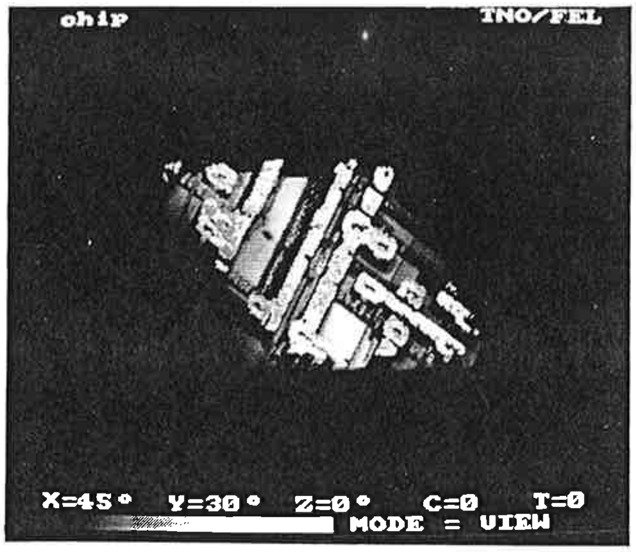


Photo 1 View Mode.

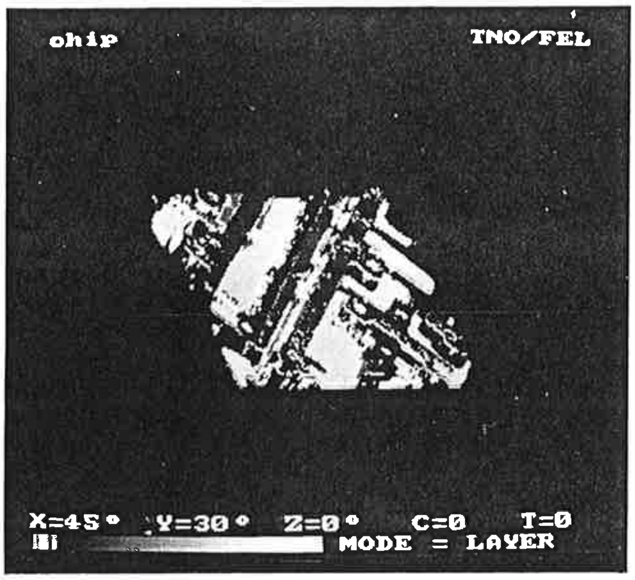


Photo 2 Layer Mode.

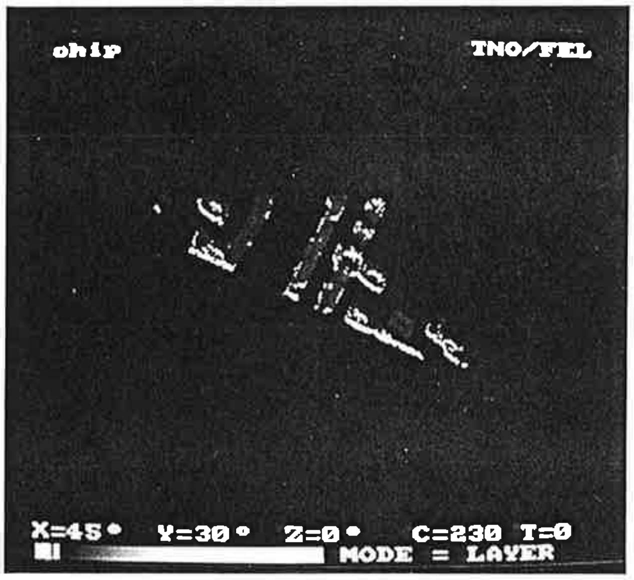


Photo 3 Volume of Interest.

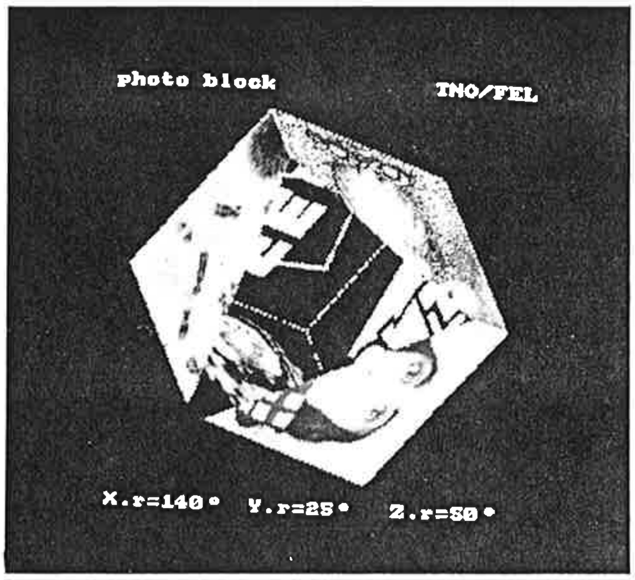


Photo 4 Cross-section.