A Transputer Network for the Arbitrary Rotation of Digitised Images

H. R. ARABNIA AND M. A. OLIVER

Computing Laboratory, The University of Kent at Canterbury, Canterbury, Kent CT2 7NF

An algorithm for the rotation of a digitised image is presented. This algorithm has been designed to work on a transputer network which has a simple topology. The data structure used to represent the image is closely related to run-length encodement. Tests have been made on a simulation of the network.

1. INTRODUCTION

Transputers[1] and the programming language Occam[2] are now available. They offer a sophisticated medium with which to approach the problems of concurrent processing. The rotation operation rotates a digitised image by θ degrees about a specified point into a resultant digitised image; there is no restriction on the value of θ. The rotation of digitised images has been studied on conventional computers.[3] In this paper we investigate how a network of transputers might be used to rotate a digitised image raster by an arbitrary angle.

The first step is to choose a suitable image data structure for the network topology. In two previous papers we presented a series of algorithms for the manipulation of digitised images on machines with SIMD (Single Instruction on Multiple Data) architecture.[4,5] In view of the success of the 'stripcode' data structure (described below) on SIMD machine architectures we decided to look for a network topology which would support stripcode in a natural way. The network is described in Section 2.

Stripcode is essentially run-length code. Run-length code exploits the horizontal coherence between adjacent pixels on a scanline. Thus some account is taken of the image structure and, in general, a reduction in the number of data objects in the image representation results. It is this reduction in the number of data objects which is important rather than the amount of memory required to represent them.

The key to the success of the rotation algorithm described here is in the careful design of the data flow. A major achievement is the avoidance of any sorting in the algorithm: where sorting might have been expected the data is merely merged.

The structure of the paper is as follows. In Section 2 the transputer network is described. The description of the rotation algorithm is given in Section 3. In 3.1 there is a general description of the algorithm which is illustrated by a simple example. In 3.2 a more precise specification of the algorithm is described. In a rather long subsection, 3.3, detailed explanations of the more complex steps in the algorithm are given (an overall view of how the algorithm works can be obtained without this subsection). An assessment of the algorithm is given in Section 4, which includes estimates of times taken for the rotation of a sample image. In Section 5 there is a description of data input to, and output from, the network. An alternative scheme for the output order of

the scanlines is described in Section 6. Some concluding remarks are made in Section 7.

The coordinate system has its origin in the lower left corner of the image space, with the x-axis to the right and the y-axis up. Scanlines are counted from the bottom up.

Stripcode is related to the run-length encodement of an image. Consider a run-length-encoded image and specify a background colour. Each run of pixels of the same colour is called a 'strip'. Each strip of colour, other than the background colour, is specified by the position coordinates of its origin (original number of its first pixel, scanline number), its length, and colour; the background colour is not explicitly coded.

The program fragments are written in Occam:[2] however, we have taken the liberty to use subscripts and primes in variable names which are printed in italic. Also, the arithmetic operator precedence of ordinary arithmetic has been assumed; accordingly, the extensive use of parentheses in arithmetic expressions has been omitted.

The image is rotated by an angle θ: the values of the trigonometric functions of θ are denoted by the names of the functions in italic. For example, *tan* denotes $\tan(\theta)$.

2. THE TRANSPUTER NETWORK

The image raster is divided into a number of blocks where each block contains an equal number of scanlines. There could be as few as two blocks or as many as there are scanlines. An example is shown in Fig. 1.
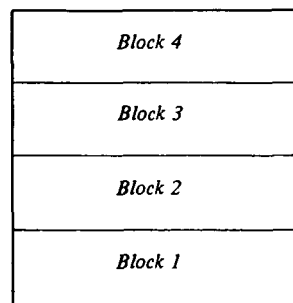

Figure 1. A raster divided into four blocks.

If the number of scanlines is not divisible by the number of blocks an appropriate number of dummy scanlines can be included.

A transputer is allocated to each block of the raster and these transputers are connected into a ring in the

order of the blocks. In Fig. 2 the channel connections are shown for the four transputers which are required for a raster subdivided into four blocks.
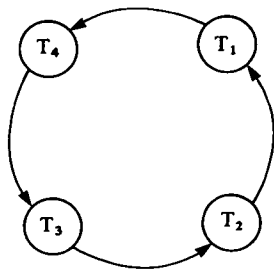

Figure 2. The transputer network for four blocks.

In order to handle input and output for the network extra channels must be used. Two ways in which input and output can be handled are described in Section 5.

3. THE ROTATION ALGORITHM

3.1. General description

The image is encoded in stripcode. The stripcode for each block is held in the local memory of the transputer to which that block is assigned. Each transputer is responsible for one particular block of scanlines in the raster. In the example shown in Figs 1 and 2, the strips in *Block* 1 will be held in the memory of T_1, *Block* 2 in T_2, and so on.

The first phase

All the transputers in the network rotate the strips which are assigned to them by θ degrees: this gives the rotated image exactly, but not digitised to the raster (Fig. 3(a)). The rotated strips are then clipped to the image space: the rectangular parts of the rotated strips that lie *completely* outside the image space are removed (Fig. 3(b)).
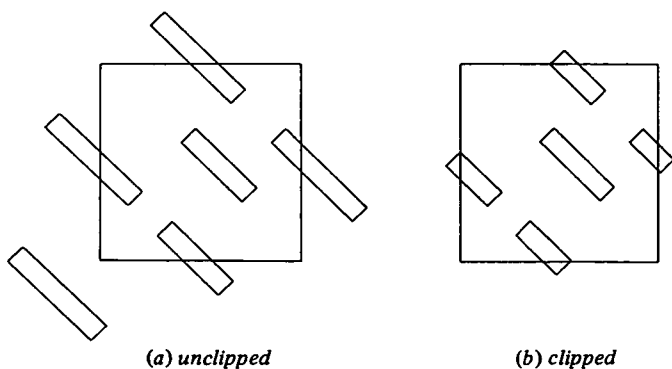

(a) unclipped   (b) clipped

Figure 3. Rotation and clip of image strips.

The second phase

In order to digitise the rotated image to the raster these rotated strips have to be represented by a new set of horizontal strips in the raster.

The rotated strips are digitised vertically into segments. A segment is defined as follows: the boundaries of the

segments are defined by the intersections of the upper and lower boundaries of the scanline (vertical digitisation) together with the intersections of the upper and lower boundaries of the rotated strip with the lower boundary of the scanline (an example is given in Fig. 4). These segments are not horizontally digitised. The end segments present a complication which is dealt with in the further details below.

The algorithm generates the segments required to build up the data for strips scanline by scanline. The segment data is generated in the transputer to which the rotated strip belongs and, in general, this transputer's block of scanlines does not include the scanline of the segment. The fundamental problem is how the generated segments can be put into the transputer which holds the block to which they belong. The solution is to pass the data round the ring: the data for a complete scanline is built up as new data is added from each transputer on the ring. This new segment data has to be horizontally digitised, merged, and compacted at each stage to form the new strips. Each step round the ring will be called a process.

Example

As an example take an image space which contains twelve scanlines and is divided into four blocks. One complete cycle of data round the ring takes four processes. The rotated strips and their segments on the first scanline of the second block are shown in Fig. 4; the horizontal digitisation is not shown in this example.
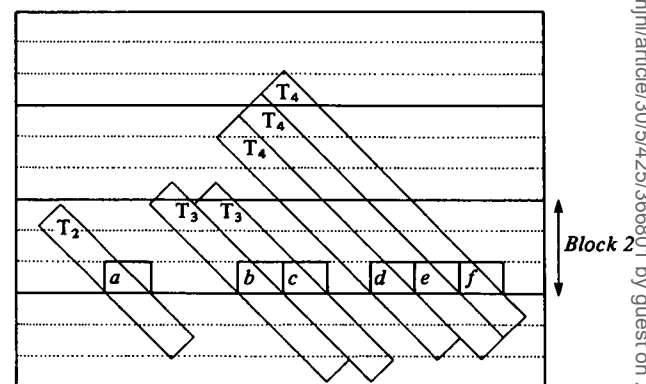

Figure 4. An example.

The transputer identities inside the rotated strips, T_2, T_3, T_4, show in which transputer memory each of these strips is held, Fig. 2.

The description which follows traces the path of the data for the first scanline of the second block. However, the strips on the first scanline of *each* block are computed concurrently in the same process. The scanline process order is shown in Fig. 5, the block order in Fig. 1, and the network in Fig. 2.

In Fig. 4 consider the first scanline of block 2: the segments formed to represent the rotated strips are shown. In Fig. 5 the ordinal numbers in the fourth scanline for each transputer give the place in the sequence of scanline processes.

● In the first process the segment a is computed in T_2, and then passed on to T_1.
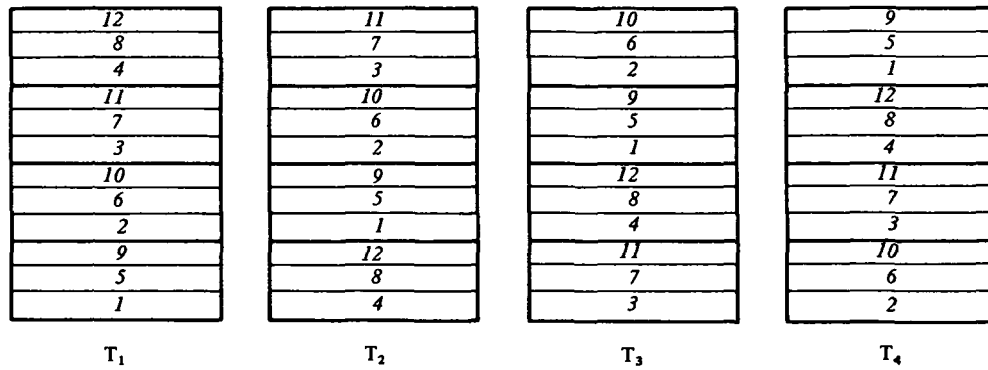
T_1	T_2	T_3	T_4
12	11	10	9
8	7	6	5
4	3	2	1
11	10	9	12
7	6	5	8
3	2	1	4
10	9	12	11
6	5	8	7
2	1	4	3
9	12	11	10
5	8	7	6
1	4	3	2

Figure 5. Process order of scanlines in ring transputers

- In the second process no segment is computed in T_1 so a is passed on to T_4.
- In the third process the segments d, e, f are computed in T_4 and these, together with a, are passed on to T_3.
- In the fourth process the segments b, c are computed in T_3 and these, together with a, d, e, f, are passed on to T_2.

After these four processes (the number of processes corresponds to the number of blocks) each transputer in the ring will hold the stripcode for the first scanline in its block. Thus three revolutions need to be performed to generate the entire rotated image of four blocks each with three scanlines.

3.2. A more precise specification

Suppose that there are S scanlines in the image divided into B blocks; there are $L = S/B$ scanlines in each block. Each transputer has three buffers: an input buffer for the block of stripcode assigned to it; an output buffer for its block of stripcode for the rotated image; a (double) buffer for the current scanline. The angle and the centre of rotation are made available to all the transputers in the ring and then the following steps are performed by each transputer:

SEQ
(a) All the strips in the input buffer are rotated by θ degrees, which gives an exact rotated image, not digitised (Fig. 3(a)).
(b) Rectangular parts of the rotated strips that lie *completely* outside the image space are clipped (Fig. 3(b)).
(c) Some preliminary computations on the stripcode of the clipped rotated strips in the input buffer are done at this point.

    $i: = L$
    WHILE $i > 0$
      SEQ
        $j: = B$
        WHILE $j > 0$
          SEQ
(d)   Compute the segments of the rotated strips that are in the input buffer on the first/next scanline. A segment occupies the portion of the scanline between the points of intersection of the rotated strip and the lower edge of the scanline: possible cases are shown in Figs 7, 10 and 11. Each transputer processes its scanlines in the 'process order' assigned to it.

(e) Digitise the ends of each segment to form strips and merge them into the strips already in the scanline buffer; thus the strips are maintained in their correct order across the scanline. This merge is trivial when the scanline buffer is empty.

(f) Compact the strips in the scanline buffer, i.e. adjacent strips with the same colour are represented by one longer strip. The compaction process is applied only to those strips that were put in the scanline buffer at the previous step.

(g) Concurrently, output the contents of the scanline buffer to the scanline buffer of the next transputer on the ring and input into the scanline buffer the contents of the scanline buffer of the previous transputer on the ring. Each transputer now holds only the strips which it received from the previous one in the ring; the data is held in the scanline buffers. To avoid deadlock and achieve concurrency the scanline buffer is a double buffer.

        $j: = j-1$
        — end of inner while loop
      Append the strips in the scanline buffer to the strips in the output buffer.
    $i: = i-1$
    — end of outer while loop

Example

For the example of Fig. 4 the description which follows traces the path of the data for the first scanline of the second block through the algorithm.

- T_2 calculates a (step (d)); puts a in the scanline buffer (step (e)); compacts a (step (f)); outputs a to the scanline buffer of T_1 (step (g)).
- T_1 finds no segment (step (d)); merge (step (e)) and compaction (step (f)) do nothing; T_1 outputs a to the scanline buffer of T_4 (step (g)).
- T_4 calculates d, e, f (step (d)); merge produces a, d, e, f in scanline buffer (step (e)); compaction produces a, D where D is the compacted d, e, f (step (f)); T_4 outputs a, D to the scanline buffer of T_3 (step (g)).

- T_3 calculates b, c (step (d)); merge produces a, b, c, D in scanline buffer (step (e)); compaction produces a, B, D where B is the compacted b, c (step (f)); T_3 outputs a, B, D to the scanline buffer of T_2 (step (g)).
- T_2 appends a, B, D to its output buffer.

As shown above, after the data has passed through all the transputers of the ring (one complete revolution), the output buffer of T_2 contains the strips on the first scanline of its block, namely *Block* 2.

3.3 Further details

Now for a detailed description of the more complex steps in the algorithm.

Step (c)

For each strip the variable *intersect* is given the value of the x-coordinate at the point where the upper side of the rotated strip, extended if necessary, intersects the x-axis. This is illustrated in Fig. 6.

The intersection point for each rotated strip is calculated as follows:

$$intersect := x_2 + y_2 * cot$$

where x_2, y_2 are the coordinates of the top right corner of the strip after rotation. These intersection points are used in step (d) for the segment calculation.

Step (d)

Computation of the segments on a given scanline from the rotated strips. A segment occupies the portion of the scanline between the points of intersection of the rotated strip and the lower edge of the scanline as shown in Fig. 7. To calculate the segments, the list of the rotated strips has to be searched in order to find those on the scanline. Because of this search this step is the most time-consuming part of the algorithm.
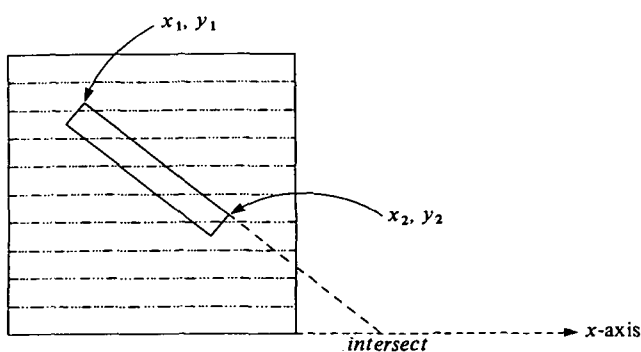

Figure 6.

Define two new y-coordinates y_1' and y_2':
SEQ
    $y_1' := \text{truncate}(y_1)$
    $y_2' := \text{truncate}(y_2)$
The variable *scanline* takes values from 0 to $S-1$, where S is the number of scanlines in the image space. A rotated strip which is on *scanline* gives rise to a segment as shown by the code which follows:
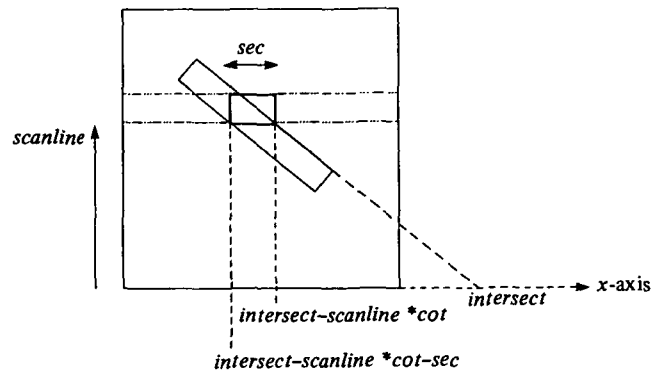

Figure 7.

IF
    $(y_1' \geqslant scanline)$ AND $(y_2' \leqslant scanline)$
    SEQ
        $decrement := scanline * cot$
        $seg.x.end := intersect - decrement$
        $seg.x.begin := seg.x.end - sec$
        $seg.colour :=$ colour of the rotated strip

where $seg.x.begin$ and $seg.x.end$ are the initial and final x-coordinates of the segment and its colour is $seg.colour$, see Fig. 7.

The previous code produces incorrect segments at either end of the rotated strip. These segments have to be modified.

The code that follows modifies the segment for the top end of the rotated strip to obtain the correct segment:

IF
    $(y_1' = scanline)$
    SEQ
        $seg.begin := x_1'$
        IF
            $(seg.x.begin < seg.x.end - sec)$
                $seg.x.begin := seg.x.end - sec$
            (TRUE)
                SKIP
    (TRUE)
        SKIP

where x_1' is the x value of the intersection of the top end of the rotated strip with *scanline*, as shown in Fig. 8,

$$x_1' := x_1 - (y_1 - y_1') * tan.$$

The first IF checks whether the segment at the top end of the rotated strip is to be considered. The second IF
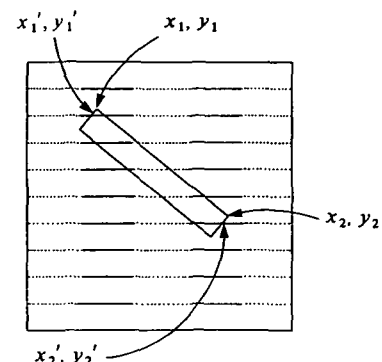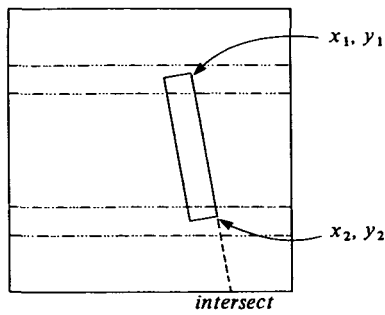

Figure 8.

Figure 9.

Figure 11.

checks to find out if the top end has intersected the scanline; Fig. 9 gives an example of the case where the ends do not intersect any scanline.

The code that follows modifies the segment for the bottom end of the rotated strip to obtain the correct segment:

```
IF
    (y'₂ = scanline)
        SEQ
            seg.x.end:= x'₂
            IF
                (seg.x.end ≤ seg.x.begin)
                    ignore this segment.
                (TRUE)
                    SKIP
    (TRUE)
        SKIP
```

where x'_2 is the x value of the intersection of the bottom end of the rotated strip with *scanline*, as shown in Fig. 8,

$$x'_2 := x_2 - (y_2 - y'_2) * tan.$$

The first IF checks whether the segment at the bottom end of the rotated strip is to be considered. The second IF checks to find out if the bottom end has intersected the scanline.

Fig. 10 shows the segments computed for the ends of a rotated strip for the two scanlines that intersect its ends.
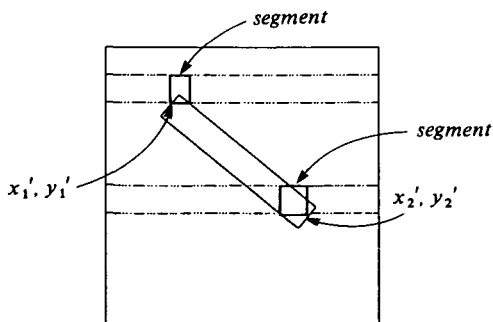

Figure 10.

The code also handles the case where both ends of a rotated strip are on the given scanline (see Fig. 11).

Step (e)

Digitise the ends of each segment to form strips and merge them into the strips already in the local memory to keep the strips in their correct order. To do this the steps that follow are performed.
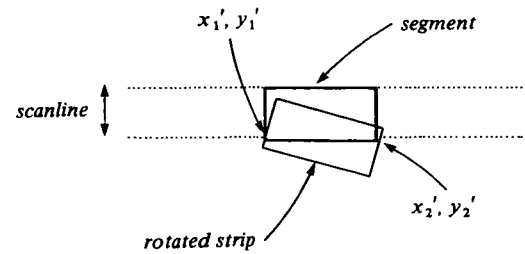
(i) Digitise the first segment; now it is a strip. Find position, p, in the list of the strips held in the scanline buffer in which the new strip should be placed. Position p is found quicker if the list is searched from the end backwards.

(ii) Move the strips between positions p and m, n places down the list, where m is the index of the last element in the list and n is the number of segments. Only a few strips have to be moved. Quite often p is $m + 1$, in which case no strip need be moved.

(iii) Place the first digitised segment in position p and the remaining segments in positions $p + 1$ onwards; digitise them as they are being placed in the scanline buffer.

Step (f)

Compact those strips just added to the list at step (e) in the sense that adjacent strips with the same colour on a scanline are represented by one longer strip.

To do this assume that the list of strips is held in four arrays (the first element of an array is indexed by 0): Ix_1 (initial x-coordinates), Ix_2 (final x-coordinates), Iy (y-coordinates), Ic (colours). The uncompacted strips (i.e. those just added to the list) are placed from position p to $p + n - 1$ (step (e)). The compaction process might need to be applied from position $p - 1$ to position $p + n$, because the first (at p) and the last (at $p + n - 1$) of these strips could be adjacent to and have the same colour as the strips at positions $p - 1$ and $p + n$ respectively. So the compaction process is applied from positions *start* to *finish*; where *start* is $p - 1$ (*start* is zero if p is zero) and *finish* is:

```
IF
    ((p + n - 1) < m) - m is the index of last element -
        finish:= p + n
    (TRUE)
        finish:= p + n - 1
```

A mask, *mask*, has to be constructed. The elements of *mask* determine which strips can be compacted.

```
SEQ
    k:= start
    SEQ i = [0 FOR (finish − start)]
        SEQ
            IF
                (Ic[k + 1] = Ic[k]) AND
                (Ix₁[k + 1] ≤ Ix₂[k])
                    mask[i]:= FALSE
                (TRUE)
                    mask[i]:= TRUE
            k:= k + 1
```

Now with the use of *mask* the actual process of compaction is carried out.

```
SEQ
    p₁:= start+1
    p₂:= p₁
    SEQ i = [0 for (finish−start)]
        SEQ
            IF
                (mask [i])
                    SEQ
                        Ic[p₁−1]:= Ic[p₂−1]
                        Ix₁[p₁]:= Ix₁[p₂]
                        Ix₂[p₁−1]:= Ix₂[p₂−1]
                        p₁:= p₁+1
                (TRUE)
                    SKIP
            p₂:= p₂+1
    Ix₂[p₁−1]:= Ix₂[finish]
    Ic[p₁−1]:= Ic[finish]
```

An example is shown in Fig. 12. Assume all strips are of the same colour. There are three lists shown in Fig. 13. The list of strips before compaction is shown in L_1. Assume the strips to be compacted are b, c, d, e, f. After compaction L_1 will have a gap shown in L_2. The second element of L_2, B, is the strip which is the compacted b, c, d, e.
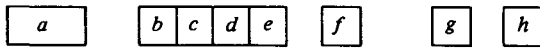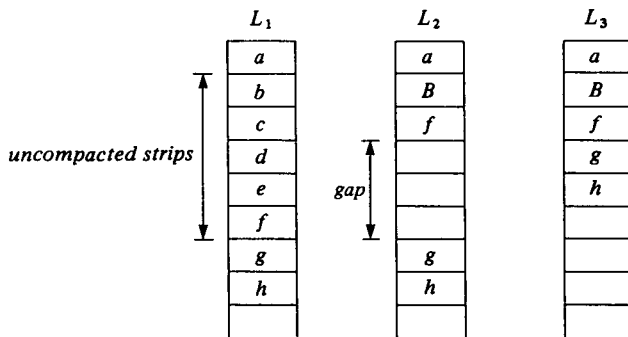

Figure 12.

Figure 13.

The code that follows will close this gap in the list of strips (producing L_3 in the example).

```
SEQ
    k:= finish+1
    SEQ i = [0 for (m−finish)]
        SEQ
            Ic[p₁]:= Ic [k]
            Ix₁[p₁]:= Ix₁[k]
            Ix₂[p₁]:= Ix₂[k]
            p₁:= p₁+1
            k:= k+1
```

4. AN ASSESSMENT

The algorithm has been implemented on an Ocean compiler which runs on a sequential machine.[6] If one block contains many strips and the others none at all then

Table 1. Execution times (simulated)

Angle of rotation (degrees)	Number of blocks	Strips in each block before rotation	Total strips after rotation	Time (ms)
5	4	640	2,952	539
	8	320		274
	16	160		142
15	4	640	3,674	558
	8	320		285
	16	160		148
85	4	640	6,426	631
	8	320		334
	16	160		183

the speed of rotation will, in general, be the same as if the other blocks had the same number of strips. An image which has a uniform number of strips in each block is used to time the system. Timings for an image of 2,560 strips on an image space of 256 scanlines each 256 pixels long are given in Table 1. The blocks are input to the transputers before rotation commences.

It is assumed that the memory is IMS2600-12 dynamic RAM (150 ns access time), the transputers are T424 (32-bit, 10 MIPS) and all data is held in external memory. From the table it is seen that when the number of transputers in the ring is doubled the execution time is almost halved. The execution times have been obtained from counts of program elements together with published times taken for each programming element.[1]

On the network of transputers the execution time depends on the *maximum number of strips in a block* and the number of scanlines in image space. In general, the more blocks (i.e. more transputers) the smaller the maximum number of strips in any block becomes.

When more transputers are used in the network ring less memory is needed in each transputer. This trade-off can be exploited to gain more speed by using more transputers with smaller and faster memories.

5. INPUT AND OUTPUT

Two methods for the input and output of the images are shown. Fig. 14 shows one simple method. The image is fed in by the transputer labelled I/O. Each transputer
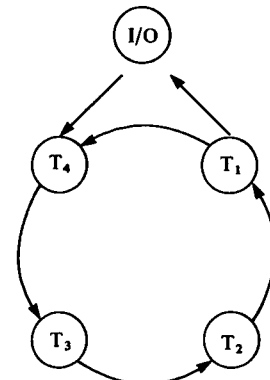

Figure 14.

keeps its own block and sends the blocks which remain on to the next transputer in the ring.

To output the rotated image after the completion of the rotation process, each transputer in the ring concurrently outputs the block it is holding to the next and inputs a block from the previous transputer, except the transputer responsible for the last block (namely T_4 in Fig. 14), which does not input any block, and the transputer responsible for the first block (namely T_1 in Fig. 14), which outputs to I/O. This input and output of blocks continues until all blocks are output to I/O. Alternatively, after the scanline data of each transputer has moved round the ring (one complete revolution), B scanlines can be sent to I/O for display, where B is the number of blocks.

Another scheme is shown in Fig. 15. Concurrently, each block is input or output directly into the transputer to which it belongs.
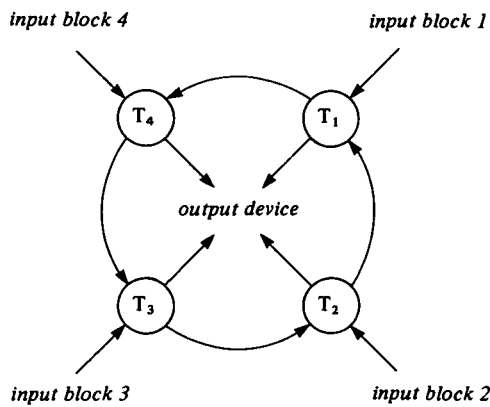

input block 4   input block 1

output device

input block 3   input block 2

Figure 15.

6. AN ALTERNATIVE SCANLINE ORDER FOR OUTPUT

As described in Subsection 3.1, the nth scanline of each block will be ready for output after the completion of the nth revolution of data flow through the transputers. Hence, if the image space has twelve scanlines numbered from 1 to 12, and is divided into four blocks, at the completion of the first revolution the scanlines 1, 4, 7, 10 (first scanline of each block) will be ready for output. We are indebted to Tony King for pointing out to us that the

same algorithm can generate the scanlines for output in natural order (i.e. 1, 2, 3, 4, ...) by a change in the process order together with a different distribution of scanlines among the blocks.

Block	Scanlines		
1	1	5	9
2	2	6	10
3	3	7	11
4	4	8	12

Figure 16. Distribution of scanlines over the blocks.

In Fig. 16 the new distribution is shown. It shows the scanlines held in each block for an image space of twelve scanlines divided into four blocks (blocks do not hold contiguous scanlines). The process order is shown in Fig. 17.

7. CONCLUSION

A transputer network has been proposed in order to facilitate operations on digitised images encoded in the image data structure 'stripcode', which is closely related to run-length encodement. Of all the fundamental geometrical operations on a digitised image, rotation through an arbitrary angle is undoubtedly the hardest to do with any efficiency. We believe it to be a considerable success that the rotation algorithm described in this paper avoids any sorting. In a second paper we shall show how the same network can be used to scale translate and combine digitised images.

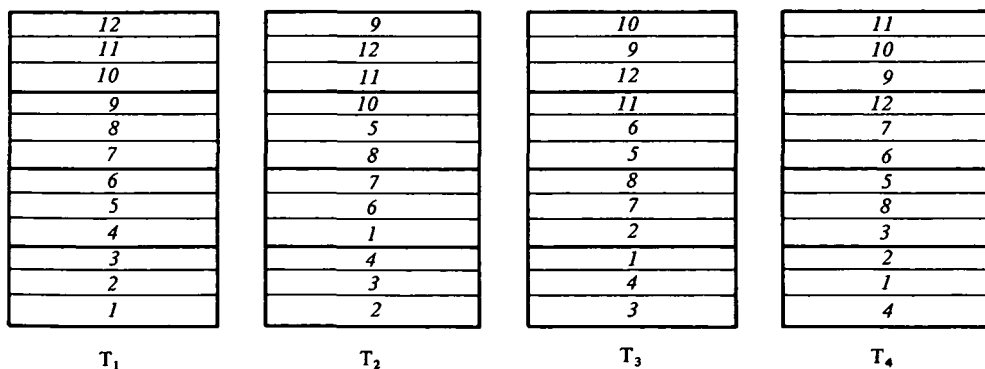| T_1	T_2	T_3	T_4
12	9	10	11
11	12	9	10
10	11	12	9
9	10	11	12
8	5	6	7
7	8	5	6
6	7	8	5
5	6	7	8
4	1	2	3
3	4	1	2
2	3	4	1
1	2	3	4

Figure 17. Process order of scanlines

REFERENCES

1. INMOS Limited, *IMS T424 Transputer*.
2. INMOS Limited, *Occam Programming Manual*. Englewood Cliffs, N.J.: Prentice-Hall (1984).
3. G. Johnston and A. Rosenfeld, Geometrical operations on digitised pictures. *Picture Processing and Psychopictorics*, pp. 217–241 (1970).
4. H. R. Arabnia and M. A. Oliver, Fast manipulation of raster images with SIMD machine architectures. *Computer Graphics Forum* 5, 179–188 (1986).
5. H. R. Arabnia and M. A. Oliver, Arbitrary rotation of raster images with SIMD machine architectures. (Submitted for publication 1986).
6. INMOS Limited, *IMS B002 Transputer Evaluation Board* (1985).

ANNOUNCEMENTS

21–24 MARCH 1988

RIAO 88, Massachusetts Institute of Technology, Cambridge MA, USA. Conference organised by the following. Centre National de la Recherche Scientifique (CNRS), Centre National de Recherche des Télécommunications (CNET), Institut National de Recherche en Informatique et Automatique (INRIA), Ecole Nationale Supérieure des Mines de Paris and Centre de Hautes Etudes Internationales d'Informatique Documentaires (CID). US participating organisations are: American Federation of Information Processing Societies (AFIPS), American Society for Information science (ASIS) and Information Industry Association (IIA).

This conference is prepared under the direction of: Professor André Lichnerowicz de l'Académie des Sciences de Paris and Professor Jacques Arsac, correspondant de l'Académie des Sciences de Paris.

Call for Papers: 'User-Oriented Content-based Text and Image Handling'

Introduction

RIAO 88 (RIAO: Recherche d'Informations Assistée par Ordinateur) is being held to demonstrate the state of the art in information retrieval, a domain that is in rapid evolution because of developments in the technology for machine control of full-text and image databases. This evolution is stimulated by the demands of end-users generated by the recent availability of CD-ROM full-text publishing and increased public access to information databases.

A group of French organisations has taken the initiative of preparing this conference. Its wish in promoting this forum is not only to stimulate and challenge researchers from all nations, but also to increase an awareness of European technology.

This 'call for papers' is being distributed world-wide. We want to reach individuals in the research communities throughout the university and industrial sectors.

The conference will be held in Cambridge, MA. We hope that it will encourage the exchange of European and American viewpoints, and establish new links between research teams in the United States and Europe, especially France.

General theme

Full-text and mixed media database systems are characterised by the fact that the structure of the information is not known *a priori*. This prevents advanced knowledge of the types of questions that will be asked, unlike the situation found in hierarchical and relational database management systems.

You are invited to submit a paper showing how the situation can be dealt with. Special attention will be given to:
● techniques designed to reduce the imprecision of full-text database searching;
● data entry and control;
● 'friendly' end-user interfaces;
● new media.
A large number of specific subjects can be treated within this general framework. Some suggestions are made in the following section.

Specific themes

(A) Linguistic processing and interrogation of full-text databases:
● automatic indexing,
● machine-generated summaries,
● natural language queries,
● computer-aided translation,
● multilingual interfaces.
(B) Automatic thesaurus construction.
(C) Expert system techniques for retrieving information in full-text and multimedia databases:
● expert systems reasoning on open-ended domains,
● expert systems simulating librarians accessing pertinent information.
(D) Friendly user interfaces to classical information retrieval systems.
(E) Specialised machines and system architectures designed for treating full-text data, including managing and accessing widely distributed databases.
(F) Automatic database construction using scanning techniques, optical character readers, output document preparation, etc.
(G) New applications and perspectives suggested by emerging new technologies:
● optical storage techniques (videodisc, CD-ROM, CD-I, Digital Optical Discs),
● integrated text, sound and image retrieval systems,
● electronic mail and document delivery based on content,
● voice-processing technologies for database construction,
● production of intelligent tutoring systems,
● hypertext and hypermedia.

Conditions for participation

The programme committee is looking for communications geared towards practical applications. Papers which have not been validated by a working model, a prototype or a simulation, or for which a realisation of such a model seems currently unlikely, may be refused.

Authors must submit a paper of about 10 pages double-spaced, and a 100-word abstract. Four copies must be sent before 30 October to one of these two addresses:

RIAO 88, Conference Service Office, MIT, Bldg 7, Room 111 Cambridge, MA 02139

RIAO 88, CID, 36 bis rue Ballu, 75009 Paris, France

Each presentation will last 20 minutes, followed by 10 minutes of discussion and questions.

Arrangements have been made with the international journal, *Information Processing and Management* to publish expanded versions of some papers.

High-quality audiovisual techniques should be used when presenting the paper.

Separate demonstration sessions can be scheduled if requested.

Particular attention will be paid to:
● the use of readily available equipment for demonstrations (IBM, PC, Apple, network connections),
● pre-recorded video or floppy disc displays.
Hardcopy printouts of results should be avoided if possible.

English is the working language of the conference.

For further information call:
(in North America) Karen Daifuku, tel. (202) 944 62 52
(in other countries) Secrétariat Général du CID in France, tel. (1) 42 85 04 75