Increased Single-Photon Emission Computed Tomography Image Processing Speed Achieved in Personal Computers With Memory-Intensive Algorithms

Jonathan P. Pratt and James L. Lear

Recent dramatic reductions in the cost of computer random access memory (RAM) and the ability of newer microprocessors and associated personal computer operating systems to address large amounts of memory make novel strategies for high-speed image processing possible. We developed image processing algorithms that use this newly available memory to achieve increases in effective processing speed. These algorithms rely on the use of precomputed lookup tables to avoid repeated use of relatively expensive machine instructions, such as multiplications and divisions. Programs using this strategy to perform single photon emission computer tomography (SPECT) analysis were written in C and assembly language and tested on a Macintosh Quadra 950 (Apple Computer, Cupertino, CA) having 64 megabytes of RAM. The measured processing times are competitive with most dedicated nuclear medicine computers. A general implementation of such programs will allow personal computers to compete with dedicated imaging systems, at a substantial reduction in cost.

KEY WORDS: Personal computers, computer processing, image processing, SPECT.

PERSONAL COMPUTERS are gaining increasing acceptance for use in basic acquisition, display, processing, and telecommunication of radiological imaging data.[1,2] However, the perception that minicomputers, workstations, and/or dedicated hardware are needed for advanced or complex processing remains. This perception partly derives from the obvious hardware-related processing superiority of the larger computers (Table 1). Minicomputer cycle times are two to three times faster than those of the most advanced personal computers; the data paths of minicomputers have twice the capacity of those in personal computers; and the floating-point units of minicomputers are faster and more sophisticated than those in personal computers. These factors result in significant differences in speed between minicomputers/workstations and personal computers using the standard measures of million instructions per second (MIPS) and million floating-point operations per second (megaflops), with personal computers suffering a 10-fold or more disadvantage in raw computing power.

Our objective was to explore methods of improving image processing capabilities of personal computers to make them more competitive with dedicated systems. We focus on the processing of single-photon emission computed tomography (SPECT) studies because this analysis is commonly used in nuclear medicine and is among the most time consuming. Additionally, SPECT analysis uses a number of image processing functions that are also useful in analysis of other types of radiological images.

Although dedicated coprocessing boards can be used to improve the performance of personal computers, the cost of such coprocessors can easily exceed that of the host personal computer. Also, nonportable code must be written to support such external hardware.

Thus our techniques for improving personal computer image processing were restricted to software enhancements. Specifically, we examined the areas of (1) computer memory usage and (2) integer arithmetic.

Computer Memory Usage

In the past 5 years, there has been a dramatic increase in the amount of random access memory (RAM) that can be addressed by personal computers. In the same period, the cost of RAM has decreased precipitously. Specifically, the amount of RAM addressable by personal computers has increased from approximately 2

From the Department of Radiology, University of Colorado Health Sciences Center, Denver, CO.

Supported in part by National Institutes of Health grant no NS-26657, Bethesda, MD.

Examples of codes will be provided by the authors to interested readers.

Address reprint requests to Jonathan P. Pratt, Nuclear Medicine (A034), University of Colorado Health Sciences Center, 4200 E Ninth Ave, Denver, CO 80262.

0897-1889/93/0604-0004$03.00/0

Table 1. Comparison of Minicomputers and Personal Computers

	Minicomputers	Personal Computers
Clock speeds (MHz)	50-100	30-50
Data paths (bits)	64	32
Floating point units	High-speed specialized	Single chip
MIPS	100-300	20-30
Megaflops	10-30	2-3

Abbreviations: MHz, Megahertz; MIPS, million instructions per second; Megaflops, million floating point operations per second.

megabytes to at least 256 megabytes, whereas the cost per megabyte has decreased from $300 to $30.

This increased availability of RAM changes cost versus time trade-offs in various types of image processing. If the result of an involved calculation is used more than once, it can be stored and recalled as necessary, rather than the calculation being repeatedly performed. Therefore, older image processing strategies that required recomputations may not be optimal in situations in which memory usage can be increased greatly.

Trading increased memory usage for reduced computing time has in fact been used in specific circumstances in image processing with minicomputers and workstations.[3-5] Similar solutions should apply to the latest personal computers. Additionally, a memory-intensive computer code that is optimized to the specific microprocessor involved should also allow substantial speed improvements on even the most basic arithmetic operations, such as multiplications and divisions. When such improvements are combined and applied to the steps in complex operations, significant increases in effective processing speed might be achieved.

Integer Arithmetic

The speed of processing on personal computers can also be improved by substituting integer arithmetic for floating-point calculations. For example, consider the Motorola 68030 microprocessor (Motorola Inc, Schaomburg, IL) and the compatible 68882 floating-point unit. A system incorporating these two devices can perform a floating-point addition in 56 clock cycles.[6] However, the same system can perform a 32-bit integer addition in only 2 cycles.[7]

Conveniently, in most medical imaging, image pixel values are acquired as integers, and usually are returned to that format even after floating-point processing. Furthermore, in calculations in which one must deal with fractional values, it may be possible to use fixed-point instead of floating-point arithmetic. With fixed-point arithmetic, one retains the speed advantage of integer arithmetic, while only sacrificing the ability to deal with numbers having very large or very small magnitudes.

The use of integers or fixed-point numbers has another important advantage over the use of floating-point numbers; the range of integer values will often be small enough so that direct calculations can be replaced by table lookups. For example, in an image having an 8-bit depth, each pixel has a value ranging from 0 to 255. If one wanted to multiply every pixel in the image by a constant, eg, 0.3, then one could create a table of 256 elements enumerating all possible integer products. Thus, for each pixel, rather than multiplying by 0.3, the pixel value would be used as an index into the precomputed product table. For the 68030/68882 processors, the savings can be illustrated as follows: a floating point multiplication requires 76 clock cycles[6]; an integer/fixed-point multiplication requires 28 cycles[7]; a table lookup requires only 8 cycles.[7]

MATERIALS AND METHODS

For the reasons previously mentioned, we concentrated on SPECT analysis in the preliminary development of memory-intensive algorithms. Specifically, we explored three image processing functions, ie, convolution, linear interpolation, and filtered backprojection.

The algorithms were implemented as a combination of C and assembly language using Think C (Symantec Corp, Cupertino, CA) as the development environment. The resulting code was tested on a Macintosh Quadra 950 (Apple Computer, Capertino, CA), which contains the Motorola 68040 microprocessor running at 33 MHz. Sixty-four megabytes of RAM were used in the system.

$$\frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Fig 1. Linear weighted-smoothing kernel.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Fig 2. Planar weighted-smoothing kernel.

Convolution

In SPECT analysis, both linear and planar convolution can be used. For example, linear convolution is used to filter the projection data before they are backprojected. In linear convolution, a resulting pixel is computed as the weighted sum of neighboring linear pixels. The kernel is a vector of the weights used in the convolution. The kernel for a linear weighted-smoothing filter is shown in Fig 1.

Planar convolution is used in two-dimensional prefiltering of SPECT projection data and postfiltering of the reconstructed slices. In planar convolution, a resulting pixel is computed as the weighted sum of a square matrix of source pixels. The kernel is a matrix of the convolution weights. The kernel for a planar weighted smooth is shown in Fig 2. In this example, a direct computation of each resulting pixel requires nine multiplications, eight additions, and one division.

The first step in creating a high-speed convolver is stipulating that integer or fixed-point arithmetic will be used throughout. Kernels can be scaled to accomplish this.

Lookup tables are then used to eliminate the multiplications. The sizes of the lookup tables can be limited to the maximum pixel value in the source image. For example, if the maximum pixel in an image has a value of 2,000, the sizes of any lookup tables can be limited to 2,001 elements.

Finally, any symmetry in the kernel can be exploited to minimize the number of lookup tables required. This is illustrated in Fig 3. The weighted smoothing filter has only three unique elements; therefore, only three multiplicative lookup tables are needed. Each table has $p_m + 1$ elements, where p_m is the maximum pixel value in the source image.

Linear Interpolation

One-dimensional linear interpolation of projection vectors can be used in SPECT analysis to create a more accurate backprojection. Two-dimensional linear interpolation can be used to increase the size of reconstructed images without "pixelation," ie, without simply enlarging the source pixels. An example of this effect is given in Fig 4.

A model for planar linear interpolation is shown in Fig 5. This model is based on the relative position of the center of the destination pixel to the centers of the four source pixels. The value of the destination pixel is computed as a distance-weighted sum of the source pixel values.

As with convolution, it is desirable to use lookup tables to eliminate multiplications. The use of lookup tables may at first appear impractical; if each destination pixel value is considered only as the weighted sum of source pixel values, then we must concede that there is usually too much variation in source weights among destination pixels to use any lookup scheme. However, if we consider the possibility of two table lookups per source pixel, then the problem is approachable.

Under these conditions, the worst-case requirement is one lookup table per destination row and one lookup table per destination column. Even this is of dubious use, because the overhead of constructing all the tables could easily exceed the resulting savings, at least for a single image. Fortunately, conditions are usually more favorable. First, if the source and destination images are square, then half as many lookup tables are required. Second, if the interpolation is mapping each image dimension into an integer number of pixels, then the number of lookup tables per dimension simplifies to:

$$\frac{d_d}{GCD(d_d, d_s)}$$

where d_d is the destination dimension, d_s is the source dimension, and GCD indicates the greatest common divisor of two numbers. If the destination dimension is some

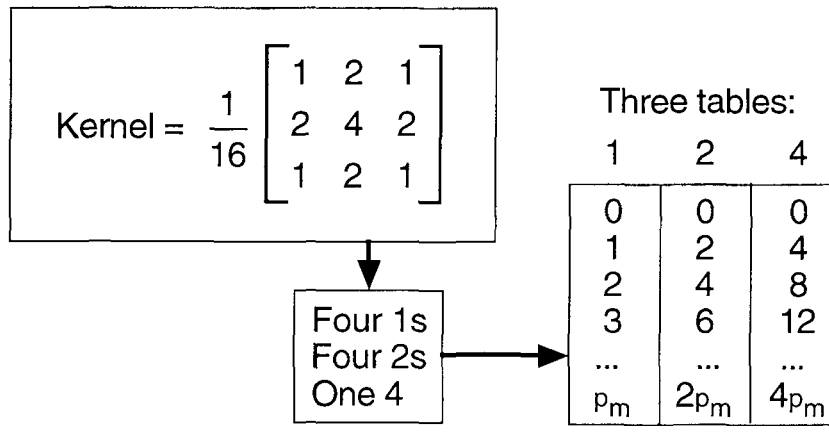


Fig 3. Lookup tables for planar weighted smooth kernel.

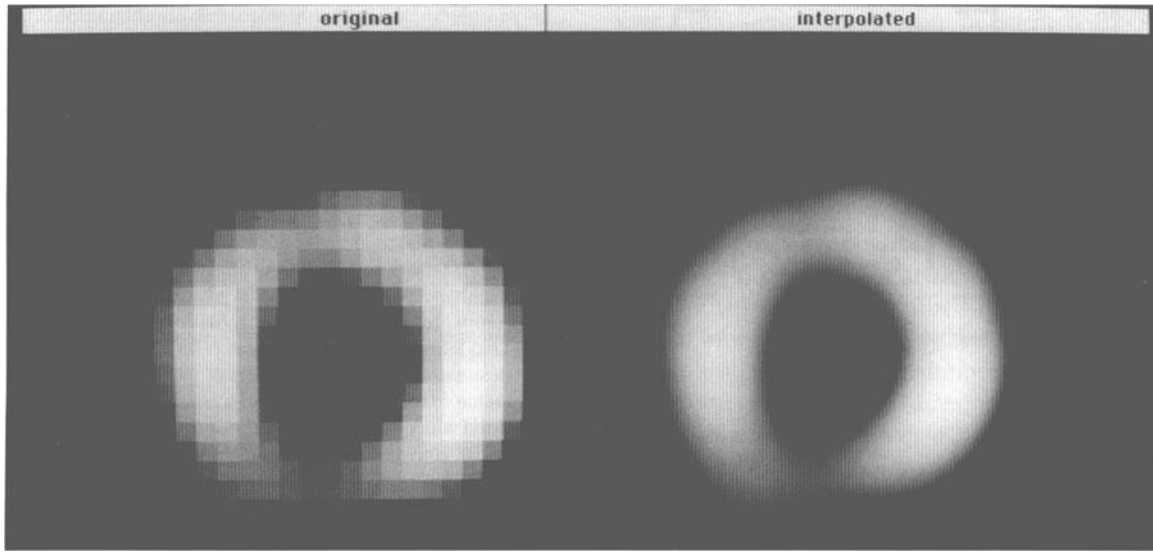


Fig 4. Uninterpolated and interpolated images.

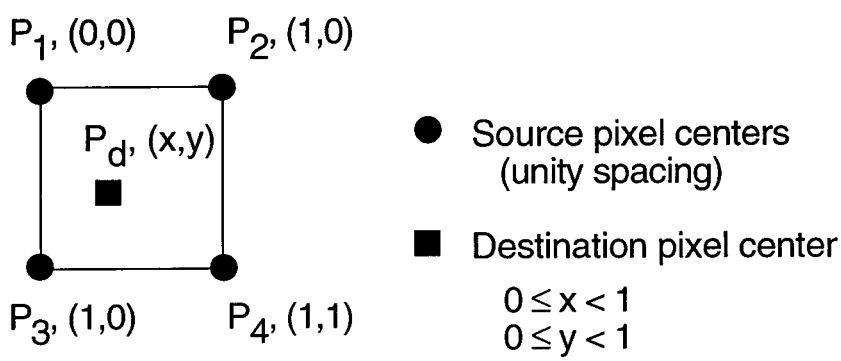
integer multiple of the source dimension, eg, k, then k is also the number of lookup tables needed for that dimension.

Although implementing this interpolation method is nontrivial, the savings make it worthwhile. The interpolation equation can be rearranged so that the computation of each destination pixel requires only six table lookups and six integer additions or subtractions.

Filtered Backprojection

Filtered backprojection used in SPECT reconstruction consists of two basic steps, filtering, and backprojection. The filtering of the data is a linear convolution of the projection vectors, which can be accomplished as previously described. The backprojection is illustrated in Fig 6. The projection vectors are the data acquired from a gamma camera, optionally prefiltered and interpolated. The reconstructed slice is the image matrix into which the projection vectors are backprojected.

Lookup tables are useful in backprojection because several slices are typically reconstructed at the same time from a given data set. It is useful to retain the results of geometric computations so that it is not necessary to repeat the computations for each slice. In the method that we implemented, a lookup table is associated with each pixel of the slice reconstruction matrix. Each table has as many elements as there are projection vectors. Each table element contains an index stating, for the corresponding projection vector, which element of the projection vector should be backprojected (ie, added) to the slice pixel owning the table. A special projection vector index is used to handle cases in which no element should be backpro-



Interpolation equation:

$$P_d = P_1(1-x)(1-y) + P_2x(1-y) + P_3(1-x)y + P_4xy$$

Fig 5. Planar linear interpolation model.

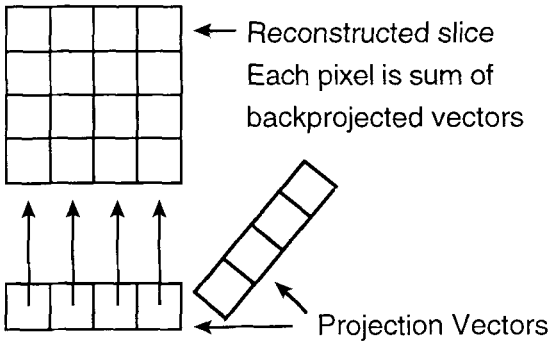


Fig 6. Backprojection element.

jected. This method is very flexible in that it allows arbitrary resolutions and relative scaling for the projection vectors and reconstructed slice.

This method is illustrated by the example in Fig 7. The lookup tables are redefined for more comprehensible two-dimensional viewing. Each table is associated with one of the projection vectors. The tables are two-dimensional arrays, and the positions of their entries correspond to the locations of pixels in the reconstructed slices. As previously mentioned, the table entries are indices into the projection vectors. Thus, every entry in the first column of the 0° table points to the first entry in the 0° projection vector, and so on. Similarly, in the 90° case, all table entries for a particular row point to the projection vector entry adjacent to that row.

RESULTS

With the exception of bilinear interpolation, the memory requirements for the various image analysis functions ranged from 3 to 10 megabytes when applied to 64 frames of 128 by 128 pixel, 12-bit data. Bilinear interpolation required a maximum of 0.2 megabytes plus the memory needed to store the images, typically less than 10 megabytes. Twelve megabytes were required to perform all of the steps of SPECT analysis without reloading or recomputing any data.

The individual tasks were complex enough that their execution times could be measured accurately. Important examples will follow.

Planar Convolution (Two-Dimensional Prefiltering)

A 13 × 13 planar kernel was used to spatially prefilter a set of 64 frames of 128 × 128 pixel images and a set of 32 frames of 64 × 64 pixel images. The operation took 60 seconds for the 128 pixel-wide images and 6 seconds for the 64 pixel-wide images.

Linear Convolution (Backprojection Filtering)

A 27 element linear kernel was used to filter a set of 64 frames of 128 × 128 pixel images and a set of 32 frames of 64 × 64 pixel images. The operation took 15 seconds for the 128 pixel-wide images and 2 seconds for the 64 pixel-wide images.

Backprojection

Sixty-four views of 128 × 128 pixel data and 32 views of 64 × 64 pixel data were backprojected. The operation took 60 seconds for the 128 pixel-wide data and 5 seconds for the 64 pixel-wide data.

Linear Interpolation (Bilinear Smoothing)

A set of 64 slices of 64 × 64 pixel images and a set of 32 slices of 32 × 32 pixel images were interpolated to 256 × 256. The operation took 4

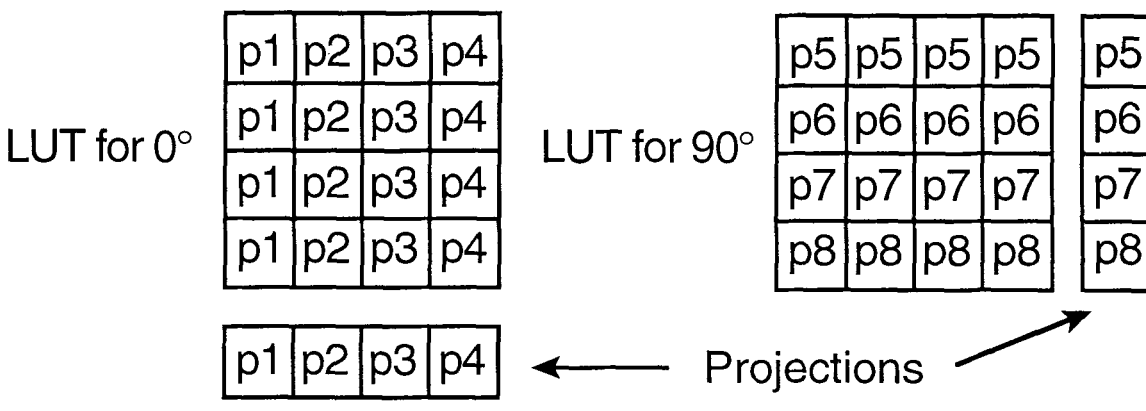


Fig 7. Backprojection lookup tables.

seconds for the 64 × 64 pixel slices and less than 2 seconds for the 32 × 32 pixel slices.

It is also useful to consider the overall time required to process a SPECT data set, starting with the raw projections and ending with the interpolated transverse images. The complete procedure could be performed in approximately 2 minutes for 128 slices of 128 pixel-wide projections. The time drops to 15 seconds for 64 slices of 64 pixel-wide projections.

Therefore, the speeds of performing both the constituent operations and the overall SPECT processing are rapid. In fact, they are comparable to those of many dedicated commercial systems. For example, using the reconstruction of 64 frames of 128 × 128 pixel data as a benchmark, we found that a speed of approximately 1 second per slice could be achieved on the Quadra 950. This compares to values of 1 second per slice on the hardware-supported Siemens ICON (Siemens Medical Systems Inc, Iselin, NJ) of 1 second per slice on the workstation-based ADAC Pegasys (ADAC Laboratories, Milpitas, CA), and of 0.3 seconds per slice on the supercomputer-based Picker Odyssey (Picker International Inc, Cleveland, OH).

DISCUSSION

Integer arithmetic and lookup tables clearly are useful tools for improving the image processing performance of the latest generation of personal computers (Intel 80486-based [Intel Corp, Santa Clara, CA] or Motorola 68040-based). The memory requirements (approximately 12 megabytes in our tests) are modest relative to the amounts of memory that personal computers can now use, and the cost for this amount of RAM is less than $1000. When extrapolating the requirements for other types of radiological imaging that use 512 × 512 pixel images as opposed to 128 × 128 pixel images, memory requirements are increased, but estimated requirements of 64 to 128 megabytes still are achieved easily with advanced personal computers.

Importantly, we have shown that effective processing speeds can be achieved on advanced personal computers that are compatible with most current dedicated computers. The Quadra 950 achieved benchmark speeds equal to workstation-based and hardware-assisted systems and speeds within a factor of three of a supercomputer-based system. In other words, clinically useful image processing can be performed with a relatively low hardware cost.

Disadvantages of this memory-intensive approach are primarily implementation issues and not end-user problems. As we have shown, implementation of time-saving algorithms can be complex compared with a straightforward use of the relevant equations. Additionally, better results are obtained when one gets "closer" to the underlying hardware. This means that one must worry about underflows and overflows in arithmetic operations. Also, the fastest routines are written in assembly language, which substantially reduces the portability of the code. However, as personal computers tend to evolve by using new processors (eg, Motorola 68040) whose microcode is compatible with that of previous processors (eg, Motorola 68030), code portability within a particular line of computers is supported.

Beyond the SPECT analysis examined in this study, lookup table-based algorithms have other important uses in two- and three-dimensional radiological imaging. For example, in gamma camera, ultrasound, and magnetic resonance imaging (MRI) data acquisition, tables can be used for fast image framing. Also, lookup table-based algorithms should be useful in applications in which SPECT, position emission tomography (PET), computed tomography (CT), and MRI images are correlated, eg, volume rotations or reorientations, when interpolation of the data cube is desired.

We conclude that the general implementation of these new memory intensive algorithms will dramatically change the types of applications in digital imaging for which personal computers are appropriate. The implementation is straightforward for programmers, once the concept of trading memory for speed is fully appreciated (see Appendix A for examples). The resulting improvement in performance achieved by personal computers, combined with their low cost and high flexibility, will result in their increasingly being used as the main computers in medical image acquisition and processing systems.

APPENDIX A. SAMPLE 68040 CODE FOR LINEAR INTERPOLATION

This code fragment illustrates how calculating a linearly interpolated pixel can be performed with a few assembly language statements. This code is primarily intended for high speed screen display; overflow and roundoff issues make it less suitable for uses when high accuracy is required.

The code fragment scans two adjacent rows of a source image to generate a destination row with more pixels than either of the source rows. The data registers p1, p2, p3, and p4 contain source pixels with this geometry:

p1 p2
p3 p4

That is, p1 and p2 are from the upper row, and p3 and p4 are from the lower row. k1 and k2 are scratch registers.

The code is driven by a state table, which contains pointers to fraction lookup tables and jump vectors for each destination pixel. The fraction lookup tables are used to avoid multiplications in the calculation. Each table is 256 bytes long, and contains all possible results of multiplying a pixel value byte a fraction f, $0 \leq f < 1$. The four jump vectors for row calculations are:

Start-of-row—This causes a duplication of the first pixel in each row. It is the first entry in the row state table.

End-of-row—This causes a duplication of the last pixel in each row. It is the last entry in the row state table.

Horizontal-source-increment—This entry point is used to advance the pixel values of the source image. To illustrate, suppose the result image has twice the width of source image. Then the source pixels are incremented horizontally once for every two destination pixels.

No-source-increment—This is the entry point for the inner loop. This is where the computation of every destination pixel takes place.

```
@Start_of_row
  Move.B (upper_row_pointer) +, p2
  Move.B (lower_row_pointer) +, p4
@End_of_row
  Move.B p2, p1
  Move.B p4, p3
  Bra.S @No_source_increment
@Horizontal_source_increment
  Move.B p2, p1
  Move.B p4, p3
  Move.B (upper_row_pointer) +, p2
  Move.B (lower_row_pointer) +, p4
@No_source_increment
  Move.B p1, k1
  Move.B p3, k2
  Sub.B (horizontal_fraction_table, p1.W), k1
  Sub.B (horizontal_fraction_table, p3.W), k2
  Add.B (horizontal_fraction_table, p2.W), k1
  Add.B (horizontal_fraction_table, p4.W), k2
  Sub.B (vertical_fraction_table, k1.W), k1
  Add.B (vertical_fraction_table, k2.W), k1
  Move.B (final_transform, k1), (dstination_pointer)+
  MoveA.L (row_state_table)+, horizontal_fraction_table
  Move.L (row_state_table)+, D0
  Jmp (D0.L) ; Jump to next entry point
```

REFERENCES

1. Lear JL, Pratt JP, Roberts DR, et al: Gamma camera image acquisition, display, and processing with the personal microcomputer. Radiology 175:241-245, 1990

2. Lear JL, Manco-Johnson M, Feyerabend AJ, et al: Ultra-high speed teleradiology using ISDN technology. Radiology 171:862-863, 1989

3. Miller TR, Wallis JW: Fast maximum-likelihood reconstruction. J Nucl Med 33:1710-1711, 1992

4. Shepp LA, Vardi Y: Maximum likelihood reconstruction for emission tomography. IEEE Trans Med Imaging 1:113-122, 1982

5. Kaufman L: Implementing and accelerating the EM algorithm for positron emission tomography. IEEE Trans Med Imaging 6:37-51, 1987

6. Motorola, MC68881/MC68882 Floating-Point Coprocessor User's Manual, ed 1. Englewood Cliffs, NJ, Prentice Hall, 1987

7. Motorola, MC68030 Enhanced 32-Bit Microprocessor User's Manual, ed 3. Englewood Cliffs, NJ, Prentice Hall, 1990