(12) PATENT
(19) AUSTRALIAN PATENT OFFICE

(11) Application No. AU 199671890 B2
(10) Patent No. 704659

(54) Title
Method of transposing data

(51)[6] International Patent Classification(s)
G06F 007/14 G06T 003/40

(21) Application No: 199671890

(22) Application Date: 1996.11.20

(30) Priority Data

(31) Number	(32) Date	(33) Country
08/562695	1995.11.27	US

(43) Publication Date : 1997.06.05
(43) Publication Journal Date : 1997.06.05
(44) Accepted Journal Date : 1999.04.29

(71) Applicant(s)
Sun Microsystems, Inc.

(72) Inventor(s)
Daniel S. Rice

(74) Agent/Attorney
DAVIES COLLISON CAVE,1 Little Collins Street,MELBOURNE VIC 3000

(56) Related Art
US 4837845
EP 200282

AU9671890

(12) PATENT ABSTRACT (11) Document No. AU-A-71890/96
(19) AUSTRALIAN PATENT OFFICE

(54) Title
METHOD OF TRANSPOSING DATA

(51)[6] International Patent Classification(s)
G06F 007/14 **G06T 003/40**

(21) Application No. : **71890/96** (22) Application Date : **20/11/96**

(30) Priority Data

(31) Number **562695** (32) Date **27/11/95** (33) Country **US UNITED STATES OF AMERICA**

(43) Publication Date : **05/06/97**

(71) Applicant(s)
SUN MICROSYSTEMS, INC.

(72) Inventor(s)
DANIEL S. RICE

(74) Attorney or Agent
DAVIES COLLISON CAVE , 1 Little Collins Street, MELBOURNE VIC 3000

(57)

A method of transposing data. Either eight bit or sixteen bit data is placed in a buffer. Each buffer is defined to contain one or more sub-buffers. Rows of the sub-buffer are selectively interleaved with the results of the selective interleaving being again interleaved in a specific order. Successive interleavings create the transpose of the original sub-buffer.

ABSTRACT

A method of transposing data. Either eight bit or sixteen bit data is placed in a buffer. Each buffer is defined to contain one or more sub-buffers. Rows of the sub-buffer are selectively interleaved with the results of the selective interleaving being again interleaved in a specific order. Successive interleavings create the transpose of the original sub-buffer.

AUSTRALIA

PATENTS ACT 1990

COMPLETE SPECIFICATION

NAME OF APPLICANT(S):

Sun Microsystems, Inc.

ADDRESS FOR SERVICE:

DAVIES COLLISON CAVE
Patent Attorneys
1 Little Collins Street, Melbourne, 3000.

INVENTION TITLE:

Method of transposing data

The following statement is a full description of this invention, including the best method of performing it known to me/us:-

BACKGROUND OF THE INVENTION

The invention relates to transposing data stored in a memory of a computer. More specifically, the invention relates to a method for performing transposition of arbitrary buffered data using an efficient interleaving technique.

Resizing graphical data is generally well-known in the art. Two commonly used approaches are forward mapping and backward mapping. In a forward map, the algorithm asks: for each source pixel, where does it land in the destination image? Conversely, for a backward map: the question is for each destination pixel, where does it come from in the source image? A forward map requires arbitration between pixels landing in the same destination, while a backward map requires a certain amount of work for each pixel to determine which source image pixel or pixels it came from, and then it is never necessary to revisit that pixel again.

In a backward mapping scheme, once it has been established for each destination pixel, where it comes from in the source or sources, it is necessary to determine for each source pixel contributing to the destination pixel, a co-efficient of contribution of that source pixel. Typically, for cases where there is more than one source pixel contributing, either two or four pixels are deemed to provide the contribution. This corresponds to bi-linear, and bi-cubic filtering, respectively. The calculation of these coefficients is generally well-known in the art and while computationally intensive, it need be done only once for each row and once for each column of pixels in the source image. The calculated coefficients are then stored in an array to be accessed during subsequent processing. The above-described scheme

will allow one to do either pure horizontal or pure vertical resizing. However, in the event that it were desirable to do both vertical and horizontal resizing at the same time, it would, of course, be necessary to choose some square of source pixels and weight them all with appropriate coefficients. This would require a pixel-by-pixel coefficient calculation and manipulation. By choosing the filter coefficients correctly, it is possible to do stretching in a separable way, i.e. first horizontal stretching and then vertical, or vice versa. Such techniques are well-known in the art.

Coefficients are selected by choosing a geometrical point that is the center of the destination pixel and establishing the function that maps points in the source to the destination, thereby generating a scaled translation where the destination is a linear function in x and y. The inverse of the function yields the point in the source from which the data comes. If the selected point is not the center of the source pixel, it is necessary to interpolate the source data to determine the relative distances between various source pixel centers and weight them accordingly. Thus, the filter coefficients are a function of subpixel position of a backward mapped destination point.

In "Resampling Algorithms for Image Resizing and Rotation" Proc. SPIE Digital Image Processing Applications, vol. 1075, pp. 260 - 269, 1989, Joseph Ward and David R. Cok provide an algorithm for quantizing the subpixel positions down to e.g. 1/32nd or 1/256th of a pixel and providing ready-made tables of filters which apply for a pixel in that area. Each area is known as a bin. Thus, for each bin, a different filter coefficient is established.

Graphical data is commonly stored in band interleaved format where, for example, three successive bytes in memory represent red, green, and blue

components of a single pixel. This complicates the issue where it is desired to multiply the adjacent pixel in memory by a filter, since for an arbitrary section of image data, it is not clear whether four bytes represent four pixels or 1-1/3 pixels.

It is relatively straight forward to write a program in C or a C variant to read individual source pixels from a source image and resize them individually. Unfortunately, there is no straightforward way to operate on multiple adjacent pixels of source data. Operating on multiple pixels of adjacent source data is further complicated by the fact that the data may well be stored in unknown formats as described above. Transposition of arbitrary data in a buffer has numerous applications. Transposition of data in a buffer is generally computationally intensive because each output row depends on every input row. Unfortunately, this makes transposition too slow to be useful in many cases.

Therefore, it would be desirable to able to operate on multiple bytes simultaneously in connection with image resizing. Moreover, it is desirable to develop an efficient transposition regime in which multiple bytes of data can be operated on simultaneously.

In accordance with the present invention there is provided a method of transposing data in a buffer comprising the steps of:

a) loading data into a buffer having a plurality of rows;

b) selectively merging pairs of rows to produce a plurality of intermediate merge results wherein each merge operation interleaves two operands of equal length to yield an output twice the length of each operand;

c) selectively merging pairs within said plurality of intermediate merge results to produce a new plurality of intermediate merge results;

d) repeating step c) until transposed rows are generated; and

e) storing the transpose rows in one of the buffer or a second buffer, thereby producing a transposed buffer.

The present invention also provides a method of transposing a buffer of data in sixteen bit representation comprising the steps of:

a) interleaving a plurality of high order bytes of a buffer row with a plurality of high order bytes in a buffer row two rows distant;

b) interleaving a plurality of low order bytes of a buffer row with a plurality of low order bytes in a buffer two rows distant;

c) repeatedly interleaving results of the interleaving steps in a predetermined manner to yield transposed rows, wherein the interleaving steps separate constituent bytes of each sixteen bit datum of the rows being interleaved and subsequent interleavings reassemble the 16-bit data; and

d) storing a plurality of transposed rows in one of the buffer or a second buffer.

The present invention further provides a computer program product which includes a computer usable medium having a computer usable code embodied therein for causing transposition of data in a buffer in a computer system having a processor coupled to a memory by a bus, the computer program product comprising:

computer readable program code device configured to load data into a buffer having a plurality of rows;

computer readable program code device configured to selectively merge pairs of rows to produce a plurality or intermediate merge results, each merge interleaving two operands of a size to yield a result equal in size to a sum of the operand sizes;

computer readable program code device configured to selectively merge pairs within said plurality of intermediate merge results to produce a new plurality of intermediate merge results;

computer readable program code device configured to repeat the selective merging of intermediate merge results until transposed rows are generated; and

computer readable program code device configured to store the transposed rows in one of a buffer or a second buffer, thereby producing a transposed buffer.

The invention is described in greater detail hereinafter, by way of example only, through

several embodiments thereof and with reference to the accompanying drawings, in which:

Figure 1 is a block diagram of a computer system employing the invention;

Figure 2a is a block diagram of the buffering scheme in one embodiment of the invention;

Figure 2b is a diagram showing the way buffers are filled in an embodiment of the invention;

Figure 2c is a diagram shown the way buffers are filled in an alternate embodiment;

Figure 3a is a sample sub-buffer of the invention and its transpose in accordance with the invention;

Figure 3b is a diagram of sub-buffer transposition in parts in accordance with one embodiment of the invention;

Figure 3c is a transposition tree for half a sub-buffer in one embodiment of the invention;

Figure 3d is a sample sub-buffer and its transpose in 16-bit representations; and

Figure 3e is a sample sub-buffer in an alternate embodiment.

A method for resizing images in a computer system is disclosed. A plurality of buffers is dynamically created in the memory of the computer system. Among the created buffers are a horizontal sampling buffer and two vertical sampling buffers. The horizontal sampling buffer is filled with data from a plurality of rows from the source image. Such filling can be performed several bytes at a time.

The horizontal sampling buffer is typically eight bytes wide and as long as the width of a source image row. Basically, a first source image row is placed in the horizontal sampling buffer eight bytes in row 0, then skip seven rows and put the next eight bytes in the buffer row 8, etc. until the entire source row is in the horizontal sampling buffer. The process is repeated for each successive source image row with an offset of one from its predecessor row until eight source rows are in the horizontal sampling buffer, eg the second source row is placed in buffer rows 1, 9, 17, etc. Thus, the vertical neghbors in the source image are vertical neighbors in the horizontal sampling buffer. The horizontal sampling buffers are composed of sub-buffers, each

source constituting eight bytes of eight adjacent source image rows. Each sub-buffer is bytewise transposed to provide easy access to different channels of the image data. The transposition exposes the bands of a band interleaved format for easy processing. A sixteen-bit representation transposition scheme buffers four rows at a time and transposes 4x8 sub-buffers with high and low order bytes of each pixel ending in the correct relation.

Once the data is transposed, a filter is then applied to the transposed data with the result being stored in another buffer, either an intermediate buffer or vertical sampling buffer. However, if the data is placed directly into the vertical sampling buffer, then transposition in place will be required. Thus, transposition is simplified by employing an intermediate buffer. The filtered data is then retransposed to assume its original configuration. The retransposed data is stored in a vertical sampling buffer previously created. The foregoing steps are repeated to fill a second vertical sampling buffer. Subsequently, throughout the processing of the source image, two vertical sampling buffers remain continuously filled to allow vertical filtering at high speed. By ping-ponging between the buffers, valid data for the vertical filtering is assured. When data is no longer needed in one of the vertical sampling buffers for the ongoing filtering, it can be refilled with new data corresponding to the next group of vertical neighbors. The filtered data may need to be converted from sixteen bit values to eight bit values clamped at the extremes. Once the conversion is complete, the resized image can be output to the screen or stored in memory for latter processing.

The ULTRASPARC® of Sun Microsystems visual instruction set supports several instructions which facilitate processing using the method described. Using the visual instruction set ("VIS"), eight bytes are read or written simultaneously. Similarly, VIS allows operations with four byte operands. Certain instructions such as the fpmerge instruction allow simplified transposition over prior methods. By aligning the data in buffers, alignment edge condition issues can be addressed early in the resizing scheme to allow more efficient use of processing resources by eliminating checks and branches in inner loops. Similarly, while the system must know the format of the source and destination data, all format specific issues can be handled in an external loop. Since horizontal and vertical processing can be done separately, multi-

threading in a multi-processor environment is easily accomplished.

In the following description, for purposes of explanation, specific applications, numbers, materials, and configurations are set forth in order to provide full understanding of the present invention. However, it would be apparent to one skilled in the art that the present invention may be practiced without the specific details. In other instances, well known systems are shown in diagrammatical or block diagram form in order not to obscure the present invention unnecessarily.

Figure 1 shows a block diagram of a computer system for use in connection with the invention. CPU 10 is coupled by bus 12 to main memory 11 and cache memory 13. Main memory 11 is composed of cacheable memory 15 and uncacheable memory 14. Sampling buffers 16 of the instant invention reside in cacheable memory 15. Filter co-efficient arrays 17 also reside in cacheable memory 15. It is desirable that sampling buffers 16 be dynamically allocatable to increase flexibility of the system. In one exemplary embodiment the CPU 10 is the ULTRASPARC® processor of Sun Microsystems which employs the visual instruction set ("VIS"). The ULTRASPARC® processor and VIS are discussed more fully in United States Patent 5,734,874 entitled A CENTRAL PROCESSING UNIT WITH INTEGRATED GRAPHIC FUNCTIONS, the disclosure of which is incorporated herein by reference.

Figure 2a shows the buffering scheme of the exemplary embodiment of the invention. A horizontal sampling buffer 20, 8 bytes wide and N rows long, is used to accept eight rows of image data from the source image. Thus, N is equal to the byte length of a source image row, which may be greater than or equal to the pixel length of the source image row. Alignment issues and edge conditions are dealt

with as the data is moved from memory into the horizontal sampling buffer. Accommodation of edge conditions and alignment issues is explained more fully below. The data in the horizontal sampling buffer is transposed and placed to expose bands for processing. The transposition will also be explained more fully below. A horizontal filter 25 is applied to the transposed data in the horizontal sampling buffer with the result of the filtering being placed in an intermediate buffer 23, 8 bytes wide and M rows long. M is equal to the length of a row in the destination image. The data in the intermediate buffer is retransposed into either a first vertical sampling buffer 21 or a second vertical sampling buffer 22. Controller 24 dictates which buffer receives the current data from intermediate buffer 23. A vertical filter 26 is applied to the data in the vertical sampling buffers 21,22. In one exemplary embodiment, application of the vertical filter involves taking eight bytes from each relevant row in the vertical sampling buffer. Relevant rows are dictated by the filter width, e.g. a filter width of four results in four relevant rows.

Parallel multiplications and additions are performed on this data to yield vertically sampled data, which may need to be converted as explained below. VIS provides fmul8x16 and fpadd16 instructions ideal for use in the parallel multiplication and addition of this vertical filtering scheme. The fpadd16 instruction accepts two arguments, each containing four 16-bit quantities resulting in four 16-bit quantities. All quantities are considered signed. The fmul8x16 instruction accepts two arguments, the first argument is four 8-bit values, the second is either a single or four 16-bit values and produce four 16-bit results. Each byte of the first argument is multiplied by either the single or the corresponding second argument. The single second argument is derived from the top or bottom half of a 32-bit word. The effect of the multiply instruction is (x*y+128)/256. Two other variants of fmul are used to approximate a 16x16-bit multiply. These variants fmul8sux16 and fmul8ulx16 both take two arguments each consisting of signed 16-

bit data. The fmul8sux16 instruction multiplies its second argument by the top 8 bits of its first argument to produce a 24-bit intermediate result which is rounded and truncated to 16 bits. The fmul8ulx16 instruction does the same, only using the lower half of each partition of its first argument and implicitly shifting its result right by an additional 8 bits to match the significance of the first product.

The backwards mapping scheme is such that as processing proceeds downward in the source image, it also proceeds strictly downward in the source image. Therefore, once the vertical filter 26 needs data beyond the second vertical sampling buffer 22, the data in the first sampling buffer 21 will never be revisited. Therefore, the first sampling buffer can be immediately refilled with new data. This assumes that the filter width is less than or equal to the number of destination rows in the buffer, in this example, eight. As the vertical filter 26 transitions through the second vertical sampling buffer 22, it re-enters the first vertical sampling buffer to process the new data. Once the vertical filter 26 has made the transition from the second sampling buffer back to the first vertical sampling buffer 21, the second vertical sampling buffer 22 can be refilled. Thus, controller 24 toggles the data directed from first to second sampling buffer 22 responsive to the transition of the vertical filter 26 from one buffer to the next. The controller 24 then ping-pongs between the vertical sampling buffers as needed data is no longer available. It will be understood by one of ordinary skill in the art that additional buffers could be used without departing from the scope or contemplation of the invention. The maximum vertical filter width in this ping-ponging arrangement is dictated by the number of buffered horizontally sampled rows in the vertical sampling buffer. Accordingly, in an embodiment having 8-bit representations with 8 source rows buffered at a time, the maximum filter width is eight, while in a 16-bit embodiment, only 4 rows are buffered in each buffer, and the maximum filter width is four. It will be understood by one of ordinary skill in the art that additional vertical

sampling buffers could be introduced to allow for greater filter width, and such is within the scope and contemplation of the invention.

Since an application of the filter may not yield an eight bit value, converter 27 converts sixteen bit outputs of the vertical filter 26 to an eight bit entry for the M byte output row 28. Moreover, converter 27 clamps the extremes at 0 and 255. When the invention is used with the ULTRASPARC® visual instruction set ("VIS"), the fpack instruction fulfills this converting function.

Significantly, because the horizontal sampling and vertical sampling are independent of each other, additional horizontal sampling can be performed simultaneously with the vertical sampling of previously horizontally sampled data. It is also within the scope and contemplation of this invention to provide additional intermediate buffers 23 so as not to create a bottleneck at the intermediate buffer 23. This arrangement is ideal for multi-threading as the workload can readily be distributed amongst multiple processors. Thus, as soon as the horizontal sampling buffer has been wholly sampled into the intermediate buffer, it can be refilled, and processing can begin again.

Figure 2b shows an example source image 30 having eight rows, each of twenty-four bytes. Horizontal sampling buffer 20 is filled with, for example, the first eight bytes of row 31 going into row 0 of the horizontal sampling buffer 20, bytes 9 - 16 of row 31 going to row 8 of the horizontal sampling buffer 20, and bytes 17 - 24 going to row 16 of the horizontal sampling buffer 20. Similarly, row 32, bytes 1 - 8 go to row 1 of the 20, bytes 9 - 16 going to row 9, and bytes 17 - 24 go to row 17 horizontal sampling buffer. One of ordinary skill in the art will recognize that rows maybe horizontal or vertical however, because references to horizontally adjacent memory locations are efficiently supported by most computer system rows in this application are generally regarded to be horizontal. However, in system supporting efficient vertically adjacent accesses, the invention would work equally on vertical rows.

The remaining source rows follow the same pattern. If the source image 30 were composed of additional rows, they would be processed on a subsequent filling of the horizontal sampling buffer 20. If the sample image 30 had rows of greater length, the length of the horizontal sampling buffer would be increased proportionally. Significantly, the length of the horizontal sampling buffer is equal to the next multiple of 8 greater than or equal to the length of a source image line. For example, a source image having lines between 25 and 32 bytes would each result in a horizontal sampling buffer of 32 line length. Each iteration of processing of the horizontal sampling buffer processes 8 complete lines of the source image. An exception exists for the last iteration. If fewer than eight lines of valid data remain, only the valid data is processed.

In the case where the pixels are in a 16-bit representation, a variation on the above-described buffering is required. **Figure 2c** shows an alternate embodiment in which the source image is twenty-four bytes wide, and the pixels are in a 16-bit representation, i.e. 1 and 1' are the high and low order bytes of a channel of a single pixel. In this case, each sub-buffer of the horizontal sampling buffer is four rows long. This implies that four rows of the source image are processed on each iteration. The four by eight buffer can be transposed while maintaining the correct relation of the high and low order bits of each pixel represented. The transposition of 16-bit data is discussed below in connection with Figures 3d and 3e.

Edge conditions are a well known concern in sizing algorithms. Basically stated, the problem is if you need a pixel to the left of the left most pixel, what do you do? By padding of the horizontal sampling buffers, edge conditions can be easily satisfied. For example, if we need two edge pixels, begin by placing the data at the third position of each row in the horizontal sampling buffer. Then entries in the first and second byte can be synthesized either with copies of the first data entered in the third byte or, for example, zero. Alignment is another issue that can

be addressed early in the buffering scheme. Basically, there is no guarantee a source image row will begin aligned with a memory word. Thus, where the system retrieves multiple bytes, e.g. 8, simultaneously, there is no assurance that the source row will begin at an even multiple of eight. Thus, the first retrieval may include data not within the source image which the user does not want to buffer. VIS provides the instructions falignddata and alignaddr which used in conjunction allow the data to be taken from memory and easily aligned in the horizontal sampling buffer through use of an internal register, the graphics status register ("gsr"). For example, if a group of eight bytes r has five bytes of junk followed by three bytes of valid image data and another group s contains eight valid bytes, alignaddr (0,5) sets the gsr align bits to five. Subsequent use of the falignddata instruction on r and s will yield eight bytes of valid data. Thus, where t contains the next valid group of eight bytes of image data, falignddata (r, s) and falignddata (s, t) will yield $r_{5\text{-}7}\ s_{0\text{-}4}$ and $s_{5\text{-}7}\ t_{1\text{-}4}$, respectively. An analogous issue exists in writing the results of a vertical pass into a possibly unaligned destination. The falignddata instruction can be used to resolve destination alignment issues as with the source data.

Transposition is performed within the horizontal sampling buffer 20 in eight row blocks, e.g. rows 0 - 7 are transposed, rows 8 - 15 are transposed, rows 16 - 23 are transposed in the example of Figure 2b. **Figure 3a** shows such a transposition. Such bytewise transposition exposes data band by band every time. For example, if in Figure 3a, RGB band interleaved format were present, 0, 3, and 6 would correspond to the red band; 1, 4, and 7 to the green band; and 2 and 5 would correspond to the blue band. Generally, transposition is inefficient because every row of the output depends on every row of the input. Serendipitously, VIS includes a functionality via the fpmerge instruction where two 4-byte words can be merged to form a group of 8 (double word) bytes. The fpmerge instruction applied to $a_0\ a_1\ a_2\ a_3$ and $e_0\ e_1\ e_2\ e_3$ yields $a_0\ e_0\ a_1\ e_1\ a_2\ e_2\ a_3\ e_3$. **Figure 3c** shows a transposition tree using the merge

functionality to effect the transposition. As shown in Figure 3c, within an 8 x 8 segment, each word is merged with a word four rows distant. This is the effect of transposing an 8 x 8 block as two 4 x 8 blocks as shown in Figure 3b. The horizontal sampling buffer can be transposed in place using this method or the data can be initially placed in a pre-buffer and transposed into the horizontal sampling buffer 20. It is anticipated that future computer systems will support the functionality of the fpmerge instruction of the VIS. Accordingly, this method of transposition and image sizing is not intended to be limited to the ULTRASPARC® system.

Figure 3d shows transposition of a sub-buffer when the data is in a sixteen bit representation. **Figure 3e** shows the transposition tree of a sixteen-bit representation embodiment. Significantly, the fpmerge instruction allows reference to high (hi) and low (lo) order bytes of an eight byte string independently. By exploiting this feature in each initial buffer row p0-p3 and with respect to each intermediate merge result, the transpose of the 4x8 buffer is easily available. Initially, the high and low order bytes of each row of a sub-buffer are merged with the corresponding bytes of a row two rows distant, e.g. row 0 hi is merged with row 2 hi and row 0 lo is merged with row 2 lo. By continuing, as shown, appropriate merging of hi and lo portions of the intermediate merge results; the four branches of the tree will yield the four rows of the original sub-buffer's transpose with high and low order bytes of each pixel correctly aligned.

Significantly, it is essential that the data be appropriately aligned prior to conducting the transposition. Thus, transposing the data in memory incurs a performance penalty because (1) care must be taken to avoid overwriting the source image data, and (2) because in memory, there is no assurance of proper alignment, e.g. the first word of a source row may begin at an arbitrary point in a memory word, and there is no assurance that successive source rows will be aligned the same

relative to a memory word. The above-described buffering scheme appropriately aligns the data so that each 8 x 8 block is aligned for transposition.

Once the data has been transposed and horizontally sampled into the intermediate buffer 23, it is then retransposed using the same transposition scheme into one of the vertical sampling buffers 21,22. It will be recognized by one of ordinary skill in the art that it is possible to transpose the data in place within the intermediate buffer 23. Accordingly, it would be possible to eliminate the intermediate buffer 23 entirely and transpose the data in place in one of the vertical sampling buffers 21,22.

If it is the desire to expand fewer than all the bands in the source image, it is possible to do so by stepping through the transposed data appropriately. For example, if the data is in red, green, blue format, and one only wishes to expand the blue band, the filtering would begin with row 2 and step by 3 to fill the intermediate buffer. This results in an intermediate buffer having valid blue data and junk in the red and green data locations. A bit mask may be maintained for channel selection to prevent the junk from being written to the vertical sampling buffer. The mask should be at least eight bits long and an even multiple of the number of output channels. After each application, the mask is rotated by eight and applied to the next eight columns in the intermediate buffer.

Sometimes it is desirable to put the output into a different format than the source format. For example, the frame buffer employed to output the image to the monitor may be of the format xbgr, while the source is in an rgb format. In such case, as the bands are processed horizontally, they are placed in the intermediate buffer such that the retransposition into the vertical sampling buffer will result in the desired ordering. For example, red data will be moved from the first entry in the horizontal sampling buffer to the entry of the intermediate buffer which will be in the fourth slot of the vertical sampling buffer. A mask is used to mask out, e.g. the

X rows of the vertical sampling buffer, which do not represent valid data from the source image. The VIS supports partial stores and edge instruction which are used in an exemplary embodiment to effect desired output formatting.

In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will however be evident that various modifications and changes can be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. Therefore, the scope of the invention should be limited only by the appended claims.

Throughout this specification and the claims which follow, unless the context requires otherwise, the word "comprise", and variations such as "comprises" and "comprising", will be understood to imply the inclusion of a stated integer or step or group of integers or steps but not the exclusion of any other integer or step or group of integers or steps.

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. A method of transposing data in a buffer comprising the steps of:

a) loading data into a buffer having a plurality of rows;

b) selectivcly merging pairs of rows to produce a plurality of intermediate merge results wherein each merge operation interleaves two operands of equal length to yield an output twice the length of each operand;

c) selectively merging pairs within said plurality of intermediate merge results to produce a new plurality of intermediate merge results;

d) repeating step c) until transposed rows are generated; and

e) storing the transpose rows in one of the buffer or a second buffer, thereby producing a transposed buffer.

2. Thc method of claim 1 wherein the data is in an 8-bit representation.

3. The mcthod of claim 2 wherein for each 8 buffer rows, the step of selectively merging comprises the steps of:

merging a first row with a fifth row;

mcrging a sccond row with a sixth row;

merging a third row with a seventh row; and

merging a fourth row with an eighth row.

4. The method of claim 3 wherein the step of selectively merging is performed in parts such that a first portion of each respective row is merged independently of a second portion of thc respective row.

5. The method of claim 1 wherein the data is in a 16-bit representation wherein a first and a second constituent byte of each 16-bit datum are separated during selective merging and reassembled by subsequent merges such that thc transposed buffer is a transpose of 16-bit data.

6. The method of claim 5 wherein the step of sclectively merging is applied separately to high and low order bytes of each buffer row.

7. The method of claim 6 wherein for each four buffer rows, the step of selectively merging comprises the steps of:

merging a plurality of high order bytes of a first row with a plurality of high order bytes of a third row;

merging a plurality of high order bytes of a second row with a plurality of high order bytes of a fourth row;

merging a plurality of low order bytes of a first row with a plurality of low order bytes of a third row;

merging a plurality of low order bytes of a second row with a plurality of low order bytes of a fourth row.

8. The method of claim 1 wherein each row contains 8 bytes.

9. A method of transposing a buffer of data in sixteen bit representation comprising the steps of:

a) interleaving a plurality of high order bytes of a buffer row with a plurality of high order bytes in a buffer row two rows distant;

b) interleaving a plurality of low order bytes of a buffer row with a plurality of low order bytes in a buffer two rows distant;

c) repeatedly interleaving results of the interleaving steps in a predetermined manner to yield transposed rows, wherein the interleaving steps separate constituent bytes of each sixteen bit datum of the rows being interleaved and subsequent interleavings reassemble the 16-bit data; and

d) storing a plurality of transposed rows in one of the buffer or a second buffer.

10. The method of claim 9 whcrein the row is each of a first and a second row in a group of four rows.

11. The method of claim 10 wherein each row contains eight bytes.

12. A computer program product which includes a computer usable medium having a computer usable code embodied therein for causing transposition of data in a buffer in a computer system having a processor coupled to a memory by a bus, the computer program product comprising:

computer readable program code device configured to load data into a buffer having a plurality of rows;

computer readable program code device configured to selectively merge pairs of rows to produce a plurality or intermediate merge results, each merge interleaving two operands of a size to yield a result equal in size to a sum of the operand sizes;

computer readable program code device configured to selectively merge pairs within said plurality of intermediate merge results to produce a new plurality of intermediate merge results;

computer readable program code device configured to repeat the selective merging of intermediate merge results until transposed rows are generated; and

computer readable program code device configured to store the transposed rows in one of a buffer or a second buffer, thereby producing a transposed buffer.

13. The computer program product of claim 12 wherein the data to be loaded into the buffer is in an 8-bit representation.

14. The computer program product of claim 13 wherein for each eight buffer rows the computer readable program code device configured to selectively merge pairs of rows merges a first row with a fifth row, a second row with a sixth row, a third row with a seventh row, and a fourth row with an eighth row.

15. The computer program product of claim 12 wherein the data is in a 16-bit representation.

16. The computer program product of claim 15 wherein for each four buffer rows the computer readable program code device configured to selectively merge a plurality of rows merges a plurality of high order bytes of a first row with a plurality of high order bytes of a third row, merges a plurality of high order bytes of a second row with a plurality of high order bytes of a fourth row, merges a plurality of low order bytes of a first row with a plurality of low order bytes of a third row, and merges a plurality of low order bytes of a second row with a plurality of low order bytes of a fourth row.

17. A method and/or computer program product substantially as hereinbefore described with reference to the drawings and/or Examples.

DATED this 23rd day of February, 1999

SUN MICROSYSTEMS, INC.
By its Patent Attorneys
DAVIES COLLISON CAVE

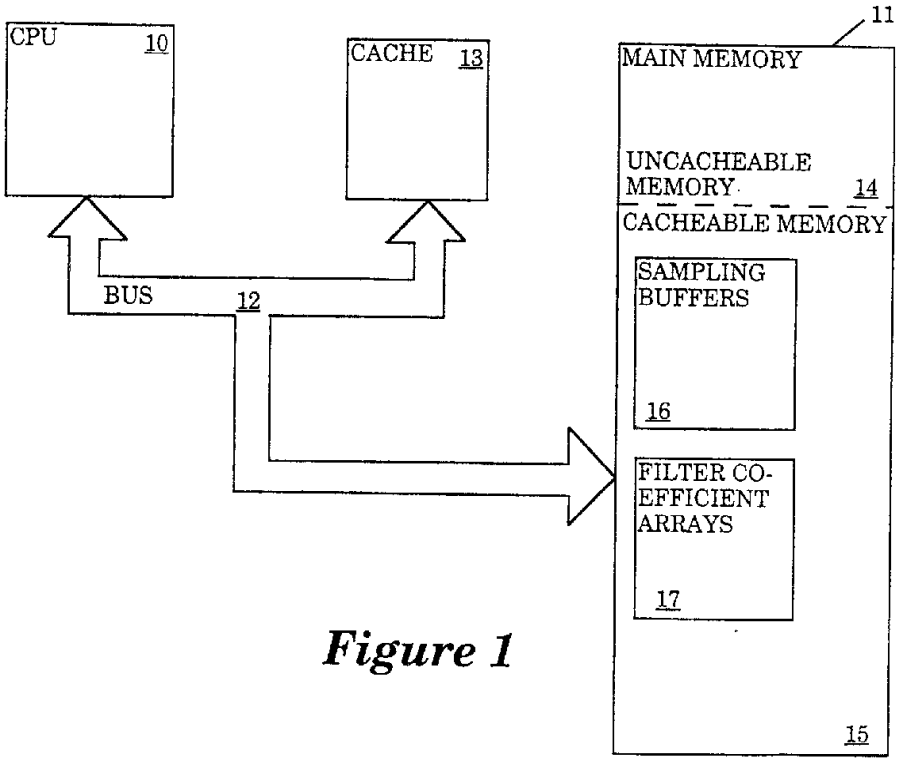
CPU
10
CACHE
13
11
MAIN MEMORY
UNCACHEABLE MEMORY
14
CACHEABLE MEMORY
SAMPLING BUFFERS
16
FILTER CO-EFFICIENT ARRAYS
17
BUS
12
15

Figure 1

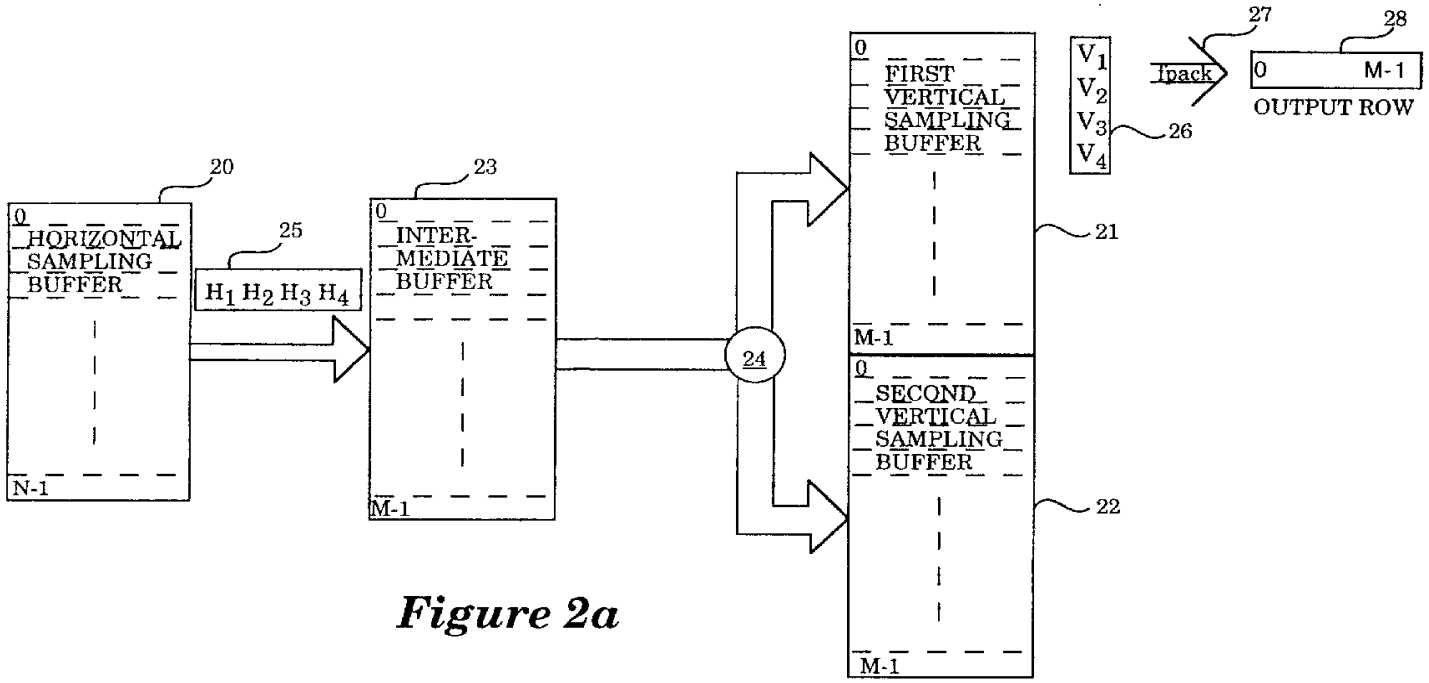
20
0
HORIZONTAL SAMPLING BUFFER
N-1
25
H_1 H_2 H_3 H_4
23
0
INTER-MEDIATE BUFFER
M-1
24
0
FIRST VERTICAL SAMPLING BUFFER
M-1
21
0
SECOND VERTICAL SAMPLING BUFFER
M-1
22
V_1
V_2
V_3
V_4
26
Ipack
27
28
0 M-1
OUTPUT ROW

Figure 2a

71890/96

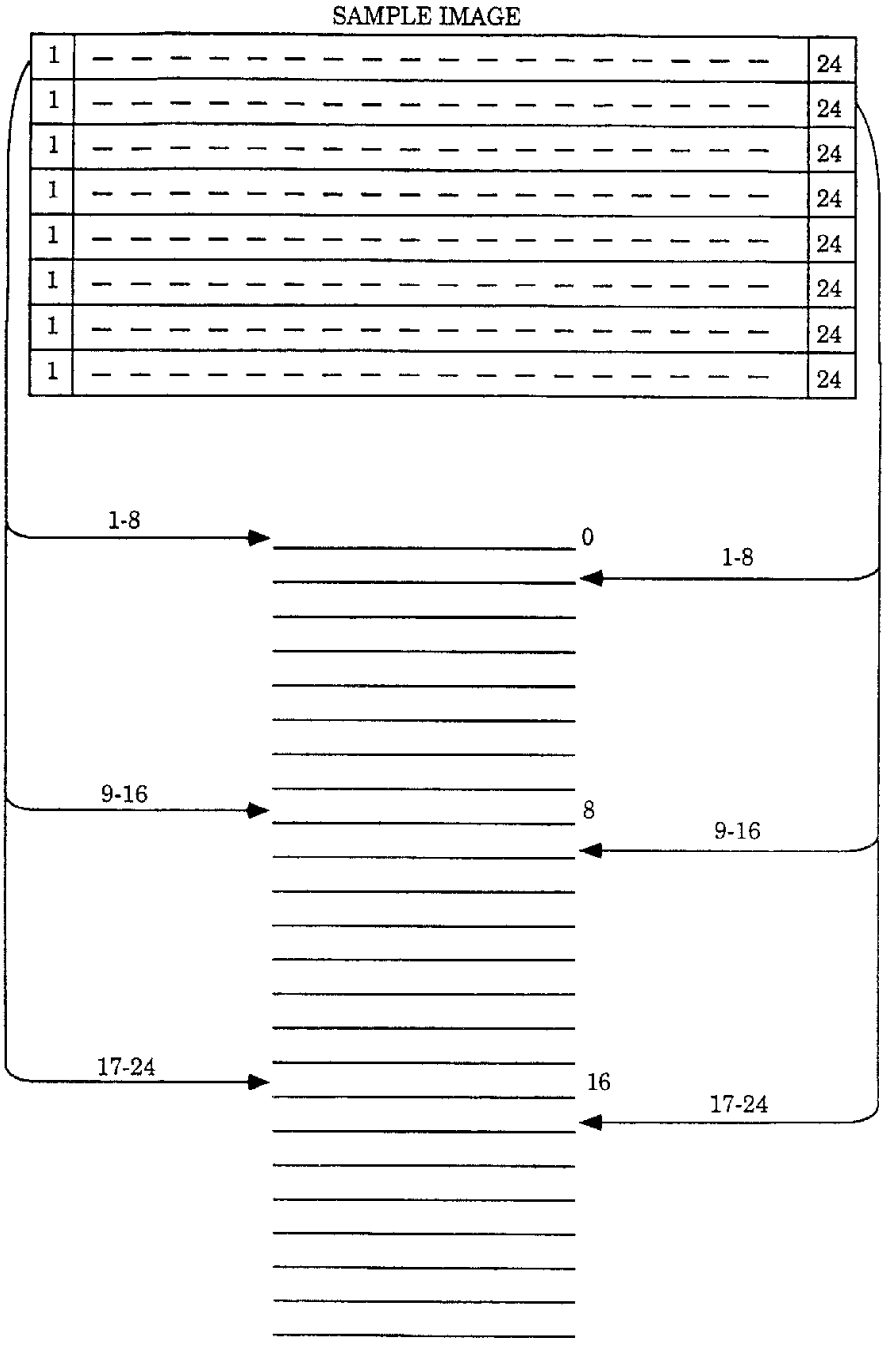

Figure 2b

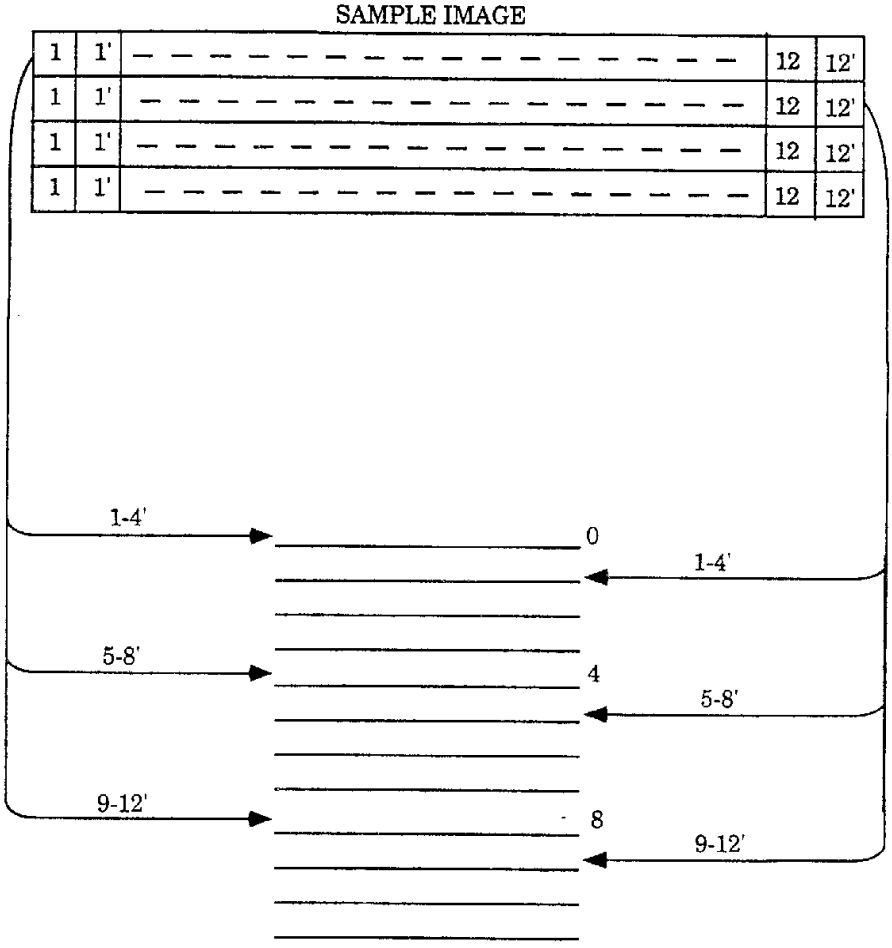

Figure 2c

a0	a1	a2	a3	a4	a5	a6	a7		a0	b0	c0	d0	e0	f0	g0	h0
b0	b1	b2	b3	b4	b5	b6	b7		a1	b1	c1	d1	e1	f1	g1	h1
c0	c1	c2	c3	c4	c5	c6	c7		a2	b2	c2	d2	e2	f2	g2	h2
d0	d1	d2	d3	d4	d5	d6	d7	⇒	a3	b3	c3	d3	e3	f3	g3	h3
e0	e1	e2	e3	e4	e5	e6	e7		a4	b4	c4	d4	e4	f4	g4	h4
f0	f1	f2	f3	f4	f5	f6	f7		a5	b5	c5	d5	e5	f5	g5	h5
g0	g1	g2	g3	g4	g5	g6	g7		a6	b6	c6	d6	e6	f6	g6	h6
h0	h1	h2	h3	h4	h5	h6	h7		a7	b7	c7	d7	e7	f7	g7	h7

Figure 3a

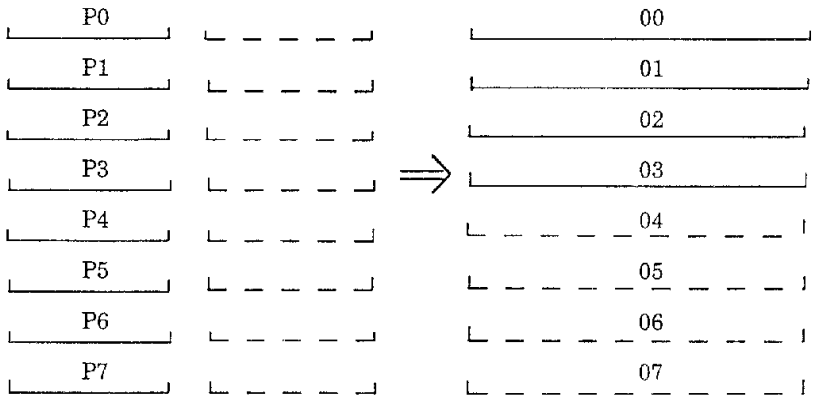

Figure 3b

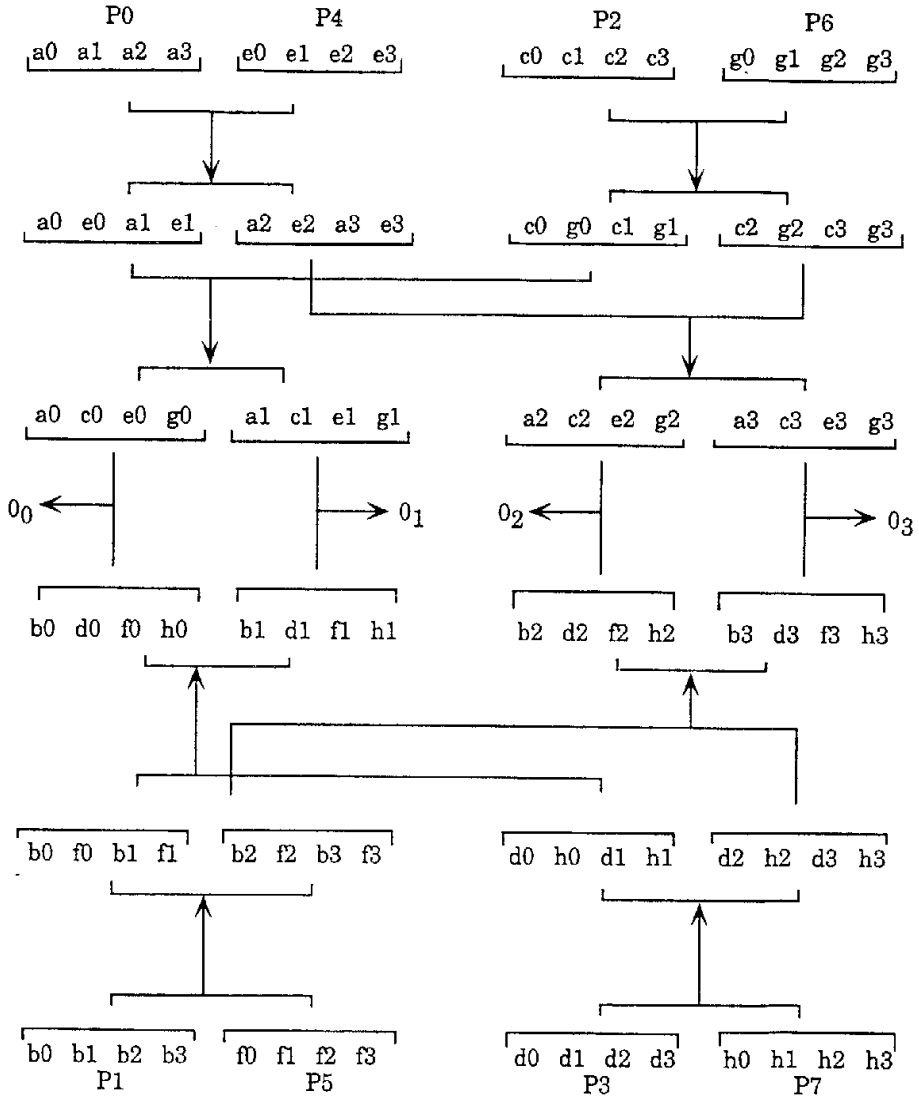
P0
P4
P2
P6
a0 a1 a2 a3
e0 e1 e2 e3
c0 c1 c2 c3
g0 g1 g2 g3
a0 e0 a1 e1
a2 e2 a3 e3
c0 g0 c1 g1
c2 g2 c3 g3
a0 c0 e0 g0
a1 c1 e1 g1
a2 c2 e2 g2
a3 c3 e3 g3
0_0
0_1
0_2
0_3
b0 d0 f0 h0
b1 d1 f1 h1
b2 d2 f2 h2
b3 d3 f3 h3
b0 f0 b1 f1
b2 f2 b3 f3
d0 h0 d1 h1
d2 h2 d3 h3
b0 b1 b2 b3
f0 f1 f2 f3
d0 d1 d2 d3
h0 h1 h2 h3
P1
P5
P3
P7

Figure 3c

Q0	=	a	a'	b	b'	c	c'	d	d'
Q1	=	e	e'	f	f'	g	g'	h	h'
Q2	=	i	i'	j	j'	k	k'	l	l'
Q3	=	m	m'	n	n'	o	o'	p	p'

⇒

a	a'	e	e'	i	i'	m	m'
b	b'	f	f'	j	j'	n	n'
c	c'	g	g'	k	k'	o	o'
d	d'	h	h'	l	l'	p	p'

Figure 3d

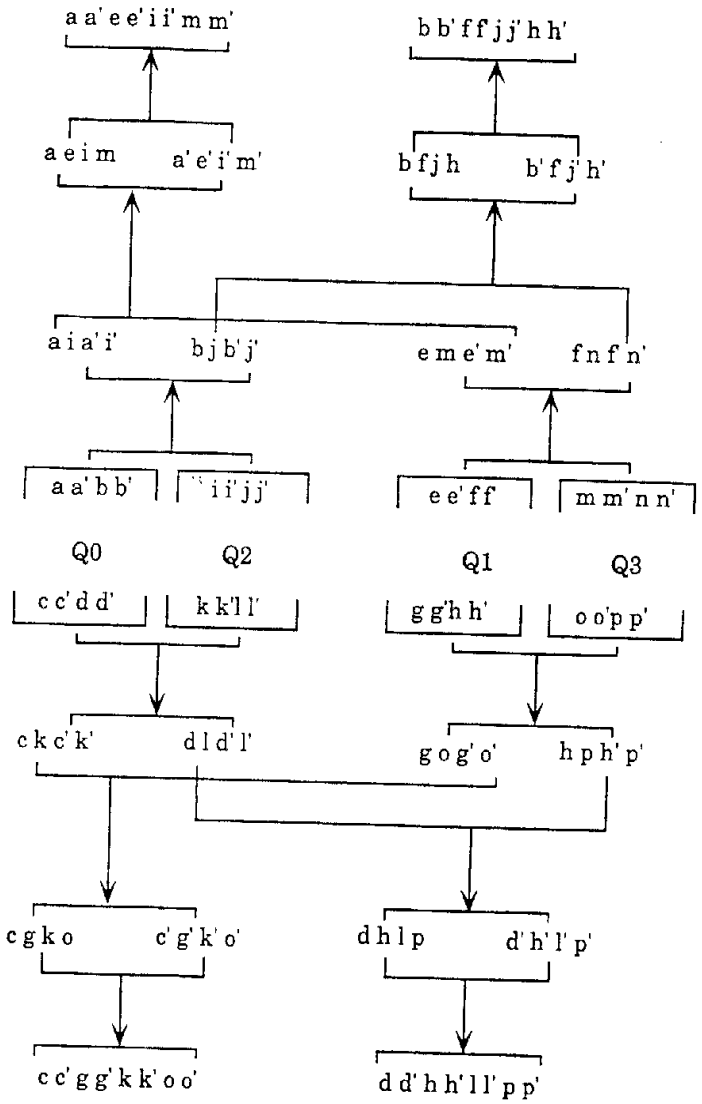
aa'ee'ii'mm'
bb'ffjj'hh'
aeim
a'e'i'm'
bfjh
b'f'j'h'
aia'i'
bjb'j'
eme'm'
fnf'n'
aa'bb'
ii'jj'
ee'ff
mm'nn'
Q0
Q2
Q1
Q3
cc'dd'
kk'll'
gg'hh'
oo'pp'
ckc'k'
dld'l'
gog'o'
hph'p'
cgko
c'g'k'o'
dhlp
d'h'l'p'
cc'gg'kk'oo'
dd'hh'll'pp'

Figure 3e