

DATA RESAMPLER FOR DATA PROCESSING SYSTEM

RELATED APPLICATION

This application is a Continuation-in-Part of commonly owned, co-pending, US Patent Application Serial No. 08/953,772, which is fully incorporated herein by reference.

5

FIELD OF THE INVENTION

This invention relates to data processing systems and more particularly, to a data resampler and data resampling method for a data processing system for processing a stream
10 of data containing logically adjacent or related data samples such as in a machine vision system.

BACKGROUND OF THE INVENTION

Machine vision systems (or other systems processing 2-
15 dimensional data) often need to resample images to adjust scale, rotation, or other distortions in the original data prior to further processing. This operation is referred to herein as warping. For each point in the output image (rendered result array), several points in the source image
20 are read and a computation performed to determine the value of the result. A well known computationally efficient technique uses bilinear interpolation for this purpose. This algorithm reads four neighboring samples surrounding the sub-sample location in the source image to be rendered
25 (2x2 aperture). The hardware described is easily extended to higher-order interpolation kernel techniques which nonetheless fetch a fixed number of surrounding samples.

The location to be rendered may be determined by a number of well-known techniques. One technique suitable for
30 efficient hardware implementation iteratively computes

horizontal and vertical difference equations which cause the sampling point to move through the source image at successive sub-sample points. In the current embodiment, these difference equations compute a linear path. This is commonly known as a linear DDA (Digital Differential Analyzer). Various other techniques such as higher order DDAs or a data structure containing pre-computed sub-sample points to be rendered could also benefit from the techniques to be described.

Complex warping (e.g. mapping an annular ring to a rectangular image) using linear DDA hardware typically requires a large number of individual linear segments to be processed. Even a simple rotation requires a new DDA operation to be executed for each row of the output image.

The need for the DDA hardware to be reinitialized frequently degrades the efficiency of operation.

A second problem occurs in the rendering process due to the nature of burst-mode memories (e.g. SDRAM, SGRAM). This is because all four neighboring pixels which must be fetched on each step do not generally lie on the same memory row. This can degrade performance as memory addressing repeatedly crosses (thrashes) memory row boundaries during the fetch process.

SUMMARY OF THE INVENTION

Accordingly, the current invention addresses the speed problem of the prior art by making a continuous stream of new parameters available to the DDA hardware.

The current invention uses three mechanisms to address the memory thrashing problem. First, address and data caches are employed for re-using pixels which may already have been fetched in a previous step. Second, the invention

exploits SDRAM/SGRAM architecture by placing odd and even image rows in opposite internal banks and pre-activating the next bank address to anticipate the next row address required. Third, the FIFO technique described in co-pending
5 Patent Application Serial No. 08/953,772 is extended to mitigate the address to data latency of the burst-mode memories.

BRIEF DESCRIPTION OF THE FIGURES

10 Figures 1A and 1B are schematic diagrams which illustrate one example of a resampling process of the present invention;

Figure 2 is a simplified block diagram of the warper according to the present invention;

15 Figure 3A is a block diagram illustrating the addition of a data buffer to hold DDA parameters;

Figure 3B illustrates the double-buffering DDA parameter register circuit;

20 Figure 4 is a block diagram showing the addition of address and data caches;

Figure 5 is a schematic diagram showing the operation of the address cache feature of the present invention;

Figure 6 is a schematic diagram which illustrates some examples of cache operation; and

25 Figure 7 is a block diagram which illustrates the operation of the data cache.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

30 Figure 1A is an overview of a resampling process according to the present invention. A 2-dimensional array of adjacent image data samples 701 is stored in memory. Typically these are taken directly from the data source.

The resampling grid 702 represents points at which samples are to be rendered. For each point in the resampling grid 702, a result is generated (rendered) and stored in the 2-dimensional rendered output sample array 703. In the
5 current embodiment, for computational efficiency, the resampling grid or array 702 consists of a series of straight line segments. These segments can piecewise approximate any type of warp distortion. Each rendering sequence involves moving along one of these lines at equally
10 spaced, arbitrary sized steps. Thus, in the illustration, moving along the six resampling grid steps from A to B generates the seven rendered output values from A' through B'.

Fig. 1B details the process of rendering one of the
15 linear segments (e.g. A to B). The initial sampling point 704 (e.g. A) is represented as an initial whole pixel address in the original sample array 701, and x and y subpixel offsets from that address. For each DDA step 705, a new sampling point is computed by adding the delta-x and
20 delta-y steps to the present x-y position with full subpixel precision.

Each sampling location can be expressed as a pair of mixed fractions for X and Y. The integer parts representing pixel address and the fractional parts representing subpixel
25 fractions. Interpolation is required except when the sampling location is exactly at an original sample point (both fractional parts are zero). Bilinear interpolation involves fetching the four pixels surrounding the sampling point. Note that movement in the Y direction involves the
30 "stride" constant (the distance between the rows in the original sample array 701. The rendered output result is computed by considering all four neighboring pixels and the

inverse distance of the sampling point from each of the four original samples.

The DDA unit 401, Fig. 2, generates a corner address and a set of interpolation coefficients for each sample to
5 be rendered. For example, a linear DDA requires the following parameters which are loaded into working parameter registers 401a:

1. Source starting address.
2. Source starting X and Y fractions.
- 10 3. Delta-X and delta-Y (slope of the DDA to be generated)
4. Clipping bounds (to supply default value when valid source address range is exceeded).
5. Number of points to render.

15 In the simple exemplary implementation shown, these parameters are loaded into DDA unit registers prior to each operation.

The DDA unit 401 generates an interpolation corner address 503. The fetch unit 402 uses the generated
20 interpolation corner address 503 to generate four source addresses 505 of samples to be fetched from the source array via memory subsystem 0 120a. The DDA unit 401 also generates a set of interpolation fractions 507 (x and y fractions) for each point to be rendered. Because of memory
25 read latency (as described elsewhere herein), there is some delay before valid data is available. The interpolation fractions 507 are held in a first, first-in, first-out memory 90d (FIFO Q), permitting the DDA unit 401 and fetch
unit 402 to continue to operate during the memory read
30 latency period.

An interpolation unit 403 receives pixel data 509 from a second FIFO memory unit, FIFO A 90a and interpolation

fractions 507 from FIFO Q 90d. The interpolation unit then computes the rendered result pixels, assembles them into memory words 17, and outputs the words to the destination memory subsystem 1 120b via a third FIFO memory unit, FIFO C 5 90c. The destination address 511 is supplied by address generator C 41c.

After each DDA rendering cycle, working parameter registers 401a must be reloaded. Host intervention would cause considerable overhead. Using the auto-sequencer 10 (described in co-pending U. S. Patent Application No. 08/953,772 incorporated herein by reference) improves performance, but still requires flushing FIFO A 90a before and after each run and fetching auto-sequencer instructions and parameter sets from memory between runs. By pre- 15 fetching large blocks of parameters into a data buffer 102 of Figure 3A, most of this overhead is avoided. In Figure 3A, DDA parameter sets are located in memory 0 120a along with the source image, but in general, they may be located in either memory. Fetching parameter sets into the data 20 buffer 102 is a separate operation, occurring whenever the data buffer 102 is empty. During this operation, memory/parameter addresses 121 are supplied by address generator B 41b. In Figure 3A, a fourth FIFO memory unit which is part of data buffer 102, is also used to extend the 25 capacity of the data buffer 102.

To further avoid overhead delays, parameter registers are double-buffered as shown in Figure 3B. Two sets of registers 420a-420b are used. During a warping operation, one set of registers 420a/420b supplies the working values 30 to the DDA circuit while the other is available to be loaded from data buffer 102 under control of sequencer 421. In this way, no parameter loading delay need be encountered when one DDA run terminates and another starts.

To make more efficient use of memory bandwidth during the rendering operation, an address cache 404 and a data cache 405 are added as shown in Figure 4. The width of FIFO Q 90d is also increased to carry a data descriptor 513 along with each set of interpolation fractions 507.

In the current embodiment, the data cache 405 consists of two cache lines (register sets) each holding two four (4) pixel memory words 17. The contents of the data cache registers represent an 8x2 pixel neighborhood. The address cache 404 has two cache lines comprising upper and lower cache lines. Each cache line is comprised of two address registers. Each address register, when valid, holds the address of a word in source memory 120a. Data from the two words thus pointed to eventually become the contents of the corresponding data cache 405 cache line. Although the present invention is described in connection with four (4) pixel words per memory word, this is not a limitation of the present invention as it also applies to wider or narrower memory words as will be appreciated by those skilled in the art.

Address cache 404 computes the four required addresses based on the incoming DDA corner address. It then compares these with all valid addresses currently resident in the address cache 404 and determines if any new data needs to be fetched from memory. It then instructs the fetch unit 402 to supply the required addresses to the source memory subsystem 120a. At the same time, it encodes and outputs a data descriptor 513 containing instructions for the data cache unit 405. The data descriptor specifies how the data cache 405 is to update it's registers from FIFO A 90a if required, and how to form the required interpolation neighborhood once it's registers contain valid data.

In the current embodiment, two memory words are always fetched whenever a cache line is re-filled, exploiting the efficient burst mode capability of SDRAM/SGRAM. The data descriptor 513 can also instruct the data cache 405 to load
5 the upper cache line from the lower cache line or vice versa. Note also that the address cache 404 can reload the upper from the lower address cache line or vice versa as the addresses are processed.

Once the data cache 405 contains valid data, the
10 required four pixels are supplied to the interpolation unit 403 using information from the data descriptor 513, along with the interpolation fractions 507 from FIFO Q 90d.

The operation of the address cache 404 is detailed in Figure 5. The lower-left pixel address 516 is generated by
15 adding the stride 514 to the corner pixel (upper-left) address 515 using adder 410. The msb's of these addresses represent a memory address containing the desired pixel. In the present example, there are four pixels in each memory word 17, and there are 26 bits of pixel address
20 corresponding to 24 bits of memory address 515a and 516a and 2 bits of pixel position 515b and 516b within the memory word.

There are four working address registers 411a-411d organized as two address cache lines. The first line
25 comprises registers 411a and 411b. The second cache line includes registers 411c and 411d. In each line, one register (for example the left registers) 411a and 411c holds the address of the memory word containing the left pixel, while one register (for example the right registers)
30 411b and 411d holds the address of the next word. This means that both the left and right pixels are contained within the two words pointed to by the left and right

address registers. This is illustrated by the examples in Figure 6.

The four memory locations addressed hold the pixel values shown. The p00 2x2 neighborhood 602 lies entirely within the cache. When the p07 neighborhood 604 is to be processed both cache lines must be updated. In the current embodiment, four new memory words are fetched in this case. Note that words 01 605 and 11, 607 could have been reused. This was not deemed necessary due to the high SDRAM/SGRAM burst bandwidth. When the p13 neighborhood 606 is to be processed, the lower cache line (words 10, 603 and 11, 607), are transferred to the upper cache line (words 00, 601 and 01 605), and the lower registers 411c and 411d are updated from the incoming lower-left word address 516a. Whenever a cache line is updated, the incrementer 412 generates the right address or addresses.

On the start of a warp operation, all four working registers' contents are invalid until loaded on the first address step. The memory addresses 515a and 516a of the upper-left 515 and lower-left 516 pixels are compared with all four working address registers 411a-411d contents when valid, using comparators 413a-413h. The results of these comparisons, plus the location of the pixel within the word, determine the actions required to update the address and data caches. Data descriptor encoder 414 generates a data cache update descriptor 517, which enters FIFO Q (90d of Figure 3A) along with the pixel location bits 515b and 516b as data descriptor 513, along with the interpolation fractions 507. The encoder 414 also instructs the incrementer 412 to update the address cache working registers 411a-411d using data switch 418 and sequencer 419. The sequencer 417 also requests words to be fetched from memory via the fetch unit 402, via a fetch signal 518.

The data descriptor 517 includes the sign of the vertical DDA (y-sign) 519, Fig. 5. This determines the order in which the fetch unit will request and the data cache will receive data from the upper and lower memory rows. The y-sign 519 also instructs the source memory controller (40 in Figure 5 of co-pending US Patent Application 08/953,772) to precharge the alternate SDRAM/SGRAM alternate bank on the previous row address rather than the next row address when asserted. This strategy reduces memory latency whenever a cache line update occurs. The encoder 414 also receives a clip input signal 520 from the DDA unit 401. When this is asserted, the DDA unit has determined that at least one source address is invalid. In this case, no data is fetched and the data cache forwards the signal to the interpolate unit 403 which outputs a default value to the result-C bus 17 for the corresponding pixel.

The data cache 405, Fig. 7, has four (4) working registers 415a-415d each holding a memory word. Pairs of registers, 415a/415b and 415c/415d left and right, each comprise a data cache line holding eight pixels. The data switch and sequencer 416 updates working registers 415a-415d based on the data descriptor 513 from the output of FIFO Q 90d. When working register update is required, data is taken from FIFO A 90a. The data switch 416 can also move data between the upper and lower cache lines when required. The data descriptor 513 insures that the sequence of updates corresponds to the sequence of addresses issued to the source data memory subsystem 120a.

Pixel output multiplexer 417 selects the four pixels forming the required 2x2 neighborhood for interpolation using the corresponding FIFO Q pixel select fields. The corresponding interpolation fractions 507 from the FIFO Q

are passed directly to the interpolation unit 403 which forms four coefficients based on the interpolation fractions; multiplies each by the corresponding pixel value to provide four products; sums the four products; and
5 normalizes the result, producing the rendered pixel value. When clip 520 from the DDA 401 is asserted, the interpolation unit outputs a default value. The rendered pixel output values are packed into a memory word and output to the destination bus. In the current embodiment, four
10 pixels are rendered for each memory word written out.

Accordingly, the present invention provides a novel data resampler for processing logically adjacent data samples.

Modifications and substitutions by one of ordinary
15 skill in the art are considered to be within the scope of the present invention which is not to be limited except by the claims which follow.

What is claimed is:

CLAIMS

1. A data resampler for a data processing system processing logically adjacent data samples, said data sampler comprising:

5 a memory subsystem, for storing at least a first plurality of data samples to be rendered;

a digital differential analyzer (DDA), for generating an interpolation corner address for said plurality of data samples to be rendered, and for also generating a set of interpolation fractions;

10 a fetch unit, for receiving the generated interpolation corner address and for generating four source addresses of data samples to be fetched from said memory subsystem;

15 a first FIFO memory, for holding said interpolation fractions and for permitting said DDA and fetch unit to continue to operate during memory read latency periods;

a second FIFO memory, for holding pixel data;
an address generator; and

20 an interpolation unit, for receiving pixel data from said second FIFO and interpolation fractions from said first FIFO, for computing rendered result pixels, for assembling said result pixels into memory words, and for outputting said memory words to a destination memory address
25 supplied by said address generator in a destination memory subsystem via a third FIFO.

2. The data resampler as claimed in claim 1 further including:



a data buffer, for holding DDA parameter sets in DDA parameter registers, said parameter sets fetched from said memory subsystem whenever said data buffer is empty; and

a second address generator, for supplying
5 memory/parameter addresses to memory said memory subsystem.

3. The data resampler as claimed in claim 2, further including a fourth FIFO to extend the capacity of said data buffer, and wherein said DDA parameter registers are double-buffered and comprise two sets of registers, a first set of
10 which supplies working values to said DDA and a second set of which is available to be loaded from said fourth FIFO under control of a sequencer in order to eliminate parameter loading delay.

4. The data resampler as claimed in claim 2 further
15 including an address cache and a data cache.

5. The data resampler as claimed in claim 4, wherein the width of said first FIFO is increased to carry a data descriptor along with each set of interpolation fractions.

6. The data resampler as claimed in claim 4, wherein
20 said data cache comprises at least two cache lines each holding two memory words and wherein said address cache comprises at least two cache lines each comprised of two address registers, each address register of which, when valid, holds an address of a word in said source memory.

25 7. The data resampler as claimed in claim 6 further comprising an adder for generating a lower-left pixel



address from an upper-left pixel address by adding a stride to said upper-left pixel address.

8. The data resampler as claimed in claim 6 further comprising four working address registers organized as two
5 address cache lines each having left and right registers for holding addresses of memory words, wherein in each line the left register holds the address of the memory word containing the left pixel, and the right register holds the address of the next word.

10 9. The data resampler as claimed in claim 8 further comprising an incrementer for generating correct memory addresses whenever a cache line is updated.

15 10. The data resampler as claimed in claim 9 further comprising at least one comparator for comparing the memory addresses of the upper-left and lower-left pixels with the contents of all four working address registers.

This Sheet Has Been Replaced



14



11. The data resampler as claimed in claim 10, further comprising a data descriptor encoder and a data switch and sequencer, said descriptor encoder for generating a data cache update descriptor and for instructing said data switch and sequencer to update the address cache working registers and to request words to be fetched from memory via said fetch unit.

12. The data resampler as claimed in claim 6, wherein said data cache comprises four working registers each holding a memory word, said registers arranged in left and right pairs, each pair comprising a data cache line holding twice the number of pixels per memory word.

13. The data resampler as claimed in claim 12, further including a second data switch and sequencer, which updates working registers based on the data descriptor from the output of said first FIFO and a pixel output multiplexer.

This Sheet Has Been Replaced



14. A method of resampling data for a data processing system for logically adjacent data samples, said method comprising:

storing samples to be rendered in a memory
5 subsystem;

generating an interpolation corner address for a sample to be rendered and a set of interpolation fractions using a digital differential analyzer (DDA);

receiving the generated interpolation corner
10 address and generating four source addresses of samples to be fetched from said memory subsystem;

holding said interpolation fractions in a FIFO memory to permit said DDA and fetch unit to continue to operate during memory read latency periods;

15 holding pixel data in a second FIFO; and

receiving pixel data from said second FIFO and interpolation fractions from said first FIFO and computing rendered result pixels, assembling said result pixels into memory words and outputting said words to a destination
20 memory address supplied by an address generator in a destination memory subsystem via a third FIFO.

15. The method of resampling data as claimed in claim 14 further comprising reloading parameter registers after each DDA rendering cycle.

25 16. The method of resampling data as claimed in claim 15, further comprising pre-fetching blocks of parameters into a data buffer.



17. The method of resampling data as claimed in claim 16 further comprising double-buffering said parameter registers into two sets of registers, a first register set of said two sets of registers of which supplies working values to said DDA, and a second register set of which is available to be loaded from said fourth FIFO under control of a sequencer in order to eliminate parameter loading delay.

18. The method of resampling data as claimed in claim 17 further comprising providing a data cache having two cache lines and holding two memory words in each of said cache lines.

19. The method of resampling data as claimed in claim 18 further comprising computing four required addresses based on an incoming DDA corner address using an address cache, comparing said required addresses with all valid addresses resident in four working address registers of said address cache, determining if any new data needs to be fetched from memory and instructing said fetch unit to supply the required addresses to the source memory subsystem.

20. The method of resampling data as claimed in claim 19 further comprising encoding and outputting a data descriptor, which specifies how the data cache is to update it's registers from said second FIFO.

21. The method of resampling data as claimed in claim 20, further comprising fetching two memory words whenever a cache line is re-filled.



22. The method of resampling data as claimed in claim 21, wherein said data descriptor also instructs said data cache to load the upper cache line from the lower cache line.

5 23. The method of resampling data as claimed in claim 21, wherein said data descriptor also instructs said data cache to load the lower cache line from the upper cache line.

10 24. The method of resampling data as claimed in claim 19, wherein a lower-left pixel address is generated by adding the stride to the upper-left (corner) pixel address using an adder.

15 25. The method of resampling data as claimed in claim 19, further comprising organizing the four working address registers as two address cache lines each containing left and right address registers holding the address of the memory word containing the left pixel in the left register and holding the address of the next word in the right address register.

20 26. The method of resampling data as claimed in claim 25, further comprising generating correct memory addresses whenever a cache line is updated using an incrementer.

27. The method of resampling data as claimed in claim 26, further comprising comparing the memory address of the

This Sheet Has Been Replaced



upper-left and lower-left pixels with the contents of all four working address registers.

28. The method of resampling data as claimed in claim 27, further comprising generating a data cache update descriptor and instructing a data switch and sequencer to
5 update the address cache working registers and requesting words to be fetched from memory.



This Sheet Has Been Replaced

