PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 15/62	A1	(11) International Publication Number: WO 94/10644
		(43) International Publication Date: 11 May 1994 (11.05.94)

(21) International Application Number: PCT/US92/09462

(22) International Filing Date: 2 November 1992 (02.11.92)

(71) Applicant: THE 3DO COMPANY [US/US]; 1820 Gateway Drive, San Mateo, CA 94404 (US).

(72) Inventors: NEEDLE, David, Lewis ; 2981 Northwood Drive, Alameda, CA 94501 (US). MICAL, Robert, Joseph ; 25 Geri Place, Redwood City, CA 94303 (US). LANDRUM, Stephen, Harland ; 179 Haas Avenue, Apartment #7, San Leandro, CA 94577 (US). GRAY, Donald, Milton, III ; 178 Church Street, Apartment #2, San Francisco, CA 94114 (US).

(74) Agents: WOLFELD, Warren, S. et al.; Fliesler, Dubb, Meyer and Lovejoy, Four Embarcadero Center - Suite 400, San Francisco, CA 94111-4156 (US).

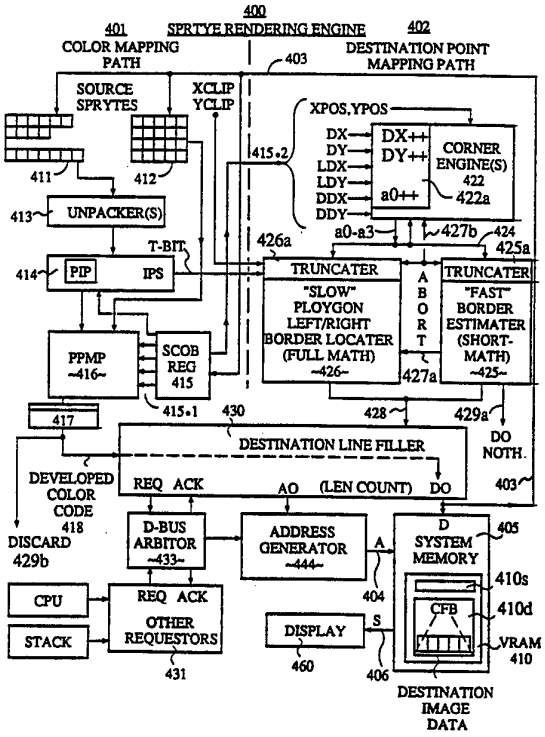
(81) Designated States: AT, AU, BB, BG, BR, CA, CH, CS, DE, DK, ES, FI, GB, HU, JP, KP, KR, LK, LU, MG, MN, MW, NL, NO, PL, RO, RU, SD, SE, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, SN, TD, TG).

Published
With international search report.

(54) Title: SPRYTE RENDERING SYSTEM WITH IMPROVED CORNER CALCULATING ENGINE AND IMPROVED POLYGON-PAINT ENGINE

(57) Abstract

The invention provides a method and apparatus for mapping a source image (401) to a destination grid (402). A general philosophy is followed in implementing the apparatus, namely, wherever possible or practical, leave undone at the start that which ultimately needs not to have been done in the end. One application of the philosophy produces a two stage delta summing unit in which a more significant portion of a result signal is left unaltered when addition of a small delta value to a less significant portion does not produce a carry. Another application of the philosophy provides a fast-condition recognizing unit (Munkee unit) (425) in combination with a slower region-paint calculating unit (Regis unit) (426). The Munkee unit (425) tests the input data set given to the Regis unit (426) and recognizes input conditions for which the Regis unit (426) will ultimately decide that only a single destination-pixel needs to be painted, or that no destination-pixel needs to be painted. In such cases, Munkee (425) terminates time-consuming calculations within Regis (426) and either issues the one pixel paint command itself or does nothing. Regis (426) is freed to begin working on new region-fill calculations.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

SPRYTE RENDERING SYSTEM WITH IMPROVED CORNER CALCULATING ENGINE AND IMPROVED POLYGON-PAINT ENGINE

BACKGROUND

1. Field of the Invention

The invention relates generally to machine-controlled image processing and display. The invention is more specifically directed to an apparatus and process for rendering a destination image on a background given a source image and a predefined mapping of the source image onto a destination grid.

2a. Copyright claims to disclosed Code-conversion and Engine operating Tables

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the U.S. Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

In particular, this application includes listings of code conversion tables (Table-II named, REDMUNK) and signal-name tables (e.g. Table-1.0, SCoB contents) and state transition states (e.g. Rgis-Fillbin). These various listings can be implemented in a number of ways, including but not limited to: a computer program, microcode, in a ROM, and so forth. The same code-conversion or other tables can also be implemented by way of combinatorial logic and/or sequential state machines.

Since implementations of the listings which are deemed to be "computer programs" are protectable under copyright law, copyrights not otherwise waived above in said listings are hereby reserved. This reservation includes the right to reproduce the listings in the form of machine-executable computer programs.

2b. Cross Reference to Related Applications

This application is related to:

PCT Patent Application Serial No. _____, entitled AUDIO/VIDEO COMPUTER ARCHITECTURE, by inventors Mical et al., filed concurrently herewith, Attorney Docket No. MDIO4222, and also to U.S. Patent Application Serial No. _____, bearing the same title, same inventors and also filed concurrently herewith;

PCT Patent Application Serial No. _____, entitled RESOLUTION ENHANCEMENT FOR VIDEO DISPLAY USING MULTI-LINE INTERPOLATION, by inventors Mical et al., filed concurrently herewith, Attorney Docket No. MDIO3050, and also to U.S. Patent Application Serial No. _____, bearing the same title, same inventors and also filed concurrently herewith;

PCT Patent Application Serial No. _____, entitled METHOD FOR GENERATING THREE DIMENSIONAL SOUND, by inventor David C. Platt, filed concurrently herewith, Attorney Docket No. MDIO4220, and also to U.S. Patent Application Serial No. _____, bearing the same title, same inventor and also filed concurrently herewith;

PCT Patent Application Serial No. _____, entitled METHOD FOR CONTROLLING A SPRYTE RENDERING PROCESSOR, by inventors Mical et al., filed concurrently herewith, Attorney Docket No. MDIO3040, and also to U.S. Patent Application Serial No. _____, bearing the same

title, same inventors and also filed concurrently herewith;

PCT Patent Application Serial No. _____, entitled METHOD AND APPARATUS FOR UPDATING A CLUT DURING HORIZONTAL BLANKING, by inventors Mical et al., filed concurrently herewith, Attorney Docket No. MDIO4250, and also to U.S. Patent Application Serial No. _____, bearing the same title, same inventors and also filed concurrently herewith;

PCT Patent Application Serial No. _____, entitled IMPROVED METHOD AND APPARATUS FOR PROCESSING IMAGE DATA, by inventors Mical et al., filed concurrently herewith, Attorney Docket No. MDIO4230, and also to U.S. Patent Application Serial No. _____, bearing the same title, same inventors and also filed concurrently herewith; and

PCT Patent Application Serial No. _____, entitled PLAYER BUS APPARATUS AND METHOD, by inventors Needle et al., filed concurrently herewith, Attorney Docket No. MDIO4270, and also to U.S. Patent Application Serial No. _____, bearing the same title, same inventors and also filed concurrently herewith.

The related patent applications are all commonly assigned with the present application and are all incorporated herein by reference in their entirety.

3. Description of the Related Art

Digital graphic processing relies on the physical transformation of digital signals representing image data from one organizational format to another. Part or all of the transformed image data is then displayed as a graphic image on an appropriate display means (e.g., a cathode ray tube or a liquid crystal display) for enjoyment by a viewer.

In many instances, it is desirable to transform digital image data from one format to another at relatively high speed. This is done to create a sense of animation in displayed images and to create a sense of real-time responsiveness to user inputs in the case of interactive systems. Such high-speed transformation is referred to as real-time digital graphic processing. Real-time digital graphic processing is particularly useful in flight or other simulation systems, interactive game systems and the like.

One function that is often called for in real-time digital graphic processing is the mapping of a source image onto a destination surface. Typically, the source image comprises one or more pixels each of which is filled with a particular color or shade. The mapping function can be a one-to-one copying of pixels from a source area to a destination area. Or alternatively, the mapping function can include a transformation of size, and/or a change of shape (e.g., skew) and/or a rotation of some angle plus a change of colors or image brightness.

By way of example, consider a simulated scene in a real-time military game. An airplane is to be pictured on a display panel such that it appears to be flying by, towards or away from a viewer in real time. The viewer controls at least part of the action on the display by way of real-time controls (e.g., a joystick). If the airplane is to be seen flying by the viewer, in a left-to-right direction, its image moves across the screen in the left to right direction at a desired speed. If the airplane is to be seen flying towards the viewer, its displayed image becomes larger as its apparent distance from the viewer decreases. Conversely, if the airplane is to be seen flying away from the viewer, its image becomes smaller as its distance from the viewer

increases. Moreover, if the airplane performs a roll during its flight, its image rotates.

If an explosive device ignites near the airplane as the airplane flies by, the displayed brightness of the aircraft body increases momentarily to simulate reflection of light from the explosion off the fuselage of the airplane.

The airplane may have transparent components, such as a large bubble-shaped cockpit window or a hole through part of its body. The hole or other transparent component may be present from the start or suddenly created by a striking projectile. In such cases, it is desirable to show background scenery passing transparently through the cockpit window and/or body hole as the airplane flies in front of a background scene.

An animated real-time scene of this type can be produced on a display means in a number of ways.

A brute force approach would separately generate each frame of the animated scene data in its entirety, store the generated frame data in high speed memory (e.g., video RAM) and transfer each complete frame (background plus airplane) to the display means at an appropriate frame rate. This brute-force approach wastes memory space and demands high-speed performance from the processing circuitry that generates the sequential frames of image data.

A better approach relies on the concept of sprite painting. One area of memory stores nonchanging, background image data and a second area of memory stores the image data of the airplane and other moving or otherwise changing objects. With each displayed frame, the image of the airplane is mapped from the second memory area onto the background image of the first memory area. The mapping function changes with time to provide size enlargement or reduction and rotation over time.

The mapping function also provides changes of color and/or brightness to simulate various illuminations such as that from a nearby explosion.

Ideally, it should be possible to take any source image and produce a mapping of it which includes arbitrary amounts of enlargement or reduction in size. It should be possible to project the mapped copy onto a destination grid with or without rotation and/or shape distortion (skew). It should also be possible to project the mapped copy onto a destination grid with or without changes of color.

High-performance electronic computer systems are available for transforming image data in such a manner. The Silicon Graphics Iris™ system is an example. Such systems rely on complex software-controlled data transformations and bulk transfers of image data from one memory region to another. A general purpose computing unit is burdened with the task of performing all calculations that define transformations of image data. These high-performance computer systems suffer from drawbacks such as excessive cost, large circuit size, complexity and/or slow image rendition speed.

A need exists within the industry for a compact image-rendering system that can be implemented at low cost on one or a few integrated circuit (IC) chips and can nonetheless perform high-speed complex image transformations.

SUMMARY OF THE INVENTION

It is an object of the invention to provide a highly efficient and compact image rendering system which is capable of bidirectional image mapping with rotation and/or skew and/or color change in a real-time environment.

The term "bidirectional" is used here to mean that mapping can be carried out to convert a large scale source image to a small scale destination image, or conversely and just as easily, to convert a small scale source image to a large scale copy. Rotation and/or skew functions are made available simultaneously and independently of size enlargement or reduction. The image rendering system allows one to enlarge the destination image along one direction of the destination surface (e.g., along the X-axis of a destination grid) while reducing the destination image along another direction (e.g., along the Y-axis of the destination grid).

Color changes and creation of transparent regions within a mapped image are also supported.

The objectives of the invention are realized by selectively following a set of primary and secondary design-rules (listed below in Design Paradigm section) and applying the rules at the macro-architectural level (see below Summarized Example 1) and/or at the micro-architectural level (see below Summarized Example 4).

TABLE OF DETAILED CONTENTS

The following, more detailed portions of this disclosure are subdivided and organized in accordance with the below listed sections and subsections:

(§1)..... DEFINITION OF GLOBAL TERMS
(§1.0)..... OVERVIEW OF DESIGN PARADIGM
(§1.1)..... SUMMARIZED EXAMPLE 1: ARCHITECTURE OF THE SPRYTE RENDERING SYSTEM
(§1.2)..... SUMMARIZED EXAMPLE 2: MULTIPLE CORNER ENGINES
(§1.3)..... SUMMARIZED EXAMPLE 3: SOME FUNCTIONS OF "SLOW" DESTINATION-PAINT DECISION-MAKING UNIT (SlowDPDMu, e.g. Regis) ARE

SUPPLEMENTED BY FastDPDMu COPROCESSOR (e.g. Munkee)
(§1.4)..... SUMMARIZED EXAMPLE 4: NONPERFORMANCE OF MATH FUNCTION ON MORE SIGNIFICANT END OF RESULT
(§1.51)..... SUMMARIZED IMPLEMENTATION OF EXAMPLE 1
(§1.52)..... SUMMARIZED IMPLEMENTATION OF EXAMPLE 2
(§1.53)..... SUMMARIZED IMPLEMENTATION OF EXAMPLE 3
(§1.54)..... SUMMARIZED IMPLEMENTATION OF EXAMPLE 4
(§2)..... BRIEF DESCRIPTION OF THE DRAWINGS
(§3)..... GLOBAL DESCRIPTION OF SPRYTE MAPPING METHOD (Figures 1A through 3B)
(§4)..... SPRYTE RENDERING ENGINE OVERVIEW (Figure 4)
(§5.1)..... GENERAL EXPLANATION OF REGION FILL METHOD (Figure 5A)
(§5.2)..... GENERAL EXPLANATION OF FastDPDMu (e.g. Munkee) SHORT-CUT METHOD (Figure 5B)
(§6)..... OVERVIEW OF PLURAL CORNER-ENGINES EMBODIMENT (Figure 6)
(§6.1)..... DETAILED DESCRIPTION OF COLOR-MAPPING PATHS (Figure 6, portion 601)
(§6.2)..... DETAILED DESCRIPTION OF DESTINATION-POINT MAPPING PATHS (Figure 6, portion 602)
(§6.3)..... DETAILED DESCRIPTION OF CORNER ENGINE AND MATH PLATFORM RESOURCES (Figure 7)
(§6.4)..... PSEUDO-INDEPENDENT OPERATION OF MULTIPLE CORNER ENGINES AND FastDPDMu (e.g. Munkee) AND Slow-DPDMu (e.g. Regis) (Figure 8)
(§6.5)..... DELTA SUMMING MECHANISM (Figures 9A, 9B)
(§6.6.1).... DETAILED DESCRIPTION OF MUNKEE SHORTCUT ALGORITHMS
(§6.6.2).... DETAILED DESCRIPTION OF REGIS_FILLBIN STATE MACHINES
(§7.0)..... DETAILED DESCRIPTION OF SCoB AND OTHER REGISTERS

(§1) DEFINITION OF GLOBAL TERMS

It is to be understood that the below descriptions are directed to physical structures (e.g., digital hardware) and to the transformation of physical signals (e.g., electrical or like signals) in real time from one form to another.

In many instances, it is convenient to speak about physical signals in terms of things they represent. By way of example, the below description will repeatedly use a phrase such as "computing the coordinates of ___". This is to be read as defining not only the underlying mathematical steps but also the physical steps of receiving signals representing input parameters, passing the received signals through a digital signal processing (DSP) unit and generating physical output signals representing the indicated result. Such a phrase is further to be read as implying that it takes finite time to propagate signals through the DSP unit and that the DSP unit consumes a finite amount of physical space within a physical system containing that DSP unit. Emphasis is to be placed on the latter reading, particularly in view of the stated objectives of the invention, which are to provide a highly efficient and compact image rendering system which is capable of bidirectional image mapping with rotation and/or skew and/or color change in a real-time environment.

In similar fashion, phrases such as "saving a value" or "transferring a value" are respectively understood to refer to storing a physical signal representing that value in a physical storage means and transmitting a physical signal representing the indicated value from one physical location to another.

The below description will also repeatedly use the term "destination pixels" and it will speak about the

"painting" of one or more destination pixels with a particular "color" or "shade". This is to be read as defining not only the underlying conceptual steps for creating a bit-mapped image but also the physical steps of providing physical signals which either immediately or at a later time change the color, brightness or other physical attributes of pixels in a projected light image containing such pixels.

The term "painting" is also to be read as meaning that a physical storage medium (e.g., a Video-speed Random Access Memory unit) is provided for storing image data signals representing attributes (e.g., colors) of individual pixels. The process of "painting" is understood to include the step of replacing pre-stored image-data signals with newly-stored image-data signals. The bit-mapped image that is represented by the stored image-data signals can be either converted directly into a light image having such pixels or it can be further processed (e.g., by means of a color look-up table [CLUT] and/or interpolator) to ultimately produce a light image which is a function of the image represented by the stored image-data. (In one embodiment, the ultimately produced light image can be selectively interpolated to produce a resolution-enhanced version of the image represented by the stored image data, where the apparent resolution of the displayed image is better than that of the stored image.)

In view of the above, it is understood that this disclosure deals with physical entities such as physical signals that represent various values; and signal processing hardware that inputs, processes and outputs such physical signals; and image data signals which ultimately cause the a physical light image to be rendered on a display means.

(§1.0) OVERVIEW OF DESIGN PARADIGM

The image rendering system disclosed here is compact but nonetheless capable of mapping complex image data at high speed and of efficiently transferring image data from one or more source memory locations to a corresponding one or more destination memory locations. The inventors have realized small size and efficient resource utilization by choosing and selectively applying the following system-design rules.

Stated generally, a first and primary design rule says:

(1) Identify and avoid unnecessary work.

A first corollary of the primary design rule says:

(1a) Wherever possible or practical, one should leave undone at the start of an image data transforming process that which ultimately needs not to have been done in the end.

A second corollary of the primary design rule says:

(1b) Wherever possible or practical, one should use compressed data formats (e.g., a minimal number of bits for representing required information) and leave the data in compressed format for as long as possible while working on it and decompress the data only when finally necessary. This avoids unnecessary work on the extra bits and helps to improve system performance.

Further and secondary system-design rules suggest the following additional guidelines:

(2) Share results among parallel resources.

(2a) When a particular signal processing step is time-consuming, parallel processing resources should be provided in the signal processing machine to speed performance; and moreover,

(2b) When a result produced by a first of the parallel processing resources is useable by a second of the parallel processing resources, and there is no performance penalty for doing so, the second parallel processing resource should take advantage of the situation and use the result produced by the first parallel processing resource rather than reproducing the same result on its own; and additionally,

(2c) Shared result signals should be left in the registers where they are stored rather than being shuttled back and forth between various locations. Back and forth transfers waste time.

(3a) Minimize circuit size. When a signal processing step is not time-consuming, serial (time-multiplexed) processing resources should be used to carry the steps out in the signal processing machine. Serial circuits are generally smaller in size than parallel circuits and thus conserve circuit space.

(3b) Mix back and forth between parallel and serial processing modalities in order to obtain the performance benefits of parallel processing where necessary and to obtain the circuit size reduction benefits of serial processing where possible.

(3c) When parallel produced datawords are to be serialized and fed to various locations of memory as a serial string of datawords, try to compress the data so that you can transfer it in minimal time and/or try to arrange the data within the

transfer string so that the order will minimize the number of memory page crossings or other delay-causing conditions that develop at the destination.

(4) Signal processing resources should have pipelined architectures to increase system throughput rates.

(5) As many parts as possible of the pipelined architecture should be doing useful work at all times.

(6) A useful singular-output of one pipeline part should be saved and repeatedly used by other parts rather than being discarded each time and thereafter reproduced by passing its parenting input data repeatedly through the front end of the pipeline.

(7) When none-useful operations are anticipated or detected to be occurring within one or more of the pipelined parts, they should be left unstarted, or terminated as soon as possible. This frees the pipeline parts to begin work on new and probably more useful data that is queued at the front (upstream end) of the pipeline. The definition of none-useful operations includes everything which ultimately produces no change in displayed or displayable image data.

(8) More commonly evoked mapping functions or math functions should be supported by more parallel resources to speed overall performance while less commonly evoked functions should be supported by serial resources or fewer parallel resources to reduce circuit size.

More specifically, the above design rules and/or guidelines are selectively applied to one or more

operations of an image rendering system according to the following summarized examples.

(§1.1) SUMMARIZED EXAMPLE 1: ARCHITECTURE OF THE SPRYTE RENDERING SYSTEM HAS SEPARATE SOURCE-TO-DESTINATION COLOR MAPPING AND SOURCE-TO-DESTINATION POINT MAPPING PATHS WHICH OPERATE IN PARALLEL PSEUDO-INDEPENDENTLY BUT SHARE PERFORMANCE-ENHANCING INFORMATION (USES DESIGN RULES 1-8)

Source-to-destination mappings are subdivided into two kinds, color-mapping and destination-point mapping. Color-mapping concerns itself with the question of what color should be assigned to a destination pixel, given a particular color or set of colors in one or a plurality of source pixels.

Destination-point mapping, on the other hand, concerns itself with the question of what destination coordinates should be assigned to each point of a destination image, given a corresponding point in the source image. Destination-point mapping is also referred to below as point-to-point mapping. The latter name indicates that, among other things, destination-point mapping is concerned with the transformation of source coordinates into destination coordinates.

Image rendition speed is increased by performing the two kinds of mappings (color and destination-point) in parallel and pseudo-independently of one another. The term "pseudo-independently" means here: independently in general, but cooperatively when a result developed in the course of one mapping process helps to minimize time consumed by a second, parallel mapping process.

In one specific embodiment of the invention, parallel, pseudo-independent circuits are provided for performing color mappings and destination-point mappings.

Image rendition speed is primarily increased by leaving undone within either a color mapping process or a corresponding destination-point mapping process that which ultimately needs not to have been done in the end.

The color mapping and destination-point mapping circuits each include a "useless-operation detecting and indicating" means for detecting conditions where, and producing an indication that, no change will occur in the ultimately displayed (or displayable) image data as a result of activities just started or about to be performed either in the color mapping process or in the corresponding destination-point mapping process.

The just-started, but ultimately-useless activities are then terminated (or more preferably, the about-to-be performed, useless activities are not begun in the first place) in order to avoid time-wasting consumption of system resources.

Aside from avoidance of unnecessary work, enhanced performance is also obtained by saving and re-using the results of a given color mapping or destination-point calculation many times rather than re-doing the same color mapping or calculation over and over again.

Here are a few specific examples which take advantage of both the "useless-operation detecting and indicating" means and the "result save and re-use" method.

When image-size reduction occurs in a source-to-destination point-to-point mapping, some finer details of the source image, including some colors and singular pixels of the source image, may fail to appear in the destination image. This happens when the finite resolution of the destination grid (a gridwork of

destination pixels) is too crude to support display of the finer source details. The finer details are therefore ultimately not rendered in the destination grid. The point-to-point mapping circuits of the invention provide an indication that such an ultimate non-rendition will occur. The color mapping circuits respond to this indication by, in essence, discarding a color code signal that had been developed for painting the corresponding destination area.

Conversely, when image enlargement occurs, singular pixels in the source image may be reproduced in the destination grid as plural pixels, each with the same mapped-over color. This happens when single pixels of the source image each become multiple pixels in the destination image as a result of the selected point-to-point mapping. The point-to-point mapping circuits of the invention provide an indication that such multiple painting with a same mapped color will occur. The color-mapping circuits retain the corresponding color-code signal in a color-code register and they wait for completion of painting with the one color before proceeding to store an overwriting, next mapped-color into the register.

Yet another example of efficient resource utilization comes from a special case where color-mapping converts a source color code into a so-called "transparency" code. The transparency code indicates that the corresponding destination pixel or pixels are not to be painted over with a new color code but rather that they should retain their original color code or codes. In such a case, the color mapping circuits send an indication to the point-to-point mapping circuits, telling them to avoid unnecessary mapping work. There is no value in calculating the exact destination coordinates where a color will not be painted.

A preferred embodiment of the invention uses a multi-path, pipelined, parallel-processing approach for transforming source color codes into destination color codes and for pseudo-independently transforming source size, shape and angular orientation into destination size, shape and angular orientation.

Two or more time-parallel, pipelined processing circuits or "paths" are provided. One or more of the processing paths develops code signals representing mapped destination colors. Each such path is referred to here as a "color-mapping path."

At the same time, one or more other processing paths develop code signals representing destination location values and destination paint/don't-paint decisions. They are referred to as "destination-point mapping paths."

A color signal develops in the color-mapping path at the same time that corresponding destination location signals and paint/don't-paint decision signals simultaneously develop in the destination-point mapping paths.

The destination decision signals indicate whether each correspondingly developed color signal is to be: (a) discarded immediately, or (b) discarded after being used only once to paint a single destination pixel whose location is defined by a position-representing signal simultaneously developed in one of the destination-point mapping paths, or (c) saved and used plural times to paint a plurality of destination pixels whose locations are defined by a plurality of position-representing signals simultaneously developed in the destination-point mapping paths.

The developed color-code of the color-mapping path can simultaneously indicate that it is not to be used in overwriting the color code of a destination pixel. This

occurs when the developed color code includes an active transparency bit (T-bit).

This multi-path parallel-processing architecture leads to faster image rendering and more efficient use of system resources and more efficient use of signals developed by the color-mapping and point-to-point mapping paths.

The architecture of the invention additionally provides performance advantages in time-multiplexed memory-access systems. The latter systems are ones where plural requestors vie for time on shared system data and address buses.

In such cases, the color-mapping and destination-point mapping paths compete with one another and with other requestors for access to shared system-memory resources. (The color-mapping path needs access to system memory to do source image fetches. The destination-point path needs memory access to do destination image writes.)

According to the invention, each parallel path shares its computational results and ultimate paint/don't-paint indications with the other so as to eliminate inefficient competition for memory or other system resources. (And also to eliminate inefficient discard and subsequent reproduction of a same piece of data over and over again.)

When one of the multiple paths decides that a particular, upcoming memory request will not ultimately produce a useable result, that path issues a uselessness indicating signal that blocks the particular memory request from being made, either by itself (by the deciding path) and/or by another path. The blockage of ultimately-useless memory requests works to speed system response.

One example of pseudo-independent cooperation between paths is the case where a color code developed by the color mapping path ends up being simply discarded rather than being used to overwrite preexisting image data. This will happen if the developed color code turns out to be a transparency code or if there is no pixel in the destination grid which can absorb the developed color. The destination-point mapping path recognizes the ultimate discard of the color code as early as possible (by, for example, responding to T-bits that are produced at a midstream portion of the color mapping path). The destination-point mapping path halts corresponding activities within itself as soon as it recognizes the futility of proceeding further. By so doing, the destination-point mapping path assures that it will not waste any extra time calculating exactly where in a destination grid the discarded color is to be NOT painted. At the same time, the destination-point mapping path also instructs the color-mapping path to permit a discard of the developed color code then residing in the color-code register. This creates a free slot within the pipelined architecture of the color-mapping path, and in essence, increases the color code production rate of the color-mapping path.

As a consequence, the color mapping path accesses system memory more often and produces new codes more rapidly when mapped color codes are being consumed on a one-to-one or many-to-one basis for each destination pixel. On the other hand, the color mapping path accesses system memory less often and produces new codes less rapidly when mapped color codes are being consumed on a one-to-many basis for each destination pixel.

(§1.2) SUMMARIZED EXAMPLE 2: MULTIPLE CORNER ENGINES (USES DESIGN RULES 1, 2, 3a, 3b, 4, 7, 8)

One step in mapping a source pixel onto a destination grid is the calculation of destination coordinates for a hypothetical polygon (or other geometric shape) that represents an ideal mapping of that source pixel onto the destination surface. This is a time consuming process.

Two or more corner engines (e.g., engines A and B) are preferably provided in accordance with the invention, each for calculating, in parallel, the destination coordinates for corner points of source pixels.

It is recognized that the bottom left corner point of a first source pixel in a first source row becomes the top left corner point of an adjacent underlying source pixel in an adjacent and underlying second source row. The mapping of the bottom left corner point of the first source pixel onto the destination grid is the same as the mapping of the top left corner point of the second source pixel onto the destination grid.

One corner engine (engine-A) begins to calculate the destination coordinates for the top and bottom corner points of a first source row, while a second corner engine (engine-B) waits. Definition of the destination coordinates for the first pair of top and bottom corners of the first pixel in the first spryte row is referred to as a row-initialization process.

As soon as the bottom left corner coordinates of the first pixel in the first row have been calculated by the first corner engine (A), they are passed by way of a shared register stack to the second corner engine (B). The first corner engine (A) exits its row-initialization procedure and proceeds on its own to calculate the destination coordinates for subsequent top and bottom corner points of the first source row.

In the mean time, the second corner engine (B) enters a row-initialization process wherein it (engine-

B) begins to calculate the destination coordinates for the top and bottom corner points of a successive second source row. As soon as the bottom left corner coordinates of the first pixel in the second source row have been calculated by the second corner engine (B), they are passed back to the shared register stack. The second corner engine (B) exits its row-initialization procedure and proceeds on its own to calculate the destination coordinates for subsequent top and bottom corner points of the second source row.

Whichever engine stops mapping the corners of its current spryte row first (because the engine has finished the task or terminated the task for some other reason) picks up the task of mapping the next, not-yet processed, source row. If the first corner engine (A) completes or terminates its calculations of all corners in the first source row before the second corner engine (B) completes or terminates calculations of all corners in the second source row, then the first corner engine (A) picks up the results left behind in the shared register stack and uses them to begin calculations of destination coordinates for the corners of pixels in third source row. On the other hand, if the second corner engine (B) ends its calculations first, it picks up the results left behind in the shared register stack and uses them to begin calculations of destination coordinates for the corners of pixels in third source row.

The first and second corner engines operate independently of one another after the point where they pick up row-starting results from the shared register stack. Each corner engine proceeds from that point forward, at its own maximum speed, using whatever shortcuts (e.g., a small delta summation method described below) may be available, to complete its assigned calculation tasks in minimum time; even if the other

corner engine can not take advantage of such shortcuts. This pseudo-independent operation is particularly useful in one embodiment of the invention where source images are defined as "spryte" constructs (described below) and/or where the area into which sprytes can be rendered is cropped by variable amounts (e.g., by XCLIP and YCLIP described below). Pseudo-independent corner-engine operation is also useful in embodiments of the invention where the source image data produces the above-mentioned "transparency bits" (T-bits).

The above noted "spryte" constructs permit successive first and second rows of a source image to have different numbers of pixels. A source row can even have no pixels. This means that one corner engine can finish its spryte row much faster than the other engine simply because there are fewer source pixels to map in that spryte row.

The above noted "cropping" function (which can optionally include a so-called "super-clipping" function described later) lets programmers halt spryte rendition for sprytes or portions of sprytes that the programmers know will never be displayed on the display means. Spryte rendering time is reduced by not wasting time generating image data that will never be displayed.

The above noted "transparency" bits are used to create a hole or window through a spryte. Mapped color codes having active transparency bits (T-bits) are not used to over write whatever exists in a corresponding memory destination area, and as a result, the pre-existing image continues shows through the "transparency" hole even though adjacent areas are painted over with new colors. In one embodiment, each corner engine bypasses the performance of comprehensive, destination-fill calculations for each mapped color code that is found to have its transparency bit activated. The corner engine

thus saves time by not performing certain point-to-point mapping calculations for a mapped color that, ultimately, will not result in the painting of a destination pixel.

(§1.3) SUMMARIZED EXAMPLE 3: SOME FUNCTIONS OF "SLOW" DESTINATION-PAINT DECISION-MAKING UNIT (SlowDPDMu, e.g. Regis) ARE SUPPLEMENTED BY FastDPDMu COPROCESSOR (e.g. Munkee) (USES DESIGN RULES 1, 2a, 3a, 5, 7, 8)

As an extension of the above-described use of multiple result-developing paths, a Fast Destination-Paint Decision-Making unit (or "Fast-DPDMu" for short, a specific example of which is later referred to as the "Munkee" border estimating unit) is provided to operate in parallel with a Slower Destination-Paint Decision-Making unit (or "Slow-DPDMu" for short, a specific example of which is later referred to as the "Regis" border locating unit).

Both of the Fast-DPDMu and Slow-DPDMu units receive the corner calculation results of the corner engines.

The slower decision-making unit ("Slow-DPDMu, e.g. Regis)") is assigned a time-consuming task of calculating relatively precise coordinates for points along left and right borders of each ideally (hypothetically) mapped source pixel, regardless of the size and/or shape of that mapped source pixel.

A so-called, ideally-mapped source-pixel is also referred to herein as a projected polygon. Each projected polygon is deemed to lie on the surface of the destination grid. The calculation of coordinates for points along left and right borders of each projected polygon is referred to below as boundary walk. Machine-implemented boundary walks tend to be difficult and time-consuming.

The slower decision-making unit (Slow-DPDMu, e.g. "Regis)") uses the results of its boundary walk to decide how many destination pixels, if any, are to be painted over with a newly-mapped color. In some embodiments this decision is made by determining how many destination pixels, if any, are fully or "effectively" bounded between left and right edges of each projected polygon. Destination pixels which are deemed to be fully or effectively bounded within the projected polygon are the ones that are later painted on a raster-scan basis with a fill-in color developed by the color-mapping path.

Unfortunately, the slower decision-making unit (Slow-DPDMu e.g., "Regis)") consumes multiple ticks of the system clock to complete its task because it is charged with the task of processing all kinds of projected polygons: big and small, rotated or unrotated, skewed or unskewed. The boundary walk algorithm that it performs is therefore relatively elaborate and time-consuming.

For certain types of projected polygons (e.g., those with zero or relatively small areas), it is possible to accurately determine, in relatively little time, without using an elaborate time-consuming algorithm, if the destination fill-in region will be zero pixels wide or one pixel wide or some other small number of pixels wide.

The fast decision-making unit (Fast-DPDMu e.g. "Munkee)") looks for projected polygons that meet a certain, easy-to-predict criteria (e.g., projected polygon with areas close to zero) and, in cases where it is sure of the result, the Fast-DPDMu makes the destination-paint or do-not-paint decision in place of the Slow-DPDMu. One embodiment of the Fast-DPDMu (e.g., the later described "Munkee" unit) completes this task within one tick of the system clock. ("Munkee" is phonetically similar to the French word "manquer," which

loosely translated means, a thing that is to be left partially undone. The Munkee unit does not handle all types of projected polygons. It leaves the task of processing the more difficult types of projected polygons to the Fast-DPDMu.)

When a projected polygon is encountered which meets the criteria of the Fast-DPDMu (where the Fast-DPDMu can accurately determine that zero, or only one or a small other number of destination pixels are to be ultimately painted), the Fast-DPDMu (e.g. Munkee) tells the Slow-DPDMu (e.g. Regis) to not perform its more elaborate boundary-walk operations.

In the case where the Fast-DPDMu determines that only one destination pixel is to be ultimately painted, the Fast-DPDMu (e.g. Munkee) sends out a request to a line filler unit to fill that single destination pixel immediately with the color developed by the color-mapping path. The developed color code is tagged as being no longer needed (discardable). The so tagged color code can then be discarded as early as the next clock tick. This saves clock ticks and frees the limited resources of the spryte rendering system to perform other, still-queued and/or not-yet completed tasks.

In cases where the Fast-DPDMu determines that no destination pixels are to be ultimately painted with a particular, developed color code, the Fast-DPDMu (e.g. Munkee) similarly tells the Slow-DPDMu (e.g. Regis) to not perform its boundary-walk operations. The developed color of the color path is tagged as being discardable. and the Fast-DPDMu (e.g. Munkee) does nothing further for the corresponding source pixel. No line-fill command is sent to the line-filler and even more clock ticks are saved. (When a line-fill command is received, the line filler performs its function by sending memory-write

requests to a system memory manager. System performance improves when useless access requests are blocked.)

In cases where the Fast-DPDMu (e.g. Munkee) can not determine for sure how many destination pixels are to be ultimately painted (even if it is zero or one), the Fast-DPDMu (e.g. Munkee) lets Slow-DPDMu (e.g. Regis) perform its more comprehensive boundary-walk operations. The Slow-DPDMu (e.g. Regis) then decides how many, if any, destination pixels are to be painted and the Slow-DPDMu issues the corresponding line-fill requests.

Significant performance improvements are seen in cases of one-to-one or one-to less-than-one image size transformations because the time-consuming and computation resource-consuming operations of the Slow-DPDMu (e.g. Regis) are circumvented. Particularly in cases where no destination pixel is ultimately painted, the Slow-DPDMu (e.g. Regis) does not waste time doing what ultimately needs not to have been done in the first place.

(It is noted here as an aside that this disclosure repeatedly refers to the term "spryte." Conventional imaging systems are built around the concept of a "sprite". The different spelling for the earlier mentioned "spryte" is intentional. A conventional "sprite" consists of a rectangularly-shaped area of image data. All scan lines of a conventional sprite have the same length. It has been found through experience that internal change of data contents within a sprite predominantly take place within non-rectangular subportions of the rectangular sprite areas. Time and memory is wasted in conventional systems by repeatedly rendering the nonchanging subportion. To avoid such waste, systems in accordance with the present invention preferably utilize an image construct that is referred to as a "spryte". (Note the pronunciation is the same

as "sprite" but the spelling is different.) A "spryte" is defined as a compilation of horizontal scan-lines extending from, and to the right of, a vertical (hypothetical) spryte edge line where each scan line includes a number of successive source pixels. The length of each spryte scan-line is independently controlled by an EOL (end-of-line) terminating code or other appropriate means. The top point on the spryte edge line is defined by a spryte corner position. The total number of horizontal lines which collectively define a spryte is given by a spryte line count. A spryte can include scan-lines with no pixels in them.)

(§1.4) SUMMARIZED EXAMPLE 4: NONPERFORMANCE OF MATH FUNCTION ON MORE SIGNIFICANT END OF RESULT (USES DESIGN RULES 1, 4, 7 and 8)

When a small delta value (Δx) is being added to, or subtracted from, a large running total ($x = x + \Delta x$), it is often found that a more significant part of the running total does not change between successive additions or subtractions of the delta value (Δx). The invention splits sum generation into at least two parts and provides carry detection at an intermediate point of sum generation. Sum generation begins at respective less significant portions of the delta value (Δx) and the running total (x), and halts at the intermediate point if the subsequent operation will produce no change to the more significant part of the running total. One example of this condition is where no carry or borrow is generated at the intermediate stop point and the more significant part of the delta value (Δx) is equal to zero. Another example of this condition is where a borrow is generated at the intermediate stop point and the more significant part of the delta value (Δx) is

equal to positive one. Yet another example of this condition is where a carry is generated at the intermediate stop point and the more significant part of the delta value (Δx) is equal to minus one. In each case, the more significant part of the running total is left unchanged rather than wasting time adding zero to it. (The term "sum generation" is understood to include addition and/or subtraction of signed or unsigned quantities.)

More specifically, the system uses an adding unit (e.g., an arithmetic logic unit or ALU) having a bit-width less than that of the final sum and a long register means for storing the sum. A sum portion stored in a more significant part of the long register means is updated only if a carry is detected during generation of an immediately preceding, less significant part of the sum. The circuitry which implements this approach uses less area on an integrated circuit than conventional circuitry for carrying out long-precision summation. The approach reduces the number of clock cycles required for generating each high-precision sum in cases where the delta is small.

This delta summing mechanism is used by the corner engines, the Slow-DPDMu, and other units (e.g. Row-Initializer) to respectively calculate polygon corner coordinates, border point coordinates, or other vlues requiring a relatively high degree of precision (e.g., better than 16 bits of precision).

(§1.51) SUMMARIZED IMPLEMENTATION OF EXAMPLE 1

A spryte rendering system in accordance with a first aspect of the invention comprises: (a) one or more color-mapping circuits for developing color signals representing potential destination color values from supplied source color signals; and (b) one or more

destination-point mapping circuits for simultaneously developing code signals representing destination location values and destination paint/don't-paint decisions.

More specifically, the color-mapping circuit includes: (1) a source data decompressor (unpacker and IPS) and (2) a source color data manipulator (PPMP).

The destination-point mapping circuits include: (1) one or more corner engines for defining corner coordinates of hypothetical geometric shapes (e.g., polygons) which are projected onto the destination grid in accordance with a selected mapping function; (2) one or more border-point locators (Slow-DPDMu (e.g. Regis) circuits) for identifying opposing points on left and right borders of each projected polygon and determining which, if any, destination pixels are sufficiently bounded by the opposing points to warrant painting by a correspondingly developed color; (3) one or more fast-decision circuits (Fast-DPDMu (e.g. Munkee) circuits) for identifying corner mapping conditions in which only one or none of the destination pixels will be painted and for instructing the border-point locators (Slow-DPDMu (e.g. Regis) circuits) to abort their operations and for further providing a one-pixel paint command in the case where only one destination pixel is to be painted; and (4) destination line fill means, responsive to the border-point locators (Slow-DPDMu (e.g. Regis) circuits) and the fast-decision circuits (Fast-DPDMu (e.g. Munkee) circuits), for generating write requests that ultimately overwrite the data of, and thereby paint a line of one or more destination pixels, in response to line fill commands supplied by the border-point locators (Slow-DPDMu (e.g. Regis) circuits) and the fast-decision circuits (Fast-DPDMu (e.g. Munkee) circuits).

The spryte rendering engine additionally includes: (c) one or more math resource engines whose computational

and data-storage resources are shared by the corner engines, Fast-DPDMu (e.g. Munkee) units and Slow-DPDMu (e.g. Regis) units, (d) memory means for storing source pixel data representing source pixels and for storing destination pixel data representing destination pixels that are to be painted in accordance with write requests sent by the line fill means; and (e) display means for displaying the pixels represented by the destination pixel data.

(§1.52) SUMMARIZED IMPLEMENTATION OF EXAMPLE 2

A spryte rendering system in accordance with a second aspect of the invention comprises a plurality of corner engines each for calculating polygon corner coordinates from an input data set, where the input data set includes position data (XPOS, YPOS) defining the coordinates of a top-left corner (a0) of a first pixel, line delta data (LDX, LDY) defining the coordinates of a bottom left corner (a3) of the first pixel, and row delta data (DX, DY, DDX, DDY) defining the coordinates of corners points (a1, a2) which succeed from the defined top-left corner (a0) of the first pixel and from the defined bottom-left corner (a3) of the first pixel. The spryte rendering system further comprises a shared storage unit (register stack) whose locations are accessible by the plurality of corner engines. The corner engines exchange results for shared destination points by way of the shared storage unit (register stack). The shared results include: (1) calculated coordinates for the bottom left corner (a3) of the first pixel in each successive source row, and (2) progressively increased or decreased delta values (DX++ = DX + DDX) calculated by a first of the corner engines and useable by a second of the corner engines.

(§1.53) SUMMARIZED IMPLEMENTATION OF EXAMPLE 3

A spryte rendering system in accordance with a third aspect of the invention comprises one or more border-point locators (Slow-DPDMu (e.g. Regis) circuits) for identifying opposing points on opposed left and right borders of each projected polygon and for determining which, if any, destination pixels are sufficiently bounded by the opposing points to warrant painting by a correspondingly developed color. For each border-point locator (Slow-DPDMu (e.g. Regis) circuit), a corresponding fast-decision circuit (Fast-DPDMu (e.g. Munkee) circuit) is provided for identifying corner conditions in which only one or none of the destination pixels will be painted and for instructing the corresponding border-point locator (Slow-DPDMu (e.g. Regis) circuit) to abort its operations and for further providing a one-pixel paint command in the case where only one destination pixel is to be painted.

(§1.54) SUMMARIZED IMPLEMENTATION OF EXAMPLE 4

A precision calculating apparatus in accordance with a fourth aspect of the invention comprises: (a) two or more part registers for storing respective more-significant and less-significant parts of a long result; (b) a math unit; (c) selective updating means for selectively coupling the math unit to one or another of the part registers and updating the result part stored in the selected part register; (d) detection means for detecting conditions where the updating of one result part necessitates the updating of another result part; and (e) control means for instructing the selective updating means to select and update the other part of the long result only if the detection means indicates a necessity of so doing.

(§2) BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be described in detail with reference to the following drawings, in which:

FIGURE 1A shows an example of a first source-to-destination image mapping, where the mapping function includes image enlargement.

FIGURE 1B shows an example of a second source-to-destination image mapping, where the mapping function includes image reduction.

FIGURE 2 shows other mapping possibilities.

FIGURE 3A illustrates one method for defining a source-to-destination point-to-point mapping.

FIGURE 3B illustrates another result using the same mapping method described for Fig. 3A.

FIGURE 4 is a block diagram of a first image rendering system in accordance with the invention.

FIGURE 5A briefly explains the operations of the Slow-DPDMu (e.g. Regis) boundary-walking unit and the region-fill math unit.

FIGURE 5B briefly explains the operations of the Fast-DPDMu (e.g. Munkee) border estimating unit.

FIGURE 5C shows how the Slow-DPDMu (e.g. Regis) and the Fast-DPDMu (e.g. Munkee) view a projected polygon after the coordinate values of its corners have been truncated.

FIGURE 6 (composed of subfigures 6A-6F) is a block diagram of a second image rendering system in accordance with the invention which includes two color-code unpacking units and a corresponding set of two corner-calculating engines.

FIGURE 7 (composed of subfigures 7A-7H) is a schematic diagram of a math platform in accordance with the invention that is used by the corner-calculating and boundary-walking engines of the invention.

FIGURE 8 is a diagram used to explain how parallel processing of information from adjacent source rows speeds performance and how the bypassing of ultimately unnecessary polygon calculations speeds performance.

FIGURE 9A shows how precision sum generation with intermediate carry detection works.

FIGURE 9B shows how double delta precision extends 20 bits to the right of the decimal point while the precision of delta addition extends a lesser 16 bits to the right of the decimal point.

(§3) GLOBAL DESCRIPTION OF SPRYTE MAPPING METHOD

The invention is now explained in more detail by sequencing through an evolutionary series of concepts.

The mapping of a source image onto a destination grid is divided into the following steps (1a) through (7):

(1a) A point-to-point mapping function is defined.

(1b) A source-to-destination color mapping function is defined concurrently.

(2) For each source pixel, a corresponding and hypothetical geometric shape (e.g., a quadrilateral or other polygon) is projected onto a destination grid in accordance with the defined point-to-point mapping function.

(3) The destination grid coordinates of top, bottom, left, right and central corners (if any) of the projected polygon are identified.

(4) Opposed left and right borders of the projected polygon are identified.

(5) Opposing points on the opposed left and right borders of the projected polygon are identified and their coordinates are calculated (not necessarily to full precision).

(6) For each pair of opposed border points, a determination is made as to how many, if any, destination pixels are to be painted because they are fully or effectively contained between the opposed border points or because they meet some other paint-decision criteria.

(7) Destination pixels that meet the paint-decision criteria are painted with a color derived from the corresponding source pixel (or derived from a plurality of corresponding source pixels) in accordance with the source-to-destination color mapping function.

Figure 1A illustrates a first, relatively elementary, mapping 100 of a source image 110 onto a destination grid 120. The illustrated mapping 100 copies the source image 110 onto the destination grid 120 in a manner which greatly enlarges the size of the destination image 121 relative to the size of pixels in the destination grid 120 and which further rotates the destination image 121 clockwise relative to a horizontal axis (X_D) and a vertical axis (Y_D) of the destination grid 120.

The source image 110 is defined as a rectangular array of source pixels: a, b, c, d, . . . , q, r, s, t, . . . , z; where each underlined lower case letter (a, b, c, etc.) refers to a specific pixel of the source image 110. The source grid and source image pixels correspond on a one-to-one basis. Each individual rectangular or square area of the source grid is filled with a single like-oriented and like-sized pixel of the source image 110.

In a preferred embodiment of the invention, the source image 110 is composed of square pixels, the destination image 120 is composed of square pixels, and the ultimately displayed light image is composed of

square pixels which are the same as those of the destination image 120 or derived therefrom.

For the sake of illustrative convenience, source pixel a is considered to be the first pixel in a first row, "A", of the source image 110. Source pixel q is considered to be the first pixel in a second row, "B", of the source image 110. Source pixel w is considered to be the first pixel in a third row, "C", of the source image 110.

Each source pixel (a, b, c, etc.) is painted with a specific, single shade or color (e.g. COLOR(a), COLOR(b), COLOR(c)). It is to be understood that the source pixels do not contain letters, a, b, c, etc. These are merely identifiers. Instead, the source pixels are each completely filled with a respective one of COLOR(a), COLOR(b), COLOR(c), etc. Likewise, the pixels of the destination image do not contain capital letters, P,P,P, etc., nor do they contain portions of enlarged letters a, b, c, etc. These are merely identifiers. The mapped projections of source identifiers a, b, c, etc. are shown enlarged and rotated (but without underlines) over the destination grid 120 to illustrate the effects of mapping function 100.

When mapping and image rendition ultimately complete, it is expected that some of the destination pixels (P,P,P, etc.) will be painted with respective colors, COLOR*(a), COLOR*(b), COLOR*(c), etc. Thusly painted destination pixels represent the mapping and ultimate rendition of the source image onto the destination grid 120. The destination colors are derived from those of the source pixels. The asterisk in the destination notation COLOR*(i) indicates that COLOR*(i) can be, but is not necessarily the same as a corresponding source color, COLOR(i). Rather the COLOR*(i) of a specific destination pixel, P_i, should be thought of as one that

is derived from (mapped from) the source color, COLOR(i), of a given source pixel, i. (And to make matters more complex, in one embodiment of the invention, the color of each destination pixel can be a function of colors found in plural source pixels, and even colors found in different sprytes, due to the the action of a so-called PPMP unit.)

Each source pixel (a, b, c, etc.) is defined as having four corner points associated with it. The four corner points of source pixel a are respectively referenced, moving clockwise from the top left corner, as a0, a1, a2 and a3. The four corner points of source pixel b are similarly referenced as b0, b1, b2 and b3. And the same referencing system is used for all other pixels of the source image 110. (The underline under the numeral portion of a0, a1, a2, a3, b0, etc. indicates that it is a corner of a source pixel. Similar symbols without underlines (a0, a1, a2, etc.) will be used below to represent corner points of projections of the source pixels onto the destination grid.)

For purposes of explanation, source coordinate system is additionally defined as having an x_s axis extending left to right in the horizontal direction and a y_s axis extending top to bottom in the vertical direction. The corner points of source pixels lie at integer values of x_s and y_s. By way of example, corner c0 has coordinates x_s=2.0 and y_s=0.0 in the source grid. (The zeroes to the right of the decimal points in these expressions can be truncated.) Similarly corner c2 has coordinates x_s=3.0 and y_s=1.0. (The fractional portions to the right of the decimal points can be truncated in these designations as well. They are included here to help explain a concept that is developed later below.)

Corner points of pixels in adjacent rows or columns are shared. By way of example, Fig. 1 shows pixels a,

b, c, and d as belonging to a first row, A, and pixels q and r as belonging to a second row, B, of the source image. Corner point a3 is the same as corner point q0. (Both have coordinates x_s=0, y_s=1.) Corner point a2 is the same as corner point q1. Corner point a2 is also the same as corner point b3 and corner point r0. (All four share coordinates x_s=1, y_s=1.)

Destination grid 120 is shown to be comprised of another array of pixels: P, P, P, . . . , P. The destination pixels (P,P,P) are arranged in rows and columns. A destination coordinate system is defined to have an X_D axis extending left to right in the horizontal direction and a Y_D axis extending top to bottom in the vertical direction. The corner points of destination pixels (P, P, ..., P) lie at integer values of X_D and Y_D.

Mapping function 100 can be divided into two subfunctions, namely, point-to-point mapping and source-color to destination-color mapping. Point-to-point mapping defines a one-to-one translation between each pair of x_S and y_S coordinates in the source image and a corresponding pair of X_D and Y_D coordinates in the destination grid 120. Color mapping, on the other hand, defines a relation between the colors of pixels in the source image and the colors that will be assigned to pixels of the destination grid 120.

Referring to Figure 1B, a second mapping 102 of source image 110 onto destination grid 120 is illustrated. This time, the source image 110 is mapped onto destination grid 120 in a manner which greatly reduces the size of destination image 122 relative to the pixels of destination grid 120. Mapping 102 further rotates destination image 122 clockwise relative to the horizontal axis (X_D) and vertical axis (Y_D) of destination grid 120. Note that destination pixels P0, P1, P2, P3, etc., are now shown to be larger than the

mapped-over pixel "a". (Note also that no underline is used for the symbol "a" which represents the mapped-over projection of source pixel a.)

As previously mentioned, the steps taken in mapping a source image 110 onto a destination grid 120 include: (1a) defining the point-to-point mapping function (100 or 102), and (2) for each source pixel (a, b, c, etc.), projecting a corresponding and hypothetical polygon (or other geometric shape) onto the destination grid in accordance with the defined mapping function. Quadrilaterals 125 and 126 represent two such projected polygons in the case of reduced image 122 (Fig. 1B). Quadrilaterals 127 and 128 represent two further examples of projected polygons in the case of enlarged image 121 (Fig. 1A).

The projected polygons 125, 126, 127, etc. represent the "ideal" mapping of each source pixel onto destination grid 120. The ideal mapping becomes the actual rendition only when the projected polygons (a, b, c, etc.) align perfectly with and cover integral numbers of destination pixels (P, P, P, etc.).

In all other cases, where all the borders of the projected polygons do not align perfectly on integral values of X_D and Y_D, the actual rendition at the destination 120 is an imperfect version of the ideal point-to-point mapping. Yes/No decisions have to be made whether or not to paint each destination pixel (P,P,P) with a specific color COLOR*(i), the specific color being the one that corresponds to the projected polygon "i" that covers that destination pixel. Since a destination pixel (P) can not be partly filled with different colors from two projected polygons, e.g., a and q, one of plural destination color candidates has to be chosen for cases where multiple projected polygons cover a given destination pixel. (Each destination pixel (P_i) is a

quantum construct which must be filled with one and only one color value, COLOR*(i).)

This concept is best understood by referring to Fig. 1B. Although destination pixel P3 is partly covered by a first quadrilateral 125 which represents the ideal mapping of source pixel a and by a second quadrilateral 126 which represents the ideal mapping of source pixel q, destination pixel P3 will only be painted with one color value; either that of mapped pixel "a" or that of mapped pixel "q".

In the illustrated case, it would make sense to a human being to choose the color of mapped pixel "a" as the one to fill destination pixel P3 because most of polygon 125 resides over P3. Similarly, it would make sense to choose the color of mapped pixel "q" for destination pixel P2 because a substantial part of polygon 126 resides over P2. This is an easy thing for a human being to see. It's a more difficult process when a machine has to make the choice automatically.

On the other hand, it should be appreciated that most destination images are formed of a large number (e.g., hundreds) of destination pixels, and when the human eye views the destination image on a grand scale, from far away, a single improperly painted pixel here or there usually makes little difference. A machine-implemented paint/don't-paint decision-making process and paint with-which-color decision making process, which operates on this grander scale, will be explained in more detail later below.

A similar decision-making problem applies to the size-enlarging map function 100 of Fig. 1A. Although most destination pixels (P,P,P,..) are completely covered by quadrilaterals such as 127 or 128, thereby making the decision easy for these, there will be a few destination pixels (P,P,P) that lie partly inside and partly outside

the borders of large quadrilaterals 127, 128. For each such partially-covered destination pixel, a decision has to be made to use either the color of mapped pixel "a" or that of mapped pixel "q" or that of another source. It is observed here again, that when the entire destination image is viewed from far away, local variances in the picking of one destination pixel over another will usually not make any differences to the perceived image as a whole.

A relatively simple first algorithm may be used in one particular embodiment of the invention: The color of the projected polygon that encloses the top left corner of each destination pixel is used as the fill color for that destination pixel. This simple algorithm is referred to here as the "top-left paint algorithm". (Other, more complex, paint-choice algorithms will be described later.) If the corners of two or more projected polygons align perfectly with that of the top left corner of a destination pixel such that neither encloses the destination pixel corner, a choice has to be made as to which source pixel is most suitable for having its color mapped into the destination pixel.

The simple top-left paint algorithm can be illustrated by way of example. In Fig. 1B, projected polygon 125 (the mapped "a" pixel) encloses the top left corner point (X_D=m.0, Y_D=n.0) of destination pixel P3 (where m and n represent integers). Accordingly, destination pixel P3 is painted with a "mapped" color that corresponds to mapped pixel "a". (The term "mapped color" means that it is either the same as that of the source pixel or related to it in some way, through a color-mapping function.)

The top left corner point (X_D=m.0, Y_D=n.0+1) of destination pixel P5 is covered by mapped pixel "q". If the simple top-left paint algorithm is applied,

destination pixel P5 will be painted with a "mapped" color that corresponds to mapped pixel "q". Obviously this choice is not good when viewed at the local (microscopic) level illustrated in Fig. 1B. Only a very small portion of projected polygon 126 ("q") overlaps with the area of destination pixel P5. On the grand-scale, however, this may not make much of difference to the perception of the overall destination image.

According to a more complex, second paint/don't-paint decision algorithm, the image rendering machine looks at both the top-left and bottom-left corners of a destination pixel and the machine determines how many and which projected polygons touch or cover each of the two corners. This is referred to here as the left top/bottom algorithm.

In Fig. 1B for example, the top-left corner of destination pixel P3 is covered by projected polygon 125 ("a") and the bottom-left corner of destination pixel P3 is covered by projected polygon 126 ("q"). One or the other of the top-left and bottom-left corners of the destination pixel has to be selected as having the most desirable one polygon covering it (or the most desirable set of plural candidates touching it). This is referred to as a "corner-select" process. If more than one polygon touches the selected corner, a second selection is made to pick the one polygon whose color will be used.

Yet a third paint-choice algorithm could be used in which the image rendering machine looks at all four corners of a destination pixel and selects one of the four corners as the one with the best candidate polygons.

It is to be understood that other paint/don't-paint algorithms can be used in conjunction with the invention. A relatively complex algorithm, named REGIS_FILLBIN which relies on preferential corner selection will be explained later. For now it is sufficient to explain it as

including a pre-processing step in which the fractional portions of signals representing mapped corner points a0 through a3 are truncated away. This has the effect of distorting the projected polygons at the microscopic level but it generally makes little difference at the macroscopic level. Each of the truncated coordinate values is thereafter considered to be a command to paint the destination pixel for which it (the truncated coordinate value) serves as a top-left corner point.

Despite its apparent simplicity, the top-left paint algorithm poses a series of further problems. (The same problems apply to the left bottom/top algorithm and to other similar paint-choice algorithms.)

A first problem is how to efficiently and economically decide whether a projected polygon (e.g., 125) encloses or does not enclose a top left corner point of a given destination pixel (e.g., P3), or any other corner point for that matter. A second problem is how to generate the corresponding series of paint decisions in minimal time. Yet another problem is how to perform these functions using circuitry of minimal size. (Ideally, one would like to incorporate all the circuitry within one or a few integrated circuit chips in order to minimize manufacturing cost, minimize circuit size and also minimize wire-length delays.)

Related but additional problems arise in the realization of the above-mentioned other mapping steps of: (3) Identifying the destination grid coordinates of top, bottom, left, right and middle (if any) corners of the projected polygon; (4) Identifying the opposed left and right borders of the projected polygon; (5) Identifying opposing points on opposed left and right borders of the projected polygon and calculating the coordinates of these points; and (6) For each pair of opposed border points, and a preceding pair of opposed

border points, determining how many, if any, destination pixel corners are fully or effectively contained between the set of four border points.

While these problems are in mind it is to be noted that there are many different kinds of size, angle and shape transformations in the field of digital graphic processing. Fig.s 1A and 1B merely show two examples. See Fig. 2 for examples of other mappings. (Fig. 2 will be described shortly.)

The size/angle transformation 100 illustrated in Fig. 1A places corner point a0 of the projected polygon 127 at a top point GT of destination grid 120. It leaves corner point a1 at a right side point, GR, of the destination grid 120, where point GR is a few rows below the top grid point GT and a few columns to the right of GT.

Mapping 100 further puts corner point a3 at grid location GL, which is below and to the left of GT. GL and GR are not necessarily on a same Y_D coordinate of the destination grid 120.

Mapping 100 further positions corner point a2 at a bottom grid point, GB, below grid points GL and GR. It is noted that although Fig. 1A places the top, bottom, left and right corners (GT,GB,GL,GR) of the projected polygon 127 at integer X_D and/or Y_D coordinates of the destination grid 120, this is not always the case. The a0 through a3 corners of the projected polygon 127 might have been just as easily placed at non-integer coordinates of the destination grid 120, and moreover, each projected corner could have been mapped to any arbitrary point of the destination grid 120, including all to the same point.

The left-side boundary of the projected polygon 127 shown in Fig. 1A is defined by lines which sequentially connect destination grid points GT, GL and GB. The

right-side boundary is defined by lines which sequentially connect grid points GT, GR and GB. A destination grid point (e.g., X_D=m.0, Y_D=n.0) which is positioned between the left and right boundaries of projected polygon 127 is deemed to be inside that polygon, otherwise it is outside the polygon.

Once a top point (e.g., GT) is identified for opposed left and right polygon boundaries, two lock-step walks are taken along those boundaries, starting at opposed tops of the boundaries and ending at opposed bottom points. The opposed top points or bottom points can be one and the same as in the case of GT and GB in Fig. 1A. For each lock step along the left boundary and right boundary, a determination is made as to how many and which destination pixel corners are bound between the opposed walk points. This aspect will be more fully explained later, in conjunction with Fig. 5A.

Before moving onwards in this discussion, it is worthwhile to consider some other geometry mapping possibilities. Referring to Figure 2, it is seen that a variety of other polygon mappings (e.g., 210, 220, 230) are possible and each one produces different left and right side boundary definitions.

Mapping 210, for example, transforms a square-shaped source pixel, a, into a bow-tied configuration 211 at the destination. Corners a0 and a1 define the respective top left and top right corners of the resulting bow-tie image 211. Corners a2 and a3 define the respective bottom left and bottom right corners of the bow tie image 211. The left-side boundary of bow-tie 211 is defined by the upper part of line a0-a3 and the bottom part of line a1-a2. The right-side boundary of image 211 is defined by the upper part of line a1-a2 and the bottom part of line a0-a3.

Another possible mapping, 220, puts corner a0 at the bottom of destination image 221. Corners a1 and a3 are squeezed very close to one another. It is possible for destination image 221 to be so thin that it fails to effectively cover any pixel (P) of the destination grid 120 (Fig. 1A). In such a case, image 221 may not even appear within the destination image.

Yet another mapping, 230, reverses the ordering of corners a0-a3 at the destination image 231. While the sequence a0, a1, a2, a3 defines a clockwise rotation in the source image, it defines a counter-clockwise rotation at the destination image 231. If source pixel a is considered to be part of a frontward-facing surface of a three-dimensional cube, destination image 231 might be considered as part of a backward-facing surface of a rotated (and skewed) reproduction of the three-dimensional cube. It is sometimes useful to indicate whether the a0-a3 sequence runs clockwise (CW) or counterclockwise (CCW). In a wire-frame rendition, CW and CCW oriented images are both shown. In a solid-surface rendition, it may be desirable to "hide" (not render) one or the other of CW and CCW oriented images.

There are many other possible mappings. Mapped image 241, for example, has three corners arranged in a line at its top, thereby forming a triangle. Mapped image 251, has two corners at its top, a third corner at a bottom-most position and a fourth corner positioned between the first three corners, thereby forming a two-horned ox-face shape. (Note: for the embodiment later described in conjunction with Fig.s 6 and 7, generation of a two-horned ox-face shape is not allowed. This simplifies circuit design because there will always be one corner for the other mappings 210-241 that can serve as a top corner from which an always-downward boundary walk can be performed.)

The mappings shown in Fig. 1A, Fig. 1B and Fig. 2 are merely examples of a relatively large number of possibilities. Any one or more such mappings (e.g. 100, 102, 210, etc.) may be used singularly, multiple times in-parallel or recursively in-series to generate a destination image.

After corners a0 through a3 are mapped onto the destination grid 120, the relative positions of corners a0-a3 of a projected polygon are identified. The relative position information defines each corner as being a top, bottom, left, right, or center corner in relation to the other corners.

A machine-implemented algorithm produces such relative position information by deciding which and how many of the projected corners have the minimum Y_D coordinates (top-most position), the maximum Y_D coordinates (bottom-most position), the minimum X_D coordinates (left-most position), and the maximum X_D coordinates (right-most position). This identifies the basic shape of the projected polygon (e.g., simple quadrilateral, bow-tie, triangle, ox-face, line, point) and also the CW or CCW orientation of the projected polygon.

Some form of precise or less-than precise top-to-bottom boundary walk is then conducted along every pair of opposed left and right linear edges of the polygon to decide which destination points lie (precisely, or in some embodiments of the invention, approximately) inside the projected polygon and which lie outside. One or more top (not necessarily "top-most") corner points on the projected polygon serve as starting points. The boundary walking operation moves two pointers (left and right side pointers) down in lock step along opposed sides of the projected polygon, from the one or more starting points

to an opposed one or more bottom points of opposed left and right edges of the projected polygon.

With each step in the boundary walk, a determination is made as to what coordinates the left and right edge pointers point to and, in some embodiments of the invention, whether one or more destination grid points, each having integral coordinate values (e.g., X_D=m.0, Y_D=n.0), lie between the opposed left and right boundaries of the projected polygon. If such interior, integer-coordinate points exist, a fill-command is generated to fill their corresponding destination pixels with a mapped color assigned to the projected polygon. (A "precise" version of the boundary walk is explained in more detail later, in conjunction with Fig. 5A.)

Just as there are many final shapes for a mapped-over pixel (as seen in Fig. 2), there are a variety of ways in which to define a source-to-destination point-to-point mapping. Figure 3A shows one such method 300 that is used in conjunction with the present invention. The illustrated method 300 is referred to here as the "double-delta method".

In Fig. 3A, the upper left corner point, a0, of the source image 110 is first mapped to coordinates X_{a0} and Y_{a0} of the destination grid 120. Coordinates X_{a0} and Y_{a0} serve as a starting point for the destination-point mapping.

Coordinates, X_{a0} and Y_{a0}, are respectively also referred to as the XPOS and YPOS coordinates in this disclosure.

Source image 110 is also referred to as a source "spryte" 110 in this disclosure. (The spelling of spryte with a "y" instead of the industry recognized spelling "sprite" is intentional and has already been explained above.)

Coordinates X_{a0} and Y_{a0} are each expressed as a 32-bit-wide dataword. The first 16 bits of the 32-bit expression define an integer value and the second 16 bits define a fraction value. (This data structure is abbreviated here by the shorthand notation, "16.16".) The double-delta mapping method 300 can therefore position the mapped starting point, a0, at non-integer coordinates of destination grid 120.

Two line-to-line delta values, LDX and LDY, are given in conjunction with starting coordinates, X_{a0} and Y_{a0}. Values LDX and LDY define the respective distances along the X axis and Y axis of the destination grid 120 which separate destination starting point a0 from destination corner point q0 (the top left corner of the second row in source image 110). Corner point q0 is the same as corner point a3. Line-to-line delta values, LDX and LDY, are each expressed in 16.16 format.

The destination grid coordinates of destination corner point a3 are then calculated as follows:

$$X_{a3} = X_{a0} + LDX \qquad \text{(Eq. 1a)}$$

$$Y_{a3} = Y_{a0} + LDY \qquad \text{(Eq. 1b)}$$

In more general terms, points along line a0-a3-q3-w3-etc. are calculated as follows:

$$X_k = X_{k-1} + LDX \qquad \text{(Eq. 1'a)}$$

$$Y_k = Y_{k-1} + LDY \qquad \text{(Eq. 1'b)}$$

where the index "k" represents a current member of the series: a3, q3, w3, etc. and the index "k-1" represents a preceding member of the series: a0, a3, q3, w3, etc.

Once the destination coordinates (X_{a0}, Y_{a0}) and (X_{a3}, Y_{a3}) of respective points along first linear path a0-a3-q3-etc. are known, a second linear path from point a0 to point a1, to point b1, etc., is defined along the destination grid by providing another set of delta values, DX and DY. Delta values DX and DY are each expressed in 12.20 format.

The destination coordinates (X_{a1}, Y_{a1}) of corner point a1 and subsequent points b1, c1, d1, etc. are then calculated according to the following equations:

$$X_i = X_{i-1} + DX \quad \text{(Eq. 2a)}$$

$$Y_i = Y_{i-1} + DY \quad \text{(Eq. 2b)}$$

where the index "i" represents a current member of the series: a1, b1, c1, d1, etc. and the index "i-1" represents a preceding member of the series: a0, a1, b1, c1, d1, etc.

Note that X_i and X_{i-1} are each represented as digital signals having 16.16 (integer.fraction) format while DX and DY are each represented as digital signals having 12.20 (integer.fraction) format. The addition result of Eq.s 2a and 2b automatically truncates the least significant four bits of DX and DY and pads their fronts with an appropriate number of sign-extension bits.

If truncation error is undesirable, the destination coordinates (X_{n1}, Y_{n1}) of successive corner points a1, b1, c1, ..., n1 can be alternatively calculated according to the following equations, Eq. 2'a and Eq. 2'b:

$$X_i = X_{i-1} + \text{INTEGER}(DX) + \text{INTEGER}(AccXerr) \quad \text{(Eq.2'a)}$$

$$Y_i = Y_{i-1} + \text{INTEGER}(DY) + \text{INTEGER}(AccYerr) \quad \text{(Eq.2'b)}$$

The INTEGER(x) function produces a result in which any fractional portion of argument "x" is stripped away, thus leaving only the integer portion of "x" as the result.

The terms, AccXerr and AccYerr, represent accumulated plotting errors. Quite often, a round-off correcting algorithm such as that of Bresenham is used to approximate a straight line in a bit-mapped image. Such a line is drawn in the destination grid 120 for connecting points a0, a1, b1, c1, etc., one to the next. The line-approximating algorithm has an accumulating fractional-error part which occasionally exceeds a threshold level (e.g. 0.5) and causes a corresponding

integer-error part to increment or decrement by the integer, one. A more detailed description of the Bresenham and other line approximating algorithms may be found in "Computer Graphics, Principles and Practice" by J. Foley, A. Van Dam, S. Feiner and J. Hughes; Addison-Wesley Publishing Company, New York, Second Edition 1990.

After the second linear path through points a0, b0, c0, etc., is plotted onto the destination grid 120, a similar procedure is used for plotting the points q0, r0, s0, etc., which define the top of the second row (B) of the source spryte 110 onto the destination grid 120. Since point a3 is the same as point q0 and point a2 is the same as point r0 and b3, Fig. 3A shows both designations.

Double delta values DX+DDX and DY+DDY are used for defining successive points a3, b3, c3, etc. as follows:

$$X_j = X_{j-1} + DX + DDX \qquad \text{(Eq. 3a)}$$

$$Y_j = Y_{j-1} + DY + DDY \qquad \text{(Eq. 3b)}$$

where the index "j" represents a current member of the series: b3, c3, d3, etc. and the index "j-1" represents a previous member of the series: a3, b3, c3, d3, etc. A Bresenham or other line approximating algorithm can be used to generate these values digitally. Second delta values DDX and DDY are each represented as digital signals having 12.20 (integer.fraction) format.

Next, a linear path extending from point q3 is defined by:

$$X_j = X_{j-1} + DX + 2 \bullet DDX \qquad \text{(Eq. 4a)}$$

$$Y_j = Y_{j-1} + DY + 2 \bullet DDY \qquad \text{(Eq. 4b)}$$

where the index "j" this time represents a current member of the series: r3, s3, t3, etc. and the index "j-1" represents a previous member of the series: q3, r3, s3, t3, etc. As before, a Bresenham or other line approximating algorithm can be used to generate these values digitally. Note that the second delta values are

now two times DDX and two times DDY. This sequence continues in the next line by using three times DDX and DDY, then four times the same values, and so forth.

Each of delta values LDX, LDY, DX, DY, DDX and DDY can be positive or negative. Figure 3B shows what happens when LDX=0, DDX=0 and DY is a small positive value and DDY is a slightly smaller negative value. The lines extending to the right from edge points a0, a3, q3, w3, etc. converge. If they extend sufficiently far to the right, a bow-tie configuration is produced.

It is to be understood that DX and DY can start as negative values (and DDX and DDY can be zero or negative at the same time), in which case a double delta mapping starts at point (Xa0, Ya0) and draws the remainder of the destination image heading towards decreasing values of X_D and Y_D.

Starting coordinates, XPOS and YPOS, can also be negative, as seen in Fig. 3B. Note that in Fig. 3B the X_D axis and Y_D axis intesect at a point other than 0,0. The 0,0 origin point is shifted for purposes of the immediately following discussion to the middle of mapped pixel "a". (As will be seen later, one embodiment of the invention has an address translater which lets the user move the origin point within a "displayable universe" by changing a so-called Base-address variable. This lets the user move the renderable area to any desired portion of the displayable universe.)

To avoid complications, regions of the destination grid 120 where $X_D<0$ or $Y_D<0$ are defined as "non-renderable". Additional regions of the destination grid, defined by $X_D>X_{CLIP}>0$ or $Y_D>Y_{CLIP}>0$ are also designated as also being non-renderable. X_{CLIP} and Y_{CLIP} are user definable signals. Images or parts of images that map into a non-renderable region are never rendered (painted) into the memory means that stores the data of the

ultimately to-be-displayed light image. This form of rendition cropping is referred to as "simple clipping" and it occurs in one embodiment of the invention due to a decision of a so-called line-fill unit (430, Fig. 4) to not send a write request to a so-called D-Bus access arbitration unit (433).

There is another, more-intelligent cropping process within one embodiment which is called "super-clipping" and which is optionally engaged or disengaged by the user. This process will be described later but is mentioned here to distinguish it from "simple-clipping".

As seen in Fig. 3B, the corner points of a projected polygon (e.g., "a") can be mapped into a non-renderable region. Projecting one or more corner points of a projected polygon into a non-renderable region is a valid operation. The main consequence of such mapping is that portions of a projected polygon that lie in a non-renderable region will never be rendered.

An already-mentioned, key feature of the invention is the avoidance of unnecessary work. This feature manifests itself in many ways. One manifestation occurs when part or all of a projected polygon is found to lie in a non-renderable region. Boundary walk and/or line-fill operations are preferably not-performed for parts of projected polygons that fall into a non-renderable region. This avoids useless work because resulting image data within the non-renderable region is, by definition, never rendered and thus never displayed. Work expended to produce such never-displayed data unproductively wastes time and system resources.

Even though a polygon lands in a nonrenderable region, the process of calculating succeeding destination coordinates for polygon corners can still be useful if there is a possibility that succeeding polygons will come to lie inside a renderable region. In Fig. 3B, for

example, the left sides of projected polygons "a", "q" and "w" lie in a non-renderable region ($X_D<0$) but succeeding pixels such as b, c, d and part of e lie in a renderable region. It is therefore useful, and necessary, to calculate the coordinates of points a0, a3, q3, etc. even though they are in a non-renderable region.

Note that projected polygon "f" and all its successors, if any, on the first row of the mapped spryte lie in non-renderable region. Corner calculating operations are preferably terminated and thereby left undone for polygon "f" and all its successors (g, h, i, etc., not shown) in order to avoid useless work. This a first form of the above mentioned "super-clipping" process. If super-clip mode is active, a test is performed to see if a row then being rendered is heading forever deeper into a non-renderable region. If it is, corner calculating and other rendition operations for that row are terminated.

A second test of the super-clip mode looks for conditions that indicate the last of a succession of renderable rows has been encountered. When such a condition is detected, termination is ordered for a row-initializing process which calculates where the next successive (but non-renderable) row of the mapped spryte starts.

A third test of the super-clip mode looks for conditions that indicate the last renderable part of the current spryte has been encountered. When such a condition is detected, termination is ordered for remaining rnedering operations associate with that spryte, and the spryte rendering engine looks to see if there is any next spryte to be rendered.

The concepts of super-clipping, corner calculations, line-fill and so forth are better understood by next

considering the structure of a first spryte rendering system in accordance with the invention.

(§4) SPRYTE RENDERING ENGINE OVERVIEW

Figure 4 shows a first image rendering system 400 in accordance with the invention that uses the above concepts.

A video random-access memory subunit (VRAM) 410 is provided within a system memory unit 405. (Memory unit 405 can include other memory devices such as DRAM and RAM-BUS™.) VRAM subunit 410 includes a first region 410s that stores compressed or non-compressed codes representing the colors of respective source pixels a, b, c, etc., in one or more source sprytes, e.g., 411 and 412. The memory contents of this first VRAM region 410s correspond to the pixels of the source image 110 (Fig. 1A). This memory region 410s is also referred to as the source memory region 410s.

A second region 410d of VRAM subunit 410 stores codes representing the colors of pixels (P,P,P) in the destination grid 120 (Fig. 1A). This memory region 410d is also referred to as the destination memory region 410d. (Memory regions 410s and 410d can overlap or even be one and the same.)

Signals representing source color codes (COLOR(a), COLOR(b), COLOR(c), etc.) move out in pipelined sequence from the source memory region 410s, over a system data bus 403, into a color-mapping portion 401 of system 400. As they do so, a register 415, which is referred to as the Source-Control Block register (or SCoB register 415 for short) outputs color-map control signals 415.1 to an IPS unit 414 and a PPMP unit 416 within the color-mapping portion 401. The color-map control signals 415.1 control various color-mapping functions of the color-mapping portion 401.

At the same time, the SCoB register 415 outputs point-map control signals 415.2 to a corner calculating unit 422 of the destination-point mapping portion 402. The point-map control signals 415.2 control various point-mapping functions of the destination-point mapping portion 402.

The SCoB register 415 is loaded with data originally placed into system memory unit 405 by a system CPU or another memory loading source. New color-map control signals 415.1 and point-map control signals 415.2 load into the SCoB register 415 in conjunction with receipt of a corresponding spryte-render command signal (not shown) that commands the system 400 to begin mapping a new source spryte (411 or 412 or a combination of both) to an image destination region 410d.

The SCoB point-map control signals 415.2 include digital signals representing the previously mentioned map control values: XPOS, YPOS, LDX, LDY, DX, DY, DDX and DDY.

The color-map control signals 415.1 output from the SCoB register 415 include digital signals for controlling a so-called Indexed PIN Substituting unit 414 (IPS 414 for short) and a so-called Pen and Palette Manipulating Processor 416 (PPMP 416 for short). The IPS 413 and PPMP 416 receive color codes extracted from source sprytes 411 and 412 and develop a final color code signal 418 (PEN signal) therefrom. The final color code signal 418 is stored in a color-code register 417.

The operational details of the IPS 414 and PPMP 416 are not important for understanding the invention at this point in the discussion. A more detailed description may be found in the above-cited copending application of R. J. Mical et al., entitled, IMPROVED METHOD AND APPARATUS FOR PROCESSING IMAGE DATA [Attorney Ref. No. MDIO4230]. It is sufficient to understand that the development of

the final color code signal 418 is a synchronous pipelined process and it takes a substantial amount of time (e.g., 3 or more ticks of the system clock) to move the data of source spryte 411 from the system memory unit 405 out over the system data bus 403 to and through color mapping path 401.

Color mapping path 401 has a pipelined architecture. Data signals move sequentially in path 401 through a data decompressor which includes one or more unpacking units 413, then through the Indexed PIN Substituting unit (IPS) 414 and then through PPMP unit 416. Data received from the IPS 414 is optionally combined within the PPMP 416 with other data received from a second source spryte 412. (The data of source spryte 412 is pulled from the system memory unit 405 out over the system data bus 403 into path 401 in a manner similar to that used for extracting source spryte 411.) PPMP unit 416 generates the final color code signal 418 (also referred to as a PEN signal 418) as either: (1) an exclusive function of the data received from IPS 414; or (2) an exclusive function of the second spryte data 412 received from memory source region 410s; or (3) a combined function of the the data received from IPS 414 and the second spryte data 412.

The PEN signal 418 that is stored in color-code register 417 therefore represents the result of a pipelined operation that takes considerable time and makes use of a number of significant system resources. The system memory unit 405 and system data bus 403 are chief among the utilized resources because system performance is most affected by bottlenecking on the system data bus 403 and in the system memory 405.

It is worthy to note here that, in many instances, one word (32 or 16 bits) of source image data 411 represents a large number of ultimately produced PEN codes 418. The compressed spryte information 411

constitutes a compressed version of unpacked information that comes out of the unpacking unit(s) 413, IPS 414 and PPMP 416. This is a manifestation of the second corollary to the primary design rule, which says: Wherever possible or practical, one should use compressed data formats and leave the data in compressed format for as long as possible while working on it and decompress the data only when finally necessary. The advantage is that source-image data fetches take little bus time on system data bus 403 thereby freeing time on the bus for more destination-image write operations (for more destination pixel paints per second).

Although significant work is expended in creating PEN signal 418, this does not mean that PEN signal 418 is always used to paint over a destination pixel. In some cases it is simply discarded (429b) without ever being so used. In cases where PEN signal 418 is indeed used to paint over a destination pixel, PEN signal 418 defines either directly or inderectly, the color and/or brightness and/or texture and/or other physical attribute of an ultimately diplayed pixel within an ultimately displayed light image.

One particular signal of interest, which develops during the PEN signal 418 developing process, is a so-called T-bit (transparency bit). The T-bit is output by the IPS unit 414. An active T-bit indicates that the downstream-produced PEN signal 418 is not to be painted (written) over a corresponding one or more destination pixels.

A soon-described destination-point mapping portion 402 of system 400 responds to active T-bits by leaving undone any activities within the destination-point mapping portion 402 that are directed to calculating which corresponding destination pixels are to be NOT painted by a final color code signal 418 that has its

T-bit activated. This is a manifestation of the prime design rule corollary which says: Where possible, leave undone that which ultimately needs not to have been done. (Activation and deactivation of the T-bit provides a quick way to respectively create and uncreate a transparent region within a rendered spryte.)

While a final color code signal 418 develops within the color-mapping portion 401 of system 400, the destination-point mapping portion 402 is set to work converting the point-map control signals 415.2 (XPOS, YPOS, LDX, LDY, DX, DY, DDX and DDY) into a series of polygon corner coordinate signals 424 (e.g., a0-a3). The earlier-mentioned corner calculating unit 422 performs this function.

The destination-point mapping portion 402 of image rendering system 400 then uses the corner coordinate signals 424 produced by the corner calculating unit 422 to decide which, if any, destination pixels are to be painted with the final color code signal 418. In one particular embodiment, the destination-point mapping portion 402 determines first, how many, if any, destination pixels are effectively bound within the borders of the projected polygon that has its corners (a0-a3) defined by corner coordinate signals 424, and second, if such destination pixels (P,P,P) exist, what their addresses are within the system memory unit 405.

In a more preferred second embodiment, fractional portions of the signals representing corner points a0-a3 are truncated and the results are used as crude indicators of which, if any, destination pixels are to be painted with the final color code signal 418.

One or both of a Slow-DPDMu (e.g. "Regis") border locating unit 426 and a Fast-DPDMu (e.g. "Munkee") border estimating unit 425 are assigned the task of converting corner coordinate signals 424 into paint/don't-paint

decision signals. The paint/don't-paint decision signals are used internally within the Slow-DPDMu (e.g. Regis) and Fast-DPDMu (e.g. Munkee) units, 426 and 425, and hence they are not seen in the block diagram of Fig. 4.

If the internal paint/don't-paint decision signals (not shown) that are generated within the Slow-DPDMu (e.g. Regis) or Fast-DPDMu (e.g. Munkee) represent an affirmative conclusion, namely, that a decision has been made to paint one or more destination pixels, the deciding one of the Slow-DPDMu (e.g. Regis) 426 and the Fast-DPDMu (e.g. Munkee) 425 outputs a set of destination paint-request signals 428 to a destination line-filling unit 430.

Destination line-filling unit 430 converts the paint-request signals 428 into a series of memory write requests which, when grnated, place the final color code signal 418 into appropriate locations of destination region 410d of the system memory unit 405 on a raster scan basis. (This function is sometimes referred to as "rasterizing".)

If the Fast-DPDMu (e.g. Munkee) 425 or the Slow-DPDMu (e.g. Regis) 426 decides that more than one destination pixel (P,P,P) will be painted, the PEN signal 418 in color code register 417 is tagged as being "not-discardable." (Register 417 is tagged as being "not empty.") This prevents loading of a new color code into register 417 until all paintings of destination pixels with the current PEN code are complete.

If only one destination pixel (P) is to be painted, the PEN signal 418 in register 417 is tagged as being "discardable." (Register 417 is tagged as being "empty.") PEN signal 418 is used one time and then, in essence, discarded. More accurately, once register 417 is tagged as empty, new data moves down the pipeline to overwrite

the old contents of register 417 thereby deleting the old contents.

When the internal paint/don't-paint decision signals of Slow-DPDMu and Fast-DPDMu units 426 and 425 represent a negative conclusion; namely that a decision has been made by one of these units to not paint any destination pixels (P,P,P), no paint request signals are sent out to line-fill unit 430.

The don't-paint decision signals (not shown) often form first within the Fast-DPDMu 425 but they can also form later within the Slow-DPDMu 426. If a don't-paint decision is made, the Fast-DPDMu 425 and Slow-DPDMu 426 do nothing further with the a0-a3 corner coordinate signals 424 they received from the corner calculating unit 422. Accordingly, no corresponding paint-request signals 428 flow to the destination line-filling unit 430 and no corresponding memory write requests are sent downstream to a soon-described D-Bus access arbitration unit 433. This non-action is indicated symbolically in Fig. 4 by the do-nothing result symbol, 429a (DO NOTH).

If the conclusion of activities within Fast-DPDMu (e.g. Munkee) 425 and/or Slow-DPDMu (e.g. Regis) 426 is to do-nothing 429a, the correspondingly developed PEN code 418 of register 417 is tagged as being immediately "discardable". It is discarded in a subsequent pipeline cycle as indicated symbolically by 429b in Fig. 4.

At any given time where it can be determined that the end result of activities then being performed either in the color-mapping portion 401 or the point-mapping portion 402 of the image rendering system 400 will simply be a do-nothing result 429a, it is advantageous to make such an anticipatory determination as soon as possible and to avoid or abort any related current or future operations of the image rendering system 400 that will ultimately lead to the do-nothing result 429a. Resources

of the image rendering system 400 such as color-code register 417 and Slow-DPDMu 426 are then made available to process new data coming down the pipeline. This is a manifestation of the prime design rule corollary which says: Where possible, leave undone that which ultimately needs not to have been done.

There are several places along the color-mapping path (401) and point-to-point mapping path (402) where the ultimate do-nothing result 429a can be anticipated.

If the T-bit output by the IPS unit 414 is active (true), the Fast-DPDMu (e.g. Munkee) and Slow-DPDMu (e.g. Regis) respond immediately by aborting (or not beginning in the first place) their operations for the corresponding source pixel. Paint-request signals 428 are not sent out for PEN codes that have an active T-bit. At the same time, an "empty" signal (discardable tag, not shown) is developed for the color-code register 417 in response to the do-nothing result 429a produced by one or the other of the Slow-DPDMu 426 and Fast-DPDMu 425. The empty signal enables an ultimate "discard" 429b of the stored PEN signal 418. The latter discard 429b effectively occurs when a new PEN code overwrites the PEN code previously stored in the color-code register 417.

Another situation where the ultimate do-nothing result 429a is anticipated is the case where the a0-a3 corner coordinate signals 424 define a series of projected polygons that are forever heading deeper into a non-renderable region. (This occurs when the earlier-mentioned super-clipping mode is active.) One or more of the corner calculating unit 422, the Slow-DPDMu 426 and the Fast-DPDMu 425 recognize this case as a do-nothing condition 429a and they abort their corresponding activities.

A "terminate" signal 427b (also referred to as a super-clip abort signal 427b) is shown in Fig. 4 to

indicate recognition of a super-clip condtion. If the condition is an encounter of the end of a renderable row (see Fig. 3B), the corner calculating unit 422 responds by aborting its current sequence of corner calculations for that row and makes itself available for calculating projected corners for a new source spryte row. If the super-clip condition is an encounter of a last renderable row in a mapped spryte (see Fig. 3B), the Slow-DPDMu 426 repeatedly compares the placement of its edge walkers (border pointers) against the coordinates of the non-displayable region boundary. when they match, it stops sending subsequent line-fill requests to the destination line-filling unit 430. If the super-clip condition is an encounter of the end of the renderable part of the mapped spryte (see Fig. 3B), the corner calculating unit 422, the Slow-DPDMu 426 and the Fast-DPDMu 425 all terminate their activities for the present spryte and make themselves available for rendering a next spryte.

Yet a third situation where the ultimate do-nothing result 429a, 429b is anticipated is the case where, given certain values of a0-a3 corner coordinate signals 424, the Fast-DPDMu 425 determines in one clock tick that the calculations about to be performed in later clock ticks by the Slow-DPDMu 426 will ultimately fail to produce any paint-request signals 428. The Fast-DPDMu 425 then sends an abort signal 427a to the Slow-DPDMu 426. (Abort signal 427a is also referred to as the Slow-DPDMu abort signal 427a.)

The cooperation between specifc embodiments of the the Slow-DPDMu (e.g. Regis) and of the Fast-DPDMu (e.g. Munkee) is explained in more detail later. For now it is sufficient to appreciate that this third situation is another case where the do-nothing result 429a is anticipated and, as a result, the Slow-DPDMu 426 (e.g. Regis) and the Fast-DPDMu 425 (e.g. Munkee) abort further

operations for the related PEN color code signal 418 and the PEN code 418 is discarded 429b.

If one or the other of the Slow-DPDMu 426 and Fast-DPDMu 425 decides to paint one or more destination pixels, paint-request signals 428 are sent to the destination line-filling unit 430. The destination line-filling unit 430 converts the paint-request signals 428 into memory-absolute requests which, when granted, write the PEN code 418 then present in color code register 417 to the system memory unit 405 on a time shared basis. Unit 430 sends a request (REQ) for memory access to a D-Bus access arbitration unit 433. If there is no higher priority requestor, the D-Bus access arbitration unit 433 returns an access acknowledge signal (ACK) to the destination line-filling unit 430 to grant access for the next available time slot. The destination line-filling unit 430 then sends a relative memory address signal (AO) and a read/write indicator to a system memory address generator 444. The system memory address generator 444 converts each such signal into a single system-absolute address signal which is placed onto a system address bus 404 that connects to the system memory unit 405. Destination line-filling unit 430 keeps track of the number (LEN COUNT) of memory writes needed to complete each line-paint request 428 sent from one of Slow-DPDMu 426 and Fast-DPDMu 425.

When the D-Bus access arbitration unit 433 returns an access acknowledge signal (ACK) to the destination line-filling unit 430, the destination line-filling unit 430 begins to place a sequence of destination pixel codes on a tri-stateable data-output port (DO) provided in it. At the same time, a corresponding sequence of system memory address signals are output onto the system address bus 404 by the system memory address generator 444. The data-output port (DO) of unit 430 connects to the system

data bus (D-bus) 403. Data transfers occur as transfers of 32-bit wide datawords.

A priority scheme defines when and for how long the destination line-filling unit 430 will retain access to the system D-bus 403. Other access requesting units 431 connect to the D-Bus access arbitration unit 433 and make requests for access to system memory unit 405. These other access requesting units 431 are divided into two classes, CPU and DMA REGISTER-STACK (labeled "STACK" in Fig. 4). When access contention occurs, STACK gets highest priority (STACK includes a DMA function of fetching source spryte data, particularly "packed" source data 411, and keeping count of spryte row length). The destination line-filling unit 430 gets second priority and the CPU gets lowest priority. However, in the case where the destination line-filling unit 430 has already obtained access for an immediately preceding D-bus transfer burst, it gets higher priority than the STACK and CPU as long as it continues to request more D-bus time.

Line-fill data is stored as destination image data in the destination region 410d of the VRAM subunit 410 of system memory unit 405. Although VRAM subunit 410 is preferred as the destination for spryte rendition, it is to be understood that other parts of system memory unit 405 (e.g. DRAM, RAMBUS™, etc.) can also be the targets of spryte rendition. Similarly, although VRAM subunit 410 is a preferred supplier of source sprytes 411 and 412, it is to be understood that other parts of system memory unit 405 (e.g. DRAM, RAMBUS™, etc.) can also serve as suppliers of source sprytes 411 and 412. Also, there is no system constraints which prevents a spryte-rendition destination area of system memory unit 405 from also serving as the supplier of one or both of source

sprytes 411 and 412. Thus image rendition can operate on a recursive basis.

Coordinate-value truncaters 426a and 425a are preferably (but not necessarily) provided at the upstream end of each of the Slow-DPDMu 426 and Fast-DPDMu 425 for truncating away fractional portions of results produced by corner calculating unit 422. The Slow-DPDMu 426 and Fast-DPDMu 425 of such an embodiment then work exclusively with integer values. This aspect of the invention will be described further in conjunction with Fig.s 5C and 6D. As a general introduction, it is noted that the human eye tends to overlook small, localized irregularities in a bit mapped image composed of hundreds or thousands of pixels. Thus, if the border of a rendered polygon is missing one pixel here or has an extra pixel there, it is usually of no matter. On the other hand, the eye does catch global irregularities such as when two lines that are expected to meet or to be orhtogonal to one another or to be parallel to one another, fail to do so by a substantial amount. High precision is maintained in the calculations of the corner calculating unit 422 to avoid the kinds of graphic errors that the eye catches more often. But, the same precision is preferably sacrificed in the operations of the Slow-DPDMu 426 and Fast-DPDMu 425 for the sake of minimizing circuit size and speeding performance. This is really another manifestation of the prime directive. The human eye is insensitive to errors at the single pixel level (when viewing an image of many pixels), so any work invested in eliminating such error is unnecessary, and thus avoided.

The prime directive manifests itself in yet another feature of the operations of image rendering system 400. Although the displayable image can be quite large in terms of number of pixels on its horizontal and vertical

sides, image rendering is preferably restricted to a small subportion of the displayable image so that the computational resources of the image rendering system 400 work with relatively small numbers (fewer bits). Circuit size and/or task-completion time can then be significantly reduced. It will be seen in the embodiment of Fig. 6, that a "Base" value is added to the destination-paint results produced by the Slow-DPDMu (e.g. Regis) 626 and/or the Fast-DPDMu (e.g. Munkee) 625 thereby shifting the results to a designated render window within the overall renderable universe. Units 626 and 625 work with relatively small numbers (ones that require less bits to represent), and only at the last necessary moment are those values converted from small relative values to potentially-large absolute values.

The VRAM subunit 410 includes a sequence output bus referred to as the S-bus 406. The S-bus 406 transfers video scan-line data to an image display unit 460 on an as-needed basis. Sprytes which are mapped and painted into the destination region 410d of system memory unit 405 then appear within the light image displayed by the image display unit 460 or define in some way portions of the displayed light image.

With regard to the light image displayed on the image display unit 460, it is to be understood that the pixels of this light image are ultimately transmitted to the eyes of a human being (not shown) and appreciated by that human being for their opto-physiological and/or psycho-visual, graphic content. As such, destination data signals that are stored in (written to) memory region 410d ultimately manifest themselves as significant parts of a physically real entity; the displayed image.

In one particular embodiment of the invention, the psycho-visual or "apparent" resolution of the image represented by the destination image data stored in

region 410d of system memory unit 405 is first improved before being converted into a light image. This is done by means of inter-line interpolation (interpolation between horizontal display lines) and/or inter-column interpolation (interpolation between vertical display lines) and/or inter-field interpolation (interpolation between sequentially displayed fields of a display frame) and/or inter-frame interpolation (interpolation between sequentially displayed frames of an animated display).

While not shown, it is to be further understood that the light image generated by the image display unit 460 typically includes a real-time animated scene. The viewer controls at least part of the action in the displayed scene by operating real-time controls (e.g., a joystick, a mouse, push-buttons). The controls are operatively coupled to the image rendering system 400 such that they can be downloaded over the system data bus 403 and used to modulate the contents of SCoB register 415 over time in response to real-time commands supplied by the viewer.

Consider now the introductory discussion in which an airplane was to be pictured on a display panel such that it appears to be flying by, towards or away from a viewer in real time. One of source sprytes 411 and 412 would be created ahead of time as a size-normalized image of parts or all of the airplane. Up/down and left/right motion across the screen is controlled by varying the XPOS and YPOS signals within the point-map control signals 415.2. On screen size is controlled by varying the LDY and DX signals within the point-map control signals 415.2. On screen rotation and skew are controlled by varying the LDX, DY, DDX and DDY signals within the point-map control signals 415.2.

Consider further that part of the introductory discussion in which parts of the airplane were to be

pictured as being suddenly brightened by light flashing off of them (or conversely, suddenly made darker to simulate a shadow passing across them), and/or suddenly made transparent because a hole is created through them (or conversely, suddenly made non-transparent to simulate mud or paint thrown on a window). In such cases, the color-map control signals 415.1 of the SCoB register 415 are changed to generated various effects including generation of an active or inactive transparency bit (T-bit) and changes in the apparent brightness of the final color code signal 418.

Since variations in the color-map control signals 415.1 and point-map control signals 415.2 are used to produce what appear to be real-time changes in the image generated on the image display unit 460, it follows that the color-mapping portion 401 and destination-point mapping portion 402 of the image rendering system 400 need to perform their respective color-mapping and point-to-point mapping functions in relatively short time.

One objective of the invention is to perform the point-to-point mapping function quickly while using circuitry which consumes relatively little space within the image rendering system 400. This is done by sharing circuit resources which perform computational functions, by sharing results produced by different circuits and by identifying and avoiding all computational activities which, if done, will ultimately be found to not have been needed for painting destination pixels. Preferably, all the circuitry shown in Fig. 4 is incorporated into a single integrated circuit chip (referred to as a System and/or Memory Address Manipulator Chip, S/MAMC) except for system memory unit 405, image display unit 460 and the system CPU unit.

Looking more closely at the corner calculating unit 422 of Fig. 4, note that the high precision a0-a3 results

are fed back into the corner calculating unit 422 for recursive use.

Note further that corner calculating unit 422 can contain one or more corner-calculating engines. When two or more corner engines are provided, they share row-initialization results. (More will be said about this in conjunction with the dual-corner engine embodiment of Fig. 6 and an explanation provided for Fig. 8.) Corner calculating unit 422 includes a result-saving register stack 422a which stores running subtotals such as DX++, DY++ and a0++.

At the start of spryte rendition, stored value a0++ is set equal to the XPOS and YPOS coordinates of mapped corner point a0. It is thereafter increased by signed values LDX and LDY to successively point to image points a3, q3, w3, etc. (see Fig.s 3A and 3B). In similar fashion, stored values DX++ and DY++ are set equal to the DX and DY values at the start of spryte rendition. Thereafter they are successively increased by respective values DDX and DDY as spryte rendition steps from a0++ = a0 to a0++ = a3, q3, w3, etc.

The Slow-DPDMu (e.g. Regis) 426 and Fast-DPDMu (e.g. Munkee) 425 cooperate with one another to produce paint-request signals 428 in minimal time. For certain values of corner coordinate signals 424, the Fast-DPDMu (e.g. Munkee) 425 can accurately determine in one clock tick that, if the decision were left to the Slow-DPDMu 426, the Slow-DPDMu 426 will ultimately decide that no paint-request signals 428 are to be sent to the destination line-filling unit 430, or that a request for painting only one destination pixel will be sent, or that a request for painting some other small number (e.g., 2 or 3) destination pixels will be sent. In such cases, the Fast-DPDMu (e.g. Munkee) steps in and sends the paint-request 428 on its own. The more-time consuming

operations of the Slow-DPDMu (e.g. Regis) are circumvented in this case. If the Fast-DPDMu (e.g. Munkee) is not sure what the Slow-DPDMu 426 would have done, it leaves it to the Slow-DPDMu (e.g. Regis) to perform the paint/don't-paint decision-making task.

The basic operating principles behind the Slow-DPDMu (e.g. Regis) 426 and the Fast-DPDMu (e.g. Munkee) 425 are now explained in more detail below.

(§5.1) GENERAL EXPLANATION OF REGION FILL METHOD

Referring to Fig. 5A, consider a projected polygon 480 having corner points with X_D, Y_D coordinates of a0=(3.125, 1.5), a1=(6.875, 1.5), a2=(2.0, 8.0) and a3=(8.0, 8.0). The line 482 between corner points a1-a2 satisfies the equation: $3Y_D = -4X_D + 32$. The line 483 between corner points a0-a3 satisfies the equation: $3Y_D = 4X_D - 8$.

It will be assumed here that the paint/don't-paint decision-making algorithm operates by first determining what, if any, destination pixels have their top-left corner inside projected polygon 480. Those that do not are not painted with the corresponding color/shade. Those that do, may be painted, provided they meet certain other criteria, e.g., they are not part of the bottom most row of the polygon or cover the right most points of polygon. (It is to be understood that this not the only way make the paint/don't paint decision. Another algorithm. REGIS_FILLBIN, will be described later.)

To determine what destination pixels have their top-left corners within the bounds of projected polygon 480, a boundary walk is conducted along the left and right sides from the top corner points of projected polygon 480 (in the illustrated case there are two topmost corner points, a0 and a1) to the bottom corner points of projected polygon 480 (in the illustrated case there are

two bottom-most corner points, a2 and a3). The Slow-DPDMu 426 is assigned the task of carrying out the boundary walks.

Boundary-walking proceeds by moving a set of hypothetical edge-pointers, EPL and EPR, from one point to a next along respective left and right edges of the polygon 480, with each next edge point being one that has a next higher integer value for its Y_D coordinate. Boundary-walking step 490, for example, moves left edge pointer EPL along line 483 from starting corner point a0=(3.125, 1.5) to a point having coordinates X_D=3.5, and Y_D= INTEGER(Y_D of a0) + 1 = 2.0. At the same time, boundary-walking step 491 moves right edge pointer EPR along line 482 from corner point a1=(6.875, 1.5) to a point having coordinates X_D=6.5, and Y_D= INTEGER(Y_D of a0) + 1 = 2.0. In both cases, the Y_D coordinate of the next point in the boundary walk is calculated first. The INTEGER function simply truncates the fractional part of the preceding Y_D coordinate and returns the remaining integer portion. The X_D coordinate is then calculated using a Bresenham or other line approximating algorithm for points along respective lines 483 (a0 to a3) and 482 (a1 to a2).

Next, the two X_D results are truncated. The lower X_D value (left border point) is incremented by one. For the illustrated boundary-walking steps 490 and 491, the results are: INTEGER(3.5)+1=(3.0)+1=4.0 and INTEGER(6.5)=6.0.

Next, paint-request signals 428 are generated to paint destination pixels having top left coordinates of Y_D=2.0 and the calculated left and right X_D values, 4.0 and 6.0, plus any in between integer values, e.g., 5.0.

The three top-left corners whose destination pixels are to be painted are shown as bold dots in Fig. 5A, located inside the boundaries of projected polygon 480.

The destination pixels are not drawn but understood to have top-left corners on Y_D= 2.0 points X_D= 4.0, 5.0 and 6.0. The same destination pixels have bottom-right corners on Y_D= 3.0 points X_D= 5.0, 6.0 and 7.0. In one embodiment of the invention, the destination line-filling unit 430 ignores the last (right most) in any series of corner point it receives. In such a case, the destination pixels having top-left corner points (4.0, 2.0) and (5.0, 2.0) are painted with a mapped color belonging to projected polygon 480 but the destination pixel having top-left corner point (6.0, 2.0) is not painted.

Boundary walking steps 492 and 493 follow steps 490, 491 and similarly produce a paint request for the destination pixel having a top left corner at X_D=5.0 and Y_D=3.0.

Boundary walking steps 494 and 495 will discover a single point 496 (X_D=5.0, Y_D=4.0) lying directly on boundary lines 482 and 483. The simple top-left corner algorithm will conclude that this point 496 is not inside the polygon and it will not send a paint request signal for this point to the destination line-filling unit 430.

After point 496, the left-edge pointer EPL continues to track the left boundary of projected polygon 480 (from point 496 to corner point a2) and the right-edge pointer EPR continues to track the right boundary of projected polygon 480 (from point 496 to corner point a3). When left-edge pointer EPL reaches its bottom corner point, a2, and right-edge pointer EPR reaches its bottom corner point, a3, the boundary walking process stops without sending a line fill request 428 to destination line-filling unit 430. The destination pixels which have their tops abutted to the bottom of projected polygon 480, are therfore not painted.

(§5.2) GENERAL EXPLANATION OF Fast-DPDMu (e.g. Munkee) SHORT-CUT METHOD

Referring to Fig. 5B it is seen that in some cases a projected polygon 497 will enclose one and only one point with integer values for X_D and Y_D. It is seen that in some other cases, 498 or 499, a projected polygon will enclose no point with integer values for X_D and Y_D. In such cases, the result of a complex (and time-consuming) boundary-walk will simply result in a decision to not paint any destination pixel or to paint only one destination pixel.

The Fast-DPDMu (e.g. Munkee) 425 obtains the corner coordinates a0-a3 (corner coordinate signals 424), estimates the distances separating the corner points, and decides from these differences whether it can accurately determine on its own if only one or no destination grid points with integer values for X_D and Y_D are contained between corner points a0, a1, a2, a3.

If the Fast-DPDMu 425 decides that it can perform the determination accurately, and it concludes that no more than one destination pixel (P) will be painted, the Fast-DPDMu 425 sends an abort signal 427a to the Slow-DPDMu 426. The Slow-DPDMu 426 aborts any calculations it may have begun for the current projected polygon and indicates that it is ready to begin a new set of calculations. The Fast-DPDMu 425 at the same time, either does nothing (for the case where the Fast-DPDMu 425 decides no destination pixel is to be painted) or issues paint-request signals 428 to the destination line-filling unit 430 for initiating the painting of a single destination pixel (for the case where Fast-DPDMu decides only one destination pixel is to be painted.

If the Fast-DPDMu border estimating unit 425 decides that it cannot accurately determine the ultimate outcome of the calculations that are being started within the

Slow-DPDMu 426 for a current polygon, the Fast-DPDMu 425 does nothing and thereby allows Slow-DPDMu 426 to complete its more comprehensive calculations. The Slow-DPDMu 426 then issues or does not issue appropriate paint-request line-fill request signals 428 to the destination line-filling unit 430 in accordance with the results of its own calculations.

In an alternate embodiment, the Slow-DPDMu (e.g. Regis) does not begin any substantive calculations until it receives a go-ahead signal (which replaces abort signal 427a) from the Fast-DPDMu (e.g. Munkee). The Fast-DPDMu (e.g. Munkee) therefore looks over the data initially, and in cases where it cannot produce the paint-only-one pixel command on its own or the don't-paint decision on its own, it will pass the decision-making task over to Slow-DPDMu (e.g. Regis).

Fig. 5C illustrates another method for making paint/don't-paint decisions. This method will be referenced here as the truncated-corners method. It explains in relatively basic and simple terms how a below described Regis unit 626a and Regis-Fillbin algorithm works.

Instead of using the precise coordinates for corner points a0-a3, as generated by the corner calculating unit 422, truncater portion 426a of the Slow-DPDMu 426 truncates the fractional portions of those signals to produce corresponding signals representing "truncated" corner points Ta0, Ta1, Ta2, and Ta3 in the illustrated case. Truncated point Ta0 is produced by finding the point with the next lowest or equal integer values of X_D and Y_D relative to the coordinates of point a0. Truncated point Ta1 is similarly produced, by performing the truncating operation on the coordinates of point a1. Truncated points Ta2 and Ta3 are similarly produced by

performing the truncating operation on the coordinates of respective corner points a2 and a3.

Truncated point Ta0 is interperted as an initial request to paint the destination pixel P1 for which it serves as the top left corner. Hence, destination pixel P1 is shown shaded in Fig. 5C. Truncated point Ta3 is interperted as an initial request to paint the destination pixel P2 for which it serves as the top left corner. Hence, destination pixel P2 is also shown shaded. And similarly, truncated points Ta1 and Ta2 are interperted as an initial request to paint respective destination pixels P3 and P% for which they serve as the top left corners. Hence, destination pixels P3 and P5 are also shown shaded.

Truncated points Ta0, Ta1, Ta3, and Ta2 in the recited order represent the respective top, right-middle, left-middle and bottom corners of a triangle. The triangle is the resultant shape that polygon "a" takes on after the values of its corner coordinates are truncated.

A boundary walk is performed with the left and right edge-pointers both starting at top point Ta0. They both send the coordinates (X_D=m, Y_D=n-1) of their starting point Ta0 to the line-filler as a paint request. Left-edge pointer EPL (not shown, see Fig. 5A) moves down along the left side of triangle Ta0-Ta1-Ta2-Ta3 using a Breshenham-like algorithm to middle-left point Ta3. Right-edge pointer EPR (not shown) moves down along the right side of triangle Ta0-Ta1-Ta2-Ta3 by also using a Breshenham-like algorithm to get to middle-right point Ta1. When the edge pointer, EPL and EPR, are both caught up with each other on a same integer Y_D coordinate (Y_D=n.0), they send their respective left and right X_D values (X_D=m-1 and X_D=m) to the line-filler.

Both edg-pointers then continue to trace their respective left and right borders of truncation-formed polygon Ta0-Ta1-Ta2-Ta3. EPL uses a Breshenham-like algorithm to get from middle-left point Ta3 to bottom point Ta2. EPR uses a similar Breshenham-like algorithm to get from middle-right point Ta1 to bottom point Ta2.

The process stops when they reach the bottom of truncation-formed polygon Ta0-Ta1-Ta2-Ta3. The coordinates of point Ta2 are not sent to the line filler 430 because it is the bottom most point of the truncation-formed polygon Ta0-Ta1-Ta2-Ta3.

The line-filler in the truncating embodiment of the invention decides on its own to not paint the right most destination pixel on any request it receives from the truncated version of the Slow-DPDMu 426 (e.g. Regis 626) to paint a horizontal having a constant Y_D coordinate value, and left and right X_D coordinates. Thus, when the first set of Y_D, X_D-left, X_D-right are received by the line-filler 430 for top point Ta0, the line filler notices that destination pixel P1 is the right most pixel of its destination grid row and the linefiller decides not to over-paint destination pixel P1 with the color of projected polygon "a". When the next set of Y_D, X_D-left, X_D-right are received by the line-filler 430 for middle points Ta3 and Ta1, the line filler over paints destination pixel P2 with the color of projected polygon "a" but the line-fiiler detects that destination pixel P3 is the right most pixel of its destination grid row and the line-filler hterefore decides not to over-paint destination pixel P3 with the color of projected polygon "a". The line-filler does not receive the coordinates of point Ta2 because the edge walkers stop their walks when they get to the bottom of the truncation-formed polygon Ta0-Ta1-Ta2-Ta3 but before they send the coordinates to the paint filler.

The end result of the above steps is that only destination pixel P2 gets painted with the color of projected polygon "a".

Note that projected polygon "a" of Fig. 5C is different from mapped polygon "a" of earlier described Fig. 1B. In Fig. 1B corner a2 resides over destination pixel P3 as does corner a1. If Fig. 1B had been used as the basis for the above example, no destination pixel would have been painted in the end. This is because the truncations of corner points a1 and a2 in Fig. 1B woulb both fold to the top-left corner of P3. Regis considers such multiple folding simply as a single point, call it Ta1a2 (not shown). Ta3 and Ta1a2 would then form the bottom-most left and right points of the truncation-formed polygon Ta0-Ta1a2-Ta3 (not shown) while Ta0 would still be the top point. Regis would not send Ta3 and Ta1a2 to the line-filler because they are the bottom-most points. The line-filler would ignore Ta0 because it is the rightmost on its horizontal line, and hence no destination pixel would be painted for the "a" polygon of Fig. 1B.

(§6) OVERVIEW OF PLURAL CORNER-ENGINES EMBODIMENT

Figure 6 (composed of subfigures 6A-6F) is a block diagram of a more complex, spryte rendering system 600 in accordance with the invention. Except where otherwise stated, all circuit components are CMOS (complementary metal-oxide-semiconductor technology) and all are implemented on a single integrated circuit chip (using 0.9 micron or smaller line widths).

The illustrated spryte rendering system 600 includes plural row initializers (701a, 701b, shown in Fig. 6D), plural corner-calculating engines (702a, 702b), plural Regis and Munkee units (625a, 226a, 625b, 226b), plural math platforms (700a, 700b, which are used on a time-shared basis by the row-initializers, corner engines,

Munkee units and Regis units), plural color code unpackers (613a, 613b, shown in Fig. 6A) and plural system memory banks (left bank 605a and right bank 605b, shown in Fig. 6C).

In many instances, like reference numbers in the "600" series are utilized in Fig. 6 for elements which have like-numbered counterparts numbered in the 400 series in Fig. 4.

(§6.1) DETAILED DESCRIPTION OF COLOR-MAPPING PATHS

System 600 is divided into a color-mapping section (or "path") 601 (see Fig.s 6A-6C) and a destination-point mapping section (or "path") 602 (see Fig.s 6D-6F). A 32-bit wide system data bus (D-bus) 603 couples the data input/output ports of two 16-bit wide system memory banks 605a and 605b (left and right memory banks) to data input portions of the color-mapping section 601 and destination-point mapping section 602.

A color-codes FIFO 609 (First-in First-out buffer unit shown in Fig. 6A) provided within the color-mapping section 601 connects to the 32-bit wide D-bus 603 for receiving compressed or noncompressed color codes representing source pixels in first and second source image rows (current and previous spryte rows). The compressed or noncompressed color codes are delivered as two, side-by-side, 16-bit wide datawords, one representing pixels of a "previous" spryte row and the other representing pixels of a "current" spryte row. As they flow out of FIFO 609, the two datawords move along generally separate and parallel, color-mapping subpaths. One subpath performs color-mapping for the pixels of the "previous" source spryte row and the other subpath performs color-mapping for the pixels of the "current" source spryte row. The "previous" and "current" source spryte datawords are originally prestored in respective

left and right banks (605a, 605b) of the system memory unit (605).

(Note: the "previous" and "current" designations refer to a graphics image interpolation process which occurs downstream, after destination pixels are painted. The interpolation process is not germane to the present discussion, but for purpose of completeness, "previous" refers to an upper row in the interpolation process and "current" refers to an underlying, next row of the interpolation process.)

A run-length compression algorithm is preferably used to minimize the amount of time required for transferring the "previous" and "current" color code datawords over the system D-bus 603 into FIFO 609. There are 2 basic formats of spryte image data which can be received by the FIFO 609: Totally Literal Format (TLF) and Non-totally Literal Format NLF). There are sub-groups within each basic format. The following discussion explains the formats in more detail.

In non-totally literal sprytes, control data is intermingled with color-defining data. NLF image data consists of groups of words that represent source scan lines of variable lengths and variable bits-per pixel assignments for representing color data. In totally literal sprytes (TLF), the image data is pure image data (there are no intermingled control codes).

Non-totally literal (NLF) sprytes can be compacted to save both memory space and rendering time. Each source scan line of data has its horizontal word size specified as part of the spryte data that is passed to FIFO 609.

Totally literal (TLF) sprytes have rectangular formats similar to that of conventional sprites. (Each row is of a same number of pixels.) The rectangle dimensions are specified in one or more preamble words

that are stored in system memory 605 and associated with the data of each spryte. There is no control data passed to FIFO 609 for this kind of image data.

First preamble word:

A first preamble word is provided in system memory 605 for ALL sprytes. This is the data-structure definition preamble. It contains the following 32 data specific control bits for the source data:

B31->B28	=	Reserved for future use.	
B27->B21	=	Reserved, set to 0.	
B20	=	PACKED	This is identical to a PACKED bit found in the SCoB
B19->B16	=	Reserved, set to 0.	
B15->B6	=	VCNT	Vertical number of source data lines in this image data -1. (10 bits)
B5	=	Reserved, set to 0.	
B4	=	LINEAR	0=use PIP for output of IPN, 1=use PIN for output of IPN
B3	=	REP8	1=replicate the bits in the linear 8 SPRYTE, 0=fill with 0
B2->B0	=	BPP	bits per pixel,=pixel type

First Preamble Notes:

VCNT is loaded into a hardware counter in a so-called SPRYTE requestor of a system DMA engine. The counter is decremented at the end of the fetching of each source scan line of data. When the count is at -1, there are no more source lines of data in the object (in the Spryte). Note that SPRYTE processing does not end here, this is merely one of the events that is required to end a SPRYTE. (A linked-list defines what "next" Spryte should be rendered. VCNT = line count -1. An initial value of -1 for VCNT is the maximum value. It will cause a REAL BIG SPRYTE to be fetched. There is no 'zero line count' value allowed for a spryte.

The LINEAR bit only applies when the BPP (bits per pixel setting, which is used by a below described IPS unit 614) is 8 bits or 16 bits. In those cases, there

are enough PIN bits (output from multiplexer 524) to provide a 15 bit portion of the 29-bit wide IPN signal output from IPS 614 without using the PIP (a look-up RAM inside the IPS 614) to fill the rest. Since the PIN bits are spread linearly across the IPN, and it will result in a linear translation from PIN to IPN, the mode is called 'LINEAR'. The only 2 valid uses are for LINEAR 8 and LINEAR 16 (as opposed to 'normal' 8 or 16).

The REP8 bit only has an effect in the 8 bit source data size.

The three BPP bits B2:B0 decode as follows:

BPP	Data Size	PIP DMA Size	IPN Trans Bits	D-bit	R-Mul	G-Mul	B-Mul
0 =	reserved	4 PIP words	Reserved,(?)	(?)	0	0	0
1 =	1 bit	4 PIP words	PIN[0]	PIP[15]	0	0	0
2 =	2 bit	4 PIP words	PIN[1..0]	PIP[15]	0	0	0
3 =	4 bit	8 PIP words	PIN[3..0]	PIP[15]	0	0	0
4 =	6 bit	16 PIP words	PIN[5..0]	PIN[5]	0	0	0
5 =	8 bit	16 PIP words	PIN[7..0]	PIP[15]	PIN[7..5]	PIN[7..5]	PIN[7..5]
6 =	16 bit	16 PIP words	PIN[14..0]	PIN[15]	PIN[13..11]	PIN[10..8]	PIN[7..5]
7 =	reserved	16 PIP words	Reserved,(?)	(?)	0	0	0

Second preamble word:

If the PACKED bit (in the SCoB) is '0', then the source data is totally literal (TLF). For totally literal sprytes, there is a second preamble word. It contains the horizontal pixel count for each line of the source data and the word offset from one line of source data to the next. It also contains the other special bits needed for totally literal sprytes. Note that these bits are only valid while the totally literal spryte is being rendered. These bits are not used ...they are GATED AWAY... when the current spryte is not totally literal.

B31->B24 =	WOFFSET(8)	Word offset from one line of data to the next (-2) (8 bits). Bits 23->16 are set to 0.
B25->B16 =	WOFFSET(10)	Word offset from one line of data to the next (-2) (10 bits). Bits 31->26 are set to 0.
B15 =	Reserved, set to 0.	
B14 =	NOSWAP	1=disable the SWAPHV bit from the general SPRYTE control word.
B13->B12 =	TLLSB	IPN PPMP blue LSB source. 0=0, 1=IPN[0], 2=IPN[4], 3=IPN[5].
B11 =	LRFORM	Left/right format.
B10->B0 =	TLHPCNT	Horizontal pixel count (-1) (11 bits).

NOTES FOR PREAMBLE WORD TWO

The TLLSB bits perform the same function that the IPNLSB bits perform in normal sprytes.

If LRFORM=1, the source data has the frame buffer format of the screen as a source format. Vertically adjacent pixels in the rectangular display space are horizontally adjacent in the 2 halves of a memory word. Only useful for 16 BPP totally literal. The unpacker will disable the 'B' FIFO data requests and alternately place pixels from the source into both FIFOs. Left 16 bits go to 'A' FIFO, right 16 bits go to 'B' FIFO. The data requests for 'A' FIFO will be made in a request 'pair' to insure the reduction of page breaks and '6 tick latencies'. The hardware will lock the corner engines (regardless of the LCE bit).

TLHPCNT is the number of pixels in the horizontal dimension (-1). This is the number of pixels that will be attempted to be rendered for each horizontal line of the spryte. This value is used by the data unpacker. A '0' in the value will attempt 1 pixel. A '-1' in the value will attempt many pixels. There is no 'zero pixel count' value.

WOFFSET is the offset in words of memory from the start of one line of data to the start of the next line (-2). If the BPP for this spryte is 8 or 16, use WOFFSET(10), else use WOFFSET(8). This number is a zero for the minimum sized spryte (2 words).

By arranging WOFFSET and TLHPCNT correctly, you can extract a rectangular area of data out of a larger sized rectangular area of data.

The DMA engine will also use WOFFSET as the length value in the normal data fetch process. If WOFFSET and TLHPCNT are set badly, WOFFSET may expire first and the DMA engine will behave abnormally.

SPRYTE Packed Data Formats

Offset

The first one or two bytes of a packed spryte block define the words-offset from the start of this block of source data to the start of the next block of data (-2). In sprytes with BPP of 6 or less, only 1 byte (bits 31->16) of offset information is used. However, the actual offset value has a maximum size of 10 bits. The rest of the bits in the 2 bytes are set to 0. 10 bits of word offset is equivalent to 2048 pixels at 16BPP. 8 bits of word offset at 6 BPP is equivalent to 1365 pixels. The requirement for four lines of 320 pixels each is 1280 pixels.

This offset is used by the DMA controller to both calculate the start of the next block of data (by adding it to the start of the current block), and to set the maximum length (by subtracting 1 and placing it in the DMA length register) of the current DMA transfer.

The word offset value (1 or 2 bytes) is not used by the data unpacker. It will arrive at the data unpacker at the start of each received block of a packed spryte and must be discarded.

Control byte and PIN data:

The next data in a packed spryte block, after the word offset byte or bytes, is comprised of 1 control byte and 0 or more bits of PIN data. The number of bits used for each PIN is specified by BPP.

The control byte consists of a 2 bit code and a 6 bit count arranged as follows. "Count" represents the run length count that is to be used by the unpacker (the number of times that the same code is to be repeated).

00 'xxxxxx' =	End of block, xxxxxx need not be present
01 'count' =	Generate literal PINs for 'count+1'
10 'count' =	Define as 'transparent' for 'count+1'
11 'count' =	Generate packed 'PIN' for 'count+1'

The 'transparent' definition will actually output an active 'transparent' bit within the 29-bit wide IPN signal output from the IPS 614. This will cause the remainder of the pixel processing pipe to ignore this pixel. For safety purposes, we set the data value output by the unpacker (613a or 613b) at this time to be zero for possible use by the IPS for the D-Mode selector. The IPS outputs a corresponding transparent bit (T-bit) as part of its output.

Unpackers 613a and 613b pop compressed spryte data signals 611a and 611b serially and respectively from the output of the color-codes FIFO 609 and begin to decompress them in accordance with the above-described control signals. The control signals are generated by a control section (not shown) of the system 600. The 16-bit wide datawords held in color-codes FIFO 609 can be coded as 1 bit per pixel, or 2 bits per pixel (BsPP), or 4 BsPP, or 6 BsPP, or 8 BsPP or 16 BsPP.

Unpacked color code signals (Pen Index Numbers or PIN's for short) flow from unpacker 613a through a pseudo-FIFO comprised of an R(OUT) register 514, an R(HOLD) register 516, multiplexer 520, and an R(IPS) register 522. IPS-input multiplexer 524 represents an example of a data consuming circuit which accepts or refuses to accept data from the pseudo-FIFO, 514, 516, 520, 522.

The pseudo-FIFO is named as such because it is not a full-fledged FIFO buffer but performs essentially the same function while using less circuit space. A pseudo-FIFO (or p-FIFO for short) functions as a variable delay means. It is used to resynchronize sequential dataword streams that are supplied out of phase relative to one another by as much as three ticks of the system clock.

A p-FIFO control circuit 510 is provided to sequence through a set of states wherein second and third registers, 516 and 522, of the p-FIFO are designated as "empty" or "full." "Full" means that the register holds data that is valid and has not yet been "popped out" of the p-FIFO.

Initially, both registers, 516 and 522, are designated as empty. In each tick of the system clock, if both of p-FIFO registers, 516 and 522, are empty, they are both simultaneously loaded with the same dataword (e.g., dataword #1) held in source register (R(OUT)) 514 and the status of third register R(IPS) 522 is set to full. In all cases, if the third register R(IPS) 522 is set to full, the status of the second register R(HOLD) is also set to full.

In the next clock cycle, if the data consuming circuit 524 accepts the output (e.g., dataword #1) of register R(IPS) 522, both of p-FIFO registers, 516 and 522, are designated as empty. At the same time, register R(OUT) 514 loads with the next dataword (e.g., dataword #2) of the sequence.

On the other hand, if the data consuming circuit 524 refuses the output (dataword #1) of register R(IPS) 522, that output is simply discarded, and the two duplicate copies (of dataword #1) in registers 514 and 516 shift down respectively for continued storage in respective registers 516 and 522. Registers 516 and 522 continue to be designated as full.

In the next clock tick, if the data consuming circuit 524 still refuses the output (dataword #1) of register R(IPS) 522, that output is simply discarded, and the duplicate copy (of dataword #1) in register 516 shifts down for continued storage in register 522. The next dataword (#2) of register 514 shifts into register 516 and register 514 thereafter loads with yet another

data word (dataword #3) of the sequence. Registers 516 and 522 continue to be designated as full.

A similar, second pseudo-FIFO (p-FIFO), formed of elements 515, 517, 521, 523, carries the decompressed PIN's output by unpacker 613b. Corner-selecting multiplexer 524 accepts the output of one of the p-FIFO's and supplies the accepted output of one or the other of R(IPS) register 522 and R(IPS) register 523 to the Indexed PIN Substituting unit (IPS) 614.

IPS unit 614 includes a PIN-Indexed Palette substitution table (a PIP RAM) 526 which may be programmably inserted into the signal path to convert PIN signals (which are initially coded as 1-to-16 bits per pixel) into 29-bit wide IPN signals (Indexed Pen Numbers). The PIP-RAM downloads the conversion data held within it from system memory 605 over the bus 603. The IPS includes other means (not shown) for converting a PIN signal into a longer IPN signal in addition to or as an alternative to the PIP look-up SRAM 526.

One bit within each 29-bit wide IPN signal the so-called T-bit (transparency bit) which has already been mentioned. It is shown being stripped away and transferred to the point-map section (into Fig. 6D) by line 527. When active, the T-bit 527 indicates that a downstream-produced PEN signal 618 will not be used to paint a corresponding destination pixel.

Note that the corner-selecting multiplexer 524 functions to funnel the parallel result signals of unpackers 613a and 613b serially into a single IPS unit 614. This saves area on system integrated circuits. The IPS is relatively fast, but consumes circuit area because it includes the PIP-RAM 526. The unpackers 613a and 613b and their respective p-FIFO's require more time to process their signals but require less circuit area. Thus a trade off is made in the design. Slow functioning

circuits (e.g., the unpackers) are provided redundantly to speed performance while fast functioning circuits (e.g., the IPS) are provided as single, serial processing units to reduce consumption of circuit area.

IPS unit 614 sends its output data (IPN's) into an IPS-OUT FIFO 528 (shown in Fig. 6B). The IPS-OUT FIFO 528 is only 28-bits wide because the T-bit does not continue to move downstream through the color-map section 601. At the output of the IPS-OUT FIFO 528, another bit, a so-called R-mode bit 531 is stripped away and sent into the point-map section 602 (to Fig. 6E), thus leaving 27 bits of the originally 29-bit wide IPN signal still flowing downstream through the color-map section 601.

At roughly the same time that 28-bit IPN code signals flow into IPS-OUT FIFO 528, corresponding second spryte code signals 612a and 612b ("previous" and "current") move from a "current frame buffer" (CFB) of system memory 605, through multiplexer 529 into a CFBD FIFO 530. (CFBD stands for "current frame buffer data.") Multiplexer 529 selects the order and timing of insertion of the CFBD data signals into FIFO 530. Signals 612c and 612d represent respective signals 612a and 612b delayed by one clock tick.

Color code signals output from IPS-OUT FIFO 528 (27-bits wide) and CFBD FIFO 530 (16-bits wide) load together synchronously into a 43-bit wide pipeline register 532. They synchronize according to their respective insertion orders into FIFOS 528 and 530. The output of pipeline register 532 feeds a PPMP unit 616. The output of pipeline register 532 also wraps back to feed a "Dolo IPN Recycle" signal back to IPS unit 614 (Fig. 6A). When the R-mode is active, application of the IPN signal to the PPMP 616 (Fig. 6B) is delayed so that a corresponding current-frame buffer PEN code can be fetched out of memory and loaded into CFBD FIFO 530 at roughly the same

time that the wrapped-back IPN signal re-enters IPS-OUT FIFO 528. A corresponding address wrap-back path 565 (Fig. 6E) is provided within the circuitry of the point-mapping section 602.

PPMP unit 616 merges the signals received from the IPS-OUT FIFO 528 and the CFBD FIFO 530 to form a "Pump-pin-ulated" 16-bit wide signal which is simply referred to as the PEN signal 618.

"Previous" and "current" PEN signals 618 are alternatingly generated by the PPMP unit 616 and alternatingly stored in registers 576 and 577. Left-side register 576 feeds one input of multiplexer 580. The PEN signal line (618) feeds a second input of multiplexer 580. Multiplexer 580 feeds the input of a 16-bit wide, tri-state left-bus driver 582. A similar arrangement is found for right-side register 577, multiplexer 581 and tri-state right-bus driver 583. The outputs of 16-bit wide drivers 582 and 583 merge (represented by circled "M" in Fig. 6C) into the 32-bit wide D-bus 603.

The outputs of 16-bit wide drivers 582 and 583 also feed the inputs of respective 16-bit wide data-output registers 594 and 595 (see Fig. 6C). Register 594 drives the data-input port of left memory bank 605a (bits 31 through 16). Register 595 drives the data-input port of right memory bank 605b (bits 15 through 00). Left and right memory banks 605a and 605b or located off-chip as indicated by the off-chip designating region.

The data-output ports of respective left and right memory banks 605a and 605b supply corresponding 16-bit wide, on-chip input latches 596 and 597. Input latch 596 drives a left-side tri-state bus driver 598. Input latch 597 drives a right-side tri-state bus driver 599. The outputs of 16-bit wide drivers 598 and 599 merge

(represented by circled "M") into the 32-bit wide, chip-internal D-bus 603.

Note that register 576 (Fig. 6B), multiplexer 580, register 594 (Fig. 6C) and an internal register (not shown) of memory bank 605a are arranged to define a left-side, memory feeding, pseudo-FIFO. Register 577, multiplexer 581, register 595 and an internal register (not shown) of memory bank 605b are similarly arranged to define a right-side, memory feeding, pseudo-FIFO.

(§6.2) DETAILED DESCRIPTION OF DESTINATION-POINT MAPPING PATHS

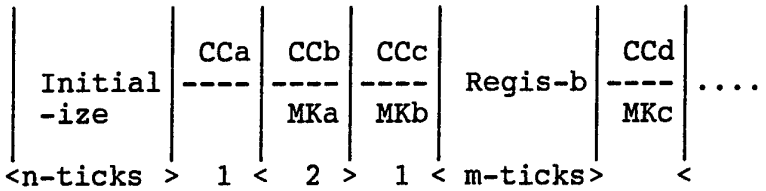
While one or more PEN signals 618 develop in the color-mapping section 601 (Fig.s 6A-C), a Spryte-Row Initialization register stack 615 (which comprises an addressable set of 16-bit registers and which is provided within the destination-point mapping section 602, see Fig. 6D) receives a corresponding set of point-to-point mapping control datawords over the D-bus 603 (on a time-multiplexed, DMA basis). The point-to-point mapping control datawords are prestored in the system memory unit (605a, 605b) within a pre-designated SCoB area.

Referring to Fig. 6D, the Spryte-Row Initialization (SRI) register stack 615 supplies point-mapping control signals 615.2 to a point-mapping subsytem 622 of section 602. The point-mapping subsytem 622 is subdivided into an "A-side" and a "B-side" and it includes respective "A-side" and "B-side" math execution platforms 700a and 700b, respective "A-side" and "B-side" row-inititializers 701a and 701b, corner engines 702a and 702b, Munkee units 625a and 625b, and Regis units 626a and 626b. The resources of math execution platform 700a are used on a time-shared basis by the "A-side" Initializer 701a, Corner-engine 702a, "A-side" Munkee unit 625a, and "A-side" Regis unit 626a. Similarly, the resources of

math execution platform 700b are used on a time-shared basis by the "B-side" Initializer 701b, Corner-engine 702b, "B-side" Munkee unit 625b, and "B-side" Regis unit 626b.

Math-platform A (700a) includes four math sections which may be identified briefly here as MathXa1a0, MathXa2a3, MathYa1a0, and MathYa2a3. (See Fig. 7 which will detailed shortly.) These resources are used first by row-initializer 701a to initialize various registers both in math-platform A (700a) and in the SRI register stack 615. Corner-engine A (702a) then takes over control of the same four math sections (MathXa1a0, MathXa2a3, MathYa1a0, and MathYa2a3) to store and/or calculate the X_D and Y_D coordinates of corner points a0, a1, a2 and a3 of a first projected polygon "a". Munkee unit 625a shares use of math-platform A (700a) concurrently with the Corner-engine A (702a). If Munkee unit 625a decides that Regis unit 626a needs to perform a boundary walk, Corner-engine A (702a) and Munkee unit 625a temporarily reliquinsh control over the math-platform A (700a) and Regis unit 626a takes over. When Regis unit 626a completes its boundary-walk, it returns control of the math-platform A (700a) to Corner-engine A (702a) and Munkee unit 625a. When Corner-engine A (702a) terminates calcuclations for corner points in a mapped spryte row, it reliquinshes its control over math-platform A (700a) and row-initializer 701a takes over to initialize data for a next row if needed.

Utilization of the calculation resources in math-platform A (700a) can be expressed as follows:



Int he above expression, CCa represents corner calculations for mapped pixel "a", CCb represents corner calculations for mapped pixel "b", and so on. MKa represents Munkee operations for mapped pixel "a", MKb represents Munkee operations for mapped pixel "b", and so on. Note that MKa takes place co-temporaneously with CCb. There is no Munkee operation at the time of CCa. Munkee operations always consume one clock tick. Corner calculations consume one or two ticks depending on whether a below-described delta-sum shortcut is available. If the MKb operations indicate that a Regis boundary walk is necessary, that follows starting in the next clock tick and can consumes a relatively large number, m, of clock ticks. When Regis-b operations complete, the corner calculations and Munkee operations regain control over the math platform. The process continues until terminated by an end-of-line (EOL) or some other condition. Each processing of a source spryte row begins with an initailization which can take a relatively large number, n, of clock ticks.

Math-platform B (700b) is identically structured to include four math sections which Corner-engine B (702b) uses to store and/or calculate the X_D and Y_D coordinates of corner points q0, q1, q2 and q3 of a second projected polygon "q". (Note the just-mentioned first and second projected polygons are typically but not necessarily arranged one on the next as shown in Fig. 1B.)

The results of the boundary walks (or Munkee shortcut decisions) are pipelined out as sets of three, integer-only coordinates: Y_D, $X_{D\text{-left}}$ and $X_{D\text{-right}}$ (the fractional parts are truncated off.) The output Y_D value is 11-bits wide. The output $X_{D\text{-left}}$ and $X_{D\text{-right}}$ values are each 12-bits wide. These coordinates (YXX for short) are given in terms of a relative display space having an origin at 0,0 and an X_D axis extending rightwardly from

the origin and a Y_D axis extending down from the origin. The relative display space is movably positioned within an absolute displayable space. This aspect of system 600 will be explained below when relative-to-absolute translater 570 (Fig. 6E) is described.

The Regis/Munkee YXX results of math platforms 700a and 700b are each accompanied by respective paint/don't-paint strobe signals 555a and 555b. If a paint strobe signal 555a/b is false, the corresponding YXX results are ignored further downstream in the pipeline flow. If the paint strobe signal 555a/b is true, the corresponding YXX results are accepted by a downstream pixel-by-pixel fill request generator 560 and converted further downstream into one-pixel at a time, memory write requests.

An A-side p-FIFO comprised of R(OUT) register 544, R(HOLD) register 546, multiplexer 550, multiplexer 556 and R(YXX) register 552 carries the YXX results of Munkee unit 625a, and/or Regis unit 626a to a first input of line-fill input multiplexer 554. A B-side p-FIFO comprised of R(OUT) register 545, R(HOLD) register 547, multiplexer 551, multiplexer 557 and R(YXX) register 553 carries the YXX results of Munkee unit 625b, and Regis unit 626b to a second input of line-fill input multiplexer 554.

The output of R(OUT) register 544 feeds back to input port 548 of math-platform A (700a). In similar fashion, the output of R(OUT) register 545 feeds back to input port 549 of math-platform B (700b). The fed back signals are used for boundary-walk tests during the Regis process.

The Y, X_{left}, X_{right} value signals held in each of registers 552 and 553 (Fig. 6D) represent a set of one or more horizontally-adjacent destination pixels that are to be painted with a corresponding PEN color code 618. Each such YXX set is converted by the next-described

downstream pix-by-pix fill-request generator (line-filler) 560 into a series of individual coordinate pairs, YX_1, YX_2, YX_3, etc. where Y represents a horizontal line in a relative display space and X_1, X_2, X_3, etc. represent individual destination pixels arranged one after the next along that horizontal line.

The one or more individual coordinate-pairs, YX_1, YX_2, YX_3, etc. that are produced by pixel-by-pixel fill requests generator 560 then stream out on a requests-carrying bus 561.

Further downstream, at an address translating unit 570 (Fig. 6E), each individual coordinate pair, YX_i, is translated into an absolute memory address signal. It turns out that certain orderings of the individual coordinate pairs, YX_1, YX_2, YX_3, etc., as put out onto the requests-carrying bus 561, produce a stream of corresponding, absolute, memory address signals that involve many page boundary crossings while other orderings produce address sequences that involve fewer page boundary crossings. An order-rearranging line 562 is optionally provided from generator 560 for changing the order in which the corresponding stream of absolute memory address signals present themselves to a downstream-located, page-based memory unit e.g., 605. Preferably, the ordering of the individual coordinate pairs, YX_1, YX_2, YX_3, etc. is rearranged to minimize the number of downstream page crossings. This works to minimize the amount of memory access time needed to complete a line-paint operation. An individual coordinate pair, YX_i, which is found to be positioned out of a desired order is routed by way of order-rearranging line 562 back to one of YXX registers 552 and 553 by way of respective multiplexers 556 and 557. (X_{left} and X_{right} are one and the same when a single destination pixel is represented in YXX format.)

A line-fill sequencing multiplexer 554 (Fig. 6D) picks the output of one of YXX registers 552 and 553 as the one to be next processed by the pixel-by-pixel fill requests generator 560. Information 558 representing the current memory page boundary is supplied to the line-fill sequencing multiplexer 554 to help the latter in its efforts at minimizing the number of downstream page crossings.

Referring to Fig. 6E, the pixel-fill request signals of bus 561 are interleaved with so-called "Dolores-fetch" address signals (565) when they reach a next in-stream multiplexer 564. The "Dolores-fetch" address signals enter multiplexer 564 on line 565. A dolo-fetch address signal is routed back from translater 570 when the R-mode bit goes high to indicate that corresponding color data is being wrapped-back in the color-map section 601.

The purpose of the "Dolores-fetch" address signals (565) is briefly explained as follows: Destination pixel paint requests 561 share access to system memory with a concurrent "Doloresizing" fetch function. The Doloresizing fetch function is used by the PPMP unit 616 to fetch desired CFB datawords into the CFBD FIFO 530 in synchronism with the wrapped-back IPN signal. Doloresizing fetches do not directly affect the concurrent line-fill operations except for the fact that they both share an access time-slot for system memory. Address signals generated by both the line-fill (LF generator 560) and the Dolo-fetch operation flow through multiplexer 564 on a time-shared basis (one for one). Register 566 synchronizes the output of multiplexer 564 to the system clock.

The output of synchronizing register 566 supplies a stream of individual XY coordinate signals to an XY-FIFO 568. The output of XY-FIFO 568 is synchronized to the

system clock and to the outputs of IPS-OUT FIFO 528 and CFBD-FIFO 530.

A relative-to-absolute address translator 570 converts the XY output signals of XY-FIFO 568 from a relative format of 16 bits for each X value and 16 bits for each Y value to an absolute memory address format of 24 bits by adding together a base address, the Y value multiplied by a "modulo" value and the X value. The result is a 24-bit wide memory address signal which is formed according to the following Eq. 5.

$$A_{MEM} = Base + (Modulo * Y) + X \qquad \text{(Eq. 5)}$$

The Doloresizing fetch function has its own Base and Modulo values which are independent from that of line-fill. The base and modulo values for the Dolo and LF functions are represented by signals stored in a MOD/BASE register 573. These values download over the D-bus 603. A select signal (SEL) supplied from a Dolores control unit 533 indicates when the base/mod values of the Doloresizing address translation are to be used (also referred to as the CFBD read translation) and when the base/mod values of the line-fill translation are to be used (also referred to as the spryte write translation). Each translated address signal is output onto a 24-bit wide Regis/Dolo address bus 571. (The Dolores control unit 533 also determines the order in which XY pairs belonging to the Dolo-fetch operation will interleave with LF XY signals as both stream out of multiplexer 564.)

In one embodiment of translater 570, the multiplication, (Modulo * Y), is performed as a sum of two binary-shifted signals. This minimizes the circuit space required for the multiply function and also minimizes the time required for completion of the multiply function. The selectable signals which enter

opposed inputs of an adder by way of respective seletion multiplexers are given in the below Table I.

TABLE I

SELECTABLE OUPUT OF MULTIPLEXER A	SELECTABLE OUPUT OF MULTIPLEXER B
0	0
32*Y	64*Y
256*Y	128*Y
1024*Y	256*Y

A four-bit wide modulo-seleting signal controls multiplexers A and B of translater 570 to pick a desired multiplying factor. By way of example, if the 32*Y output of multiplexer A is selected and the 64*Y output of multiplexer B is selected, these selected signals are next fed into an adder to produce a corresponding (Modulo * Y) value of 96*Y.

In the case of the Line-fill function, the Base term in Eq. 5 functions to position the relative (X,Y) = (0,0) point of the relative spryte-rendering space (see Fig. 3B) within a larger, absolute displayable space (not shown). The selected Modulo term functions to define the address space spread between the stored data of successive destination scan lines. The absolute region into which a spryte is to be rendered can be changed by changing the LF Base value. The address space spread between the stored data of successive destination scan lines can change from region to region.

Given the above relative-to-absolute translation, line-fill sequencing multiplexer 554 and Dolores control unit 533 are preferably (but not necessarily) operated such that the XY values queued up in XY-FIFO 568 are ordered to minimize the number of downstream page crossings which will occur in a downstream page-mode memory system that receives the translated address signals according to the order they are output on bus 571.

Referring to Fig. 6F, each Regis/Dolo address signal 571 is accompanied by a Regis D-bus request signal 572 that is applied to a D-bus arbitration unit 633. The D-bus arbitration unit 633 controls a D-bus arbitration multiplexer 644. The D-bus arbitration multiplexer 644 receives competing address signals from an ARM CPU and a so-called DMA REGISTER STACK unit. Contentions are resolved as indicted previously.

The DMA REGISTER STACK unit includes a first auto-incrementing register which holds an address word named "Engine-A Fetch Address." This address word points to the color code 411 or 412 of a source spryte row that is then being processed by Corner-engine A (702a). The DMA REGISTER STACK unit also includes an auto-decrementing register which holds a DMA count word named "Engine-A Length." This count word defines the number of words still remaining in memory, which need to be fetched for the source spryte row then being processed by Corner-engine A (702a). Similarly, the DMA REGISTER STACK unit further includes a second auto-incrementing register which holds an address word named "Engine-B Fetch Address." This address word points to a color code of a source pixel in a source spryte row then being processed by Corner-engine B (702b). The DMA REGISTER STACK unit also includes an auto-decrementing register which holds a DMA count word named "Engine-B Length." This count word defines the number of words still remaining in memory and representing pixels of the source spryte row then being processed by Corner-engine B (702b).

An access protection unit 645 unit checks the output of D-bus arbitration multiplexer 644 for out-of-range violations. An address splitting unit 646 designates address signals received from the D-bus arbitration multiplexer 644 as belonging either to the left memory bank 605a (Fig. 6C) or the right memory bank 605b of

system memory. Register 586, multiplexer 590 and register 592 are connected in series to define a p-FIFO which drives left-bank memory address pads 604a. Register 587, multiplexer 591 and register 593 are connected in series to define a p-FIFO which drives right-bank memory address pads 604.

Referring back to Fig. 6C, 24-bit wide left and right address signals from respective registers 593 and 592 enter system memory units 605a and 605b in synchronism with corresponding PEN code signals 618 developed by PPMP unit 616. Register 576, multiplexer 580, D-bus driver 582 and D-input register 594 define a p-FIFO which delivers PEN codes to left system memory bank 605a. Register 577, multiplexer 581, D-bus driver 583 and D-input register 595 define a p-FIFO which delivers PEN codes to right system memory bank 605b.

(§6.3) DETAILED DESCRIPTION OF CORNER ENGINE AND MATH PLATFORM RESOURCES

Figure 7 shows details of a first math platform 700a in accordance with the invention and also details of a SRI register stack 615 in accordance with the invention. Corner-engine A (702a) uses the resources of this first math platform 700a, in combination with the Spryte-Row Initialization register stack 615, to calculate the destination coordinates (e.g., c1 and c2) for successive pairs of top and bottom corners (e.g., c1 and c2) of successive source pixels (e.g., a, b, c, etc.) found in a given spryte row (e.g., row A of Fig. 1A.

Math-platform A (700a) is shown divided into a left portion which constitutes an X-coordinates processing section and a right portion which constitutes a Y-coordinates processing section. Each of the X-coordinates and Y-coordinates processing sections is further subdivided to have at its top, an a1-producing

section and to have at its bottom, an a2-producing section. The upper area of each of the X-coordinates and Y-coordinates processing sections further includes an a0-storing section and a respective DX or DY-updating section. An a2-a1 difference calculating section is provided in each of the X-coordinates and Y-coordinates processing sections between the respective upper and lower areas.

Corner-engine A (702a) uses the resources of math-platform A (700a), as will be explained in more detail when Fig. 8 is disscussed, to store and/or generate signals representing the coordinate values for mapped corners a0, a3, a1, a2 and corner-to-corner distance values (a3 - a0), (a1 - a0), (a2 - a3), and (a2 - a1). After Corner-engine A (702a) completes its computations, it leaves its result signals stored in registers of the math-platform A (700a) and goes to sleep.

Signals representing the destination coordinates for two successive pairs of top and bottom source corners (e.g., c0 and c3, c1 and c2), are left behind in a respective set of four X-registers (1740, 1743, 1741, 1742) and four Y-registers (1780, 1783, 1781, 1782) of first math platform 700a. Signals representing the corner-to-corner distance values (e.g. [c3 - c0], [c1 - c0], [c2 - c3], and [c2 - c1] are left behind in another respective set of three X-registers) and three Y-registers of first math platform 700a or output by subtractors .

Referring still to Fig. 7, it is seen that the SRI register stack 615 has sixteen registers each of which is 16-bits wide. The registers are divided into two groups of eight addressable registers each. One group (the left group) stores X values and the other (the right group) stores Y values.

The X group of SRI registers has a DDXL\ register (address=000) for storing the less significant sixteen bits of a 32-bit wide binary-coded signal that represents delta value DDX in binary-inverted format. The last "L" in the notation "DDXL\" indicates it is the lower half of the 32 bits and the "\" indicates that the data is inverted.

The X group of registers includes the following similarly named, additional registers: DXL\ (address=001), DDXH\ (address=010), DXH\ (address=011), LDXL\ (address=100), CXL\ (address=101), and LDXH\ (address=110), CXH\ (address=111). The DXL\ register stores the less significant sixteen bits of a 32-bit wide binary-coded signal that represents double delta value DX+n•DDX in binary-inverted format, where n=0,1,2,3, etc. for successive rows of a source spryte. The DXH\ register stores the more significant sixteen bits of a 32-bit wide binary-coded signal that represents double delta value DX+n•DDX in binary-inverted format.

The CXL\ register stores the less significant sixteen bits of a 32-bit wide binary-coded signal that represents line-delta incremented value XPOS+n•LDX in binary-inverted format, where n=0,1,2,3, etc. for successive rows of a source spryte. The CXH\ register stores the more significant sixteen bits.

The Y group of registers includes the following similarly named and similarly functioning registers: DDYL\ register (address=000), DYL\ (address=001), DDYH\ (address=010), DYH\ (address=011), LDYL\ (address=100), CYL\ (address=101), and LDYH\ (address=110), CYH\ (address=111).

A stack control unit 710 supplies the SRI register stack 615 with a 3-bit write address signal, WA, a 3-bit read address signal, WR, and a 1-bit write-enable signal, WE. The data-input side of SRI register stack 615 is fed

by an X-input multiplexer 706 (which has an inverting output) and a Y-input multiplexer 707 (which has an inverting output). Stack control unit 710 supplies selection control signals to multiplexers 706 and 707.

A first of the selection control signals, LOAD-D, instructs multiplexers 706 and 707 to transfer data from D-bus 603 into the SRI register stack 615. A lower 16 bits (15:0) of D-bus 603 load into register 703. An upper 16 bits (31:16) of D-bus 603 pass through multiplexer 704 (has an inverting output) and load into register 705 (also has an inverting output). The output of register 705 connects to a D input of multiplexers 706 and 707. With each D-bus input transfer, the upper 16 D-bus bits load into register 705 first and then the lower bits of register 703 pass through multiplexer 704 and load into register 705 afterwards. Note that the respective outputs of multiplexer 704, register 705 and multiplexers 706 and 707 are inverting. SRI register stack 615 therefore stores an inverted version of the D-bus data. The 16-bit wide outputs of the X and Y register groups of SRI register stack 615 feed respective output registers 708 and 709. Registers 708 and 709 each has both an inverting and noninverting output.

A second selection control signal, LOAD-XYO, instructs multiplexers 706 and 707 to transfer respective data signals, XSO and YSO, from inverting outputs (Q-bar outputs) of registers 708 and 709 to the inputs of the X and Y register groups. Respective data signals, XSO and YSO, also feed to respective D-bus tri-state drivers 798 and 799. Note that data signals XSO and YSO are in true rather than inverted form.

A third selection control signal, INIT, instructs multiplexers 706 and 707 to transfer respective data signals, XIN and YIN, to the inputs of the X and Y register groups. These signals, XIN and YIN, develop on

tri-state buses which are shared by math-platform A (700a) and math-platform B (700b). Stack control unit 710 supplies tri-state bus control signals, W-A and W-B, to respective platforms A and B (700a and 700b) for enabling each to exclusively drive the tri-state XIN and YIN buses.

Math-platform A (700a) includes a pair of tri-state bus-drivers, 744 and 784, which are enabled by the W-A signal to output respective engine-A signals, XA and YA, onto the XIN and YIN buses. Math-platform B (700b) is similarly structured (not shown) to place engine-B signals, XB and YB, onto the XIN and YIN buses. Bus-drivers 744 and 784 have inverting inputs. The signals at the inputs of drivers 744 and 784 are accordingly referenced as XA\ and YA\.

A 16-bit wide engine bus 743, which carries an Xa3\h signal, connects to the input of bus-driver 744. Another engine bus 783, which carries a Ya3\h signal, connects to the input of bus-driver 784. The "h" suffix in notations Xa3\h and Ya3\h indicates that these 16 bit signals represent "half" of a 32-bit wide signal. The represented "half" can be either the left-hand 16 bits (integer part) of a 32-bit wide value formatted as 16.16 or the right-hand 16 bits (fractional part) of the 16.16 formatted signal. Thirty-two bit wide signals are trnasferred over.

During a so-called spryte row initialization (SRI) operation, Xa3\h engine bus 743 and Ya3\h engine bus 783 each transfer the left and right halves of respective values Xa3 (the X_D coordinate of mapped point a3) and Ya3 (the Y_D coordinate of mapped point a3) to SRI register stack 615 for storage in respective locations CXH\, CXL\, CYH\, CYL. (The upper halves transfer first, then the lower halves.)

In addition to Xa3\h engine bus 743 and Ya3\h engine bus 783, math-platform A (700a) includes a 16-bit wide Xa0\h engine bus 740 for carrying respective upper and lower halves of an Xa0 signal which represents the X_D coordinates of mapped corner a0 and a 16-bit wide Ya0\h engine bus 740 for carrying respective upper and lower halves of an Ya0 signal which represents the Y_D coordinates of mapped corner a0. Math-platform A (700a) further includes, a similarly named and similarly functioning: Xa1\h engine bus 741, Ya1\h engine bus 781, Xa2\h engine bus 742 and Ya2\h engine bus 782.

Engine buses 740-743 and 780-783 receive their respective signals from a corresponding set of 16-bit wide engine-bus registers, 1740-1743 and 1780-1783, by way of inverting bus-drivers 3740-3743 and 3780-3783. Two bit data-save registers 2740-2743 and 2780-2783 are provided for storing 1 tick-delayed data.

Referring to the Xa1-producing section of math-platform A (700a), two signal-routing multiplexers, 711 and 712, are provided for routing signals representing addends to respective A and B input ports of a 16-bit wide adding unit 715. Exclusive OR units 713 and 714 are respectively interposed between the outputs of signal-routing multiplexers 711 and 712 and the A and B input ports of adding unit 715 for selectively inverting or not-inverting the 16-bit wide signals passing from multiplexers 711 and 712 to the A and B input ports of adding unit 715.

A 16-bit wide result bus 716 carries the result signal produced by adding unit 715 for loading into either a corner-low holding register (CL) 717, or a lower 16-bits of an 18-bit wide, corner-high holding register (CH) 1741, or an X1-holding register 765 or a DX1-holding register 765, or by way of a multiplexer 719 into an upper 2-bit portion of the 18-bit wide, corner-high

holding register (CH) 1741. A programmable control means (not shown) determines which of these destination registers will load the result signal produced by adding unit 715.

(§6.4) PSEUDO-INDEPENDENT OPERATION OF MULTIPLE CORNER ENGINES AND MUNKEE AND REGIS UNITS

Figure 8 diagrams a math resource sharing and result sharing process 800 of the invention.

At starting step 801, the coordinates for destination point a0 (XPOS, YPOS) are downloaded into SRI register stack 615 from a SCoB region of system memory. The other point-map control signals 415.2 (e.g., LDX, LDY, DX, DY, DDX, DDY) are also downloaded into stack 615. Corner-engine A (702a) and Corner-engine B (702b) are both dormant at this time.

At step 802, row-initializer 701a uses the resources of math platform-A (700a) to compute the destination coordinates for point a3 (by adding LDX and LDY to XPOS and YPOS respectively). Row-initializer 701a also computes the DX++ and DY++ values for source spryte row-A (by adding DDX and DDY to DX and DY respectively). Row-initializer 701a then returns the calculated coordinates for point a3 and the updated DX++ and DY++ values to the SRI register stack 615. This is indicated by step 803.

The a3 coordinate values returned to SRI register stack 615 are then used as the coordinates for destination point q0. When step 803 completes, row-initializer 701b awakens from its dormant state and begins its row initialization procedure for source spryte row-B. At step 804, row-initializer 701b uses math platform-B (700b) to calculate the coordinate values for destination point q3 (by adding LDX and LDY respectively to the X_D and Y_D coordinates of point q0). It also

computes the DX++ and DY++ values for source spryte row-B (by adding DDX and DDY respectively to the DX++ and DY++ values received from row-initializer 701a. Row-initializer 701b then returns the q3 coordinate values and the updated DX++ and DY++ values back to the SRI register stack 615 in step 805. The values returned in step 805 then become the coordinate values for next corner point w0 and the point-map control signals for next source row C.

In a more preferred embodiment, rather than initializing points a0 and a1, a hypothertical pixel having corner points a-1 and a-2 is created and the values for a0 and a3 are initialized instead as values for hypothetical corner points a-1 and a-2. When the corner engine next takes over control of the math platform, the first thing done is to transfer the values from a-1 and a-2 respectively to a0 and a1.

@ As soon as row-initializer 701a completes its row initialization steps, 802 and 803, corner engine-A (702a) takes over control of math-platform A (700a) and it (702a) enters a row-servicing mode in which it dedicates itself to calculating destination coordinates for successive top and bottom corner points of spryte row A. Row-initializer 701b is entering its row-B initialization mode at this time.

At step 810 (after having transferred the a-1 and a-2 values respectively to a0 and a3), corner engine-A (702a) calculates the coordinate values for destination point a1 (by adding DX and DY to the respective X_D and Y_D coordinates of starting point a0). At step 812, corner engine-A (702a) calculates the coordinates for destination point a2 (by adding the row-A version of DX++ and DY++ to the respective X_D and Y_D coordinates of point a3). The X_D and Y_D coordinates for destination points

a0, a3, a1 and a2 are saved within registers of the first math platform 700a.

Munkee performs its functions contemporaneously but one tick out of phase with the corner engine. If a Regis boundary walk is required, Corner engine-A (702a) goes to sleep and the Regis unit (626a) takes over control of the first math platform 700a. The functions of the Regis unit is represented by boundary-walking (BW) function module 825. ("BW ppa" is to be read as "Boundary Walk for projected polygon 'a' ".) Steps 820, 821, 822 and 823 represent the respective handing-over of the coordinate values for points a0, a1, a2 and a3 to boundary-walking function module 825. Steps 853 and 854 (which for purposes of illustrative clarity are shown only in the b0-b3 projected polygon but are understood to have counterparts in projected polygon 'a') represent the respective handing-over of the coordinate difference values: (a1-a0) and (a2-a3) to boundary-walking function module 825. (Note that these difference values are inherently represented by the DX, DY, DX++ and DY++ signals of row "A" and, accordingly, they are handed over rather than being calculates anew.) Preferably, the "handing-over" functions 820-822 and 853, 854 do not include a physical movement of signals from one register to another. Instead, the signals are left in the registers where they are originally stored, and the BW function module 825 accesses the contents of those same registers to decide whether destination pixels will be painted, and if so, to produce the corresponding line-fill command signals (e.g., YXX, YXX, etc.).

While difference values such as (b1 - b0) and (b2 - b3) are handed over by steps 853 and 854 to the boundary-walking function module 855 (or 825), it is to be noted that the Y direction difference values, (b3 - b0) and (b2 - b1) are, in general, not available

for handing over to the boundary-walking function module. The boundary-walking function module 855 (or 825) therefore pre-calculates difference values (b3 - b0) and (b2 - b1) on its own one tick before it takes over control of the first math platform 700a (using an available set of subtractors in the math platform). On the other hand, it is to be noted that difference value (c3 - c0) is the same as difference value (b2 - b1). After difference value (b2 - b1) is calculated once, its representative signal is handed over as difference value (c3 - c0) rather than being regenerated anew when BW-ppc module 885 later needs that value.

The four difference values (b1 - b0), (b2 - b3), (b3 - b0) and (b2 - b1) are used by Regis unit 626a to determine whether one, none, or many destination pixels are to be painted as a result of the geometry of polygon "b" (or "a").

Munkee determines ahead of time, for each difference value whether it is zero or not-zero, and if not, whether it is positive or negative. The equals-zero and positive/negative indicating signals are used to reach some simple conclusions about the projected polygon. By way of example, if all four difference values, (b1 - b0) through (b3 - b0), equal zero, Munkee unit 425a can quickly decide that no destination pixels are to be ultimately painted. In such a case, Munkee 425a blocks Regis 426a from taking control and performing its more precise border-point computations. No time is wasted by the more comprehensive and more precise calculations that would have been carried out by the Regis unit 426a had there been many destination pixels to be painted.

Once the boundary-walking functions of the BW-ppa module 825 terminate (for whatever reason), corner engine-A (702a) reawakens and begins to calculate the destination coordinates of subsequent points b1 and b2.

The coordinate values for points b0 and b3 do not have to be calculated because they are the same as a1 and a2. The difference value (b3 - b0) does not have to be recalculated because it is the same as (a2 - a1). Accordingly, at steps 831 and 832, the signals representing the coordinates for points a1 and a2 and representing the difference value (a2 - a1) are transferred such that they now serve as the signals representing destination points b0 and b3. Then, at steps 840 and 842, corner engine-A (702a) computes the coordinates for b1 and b2.

If another Regis boundary walk is needed, Corner engine-A (702a) goes to sleep once again, and in the process, hands over values for coordinates b0, b1, b2, and b3 to a BW-ppb function module 855. Difference values b1 - b0 and b2 - b3 are also handed over as indicated at 853 and 854 respectively. The BW-ppb function module 855 then takes over control of the math platform 700a. Line-fill commands 856 (YXX, YXX, etc.) are generated in the case where the BW-ppb function module 855 decides that destination pixels need to be painted for projected polygon "b".

Upon termination of the functions by the BW-ppb module 855, corner engine-A (702a) reawakens to repeat the process for projected polygon "c". Coordinate values b1 and b2 become respective new values c0 and c3 as indicated at 861 and 862. Corner-engine A (702a) calculates coordinates c1 and c2 respectively from c0 and c3. A BW-ppc function module 875 thereafter takes over control of math platform 700a to produce yet more line-fill commands 876 (YXX, YXX, etc.) if needed.

This map-and-paint process for source spryte row-A continues until terminated by the encounter of an end-of-line (EOL) code in the source spryte row (A) or until terminated by some other cause such as a decision by an

out-of-bounds detecting unit (not shown) that calculations are proceeding ever deeper into a nondisplayable region (e.g., a super-clipped region).

While this is occurring in the first math platform 700a, a similar process is preferably occurring in parallel in second math platform 700b. Corner engine-B (702b) uses the q0 and q3 coordinate values produced during the row-B initialization mode to calculate successive destination points q1 and q2. BW-ppq function module 885 then takes over control of second math platform 700b to produce line-fill command signals 886 (YXX, YXX, YXX, etc.), if needed.

Note that the BW-ppq function module 885 is illustrated in Fig. 8 as being much longer than the BW-ppa function module 825. This is intentionally done to convey the idea that, for each source pixel a, b, c, ..., q, r, s, ..., etc., a substantially different amounts of time may be consumed by the respective first and second math platforms, 700a and 700b, to complete their respective corner calculations and Regis operations. There are many reasons why this can happen. Corner engine-A (702a) might be able to take advantage of certain calculation shortcuts (see explanation of Fig. 9A) which are not available to corner engine-B (702b). Boundary-walking function module 825 may be skipped because of a fast decision by Munkee to paint one or no destination pixels, whereas no such fast decison is provided in engine B to circumvent the time consumed by BW-ppq function module 885.

The operations of first and second math platforms, 700a and 700b, can therefore quickly become nonsynchronized with respect to one another. Each math platform, 700a and 700b, is preferably operated pseudo-independently of the other, rather than in lock step, so that each can take advantage of all time-saving methods

that become available to that math engine even if the same shortcuts do not become available to the other math engine.

Either one of math platforms 700a and 700b can complete or end the processing of its assigned source spryte row (A or B) before the other for any number of reasons. Math platform-B (700b) for example, may end its processing of source spryte row-B well ahead of the time that math platform-A (700a) ends its processing of source spryte row-A simply because source spryte row-B has fewer pixels than source spryte row-A. The first math platform, 700a or 700b, which completes or otherwise terminates processing of its assigned source row (A or B) then picks up the next-queued task in the SRI register stack 615. That task is to initialize row C (by calculating w3 and updating DX++ and DYY), and thereafter proceeding to map the remaining pixels if any of source row C.

This map-and-paint process 800 continues until either all the rows of a given source spryte are processed or termination occurs due to encounter of a cropping border or some other terminating cause. The SCoB of a next-to-be rendered spryte is then downloaded into SRI register stack 615 and process 800 begins all over again.

(§6.5) DELTA SUMMING MECHANISM

The above discussion mentions the idea that certain calculation short-cuts may be available to one math engine but not the other. One important short-cut is found in calculating the sum of a relatively large first value and a relatively smaller second value (delta value).

Figure 9A shows a first summing unit 900 in accordance with the invention that takes advantage of

such a short-cut. Similar, more complex summing units are found within math platforms 700a and 700b of Fig. 7. The more complex structures will be explained later in conjunction with Fig. 9B.

Referring first to Fig. 9A, respective first and second result-part registers, 910 and 920, each of which is 16-bits wide, are provided for respectively storing a less significant part, XL, and a more significant part, XH, of a 32-bit wide result signal, XHL. The result signal XHL is iteratively updated over successive clock cycles according to the delta summing equation:

$$XHL_{i+n} = XHL_i + \Delta X_i \qquad \text{(Eq. 5)}$$

In equation Eq. 5, XHL_i is a 32-bit wide value represented by a signal XHL that is stored in registers 910 and 920 at clock cycle, i. ΔX_i is a 32-bit wide value represented by a signal DXHL that is stored in another set of registers, 915 and 925, at clock cycle, i. XHL_{i+n} is a value of the XHL signal stored in registers 910 and 920 at a later clock cycle, i+n, where n is an integer greater than 0 (n= 1, 2, etc.).

It will be assumed here that the more significant 16 bits (XH) of the 32-bit long result signal represent an integer portion of XHL_{i+n}. A hypothetical decimal point is placed between parts XH and XL as indicated by the notation: XH.XL. The less significant 16 bits (XL) of the 32-bit long result signal XHL therefore represent the fractional portion of XH.XL.

It will be further assumed here that the delta value, ΔX_i, changes slowly over time and it is typically, but not always, in the range:

$$+2.0 > \Delta X_i > -2.0. \qquad \text{(Eq. 6.1)}$$

and it is more often, but not always, in the range:

$$+0.5 > \Delta X_i > -0.5. \qquad \text{(Eq. 6.2)}$$

A 16-bit wide adding unit 930 is provided for selectively updating the contents of one or both of the

first part register 910 and the second part register 920. Adding unit 930 has two input ports, 931 and 932 (input-A and input-B), each 16-bits wide, and a corresponding 16-bit wide output port 933 for producing a sum signal representing the sum of values applied to its A and B input ports, 931 and 932. All 16 bits of the adder output port 933 connect to the 16-bit wide data inputs of both the first part register 910 and the second part register 920.

Adding unit 930 has a carry and/or borrow (CY/BW) output port 934 for outputting a carry or borrow indicating signal (CY/BW indicating signal) in the event that a carry or borrow is generated by a respective addition or subtraction operation performed within the adding unit 930. Adding unit 930 also has a carry and/or borrow (CY/BW) input port 936 for inputting a carry or borrow indicating signal in the event that a carry or borrow is to be applied to a respective addition or subtraction operation being performed within the adding unit 930. A carry/borrow storing register 935 is provided for receiving and storing a CY/BW indicating signal output by the C/W output port 934 of the adding unit 930 during a first tick of the system clock (CLK) and for supplying the stored CY/BW indicating signal to the C/W input port 936 of the adding unit 930 during a later, second tick of the system clock. For purposes of explanation, CY/BW = +1 will represent an active carry, CY/BW = -1 will represent an active borrow, and CY/BW = 0 will represent the condition of no carry and no borrow.

As already mentioned, a supplied 32-bit wide signal, DXHL, represents the delta value, ΔX_i, of above Eq. 5. Like the XHL signal, the DXHL signal is divided into integer and fractional subportions, DXH and DXL, each 16-bits wide. A hypothetical decimal point is placed

between parts DXH and DXL as indicated by the notation: DXH.DXL.

Respective first and second delta-part registers, 915 and 925, are each 16-bits wide, and they are provided for respectively storing the less significant part, DXL, and a more significant part, DXH, of the supplied 32-bit wide delta signal, DXHL. The delta signal DXHL generally remains constant over many ticks of the system clock but it can be changed over successive ticks of the system clock.

First and second multiplexers, 940 and 950, are provided respectively for selecting and supplying corresponding first and second addend signals to the A-input 931 and B-input 932 of adding unit 930.

The output of the first result-part register 910 connects to a first input (input-A) 941 of the first addend-selecting multiplexer 940. The output of the first delta-part register 915 connects to a first input (input-A) 951 of the second addend-selecting multiplexer 950. The output of the second result-part register 920 connects to a second input (input-B) 952 of the second addend-selecting multiplexer 950. The output of the second delta-part register 925 connects to a second input (input-B) 942 of the first addend-selecting multiplexer 940.

A math sequence control unit 960 is provided with a first state input 961 coupled to detect the state of the C/W output port 934 (whether it is equal to 0, +1 or -1) and a second state input 962 coupled to detect the state of the output of the second delta-part register 925 (whether it is equal to 0, +1 or -1). The math sequence control unit 960 outputs first and second selection signals, 964 and 965, for respectively controlling selections made by the first and second addend-selecting multiplexers, 940 and 950. The math sequence control

unit 960 also outputs first and second load-control signals, 967 (LDL) and 968 (LDH), for respectively controlling the input loading operations of the corresponding first and second result-part registers, 910 (XL) and 920 (XH).

The process of sum generation is subdivided into at least two successive parts. Carry/borrow detection and analysis is performed at an intermediate point of the sum generation process to see if a next-to-be performed part of the sum generation process can be bypassed. This is another manifestation of the prime design rule: identify and avoid unnecessary work.

In the case of math unit 900, the sum generation process begins by summing together the corresponding less significant portions, DXL and XL, of the delta value (DXH.DXL) and the running total (XH.XL). The math sequence control unit 960 accordingly outputs first and second selection-control signals, 964 and 965, for respectively causing the first addend-selecting multiplexer 940 to select its input-A 941 (XL) as its output and for causing second addend-selecting multiplexer 950 to select its input-A 951 (DXL) as its output. When the corresponding result, XL = XL + DXL, appears at the adder output port 933, the math sequence control unit 960 outputs an active LDL signal 967 and thereby loads the result into the first result-part register 910.

While the adder output signal (933) loads into the first result-part register 910, the math sequence control unit 960 tests the conditions present at its first and second state inputs, 961 and 962, to determine if the next-planned step of summing together the corresponding more significant portions, DXH and XH, of the delta value (DXH.DXL) and the running total (XH.XL), plus the carry or borrow of the just completed summation, will

ultimately fail to produce any change in the more significant part (XH) of the running total XHL. One example of this condition is where no carry or borrow is generated at the intermediate point and the more significant, DXH part of the delta value is equal to zero. Another example of this condition is where a borrow is generated at the intermediate point and the more significant DXH part of the delta value is equal to positive one. Yet another example of this condition is where a carry (CY/BW=+1) is generated at the intermediate point and the more significant, DXH part of the delta value is equal to minus one.

In each such case, performance of above equation Eq. 5 is deemed complete after summing only the less significant parts, XL and DXL. The more significant, XH part of the running total (XHL) is left unchanged in the second result-part register 920. Time is saved by avoiding the unnecessary step of adding zero to the value already stored in second result-part register 920. Moreover, it is to be appreciated that circuit space is saved by using the 16-bit wide adding unit 930 rather than a larger 32-bit wide adder.

When the earlier made assumption about the delta value holds true (that it often satisfies the condition: $+0.5 > \Delta X_i > -0.5$), summations will often complete in one tick of the system clock although the result is 32 bits and the adding unit 930 is only 16-bits wide. By way of example, consider the case where XH.XL = 0.0 at clock tick number zero and DXH.DXL = 0.1 over a period of 100 clock ticks. Addition will complete in one clock tick 90% of the time.

If the math sequence control unit 960 determines at the intermediate result point that the next-planned step of summing together the corresponding more significant portions, DXH and XH, of the delta value (DXH.DXL) and

the running total (XH.XL), plus the carry or borrow of the just completed summation, will produce a change in the more significant part (XH) of the running total XHL, then summation completes in a conventional manner. The math sequence control unit 960 switches its first and second selection-control signals, 964 and 965, for respectively causing the first addend-selecting multiplexer 940 to select its input-B 942 (DXH) as its output and for causing second addend-selecting multiplexer 950 to select its input-B 952 (XH) as its output. The carry or borrow from the less significant portion of the sum generation process is saved in the carry/borrow storing register 935 and applied in the next clock tick to the C/W input port 936 of the adding unit 930 in conjunction with the DXH and XH signals (942 and 952). When the corresponding result, XH = XH + DXH + CY/BW, appears at the adder output port 933, the math sequence control unit 960 outputs an active LDH signal 968 and thereby loads the result into the second result-part register 920.

Figure 9B shows another adding unit 1000 in accordance with the invention. Sum generation proceeds here in basically the same manner as described for Fig. 9A. Reference symbols in the "1000" number series are used to represent elements having like counterparts in the "900" number series in Fig. 9A. For the sake of illustrative simplicity, the various registers which store the running sub-total values and the applied delta values are not shown. The math sequence control unit (1060) which loads result values into the sub-total result registers and which controls the adding selecting function of multiplexers 1040 and 1050 is also not shown.

Referring to the top of Fig. 9B, it is seen that three input signals, X, DX, and DDX, are replied to sum generating unit 1000. Each of these signals is 32-bits

wide. Part of the 32 bits represents a binary-coded integer portion and the other part represents a binary-coded fractional portion. The notation (i.f) is used to represent the partitioning of each signal relative to the decimal point. The symbol "i" is an integer defining the number of bits to the left of the decimal point and the symbol "f" is an integer representing the number of bits to the right of the decimal point.

Input signal X(16.16) is partitioned into an 18-bit wide, less significant part signal, XLP(2.16) and a 14-bit wide, more significant part signal, XMP(14[redacted].). The notation "14[redacted]." indicates the most significant 14 bits of a value that has 16 bits to the left of the decimal point.

Note that input signal X(16.16) is arranged to have an equal number of bits to the left and right of the decimal point while input signals DX(12.20) and DDX(12.20) are arranged to have 12 bits to the left of the decimal point and 20 bits to the right of the decimal point. The delta input signal, DX(12.20) is used for two summing operations of different precisions. In a first of the summing operations, precision is limited to 16 bits to the right of the decimal point and delta value DX is added iteratively to the running subtotal value X. Four sign-extension bits are hypothetically padded to the left and a value DX(12.20) and the four right most bits are truncated away.

The second operation is carried out to a precision of twenty bits to the right of the decimal point. In this second operation, delta value DX(12.20) serves as the running subtotal and the double hyphen delta value DDX(12.20) is iteratively added to running subtotal DX(12.20).

The delta value signal DX(12.20) is accordingly partitioned into a plurality of different component

signals depending on the operation to be performed. For its addition to running-total value X(16.16), the delta signal DX(12.20) is subdivided to form a 10-bit wide first more significant signal part, $DXMP_1$(10[redacted].), a 14-bit wide first less significant signal part $DXLP_1$(2.16) and a 4-bit wide, not used portion $DXNP_1$(.[redacted]4). The notation "[redacted] " is used to represent a large plurality of empty bit positions. In the case of not-used signal part $DXNP_1$, the ".[redacted] " symbol represents sixteen empty bit positions preceding the last four, filled bit positions.

As seen in Fig. 9B, a 2-bit wide adder 1130 is provided in addition to 16-bit wide adder 1030. In order to add the delta value DX([redacted]12.16[redacted]) to running total value X(16.16) the lower sixteen bits of signal $DXLP_1$(2.16) is applied to a first input of multiplexer 1050 while the lower sixteen bits of signal XLP(2.16) is applied to a corresponding input of multiplexer 1040. The upper two bits of $DXLP_1$(2.16) are applied to one input of 2-bit adder 1130 while the upper two bits of XLP(2.16) are applied to the opposed input of 2-bit adder 1130. In a first clock tick, the 16-bit adder 1030 produces the result signal XLP(.16)++ on its output port 1033 together with any corresponding carry-indicating bit (CYO_{16}) on carry output port 1034. In the same tick of the system clock, the 2-bit adder 1130 produces the 2-bit wide result signal XP(2.)++ on its output port 1133 together with any corresponding carry-indicating signal (CYO_2) on its carry output port 1134. Result signals XP(2.)++ and XP(.16)++ combine to form the 18-bit result signal XLP(2.16)++. If the CYO_{16} indicating signal indicates that no carry or borrow has been generated by the operation just performed in 16-bit adder 1030, the XLP(2.16)++ signal is saved as is. Otherwise, the 2-bit addition of 2-bit adder 1130 is reperformed in the next

clock tick with the appropriate carry or borrow signal applied to its CYI_2 input. If the condition of the CYO_2 signal and $DXMP_1$(10[redacted].) signal indicate that no change is required for the more significant portion XMP(14[redacted].)++ of the running total, that subsequent operation is foregone and time is saved. Otherwise, $DXMP_1$([redacted] 10[redacted].) is passed through a second input of multiplexer 1050 while corresponding signal XMP([redacted]14[redacted].) passes through an opposed second input of multiplexer 1040 for addition within 16-bit adder 1030.

Similarly, when double hyphen delta value DDX is added to single hyphen delta value DX, their corresponding less-significant parts, $DXLP_2$(.[redacted]-[redacted] 16) and DDXLP(.[redacted]-[redacted] 16) are first added together. If an upper part adjustment is found to be necessary, corresponding signals $DXMP_2$(12.4) and DDXMP(12.4) are subsequently summed through 16-bit adder 1030.

As such, it is seen that the invention may be applied advantageously to add numbers of different precisions and that a 16-bit adder may be advantageously operated to produce 32-bit wide results. It is worthy to note that the $DXLP_1$ signal has a format of (2.16) rather than simply ([redacted].16). This partitioning is specifically adapted for optimizing performance in the spryte rendering engine of the invention. It was observed that most spryte mapping functions use a delta value in the range +4.0 > DX > -4.0.

Referring momentarily back to Fig. 3A, it is noted that value DX represents magnification in the X_D direction. The map control signal LDY defines magnification in the Y_D direction. Most zoom and unzoom operations use a magnification factor of less than four. The sum generating unit 1000 of Fig. 9B is structured to minimize the number of clock ticks consumed by magnifications of less than four while minimizing the

amount of circuit space consumed by unit 1000. Other arrangement may, of course, be used to suit the statistical performance requirements of other applications.

(§6.6.1) DETAILED DESCRIPTION OF MUNKEE SHORTCUT ALGORITHMS

In the below code conversion Table II, input data bits are presented in a left-hand column in the same left to right order as recited next to the introduction "INPUTS: ...". The corresponding output data bits are indicated on the same line in a right-hand column in the same left to right order as recited next to the introduction "OUTPUTS: ...". For purposes of speed and circuit compactness, the code-conversion functions are preferably implemented in the form of combinatorial logic circuitry which is designed using conventional Karnough-mapping techniques or the like. The code-conversion functions can be alternatively implemented in the form of a ROM (read only memory) circuit or a computer program or the like. (Copyright Notice: In so far that the subject matter of the code-conversion tables is coverable by copyrights, the copyright owner reserves all such rights except those expressly waived above.)

The below Munkee code-conversion table was created by first running Regis with all possible input combinations. Then the results produced by Regis were reviewed and all cases where the end result was to paint no pixels were extracted. Then a first code conversion table was developed based on inputs defining whether the deltas were equal to zero or the deltas were negative. The idea was to cover as many cases as practical without requiring an excessively large ROM or logically

equivalent (but faster) combinatorial circuit. Suprisingly, the table turned out to be relatively small because it included a simple rule: if any two [ACCURATE ???] of distances: (a1-a0) OR (a2-a3) OR (a3-a0) between the corners of the projected polygon are zero, the projected polygon collapses to have an area of zero and no destination pixel will be painted.

As a further improvement, all combinations of results produced by Regis were again reviewed and all cases where the end result was to (1) paint no pixels OR to (2) paint one pixel were extracted. Then a second code conversion table was developed based on inputs defining whether the deltas were equal to zero or the deltas were negative. The idea again, was to cover as many cases as practical without requiring an excessively large ROM or logically equivalent (but faster) combianatorial circuit. The resultant table was larger than the first but, when certain cases were converted to an "I'm not sure" output rather than a definitive assertion that zero or one pixel will be painted, it was possible to reduce the second table to a sufficiently small size where it could be economically and practically contained within the integrated circuit housing the math platfrom, corner engine and Regis. The result of this approach is given by below TABLE II (REDMUN.FDS).

It is to be understood that if desired, one could repeat the same procedure to develop a table that tries to cover all or most of the cases where Regis ultimately decides to paint 0, 1 or 2 pixels. In such a case the input parameter list will be larger, and the corresponding fast-decision unit will be larger in size and/or slower in speed. A trade-off should be made between performance gain in the image rendering system and the costs involved in trying to cover more and more of the Regis outcomes with a fast-deciding circuit.

There will be a point of diminishing returns at which the time consumed by the line-filler to paint more than, say 4 or 5 pixels, dominates over the time saved by circumventing the Regis operations with a fast decision. There will be a further point of diminishing returns at which the size of the fast-deciding circuit no longer makes economic sense (e.g., it is impractically large). For a 0.9 micron line-width technology used in one embodiment of the invention, the inventors here decided that it was prudent to use a fast-deciding circuit which covers most or all zero paint results and a majority of the paint-one pixel results. For all other cases, Munkee esentially says "I don't know" and leaves it to Regis to perform the operations.

In the below Table II, the inputs are defined by multi-character mnemonics. An initial D means "delta", the letter after the D defines the direction of the delta as X or Y. The next to digits identify the from and to corners. 01, for example means the distance going from a0 to a1. (DX01= a1-a0). The next letter identifies the input parameter as representing whether the delta is Equal to zero or whther the delta is negative.

The outputs are: NP=1 means that no destination pixels are to be painted. MF=1 (Munkee function) means that only one pixel will be painted. An "I don't know" condition is defined by the logic NOR of MF and NP (both are false). CW means that the to-be-painted pixel is clockwise. MX identifies which of corners a0-a3 should be used as the X for the output YX value. MY identifies which of corners a0-a3 should be used as the Y of the output YX value. Each input/output "1" represents a logic true electrical or other signal, each input/output "0" represents a logic false electrical or other signal, each input/output "x" bit represents a logic don't-care electrical or other signal.

TABLE II (REDMUN.FDS)

```
COPYRIGHT © 1992 The 3DO Company
CKTNAME : REDMUNK;
TYPE    : COMB;
INPUTS  : DX01N, DX01E, DX23N, DX23E, DX02N, DX02E,
          DY01N, DY01E, DY23N, DY23E, DY02N, DY02E;
OUTPUTS : MF, NP, MX1, MX0, MY1, MY0, CW;
/* trymunk.eqn */
<TT>

   000000000000   01xxxxx
   000000000001   1000x00
   000000000010   00xxxxx
   000001000000   10x0011
   000001000001   01xxxxx
   000001000010   10x0000
   000010000000   00xxxxx
   000010000001   1010x01
   000010000010   01xxxxx
   000000000100   01xxxxx
   000000000101   00xxxxx
   000000000110   00xxxxx
   000001000100   01xxxxx
   000001000101   10x0x00
   000001000110   00xxxxx
   000010000100   01xxxxx
   000010000101   01xxxxx
   000010000110   10101x1
   000000001000   xxxxxxx
   000000001001   00xxxxx
   000000001010   00xxxxx
   000001001000   xxxxxxx
   000001001001   10x0x00
   000001001010   00xxxxx
   000010001000   xxxxxxx
   000010001001   01xxxxx
   000010001010   1010101
   000100000000   01xxxxx
   000100000001   1000x00
   000100000010   1000000
   000101000000   10x0011
   000101000001   01xxxxx
   000101000010   01xxxxx
   000110000000   00xxxxx
   000110000001   101xx01
   000110000010   101x001
   000100000100   01xxxxx
   000100000101   1000x00
   000100000110   1000000
   000101000100   01xxxxx
```

000101000101	01xxxxx
000101000110	01xxxxx
000110000100	01xxxxx
000110000101	101xx01
000110000110	01xxxxx
000100001000	xxxxxxx
000100001001	1000x00
000100001010	1000000
000101001000	xxxxxxx
000101001001	01xxxxx
000101001010	01xxxxx
000110001000	xxxxxxx
000110001001	01xxxxx
000110001010	01xxxxx
001000000000	01xxxxx
001000000001	1000x00
001000000010	01xxxxx
001001000000	10x0011
001001000001	01xxxxx
001001000010	1011001
001010000000	00xxxxx
001010000001	1010x01
001010000010	00xxxxx
001000000100	01xxxxx
001000000101	01xxxxx
001000000110	10001x1
001001000100	01xxxxx
001001000101	1011x01
001001000110	10111x1
001010000100	01xxxxx
001010000101	00xxxxx
001010000110	00xxxxx
001000001000	xxxxxxx
001000001001	01xxxxx
001000001010	1000101
001001001000	xxxxxxx
001001001001	01xxxxx
001001001010	1011101
001010001000	xxxxxxx
001010001001	1010x01
001010001010	01xxxxx
000000010000	10000x1
000000010001	01xxxxx
000000010010	01xxxxx
000001010000	00xxxxx
000001010001	10x00x1
000001010010	01xxxxx
000010010000	00xxxxx
000010010001	00xxxxx
000010010010	01xxxxx
000000010100	10000x1
000000010101	01xxxxx

000000010110	10011x0
000001010100	10x00x1
000001010101	01xxxxx
000001010110	10x01x0
000010010100	10000x1
000010010101	01xxxxx
000010010110	10101x0
000000011000	00xxxxx
000000011001	01xxxxx
000000011010	1001100
000001011000	10x00x1
000001011001	01xxxxx
000001011010	10x0100
000010011000	01xxxxx
000010011001	01xxxxx
000010011010	1010100
000100010000	10000x1
000100010001	01xxxxx
000100010010	01xxxxx
000101010000	00xxxxx
000101010001	10x00x1
000101010010	01xxxxx
000110010000	00xxxxx
000110010001	00xxxxx
000110010010	01xxxxx
000100010100	10000x1
000100010101	01xxxxx
000100010110	01xxxxx
000101010100	10x00x1
000101010101	01xxxxx
000101010110	01xxxxx
000110010100	10000x1
000110010101	01xxxxx
000110010110	01xxxxx
000100011000	10000x1
000100011001	01xxxxx
000100011010	01xxxxx
000101011000	01xxxxx
000101011001	01xxxxx
000101011010	01xxxxx
000110011000	101x0x0
000110011001	01xxxxx
000110011010	101x100
001000010000	10000x1
001000010001	01xxxxx
001000010010	01xxxxx
001001010000	10x00x1
001001010001	10x00x1
001001010010	01xxxxx
001010010000	00xxxxx
001010010001	00xxxxx
001010010010	01xxxxx

001000010100	10000x1
001000010101	01xxxxx
001000010110	10001x1
001001010100	10x00x1
001001010101	01xxxxx
001001010110	10111x1
001010010100	10000x1
001010010101	01xxxxx
001010010110	10111x1
001000011000	01xxxxx
001000011001	01xxxxx
001000011010	1000101
001001011000	10110x0
001001011001	01xxxxx
001001011010	01xxxxx
001010011000	00xxxxx
001010011001	01xxxxx
001010011010	01xxxxx
000000100000	1000001
000000100001	01xxxxx
000000100010	xxxxxxx
000001100000	00xxxxx
000001100001	10x0x01
000001100010	xxxxxxx
000010100000	00xxxxx
000010100001	00xxxxx
000010100010	xxxxxxx
000000100100	1000001
000000100101	01xxxxx
000000100110	01xxxxx
000001100100	10x0001
000001100101	01xxxxx
000001100110	10x0x10
000010100100	1000001
000010100101	01xxxxx
000010100110	00xxxxx
000000101000	00xxxxx
000000101001	1001x11
000000101010	01xxxxx
000001101000	10x0001
000001101001	01xxxxx
000001101010	10x0010
000010101000	01xxxxx
000010101001	1000x10
000010101010	00xxxxx
000100100000	1000001
000100100001	01xxxxx
000100100010	xxxxxxx
000101100000	00xxxxx
000101100001	10x0x01
000101100010	xxxxxxx
000110100000	00xxxxx

000110100001	101xx01
000110100010	xxxxxxx
000100100100	1000001
000100100101	01xxxxx
000100100110	01xxxxx
000101100100	10x0001
000101100101	01xxxxx
000101100110	10x0x10
000110100100	01xxxxx
000110100101	01xxxxx
000110100110	00xxxxx
000100101000	1000001
000100101001	01xxxxx
000100101010	01xxxxx
000101101000	01xxxxx
000101101001	10x0x10
000101101010	10x0010
000110101000	101x000
000110101001	00xxxxx
000110101010	00xxxxx
001000100000	1000001
001000100001	01xxxxx
001000100010	xxxxxxx
001001100000	10x0001
001001100001	01xxxxx
001001100010	xxxxxxx
001010100000	01xxxxx
001010100001	1010x01
001010100010	xxxxxxx
001000100100	1000001
001000100101	01xxxxx
001000100110	01xxxxx
001001100100	01xxxxx
001001100101	01xxxxx
001001100110	10x0x10
001010100100	01xxxxx
001010100101	01xxxxx
001010100110	00xxxxx
001000101000	01xxxxx
001000101001	1000x10
001000101010	01xxxxx
001001101000	1011000
001001101001	00xxxxx
001001101010	10x0010
001010101000	00xxxxx
001010101001	00xxxxx
001010101010	00xxxxx
010000000000	100x010
010000000001	100xx00
010000000010	00xxxxx
010001000000	01xxxxx
010001000001	01xxxxx

010001000010	100x000
010010000000	1010011
010010000001	1010x01
010010000010	01xxxxx
010000000100	01xxxxx
010000000101	00xxxxx
010000000110	00xxxxx
010001000100	01xxxxx
010001000101	100xx00
010001000110	00xxxxx
010010000100	01xxxxx
010010000101	01xxxxx
010010000110	10101x1
010000001000	xxxxxxx
010000001001	100xx00
010000001010	00xxxxx
010001001000	xxxxxxx
010001001001	100xx00
010001001010	00xxxxx
010010001000	xxxxxxx
010010001001	01xxxxx
010010001010	1010101
010100000000	100x010
010100000001	100xx00
010100000010	100x000
010101000000	01xxxxx
010101000001	01xxxxx
010101000010	01xxxxx
010110000000	101x011
010110000001	101xx01
010110000010	101x001
010100000100	01xxxxx
010100000101	100xx00
010100000110	100x000
010101000100	01xxxxx
010101000101	01xxxxx
010101000110	01xxxxx
010110000100	01xxxxx
010110000101	101xx01
010110000110	101x001
010100001000	xxxxxxx
010100001001	100xx00
010100001010	100x000
010101001000	01xxxxx
010101001001	01xxxxx
010101001010	01xxxxx
010110001000	xxxxxxx
010110001001	101xx01
010110001010	101x001
011000000000	100x010
011000000001	100xx00
011000000010	01xxxxx

011001000000	01xxxxx
011001000001	01xxxxx
011001000010	1011001
011010000000	1010011
011010000001	1010x01
011010000010	00xxxxx
011000000100	01xxxxx
011000000101	01xxxxx
011000000110	100x1x1
011001000100	01xxxxx
011001000101	1011x01
011001000110	00xxxxx
011010000100	01xxxxx
011010000101	00xxxxx
011010000110	00xxxxx
011000001000	xxxxxxx
011000001001	01xxxxx
011000001010	100x101
011001001000	xxxxxxx
011001001001	1011x01
011001001010	00xxxxx
011010001000	xxxxxxx
011010001001	1010x01
011010001010	00xxxxx
010000010000	01xxxxx
010000010001	100x0x0
010000010010	01xxxxx
010001010000	01xxxxx
010001010001	01xxxxx
010001010010	01xxxxx
010010010000	1010101
010010010001	10100x1
010010010010	01xxxxx
010000010100	01xxxxx
010000010101	01xxxxx
010000010110	10101x0
010001010100	01xxxxx
010001010101	01xxxxx
010001010110	100x1x0
010010010100	01xxxxx
010010010101	01xxxxx
010010010110	10101x0
010000011000	00xxxxx
010000011001	01xxxxx
010000011010	01xxxxx
010001011000	100x0x1
010001011001	01xxxxx
010001011010	100x100
010010011000	01xxxxx
010010011001	01xxxxx
010010011010	1010100
010100010000	100x100

010100010001	100x0x0
010100010010	01xxxxx
010101010000	01xxxxx
010101010001	01xxxxx
010101010010	01xxxxx
010110010000	101x101
010110010001	101x0x1
010110010010	01xxxxx
010100010100	01xxxxx
010100010101	01xxxxx
010100010110	01xxxxx
010101010100	01xxxxx
010101010101	01xxxxx
010101010110	01xxxxx
010110010100	01xxxxx
010110010101	01xxxxx
010110010110	01xxxxx
010100011000	100x0x1
010100011001	01xxxxx
010100011010	01xxxxx
010101011000	01xxxxx
010101011001	01xxxxx
010101011010	01xxxxx
010110011000	101x0x0
010110011001	01xxxxx
010110011010	01xxxxx
011000010000	100x100
011000010001	100x0x0
011000010010	01xxxxx
011001010000	01xxxxx
011001010001	01xxxxx
011001010010	01xxxxx
011010010000	01xxxxx
011010010001	10100x1
011010010010	01xxxxx
011000010100	01xxxxx
011000010101	01xxxxx
011000010110	100x1x1
011001010100	01xxxxx
011001010101	01xxxxx
011001010110	10111x1
011010010100	01xxxxx
011010010101	01xxxxx
011010010110	10111x1
011000011000	01xxxxx
011000011001	01xxxxx
011000011010	100x101
011001011000	10110x0
011001011001	01xxxxx
011001011010	1011101
011010011000	00xxxxx
011010011001	01xxxxx

011010011010	01xxxxx
010000100000	01xxxxx
010000100001	01xxxxx
010000100010	xxxxxxx
010001100000	01xxxxx
010001100001	01xxxxx
010001100010	xxxxxxx
010010100000	1010101
010010100001	1010x01
010010100010	xxxxxxx
010000100100	100x001
010000100101	01xxxxx
010000100110	100xx11
010001100100	01xxxxx
010001100101	01xxxxx
010001100110	01xxxxx
010010100100	01xxxxx
010010100101	01xxxxx
010010100110	1010x10
010000101000	00xxxxx
010000101001	00xxxxx
010000101010	100x011
010001101000	100x001
010001101001	100xx11
010001101010	01xxxxx
010010101000	01xxxxx
010010101001	01xxxxx
010010101010	1010010
010100100000	100x100
010100100001	100xx00
010100100010	xxxxxxx
010101100000	01xxxxx
010101100001	01xxxxx
010101100010	01xxxxx
010110100000	101x101
010110100001	101xx01
010110100010	xxxxxxx
010100100100	01xxxxx
010100100101	01xxxxx
010100100110	100xx11
010101100100	01xxxxx
010101100101	01xxxxx
010101100110	01xxxxx
010110100100	01xxxxx
010110100101	01xxxxx
010110100110	101xx10
010100101000	100x001
010100101001	100xx11
010100101010	100x011
010101101000	01xxxxx
010101101001	01xxxxx
010101101010	01xxxxx

010110101000	101x000
010110101001	101xx10
010110101010	101x010
011000100000	100x100
011000100001	100xx00
011000100010	xxxxxxx
011001100000	01xxxxx
011001100001	01xxxxx
011001100010	xxxxxxx
011010100000	01xxxxx
011010100001	01xxxxx
011010100010	xxxxxxx
011000100100	01xxxxx
011000100101	01xxxxx
011000100110	100xx11
011001100100	01xxxxx
011001100101	01xxxxx
011001100110	01xxxxx
011010100100	1010000
011010100101	01xxxxx
011010100110	1010x10
011000101000	01xxxxx
011000101001	01xxxxx
011000101010	100x011
011001101000	1011000
011001101001	1011x10
011001101010	01xxxxx
011010101000	00xxxxx
011010101001	00xxxxx
011010101010	1010010
100000000000	00xxxxx
100000000001	1000x00
100000000010	00xxxxx
100001000000	1001010
100001000001	01xxxxx
100001000010	10x0000
100010000000	01xxxxx
100010000001	1001x01
100010000010	01xxxxx
100000000100	01xxxxx
100000000101	00xxxxx
100000000110	00xxxxx
100001000100	01xxxxx
100001000101	10x0x00
100001000110	10x01x1
100010000100	01xxxxx
100010000101	01xxxxx
100010000110	10011x1
100000001000	xxxxxxx
100000001001	1000x00
100000001010	01xxxxx
100001001000	xxxxxxx

100001001001	01xxxxx
100001001010	10x0101
100010001000	xxxxxxx
100010001001	01xxxxx
100010001010	1001101
100100000000	00xxxxx
100100000001	1000x00
100100000010	1000000
100101000000	1001010
100101000001	01xxxxx
100101000010	01xxxxx
100110000000	01xxxxx
100110000001	10x1x01
100110000010	10x1001
100100000100	01xxxxx
100100000101	1000x00
100100000110	01xxxxx
100101000100	01xxxxx
100101000101	01xxxxx
100101000110	01xxxxx
100110000100	01xxxxx
100110000101	10x1x01
100110000110	10x1001
100100001000	xxxxxxx
100100001001	01xxxxx
100100001010	01xxxxx
100101001000	xxxxxxx
100101001001	01xxxxx
100101001010	01xxxxx
100110001000	xxxxxxx
100110001001	10x1x01
100110001010	10x1001
101000000000	00xxxxx
101000000001	1000x00
101000000010	01xxxxx
101001000000	10x1010
101001000001	01xxxxx
101001000010	10x1001
101010000000	01xxxxx
101010000001	1001x01
101010000010	00xxxxx
101000000100	01xxxxx
101000000101	01xxxxx
101000000110	10001x1
101001000100	01xxxxx
101001000101	10x1x01
101001000110	00xxxxx
101010000100	01xxxxx
101010000101	00xxxxx
101010000110	00xxxxx
101000001000	xxxxxxx
101000001001	01xxxxx

101000001010	1000101
101001001000	xxxxxxx
101001001001	10x1x01
101001001010	00xxxxx
101010001000	xxxxxxx
101010001001	00xxxxx
101010001010	00xxxxx
100000010000	00xxxxx
100000010001	00xxxxx
100000010010	01xxxxx
100001010000	10010x1
100001010001	10010x0
100001010010	01xxxxx
100010010000	10010x1
100010010001	01xxxxx
100010010010	01xxxxx
100000010100	10010x0
100000010101	01xxxxx
100000010110	10101x0
100001010100	10010x0
100001010101	01xxxxx
100001010110	10x01x0
100010010100	10010x0
100010010101	01xxxxx
100010010110	10011x0
100000011000	00xxxxx
100000011001	01xxxxx
100000011010	01xxxxx
100001011000	10x00x1
100001011001	01xxxxx
100001011010	01xxxxx
100010011000	01xxxxx
100010011001	01xxxxx
100010011010	1001100
100100010000	00xxxxx
100100010001	00xxxxx
100100010010	01xxxxx
100101010000	00xxxxx
100101010001	10010x0
100101010010	01xxxxx
100110010000	10x10x1
100110010001	01xxxxx
100110010010	01xxxxx
100100010100	10010x0
100100010101	01xxxxx
100100010110	01xxxxx
100101010100	10010x0
100101010101	01xxxxx
100101010110	01xxxxx
100110010100	10x10x0
100110010101	01xxxxx
100110010110	01xxxxx

100100011000	10000x1
100100011001	01xxxxx
100100011010	1000101
100101011000	01xxxxx
100101011001	01xxxxx
100101011010	01xxxxx
100110011000	10x10x0
100110011001	01xxxxx
100110011010	01xxxxx
101000010000	00xxxxx
101000010001	00xxxxx
101000010010	01xxxxx
101001010000	00xxxxx
101001010001	10x10x0
101001010010	01xxxxx
101010010000	10010x1
101010010001	01xxxxx
101010010010	01xxxxx
101000010100	10010x0
101000010101	01xxxxx
101000010110	10001x1
101001010100	10x10x0
101001010101	01xxxxx
101001010110	10x11x1
101010010100	10010x0
101010010101	01xxxxx
101010010110	10111x1
101000011000	01xxxxx
101000011001	01xxxxx
101000011010	1000101
101001011000	10x10x0
101001011001	01xxxxx
101001011010	10x1101
101010011000	00xxxxx
101010011001	01xxxxx
101010011010	1011101
100000100000	01xxxxx
100000100001	1000x00
100000100010	xxxxxxx
100001100000	1001001
100001100001	01xxxxx
100001100010	xxxxxxx
100010100000	1001001
100010100001	01xxxxx
100010100010	xxxxxxx
100000100100	01xxxxx
100000100101	01xxxxx
100000100110	00xxxxx
100001100100	01xxxxx
100001100101	01xxxxx
100001100110	1001x11
100010100100	1001000

100010100101	01xxxxx
100010100110	01xxxxx
100000101000	00xxxxx
100000101001	00xxxxx
100000101010	00xxxxx
100001101000	10x0001
100001101001	00xxxxx
100001101010	1001011
100010101000	01xxxxx
100010101001	1001x11
100010101010	01xxxxx
100100100000	00xxxxx
100100100001	1000x00
100100100010	xxxxxxx
100101100000	00xxxxx
100101100001	1001x00
100101100010	xxxxxxx
100110100000	10x1001
100110100001	01xxxxx
100110100010	xxxxxxx
100100100100	01xxxxx
100100100101	01xxxxx
100100100110	00xxxxx
100101100100	1001000
100101100101	01xxxxx
100101100110	1001x11
100110100100	10x1000
100110100101	01xxxxx
100110100110	01xxxxx
100100101000	1000001
100100101001	00xxxxx
100100101010	00xxxxx
100101101000	01xxxxx
100101101001	1001x11
100101101010	1001011
100110101000	10x1000
100110101001	01xxxxx
100110101010	01xxxxx
101000100000	00xxxxx
101000100001	00xxxxx
101000100010	xxxxxxx
101001100000	00xxxxx
101001100001	10x1x00
101001100010	xxxxxxx
101010100000	1001001
101010100001	01xxxxx
101010100010	xxxxxxx
101000100100	1001000
101000100101	01xxxxx
101000100110	00xxxxx
101001100100	10x1000
101001100101	01xxxxx

```
        101001100110   10x1x11
        101010100100   1001000
        101010100101   01xxxxx
        101010100110   01xxxxx
        101000101000   01xxxxx
        101000101001   1001x11
        101000101010   00xxxxx
        101001101000   10x1000
        101001101001   01xxxxx
        101001101010   10x1011
        101010101000   00xxxxx
        101010101001   1011x10
        101010101010   01xxxxx
        11xxxxxxxxxx   xxxxxxx
        xx11xxxxxxxx   xxxxxxx
        xxxx11xxxxxx   xxxxxxx
        xxxxxx11xxxx   xxxxxxx
        xxxxxxxx11xx   xxxxxxx
        xxxxxxxxxx11   xxxxxxx
<END>
```

The above input signals of form: D[X,Y][01,32,03][N,E] are derived from the corner engine stored and/or calculated signals representing difference values a1-a0 (same as DX, DY), a2-a3 (same as DX++, DY++) and a3-a0 (which calculated by the corner engines in successive steps). The N (negative) indicating signals are simply a saved most-significant bit from each of difference signals a1-a0 , a2-a3 and a3-a0. The E (equal to zero) indicating signals are simply formed by performing a logical NOR operation on all bits of difference signals a1-a0 , a2-a3 and a3-a0.

(§6.6.2) DETAILED DESCRIPTION OF REGIS_FILLBIN STATE MACHINES

The Regis boundary-walking unit preferably comprises two boundary-walking state machines: BW1 and BW2. (These have respective re-entrant state points: R1, and R2 in the below state-transition definitions.) State machine BW1 is typically assigned the task of

walking down a left-side line of a geometric shape, from a specified FROM point (F1) toward a specified Termination point (T1). At the same time, state machine BW2 is typically assigned the task of walking down an opposed right-side line of the geometric shape, from a specified FROM point (F2) toward a specified Termination point (T2). The two state machines (BW1 and BW2) step in unison from each Y_D coordinate to a next, $Y_D = Y_D+1$ coordinate.

While state machines BW1 and BW2 are operating, the basic corner-calculating logic is disabled.

The following hardware resources are used in the math engines:

(a) Five 2-bit storage/counter elements for respectively identifying which of coordinates a0 through a3 constitutes: a top point (Ytop), the T1 target point, the T2 target point; and for further keeping count of the number of polygon corner points which the first state machine BW1 has already walked past, CC1 (corner count for edge walker BW1); and for also keeping count of the number of polygon corner points which the second state machine BW2 has already walked past, CC2 (corner count for edge walker BW2). The FROM point identifiers, F1 and F2, are not stored but rather calculated as next indicated.

(b) Three 2-bit combinatorial logic circuits for generating signals according to the following functions: F1=T1+1, F2=T2-1, CS=CC1+CC2. Note that the left walker adds one to define its FROM point while the right walker subtracts one define its FROM point. CS represents a corner stop value. When the sum of corners walked past by the left and right walkers is greater

than 3 (CS > 3), the polygon walk is deemed to be complete.

(c) Three 1-bit flag storing latches: ACW is set true to allow rendering of clockwise-oriented (CW) projected polygons, ACCW is set true to allow rendering of counterclockwise-oriented (CCW) projected polygons, TWD is set true to cause termination of the rendition if a nonallowed direction (CW or CCW) is encountered at the first pixel of the spryte then being rendered.

(d) One 1-bit input line: Indicates whether the first pixel of a spryte is now being rendered.

ASSUMPTIONS:

(1.) Corners are identified in CW order with corner a0 being the top left of the source pixel and a3 being the bottom left.

(2.) Projected polygons with vertically crossed corners (bad, sideways turned bowtie) will not be rendered.

OTHER NOTES:

When below wait states W1 and W2 exist, AND (CE_hold OR CE_YX) is available, the Regis can move on to perform the next line walk step. The LF (Line-filling unit) can take the just-produced X1, X2, and Y values into its own registers and generate the memory write requests in parallel.

Exit from the W1 and W2 states will be simultaneous. Only one of the controllers needs to initiate the 'Y=Y+1' event. Both walkers (BW1 and BW2) step to the next Y position together.

Each machine, BW1 and BW2, waits at its respective W1/W2 state until the other catches up.

The CC1 and CC2 counters permit simultaneous bumps of T1 and T2.

If, at entry to Regis, Ytop = T1 (or T2), we actually set T1 to T1-1 (or T2 to T2+1), and we set CC1 (CC2) to 1 vice 0. Use the top detector logic to determine this condition. Use pre-top, pre-t1, pre-t2.
ADDITIONAL NOTE:

We reverse the subtraction order in the DYn calculation. This results in -DYn as the answer which can then be directly deposited into XnFRAC and the -DYn holder.

Xnslope is used to determine polarity of XnADD and +/- of DXn. If Xnslope is negative, then XnADD=-1 and DXn is subtracted in the below R1/R2 states. The -DYn and the Xnslope operations make it unnecessary to check for the DYn=1 shortcut.

(§6.6.2.1) REGIS-FILLBIN STATES

Initial entry into the Regis operation is at left and right start states, S1 and S2. Thereafter, re-entry states R1 and R2 are used.

The below state transition tables are to be read as left to right as a series of embedded IF clauses with the THEN operation at the right followed by a comments column. CMP indicates a comparison of two arguments. EQ defines the case where the argument variables are equal. NE defines the case where the argument variables are not equal. "X" indicates a don't care condition or a do-nothing action.

Thus, when machine BW1 is in the below R1 state, and a comparison of the current Y coordinate against the Y of the specified TO point shows an equality condition, and if corner sum CS= 0 OR 1, then the next state is S1 and the value of the new X1(fraction) is irrelevant. @

STATE: R1 (Re-entry point for the BW state machine)

IF: CMP of Y, Y(T1)	AND IF: CS=	ALWAYS DO: X1FRAC=X1FRAC+/-DX1 THEN CHECK FOR:	AND DO ON COND:	(COMMENTS)
EQ	2 or 3	x	TERMINATE(R)	(last corner)
EQ	0 or 1	x	GOTO S1 [T1=T1-1], [CC1=CC1+1]	(Get Next Corner)
NE	x	new X1FRAC<0	GOTO W1	(Still on same edge, end of scan line)
NE	x	new X1FRAC>=0	GOTO E1	(Still on same edge, walk the edge)

STATE: S1 (Start)

(Get Breshenham deltas, do simultaneous with the CMPs)
also do X1FRAC = Y(F1) - Y(T1); and do X1=X(F1);

Y, Y(T1);	Check CS:		
EQ	2 or 3	TERMINATE(R)	(New Corner, last corner)
EQ	0 or 1	GOTO S1	(New Corner, try again) DO[T1=T1-1], CC1=CC1+1
NE	x	GOTO W1	(New Corner, use deltas)

STATE: W1 (Wait)

Wait here until W2 exists AND LF is available.

Then provide Y and X1 and X2 to LF (Line-filler).

If Y is neg, Don't initiate LF.

THE BELOW ACTIONS ARE PERFORMED BY THE LINE-FILLER MATH SECTION

SUB X1-X2:	ACW=	ACCW=	
X1 = X2	x	x	XL==xDon't initiate LF;
X1 > X2	x	0	XL==X2Don't initiate LF; if NE, set 'wrong direction' flag
X1 > X2	x	1	XL==X2initiate LF
X1 < X2	0	x	XL==X1Don't initiate LF, set 'wrong direction' flag
X1 < X2	1	x	XL==X1initiate LF

Upon exit from Line-filler:
Y=Y+1; DX1 = X(T1) - X(F1); Save Sign of DX1 as X1 slope;
GOTO R1.

STATE: E1 (Edge walk)

X1FRAC = X1FRAC -DY1; X1 = X1 +/- X1ADD (Walk the edge)

X1FRAC < 0	GOTO W1	(End of Scan Line)
X1FRAC >=0	GOTO E1	(Walk more)

Super-clipping:

At any state, if Y>CLIP, TERMINATE(R) (do not initiate LF). {Skip is performed by LF}
At any state, if (WDF and not-crossed), TERMINATE(R) (do not initiate LF).

At any state, if (TWD and WDF and 1st pixel), TERMINATE(S) (do not initiate LF).

STATE: R2 (Re-entry)

CMP Y,Y(T2);	Check CS:	X2FRAC=X2FRAC+/-DX2		
EQ	2 or 3	x	TERMINATE(R)	(last corner)
EQ	0 or 1	x	GOTO S2	(Get Next Corner) DO[T2=T2+1], [CC2=CC2+1]
NE	x	new X2FRAC<0	GOTO W2	(Still on same edge, end of scan line)
NE	x	new X2FRAC>=0	GOTO E2	(Still on same edge, walk the edge)

STATE: S2 (Start)

(get bresh deltas, do simultaneous with the CMPs)

CMP Y,Y(T2);	Check CS:	also do X2FRAC = Y(F2) - Y(T2); and do X2=X(F2);	
EQ	2 or 3	TERMINATE(R)	(New Corner, last corner)
EQ	0 or 1	GOTO S2	(New Corner, try again) DO[T2=T2+1], CC2=CC2+1
NE	x	GOTO W2	(New Corner, use deltas)

STATE: W2 (Wait)

Wait here until W1 exists AND LF is available.
Then provide Y and X1 and X2 to LF.
If Y is neg, Don't initiate LF.

SUB X1-X2;	ACW:	ACCW:		
X1 = X2	x	x	XL==x	Don't initiate LF;
X1 > X2	x	0	XL==X2	Don't initiate LF; if NE, set 'wrong direction' flag
X1 > X2	x	1	XL==X2	initiate LF
X1 < X2	0	x	XL==X1	Don't initiate LF, set 'wrong direction' flag
X1 < X2	1	x	XL==X1	initiate LF

Upon exit,
Y=Y+1. copy X2FRAC TO -DY2 register; DX2 = X(T2) - X(F2); Save Sign of DX2 as X2slope;
GOTO R2.

STATE: E2 (Edge walk)

X2FRAC = X2FRAC -DY2; X2 = X2 +\- X2ADD (Walk the edge)

X2FRAC < 0	GOTO W2	(End of Scan Line)
X2FRAC >=0	GOTO E2	(Walk more)

***END OF REGISFILL-BIN STATE TRANSITION TABLE

§7.0 SCoB contents and other Miscellaneous Tables

TABLE 1.0

SCoB Data Structure

Number of Bits	Name	Description
32	FLAGS	Assorted flags. This is the first word read by the spryte-rendering hardware!! (The flag bits are detailed in below TABLE 1.1)
24	NEXTPTR	Address of next SCoB to process. (Format is absolute or relative.) Spryte rendition takes place by stepping through a linked list having one or more SCoB's. After a first source spryte is mapped to, and painted onto a destination grid area defined by its SCoB, the spryte-rendering engine processes the next SCoB, if any, and renders its source spryte onto its designated destination surface. The linked list can be circular if desired so that the process is repeated iteratively.
24	SOURCEPTR	Address of image data that is to be rendered as a spryte.
24	PIPPTR	Address of Pen Index Palette (PIP) that is to be loaded into the IPS unit (103 in Fig. 1A).
32	XPOS	Horizontal position (in 640-max pixels format) in the destination grid of the upper left corner of the to-be rendered SPRYTE, including 16 bits which represent a fraction (noninteger) position-defining portion.
32	YPOS	Vertical position (in 480-max pixels format) in the destination grid of the upper left corner of the to-be rendered SPRYTE, including 16-bit fraction part.
32	DX	Horizontal position increment from mapped first corner of a source pixel to mapped second corner of a source pixel when scanning and re-mapping the first spryte row onto the destination grid (format is two 16-bit half-words which are expressed in integer.fraction form as: 12.20).
32	DY	Vertical position increment from mapped first corner of a source pixel to mapped third corner of a source pixel when scanning and re-mapping the first spryte row (12.20).
32	LINEDX	Horizontal position increment in destination grid from top left corner of 1st mapped spryte row to top left corner of 2nd mapped spryte row (16.16).

32	LINEDY	Vertical position increment from 1st line to 2nd (16.16).
32	DDX	Increment to DX for each successive row after the 1st row of the spryte being rendered (12.20).
32	DDY	Increment to DY for each successive line processed (12.20) after 1st line.
32	PPMPC	PPMP control word (two halfwords: 16, 16).
32	PRE0	Possible 1st preamble word.
32	PRE1	Possible 2nd preamble word.

----------<END OF TABLE 1.0>----------

TABLE 1.1

FLAGS Data Structure

FLAGS: (These flag bits control specific fetching and rendering operations of the spryte-rendering engine. The data specific control bits are found in the preamble word of the source data.)

Bits	Name	Description
B31 =	SKIP	If set, skip this SCoB.
B30 =	LAST	If set, this is the last SCoB to process.
B29 =	NPABS	1=Absolute, 0=Relative address for NEXTPTR.
B28 =	SPABS	1=Absolute, 0=Relative for SOURCEPTR.
B27 =	PPABS	1=Absolute, 0=Relative for PIPPTR.
B26 =	LDSIZE	Load 4 words of size and slope data. (DX, DY, LINEDX, LINEDY).
B25 =	LDPRS	Load 2 words of perspective (skew control) data. (DDX, DDY).
B24 =	LDPPMP	Load new PPMP control word (PPMPC) into PPMP control registers.
B23 =	LDPIP	Load new PIP data into PIP.
B22 =	SCoBPRE	Preamble location. 1=At end of SCoB, 0=At start of source data.
B21 =	YOXY	Translate the XY values to a system memory address value and write the corresponding data to the hardware.
B20:B19 =	xx	Reserved.
B18 =	ACW	Allow rendering of a CW (clock-wise) oriented destination pixel.
B17 =	ACCW	Allow rendering of a CCW oriented destination pixel.
B16 =	TWD	Terminate rendition of this Spryte if wrong direction is encountered (CW-CCW).
B15 =	LCE	Lock the operations of the 2 corner-calculating engines together. (at H change).
B14 =	ACE	Allow the second corner-calculating engine to function.
B13 =	ASC	Allow Super-Clipping (the local switch is ANDed with ASCALL).

B12 =	MARIA	1=disable full-math region-fill action and use only the faster Munkee decisions as instructions to the destination line-filler.
B11 =	PXOR	1=set PPMP XOR mode. (XOR the A and B sources while disabling adder.)
B10 =	USEAV	1=use the "AV" bits in PPMPC to control PPMP math functions.
B9 =	PACKED	Primary source spryte type, 1=packed, 0=totally literal. (Secondary source spryte is always totally literal.)
B8:B7 =	DOVER	D-Mode override. 00=use the D-bit generated by the IPS unit to select the output of CMUX , 01=reserved, 10=select the A input of the CMUX, 11=select the B input of the CMUX.
B6 =	PIPPOS	Use PIP generated bits as the subposition bits (B0 & B15 of the Output-PEN signal) instead of the SCoB selection made below by B15POS and B0POS.
B5 =	BGND	1=background SPRYTE type.
B4 =	NOBLK	1=no black SPRYTE type.
B3:B0 =	PIPA	PIP address bits, these are used to pad the 5-bit wide PIP address input signal when BPP (Bits Per Pixel) output of unpacker is less than 5 bits wide.

-------<END OF TABLE 1.1>

TABLE 1.2

PPMC Data Structure

The PPMP control word has two 16-bit wide halves. One half is used when the CMUX select control bit=0, the other half is used when the CMUX select control bit=1. Only the upper half is described here. The lower half has identical structure.

Bits	Name	Description
B31=	S1	Select 1st multiplier input signal. 0=use IPN (Source A), 1=use cFB data (Source B).
B30:B29=	MS	Select 2nd multiplier input signal. 0=MxF (source is SCoB), 1=MUL (source is IPS), 2=IPNM (source is IPS), 3=xx (multiply by default value, 1 or 0).
B28:B26=	MxF	Multiply Factor. 0->7 means multiply by 1->8. (only used if MS=0).
B25:B24=	Dv1	First divide-by Scaling-factor. 1=divide by 2, 2=divide by 4, 3=÷8, 0=divide by 16.
B23:B22=	S2	Selector of secondary input signal. 0="apply 0 value to Adder port B", 1=use AV word (from SCoB), 2=use cFBD (Source B), 3=use IPS output (Source A).

B21:B17=	AV	Adder Value. 5 bit value to be added if S2=1. This 5-bit signal is also used as a math control word if USEAV=1.
B16=	Dv2	Post addition, 2nd divider. 0=divide by 1, 1=divide by 2.

-------<END OF TABLE 1.2>

TABLE 1.3

Secondary AV Bit Functions

Aside from providing an "add value", the AV bits serve a secondary function as follows when USEAV=1.

Bits		Function
AV0	=	Invert the output of the second divider in the PMPP and set the carry-in of the adder.
AV1	=	Enable the sign-extend function for the signals flowing down the second math side of the PMPP (Post possible XOR).
AV2	=	Disable the wrap-limiter function. (Use the 5 LSB's of the 8-bit adder output and ignore possibility that it wrapped above decimal 31 or below decimal zero.)
AV3:AV4	=	Select second side divider value as: 00=divide by 1, 01=divide by 2, 10=divide by 4, 11=divide by __ (reserved).

-------<END OF TABLE 1.3>

TABLE 1.4

Engine Control Data Structure

There is also a general SPRYTE-rendering engine control word. It is loaded only by the CPU. Its bits are:

Bits	Name	Description
B31:B30=	B15POS	B15 oPEN selector for output of PMPP. (This bit can function as a subposition defining bit that is used by the pre-display interpolater.) 0=0, 1=1, 2=xx, 3=same as Source data
B29:B28=	B0POS	B0 oPEN selector for output of PMPP. (This bit can also function as a subposition defining bit that is used by the pre-display interpolater.)

		0=0, 1=1, 2=PPMP math, 3=same as Source data
B27=	SWAPHV	1=Swap the H and V subpositions prior to their entry into the PPMP
B26=	ASCALL	1=Allow super clipping function (master enable switch)
B25=	xx	Reserved
B24=	CFBDSUB	1=use the H and V subposition bits of the cFB data in place of (vice) the SPRYTE source values when the cFB data is selected as a PPMP source. (Note: CFBDsel=(S1=1) OR (S2=2).)
B23:B22=	CFBDLSB	cFBD PPMP Blue LSB source. 0=0, 1=cFBD[B0], 2=cFBD[B4], 3=x
B21:B20=	IPNLSB	IPN PPMP Blue LSB source. 0=0, 1=IPN[B0], 2=IPN[B4], 3=x

NOTE#1: When 'relative' has been specified in the flags for NEXTPTR, SOURCEPTR, or PIPPTR, the value that should (must) be placed in the SCoB is the word distance from the address in RAM that has the relative value in it to the address in RAM that is desired to be the new address MINUS FOUR.

REL= Target - PC - 4

NOTE#2: The B0POS value of '2' is the only setting that uses PPMP math to control the B0 bit in the actually output oPEN signal. When this setting is chosen, the Blue LSB will also be included in the input parameters of the black detector.

NOTE#3: SCoB Loading Process.

The first 6 words of a SCoB (FLAGS, NEXTPTR, SOURCEPTR, PIPPTR, XPOS, YPOS) are always read out of VRAM by the DMA engine (inside the memory address manipulator chip (MAMC) and generally placed into corresponding hardware registers. The last 2 words (XPOS, YPOS) are always read as part of the general 6-word SCoB read, but they are not written to the hardware registers if this Spryte is to be skipped (SKIP=1) or YOXY is set to 0.

The remainder of the SCoB words are optionally downloaded into the hardware. The hardware registers are left in their pre-existing states if the corresponding SCoB words are not downloaded.

The optionally downloaded 7 SCoB words are be divided into the folowing successive groups: (a) Size and slope data (4 words), (b) Prespective or skew-control data (2 words: DDX & DDY), and finally (c) the PPMP control word (1 word). These 7 SCoB words are read out of VRAM as one burst regardless of which specific ones have been requested.

After the nonoptional and/or optional SCoB words are read out of VRAM, the preamble word or words are read. This is not optional.

The last group of words read (optionally) out of VRAM is the PIP data (max 16 words). Note that, if read, the PIP is tha LAST element of the SCoB that is read. The length of the PIP read wil vary depending on the BPP setting of the current Spryte. If LINEAR=1, the PIP can still be loaded even though it is not used.@

-------<END OF TABLE 1.4>

TABLE 1.5 Spryte Image Data Formats

There are 2 basic formats of Spryte image data, Totally literal format and non-totally literal format. There are sub-groups within each basic format. In non-totally literal Sprytes, the image data consists of groups of words that represent source scan lines of data. In totally literal Sprytes, the image data consists of purely image data (no intermingled control functions).

Non-totally literal Sprytes require 1 word preamble. Totally literal Sprytes require 2 words of preamble. These preamble words may be located at the end of the SCoB words (but before the PIP) or at the start of the image data. The normal location for these words is at the start of the image data, but totally literal Sprytes that are in frame buffer format will want to not damage their rectangular space with 2 extra words. We are using a stand alone bit for the selection just to keep is all simple.

Non-totally literal Sprytes can be compacted to save both memory space and rendering time. Each source scan line of data has its horizontal word size specified as part of the data.

Totally literal Sprytes have a rectangular format that is specified in the preamble of the data.

-------<END OF TABLE 1.5>

TABLE 1.6 Spryte Data Preamble Words

First preamble word:

The first preamble word for ALL SPRYTES is the data structure preamble. It contains the data specific control bits for the source data.

B31->B28	=	PRESERVED for future use. The current hardware will ignore writes to these bits and return zero on read.	
B27->B21	=	Reserved, set to 0.	
B20	=	PACKED	This is identical to the PACKED bit in the SCoB
B19->B16	=	Reserved, set to 0.	
B15->B6	=	VCNT	Vertical number of source data lines in this image data -1. (10 bits)
B5	=	Reserved	set to 0.
B4	=	LINEAR	0=use PIP for generating IPN output of IPS unit, 1=use PIN for outputting IPN
B3	=	REP8	1=replicate the bits in the linear 8 Spryte, 0=fill with 0
B2->B0	=	BPP	bits/pixel, pixel type

PRESERVED bits are currently ignored but someday may be used. It is required that the software correctly create these bits. The current hardware will not check that you did it right.

VCNT is loaded into a hardware counter in the Spryte requestor that is decremented at the end of the fetching of each source scan line of data. When

the count is at -1, there are no more source lines of data in the object. Note that Spryte processing does not end here, this is merely one of the events that is required to end a Spryte. VCNT = line count -1.

An initial value of -1 for VCNT will cause a REAL BIG Spryte to be fetched. Sorry, there is no 'zero line count' value.

The LINEAR bit only applies when the BPP type is 8 bits per pixel or 16 bits per pixel. In those cases, there are enough PIN bits to provide a 15 bit IPN without using the PIP. Since the PIN bits are spread linearly across the IPN, and it will result in a linear translation from PIN to IPN, the mode is called 'LINEAR'. The only 2 valid uses are for LINEAR 8 and LINEAR 16 (as opposed to 'normal' 8 or 16).

The REP8 bit only has an effect in the 8 bit source data size.

The BPP control bits decode as follows:

BPP	Data Size	PIP DMA Size	IPN Trans Bits	D-bit	R-Mul	G-Mul	B-Mul
0 =	reserved	4 PIP words	Reserved,(?)	(?)	0	0	0
1 =	1 bit	4 PIP words	PIN[0]	PIP[15]	0	0	0
2 =	2 bit	4 PIP words	PIN[1..0]	PIP[15]	0	0	0
3 =	4 bit	8 PIP words	PIN[3..0]	PIP[15]	0	0	0
4 =	6 bit	16 PIP words	PIN[5..0]	PIN[5]	0	0	0
5 =	8 bit	16 PIP words	PIN[7..0]	PIP[15]	PIN[7..5]	PIN[7..5]	PIN[7..5]
6 =	16 bit	16 PIP words	PIN[14..0]	PIN[15]	PIN[13..11]	PIN10..8]	PIN[7..5]
7 =	reserved	16 PIP words	Reserved,(?)	(?)	0	0	0

Second preamble word:

If the PACKED bit (in the SCoB) is '0', then the source data is totally literal. For totally literal Sprytes, there is a second preamble word. It contains the horizontal pixel count for each line of the source data and the word offset from one line of source data to the next. It also contains the other special bits needed for totally literal Sprytes. Note that these bits are only valid while the totally literal Spryte is being rendered. These bits are not used ...GATED AWAY... when the current Spryte is not totally literal.

B31->B24 = WOFFSET(8). Word offset from one line of data to the next (-2) (8 bits).
bits 23->16 of offset are set to 0.

B25->B16 = WOFFSET(10). Word offset from one line of data to the next (-2) (10 bits).
bits 31->26 of offset are set to 0.

B15 = Reserved, set to 0.

B14 = NOSWAP 1=disable the SWAPHV bit from the general Spryte control word.

B13->B12 = TLLSBIPN PPMP blue LSB source. 0=0, 1=IPN[0], 2=IPN[4], 3=IPN[5].

B11 = LRFORM Left/right format.

B10->B0 = TLHPCNT Horizontal pixel count (-1) (11 bits).

The TLLSB bits perform the same function that the IPNLSB bits perform in normal Sprytes.

If LRFORM=1, the source data has the frame buffer format of the screen as a source format. Vertically adjacent pixels in the rectangular display

space are horizontally adjacent in the 2 halves of a memory word. This is useful for 16 BPP totally literal. The unpacker will disable the 'B' FIFO data requests and alternately place pixels from the source into both FIFOs. Left 16 bits go to 'A' FIFO, right 16 bits go to 'B' FIFO. The data requests for 'A' FIFO will be made in a request 'pair' to insure the reduction of page breaks and '6 tick latencies'. The hardware will lock the corner engines (regardless of the LCE bit).

TLHPCNT is the number of pixels in the horizontal dimension (-1). This is the number of pixels that will be attempted to be rendered for each horizontal line of the Spryte. This value is used by the data unpacker. A '0' in the value will attempt 1 pixel. A '-1' in the value will attempt many pixels. There is no 'zero pixel count' value.

WOFFSET is the offset in words of memory from the start of one line of data to the start of the next line (-2). If the BPP for this Spryte is 8 or 16, use WOFFSET(10), else use WOFFSET(8). This number is a zero for the minimum sized Spryte (2 words).

By arranging WOFFSET and TLHPCNT correctly, you can extract a rectangular area of data our of a larger sized rectangular area of data.

The DMA engine will also use WOFFSET as the length value in the normal data fetch process. If WOFFSET and TLHPCNT are set badly, WOFFSET may expire first and the DMA engine will not cope properly.

-------<END OF TABLE 1.6>

TABLE 1.7 Spryte Packed Data Formats

Offset

The first one or two bytes are the word offset from the start of this line of source data to the start of the next line of data (-2). In Sprytes with BPP of 6 or less, only 1 byte (bits 31->16) of offset are used. However, the actual offset has a maximum size of 10 bits. The rest of the bits in the 2 bytes are set to 0. 10 bits of word offset is 2048 pixels at 16BPP. 8 bits of word offset at 6 BPP is 1365 pixels. The requirement is 1280 pixels.

This offset is used by the DMA controller to both calculate the start of the next line of data (by adding it to the start of the current line), and to set the maximum length (by subtracting 1 and placing it in the DMA length register) of the current DMA transfer.

This offset value (1 or 2 bytes) is not used by the data unpacker. It will arrive at the data unpacker at the start of each line of a packed Spryte and must be discarded.

Control byte and PIN data:

The next data after the offset is comprised of 1 control byte and 0 or more bits of PIN data. The number of bits used for each PIN is specified by BPP.

The control byte consists of a 2 bit code and a 6 bit count:

00 'xxxxxx'= end of line, xxxxxx need not be present
01 'count'= literal PINs for 'count+1'
10 'count'= Defined 'transparent' for 'count+1'
11 'count'= packed 'PIN' for 'count+1'

The 'transparent' definition will actually output a 'transparent' bit from the unpacker. This will cause the remainder of the pixel processing pipe to ignore this pixel. For safety purposes, we will set the data value at this time to be zero for possible use by the IPS for the D-Mode selector. It probably will not be seen by the IPS, but we are not sure.

-------<END OF TABLE 1.7>

TABLE 1.8 Spryte Render Stopping and Starting

The Spryte rendering process is started from the CPU by writing a meaningless value to the 'SPRSTRT' address in the memory address manipulator chip (MAMC). This sets the 'Spryte-ON' flipflop. Writing to SPRSTART while the 'Spryte-ON' flipflop is already on will have no effect at all. BUT DON'T DO IT. The race condition of Spryte just now ending and you just now writing is not preventable.

A start is always a 'cold' start. Previously running Spryte things have been canceled. It is required that the software has setup the data in memory and the first SCoB address in the DMA stack correctly. The SYSTEM WILL CRASH if the Spryte data or the first SCoB address are faulty.

The Spryte process is stopped cold when the CPU writes to 'SPRSTOP'. All intermediate states of the engines have been RESET. The data pointers are now WRONG. Spryte processing is dead. Except, of course, for the SPRPAUS flipflop.

When the Spryte engine is completely finished with all of its functions including sending the last of the rendered data to memory, it will reset its Spryte-ON flipflop.

Spryte processing is not overlapped in the hardware. One Spryte is allowed to totally complete prior to starting the next Spryte in the list. This total completion includes the processing of trailing transparent pixels and the outputting of the last data values to memory.

The CPU can request that the Spryte rendering engine stop operating at the end of the current Spryte but not reset itself by writing to 'SPRPAUS'. This will set a 'SPRPAUS' flipflop that will cause the Spryte rendering engine to set its 'PAUSE' flipflop at the end of the current Spryte. When the PAUSE flipflop is set by this mechanism (not by other means), the SPRPAUS flipflop will be cleared. The SPRPAUS event is only a one shot thing. Obviously this can only be done if the CPU already has access to the system bus. SPRPAUS can be set by the CPU even if the Spryte engine is off.

When an interrupt is present, the PAUSE flipflop is held in a 'set' condition. This will allow the Spryte rendering engine to pause cleanly and allow the CPU to notice the interrupt.

The CPU can reset the PAUSE flipflop by writing to 'SPRCNTU'. If the Spryte engine was on but paused it will now continue. If not, it won't. This is also when the CPU might avail itself of the SPRPAUS function.

Once a render process has started, the CPU is effectively asleep. Nothing has actually been done to the CPU, it just can't get any cycles on the system bus. When the CPU finally does get some cycles, it needs to decide why and service the situation. The reasons it got a cycle are that an interrupt is

present, or that the Spryte engine is paused or done, or somehow, the bus has temporarily become available. After servicing the interrupt or CPU requested Spryte pause, it will be up to the CPU to then decide whether or not to continue the render process. The CPU can determine the status of the render process by reading the appropriate status bit(s).

If the Spryte rendering engine is on but not paused, the CPU should just sit in its status bit check loop. This is a temporary situation and will soon change.

If the engine is on and paused, the CPU could just issue the SPRCNTU. If there are no Sprytes left, the Spryte engine will turn itself off and no harm done.

If on and paused and at end of current Spryte, then the CPU can use the Spryte rendering engine for its purpose without fear of damaging a Spryte in process.

-------<END OF TABLE 1.8>

The above disclosure is to be taken as illustrative of the invention, not as limiting its scope or spirit. Numerous modifications and variations will become apparent to those skilled in the art after studying the above disclosure.

Given the above disclosure of general concepts and specific embodiments, the scope of protection sought is to be defined by the claims appended hereto.

CLAIMS

What is claimed is:

[Note: Bracketed bold text is provided in the below claims as an aid for readability and for finding corresponding support in the specification. The bracketed text is not intended to add any limitation whatsoever to the claims and should be deleted in any legal interpretation of the claims.]

1. An image rendering system **[400]** for producing displayable image data **[410d]** representing a mapping of source pixels **[110]** onto a destination grid **[120]**, said system **[400]** comprising:

color/shade-mapping means **[401]** for receiving one or more source color/shade defining signals **[411,412]** that define colors and/or shades of a respective one or more source pixels **[110]** and for converting said source color/shade defining signals **[411,412]** into a corresponding one or more destination color/shade defining signals **[418]** that define colors and/or shades which are potentially assignable to a respective one or more destination pixels **[PPP]** in the destination grid **[120]**; and

destination-point mapping means **[402]** for receiving point-map control signals **[415.2]** defining a source-to-destination point-to-point mapping function **[300]** and for producing corner coordinate signals **[424]** representing destination coordinates of corner points **[a0-a3]** of projections **[211-251]** of the source pixels **[110]** onto the destination grid **[120]** and for further producing paint-request signals **[428]** in accordance with the number of destination pixels **[PPP]**, if any, that are effectively covered by said projections **[211-251]** of the source pixels;

where, in a general case, the destination-point mapping means [402] requires a relatively long, general time period to determine the number of destination pixels [PPP], if any, that are effectively covered by said projections [211-251] of the source pixels; and

where the destination-point mapping means [402] includes:

early-termination means [427a,427b] for recognizing ahead of time conditions [429a] where the destination-point mapping means [402] will ultimately not produce paint-request signals [428] if allowed to follow its general course of making such a determinations over the general time period and for terminating operations within the destination-point mapping means [402] that relate to the paint-request signals [428] that will not be produced in a time period substantially shorter than the general time period. [LINE NUMBERS OFF]

2. An image rendering system [400] according to Claim 1

wherein the color/shade-mapping means [401] includes:

transparency-code generating means [414] for selectively converting one or more of said source color/shade defining signals [411,412] into a corresponding one or more transparency defining signals [T-bit] that, when active, prevent assignment of new colors and/or shades to a respective one or more destination pixels [PPP] in the destination grid [120]; and

wherein said early-termination means [427a,427b] responds to active transparency defining signals [T-bit] by recognizing there active state as being a condition [429a] where the destination-point mapping means [402]

will ultimately not produce paint-request signals [428].

3. An image rendering system [400] according to Claim 1 further comprising:

light image producing means [460] for converting said displayable image data [410d] into a displayed light image.

4. A point-to-point source-to-destination mapping system [402] comprising:

one or more corner engines [422,702a,702b] each for receiving point-map control signals [415.2], where the point-map control signals [415.2] define a mapping [300] of one or more source pixels [110] onto a destination grid [120] composed of plural destination pixels [P,P,P], the one or more corner engines [422] generating corner signals [424] representing the coordinates [a0-a3] of source pixel points mapped onto the destination grid [120] in accordance with the defined mapping [300];

one or more boundary locating units [426,626] for receiving the corner signals [424] generated by the one or more corner engines [422] and for generating therefrom boundary signals [YXX] representing opposed points on opposed boundaries of the mappings [125,127] of the source pixels; and

one or more border estimating units [425,625] for receiving the corner signals [424] generated by the one or more corner engines [422] and for generating therefrom paint-decision signals [427a,429a,b] indicative of whether one, none or plural destination pixels [P,P,P] are effectively bound within the borders of each mapping [125,127] of the source pixels.

5. A point-to-point source-to-destination mapping system [602] according to Claim 4

wherein said corner engines include a first corner engine [702a] and a second corner engine [702b];

wherein said source pixels [110] include a first source pixel ["a"] having respective corners a0, a1, a2, a3 and a second source pixel ["q"] having respective corners q0, q1, q2, q3, said corner q0 being coincident with corner a3;

wherein the first corner engine [702a] receives an initializing signal [801] defining the destination coordinates of mapped corner a0 and the first corner engine [702a] responsively generates therefrom [802] a corner-a3 defining signal representing the destination coordinates of mapped corner a3; and

wherein the second corner engine [702b] receives [803] the corner-a3 defining signal from the first corner engine [702a] and the second corner engine [702b] responsively generates therefrom [804] a corner-q3 defining signal representing the destination coordinates of mapped corner q3.

6. A point-to-point source-to-destination mapping system [602] according to Claim 5

wherein after the first corner engine [702a] generates [802] said corner-a3 defining signal, the first corner engine [702a] proceeds to independently generate [810] a corner-a1 defining signal representing the destination coordinates of mapped corner a1 and to independently generate [812] a corner-a2 defining signal representing the destination coordinates of mapped corner a2; and

wherein after the second corner engine [702b] generates [804] said corner-q3 defining signal, the second corner engine [702b] proceeds to independently

generate a corner-q1 defining signal representing the destination coordinates of mapped corner q1 and to independently generate a corner-q2 defining signal representing the destination coordinates of mapped corner q2.

7. A point-to-point source-to-destination mapping system [602] according to Claim 6

wherein said source pixels [110] include a third source pixel ["w"] having respective corners w0, w1, w2, w3 and said corner q3 is coincident with corner w0;

wherein after the first corner engine [702a] generates [802] said corner-a3 defining signal, and after the second corner engine [702b] generates [804] said corner-q3 defining signal, the first and second corner engines [702a,702b] proceed to independently map corner points of respective first and second source rows [A and B], and

wherein whichever of the first and second corner engines [702a,702b] ends its task first, that first-finished corner engine receives [805] the corner-q3 defining signal produced by the second corner engine [702b] and that first-finished corner engine responsively generates therefrom [806] a corner-w3 defining signal representing the destination coordinates of mapped corner w3.

8. A spryte rendering system [400] comrising:

a plurality of corner engines [422] each for generating corner signals [424] representing polygon corner coordinates [a0-a3] whose values are calculated from an input data set [415.2], where the input data set includes position data (XPOS, YPOS) defining the destination coordinates [Xa0,Ya0] of a top-left corner

(a0) of a first source pixel [a], line delta data (LDX, LDY) defining the destination coordinates [Xa3,Ya3] of a bottom left corner (a3) of the first source pixel, and row delta data (DX, DY, DDX, DDY) defining the destination coordinates [Xa1,Ya1,Xa2,Ya2] of corners points (a1, a2) which respectively succeed from the defined top-left corner (a0) of the first pixel and from the defined bottom-left corner (a3) of the first pixel; and

a shared storage unit [422a,615] having locations that are accessible by the plurality of corner engines [422,702a,702b];

where the corner engines exchange results for shared destination points [a3,q3] by way of the shared storage unit [422a]; and

where the shared results include one or more of the following items: (1) calculated coordinates for the bottom left corner (a3) of the first pixel in each successive source row, and (2) progressively increased or decreased delta values (DX++ = DX + DDX) calculated by a first of the corner engines and useable by a second of the corner engines.

9. A spryte rendering system [400,600] comprising:

corner engine means [422,702a,702b] for generating corner point signals [424] representing destination coordinates of polygons [125-128] projected onto a destination grid [120], where the destination grid [120] includes an array of destination pixels [P,P,P];

one or more border-point locators (Regis circuits) [626a,626b] for identifying opposing points on opposed left and right borders of each projected polygon and for determining which, if any, destination pixels are effectivey bounded by the opposing points to warrant

painting cf those destination pixels by a color code correspondingly developed for the associated, projected polygon; and

one or more fast-decision circuits (Munkee circuits) [425,625a,625b], each associated with one of the border-point locator (Regis circuits), for identifying corner mapping conditions in which only one or none of the destination pixels will be painted by a corresponding color code [418,618] and for instructing the corresponding border-point locator (Regis circuit) to abort its operations and for further providing a one-pixel paint command in the case where only one destination pixel is to be painted.

10. A method [800] for mapping and rendering adjacent rows of a source image [110] onto a destination grid [120], where the source image [110] includes in a first row [A] thereof, a first source pixel [a] having zeroth through third corner points, a0, a1, a2, a3, where the source image [110] includes in a second row [B] thereof, a second source pixel [q] having zeroth through third corner points, q0, q1, q2, q3, and where corner point q0 is coincident with corner point a3, said method comprising the steps of:

defining [801] corner-a0 coordinates [XPOS,YPOS] which map the first corner a0 of the first source pixel onto the destination grid [120];

from the defined corner-a0 coordinates, generating [802] corner-a3 signals representing a mapping of the third corner a3 of the first source pixel onto the destination grid [120]; and

from the generated corner-a3 signals, generating [804] corner-q3 signals representing a mapping of the third corner q3 of the second source pixel onto the destination grid [120].

11. A mapping and rendering method [800] according to Claim 10 further comprising the steps of:

after the corner-a3 signals are generated:

from the defined corner-a0 coordinates, generating [810] corner-a1 signals representing a mapping of the first corner a1 of the first source pixel onto the destination grid [120]; and

from the generated corner-a3 signals, generating [812] corner-a2 signals representing a mapping of the third corner a2 of the second source pixel onto the destination grid [120];

defining [803] corner-q0 coordinates to be the same as those of the generated corner-a3 signals,

after the corner-q3 signals are generated:

from the defined corner-q0 coordinates, generating corner-q1 signals representing a mapping of the first corner q1 of the second source pixel onto the destination grid [120]; and

from the generated corner-q3 signals, generating corner-q2 signals representing a mapping of the third corner q2 of the second source pixel onto the destination grid [120].

12. A mapping and rendering method [800] according to Claim 11 further comprising the steps of:

after the corner-a0 through corner-a3 signals are generated, using [820-823] the corner-a0 through corner-a3 signals to determine [825] what destination pixels, if any, will be painted with a color code [418] associated with the first source pixel [a]; and independently thereof,

after the corner-q0 through corner-q3 signals are generated, using the corner-q0 through corner-q3 signals to determine [885] what destination pixels, if any, will

be painted with a color code [418] associated with the second source pixel [g].

13. A precision calculating apparatus comprising:

(a) two or more part registers for storing respective more and less significant parts of a long result;

(b) a math unit;

(c) selective updating means for selectively coupling the math unit to one or another of the part registers and updating the result part stored in the selected part register;

(d) detection means for detecting conditions where the updating of one result part necessitates the updating of another result part; and

(e) control means for instructing the selective updating means to select and update the other part of the long result if the detection means indicates a necessity of so doing.

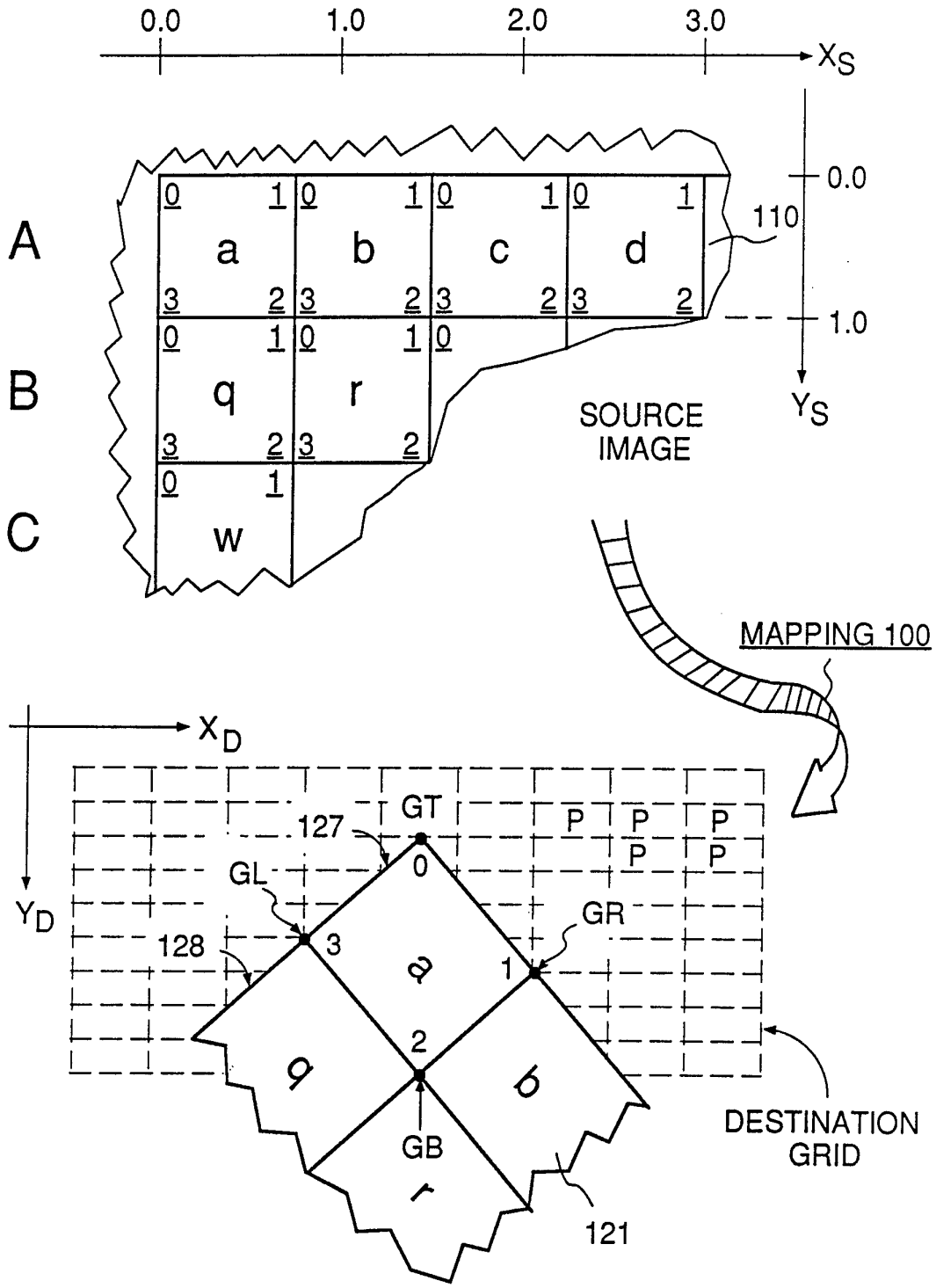

0.0
1.0
2.0
3.0
X_S
A
B
C
a
b
c
d
q
r
w
110
0.0
1.0
Y_S
SOURCE IMAGE
MAPPING 100
X_D
Y_D
GT
127
GL
GR
128
GB
121
P
DESTINATION GRID

FIGURE 1A

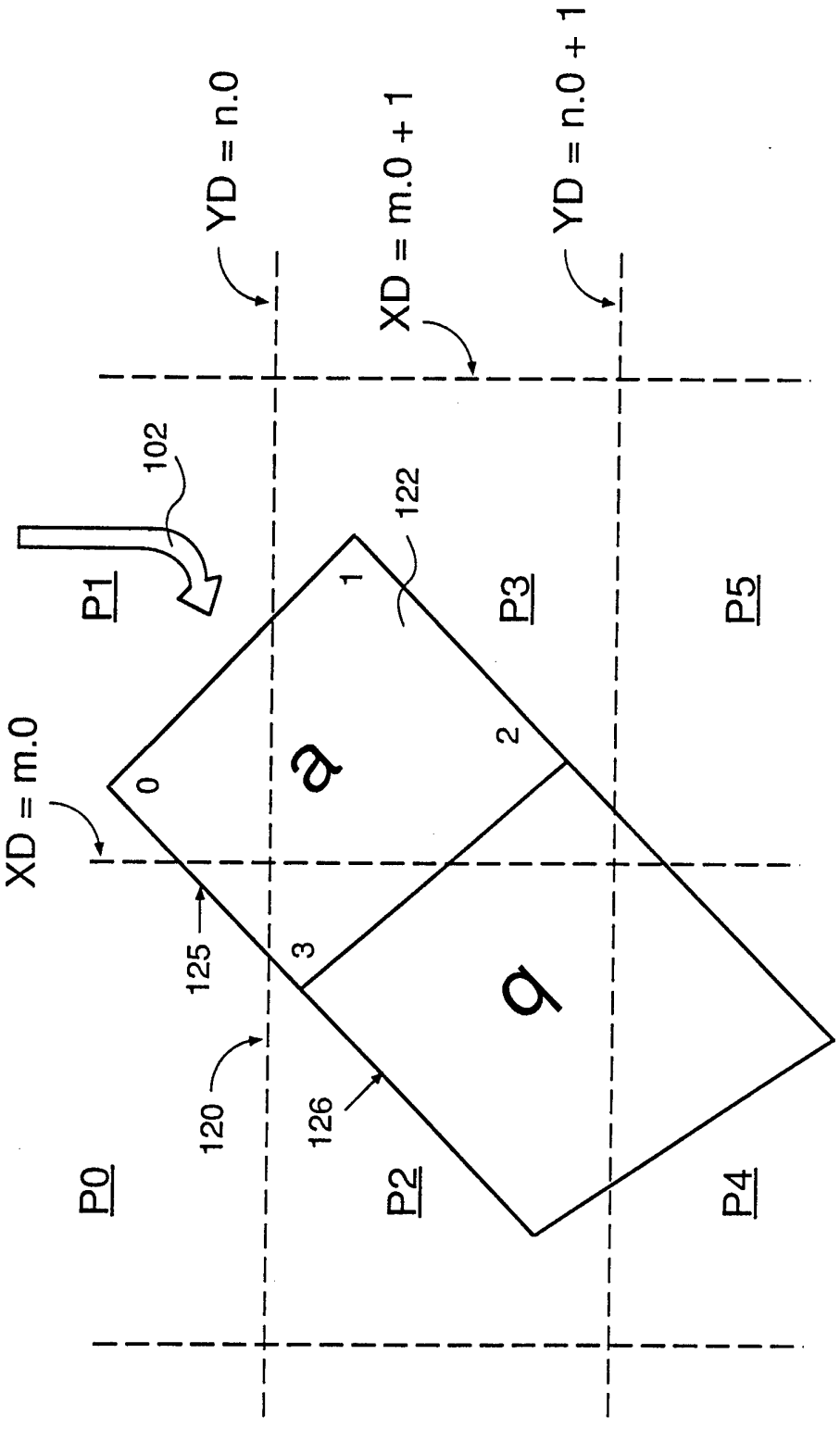

YD = n.0
XD = m.0 + 1
YD = n.0 + 1
XD = m.0
102
122
P1
P3
P5
a
q
0
1
2
3
125
120
126
P0
P2
P4

FIGURE 1B

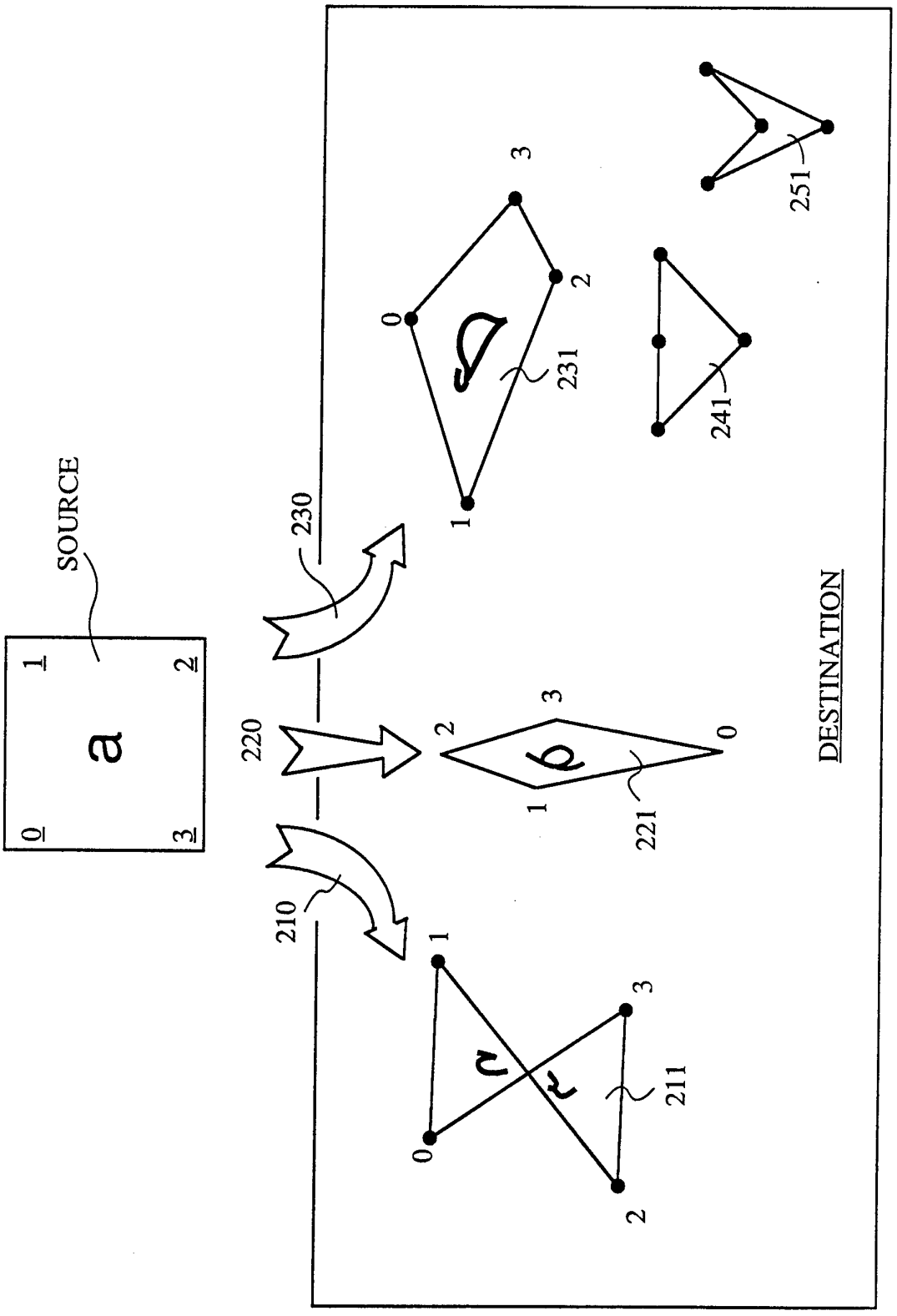


FIGURE 2

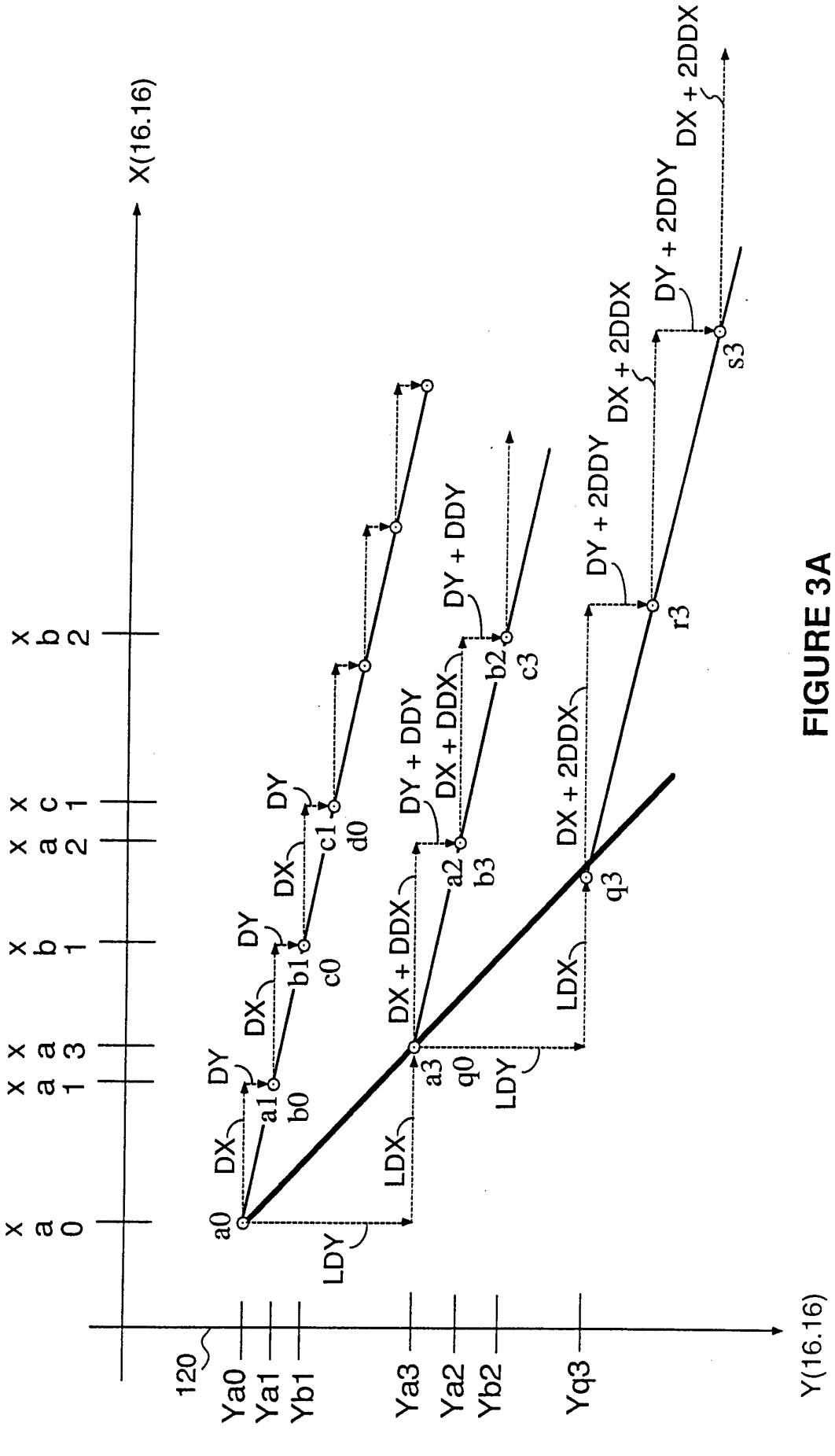

X(16.16)
Y(16.16)
x a 0
x a a 1 3
x x a a 1 3
x x b 1
x x a c 2 1
x x b 2
a0
a1
b0
b1
c0
c1
d0
a2
b3
b2
c3
a3
q0
q3
r3
s3
DX
DY
LDX
LDY
DX + DDX
DY + DDY
DX + 2DDX
DY + 2DDY
120
Ya0
Ya1
Yb1
Ya3
Ya2
Yb2
Yq3

FIGURE 3A

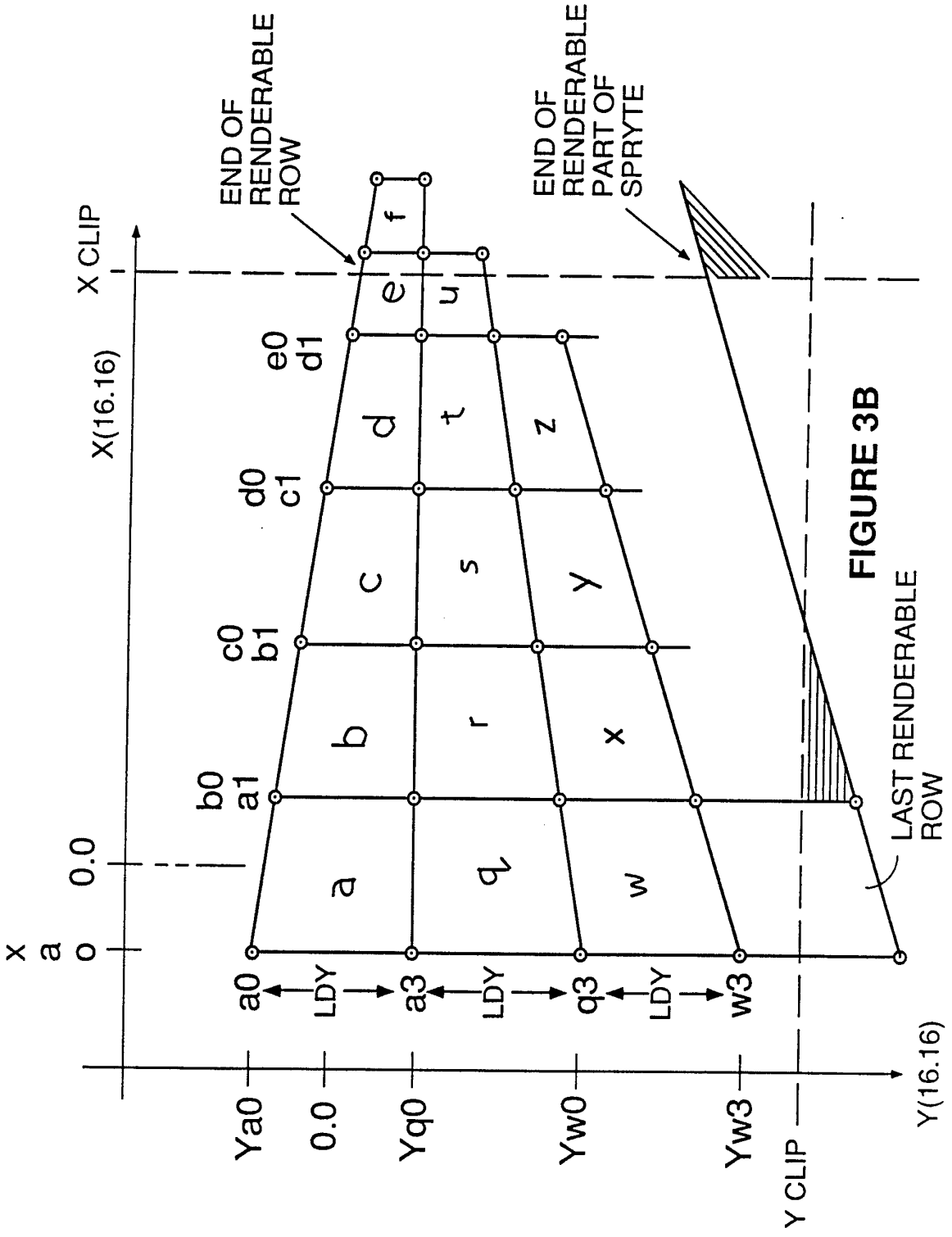

END OF RENDERABLE ROW
END OF RENDERABLE PART OF SPRYTE
X CLIP
X(16.16)
X
a
o
0.0
b0
a1
c0
b1
d0
c1
e0
d1
a
b
c
d
e
f
q
r
s
t
u
w
x
y
z
a0
LDY
a3
LDY
q3
LDY
w3
Ya0
0.0
Yq0
Yw0
Yw3
Y CLIP
Y(16.16)
LAST RENDERABLE ROW

FIGURE 3B

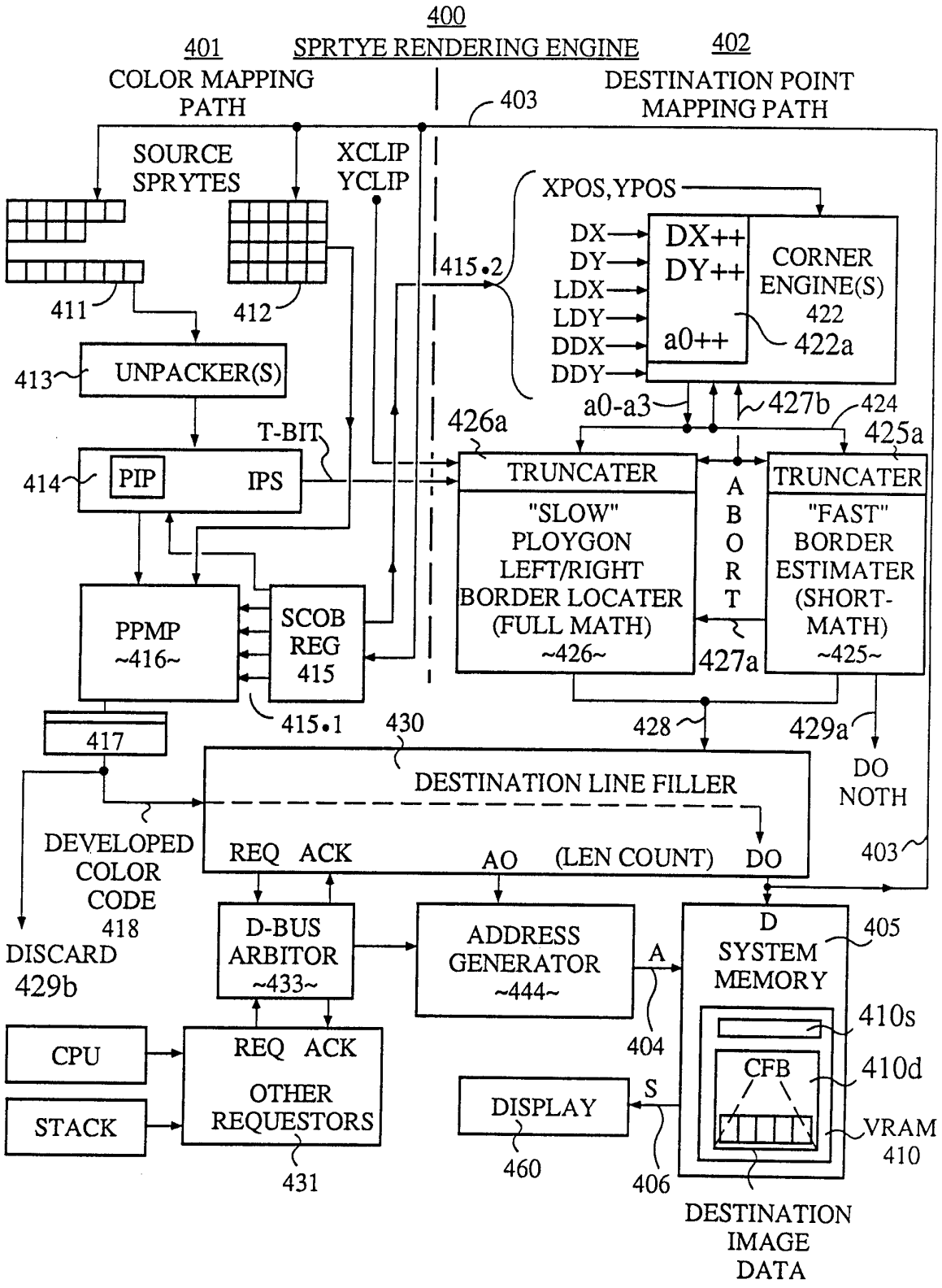

400
401
SPRTYE RENDERING ENGINE
402
COLOR MAPPING PATH
DESTINATION POINT MAPPING PATH
403
SOURCE SPRYTES
XCLIP YCLIP
XPOS,YPOS
DX
DY
LDX
LDY
DDX
DDY
DX++
DY++
a0++
CORNER ENGINE(S) 422
422a
415•2
411
412
413
UNPACKER(S)
a0-a3
427b
424
425a
426a
T-BIT
414
PIP
IPS
TRUNCATER
"SLOW" PLOYGON LEFT/RIGHT BORDER LOCATER (FULL MATH) ~426~
ABORT
TRUNCATER
"FAST" BORDER ESTIMATER (SHORT-MATH) ~425~
427a
PPMP ~416~
SCOB REG 415
415•1
417
430
428
429a
DO NOTH
DESTINATION LINE FILLER
DEVELOPED COLOR CODE 418
REQ ACK
AO
(LEN COUNT)
DO
403
DISCARD 429b
D-BUS ARBITOR ~433~
ADDRESS GENERATOR ~444~
A
D
SYSTEM MEMORY
405
410s
CFB
410d
VRAM 410
CPU
STACK
REQ ACK
OTHER REQUESTORS
431
404
DISPLAY
S
460
406
DESTINATION IMAGE DATA

FIGURE 4

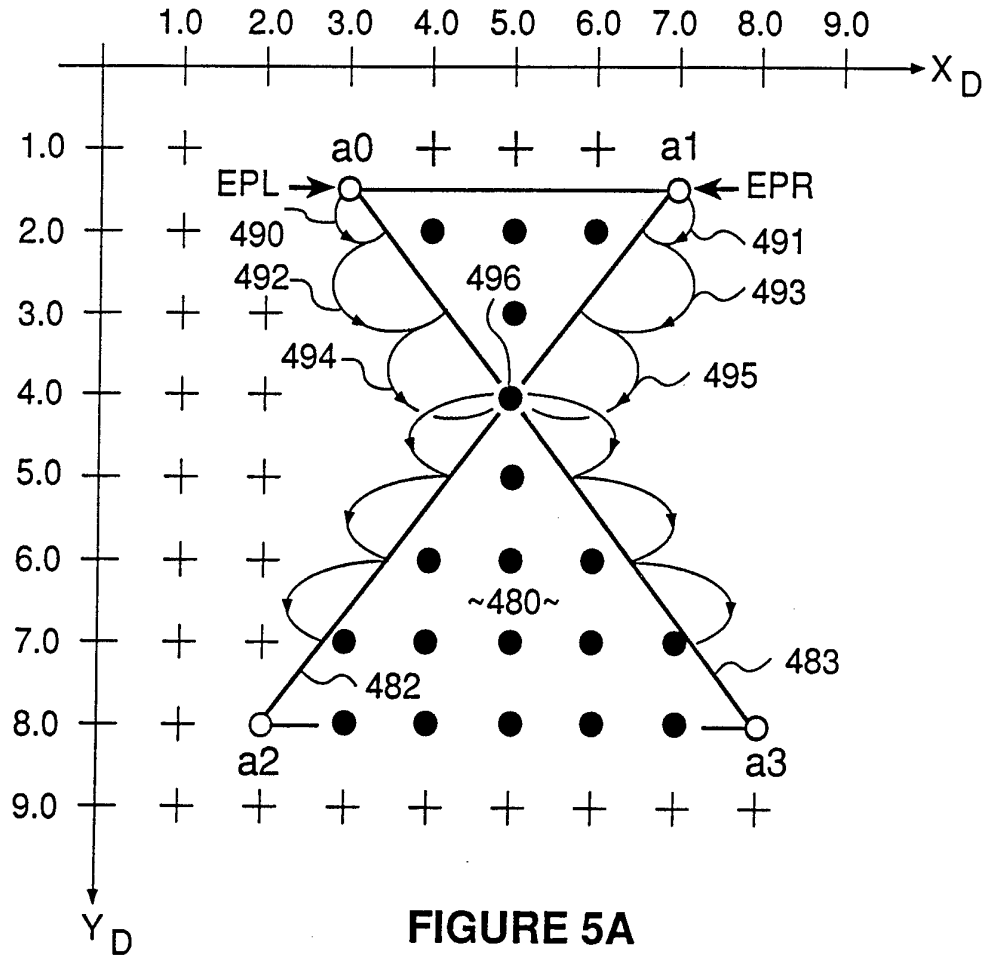


FIGURE 5A

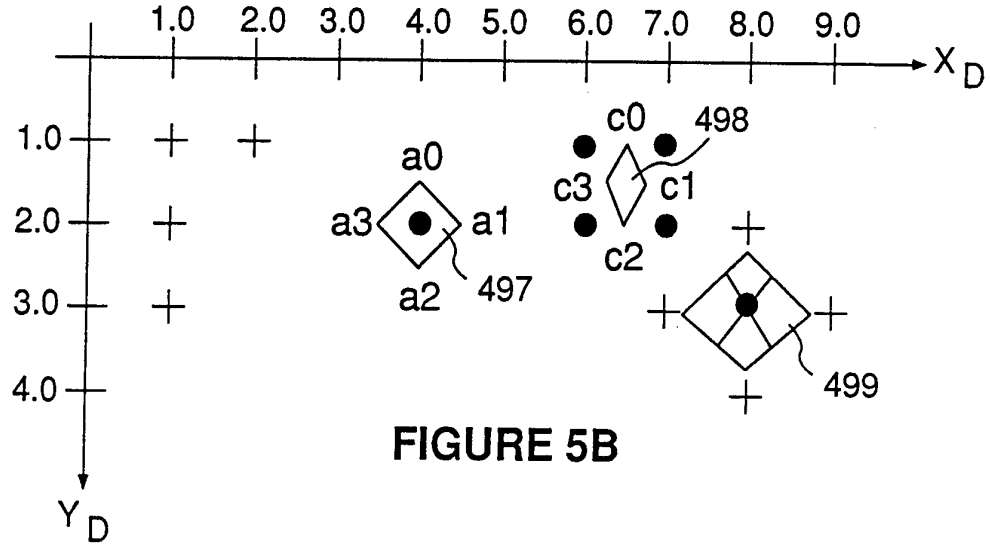


FIGURE 5B

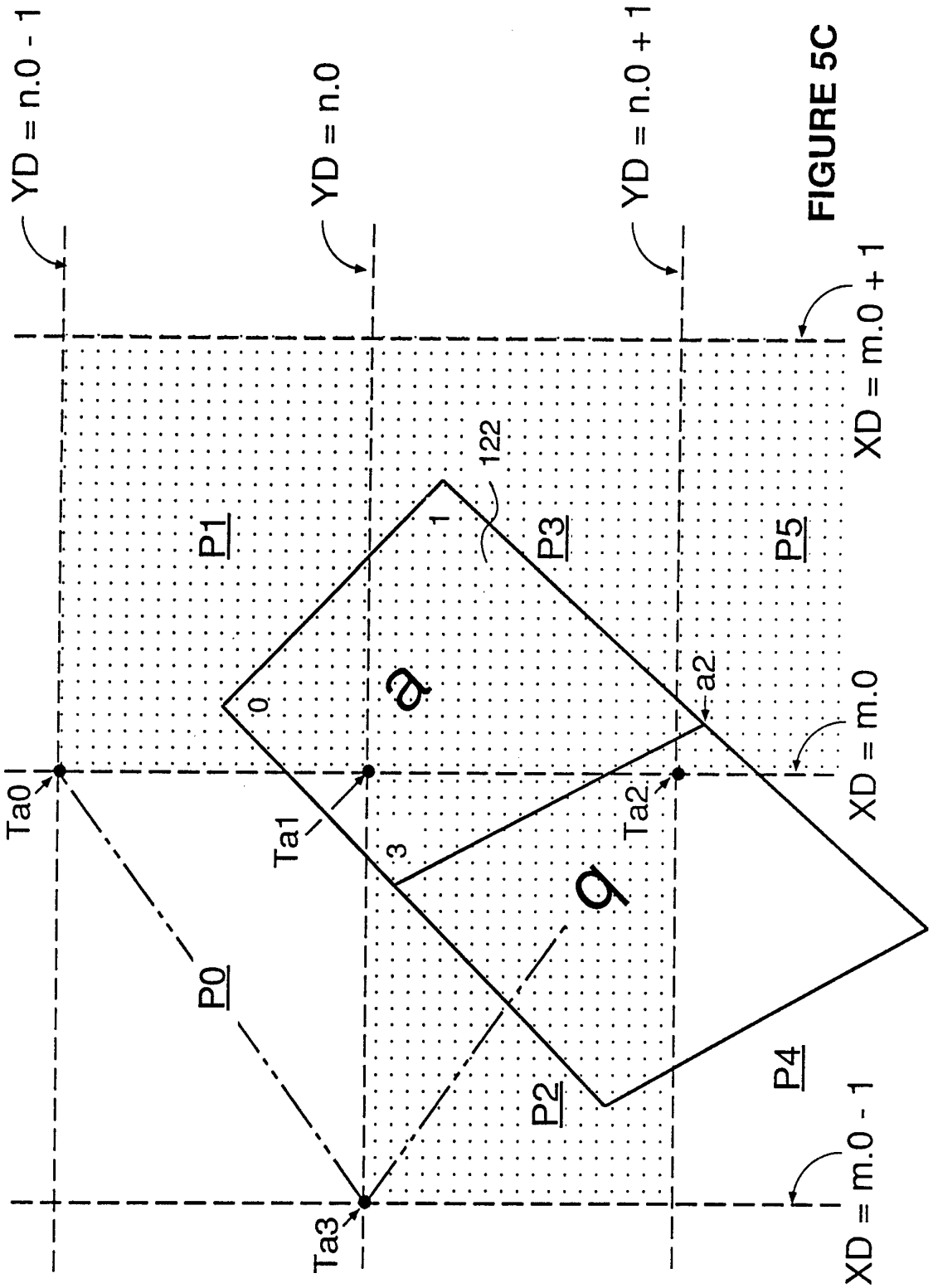

YD = n.0 - 1
YD = n.0
YD = n.0 + 1
XD = m.0 + 1
XD = m.0
XD = m.0 - 1
P0
P1
P2
P3
P4
P5
Ta0
Ta1
Ta2
Ta3
a
a2
122
0
1
3

FIGURE 5C

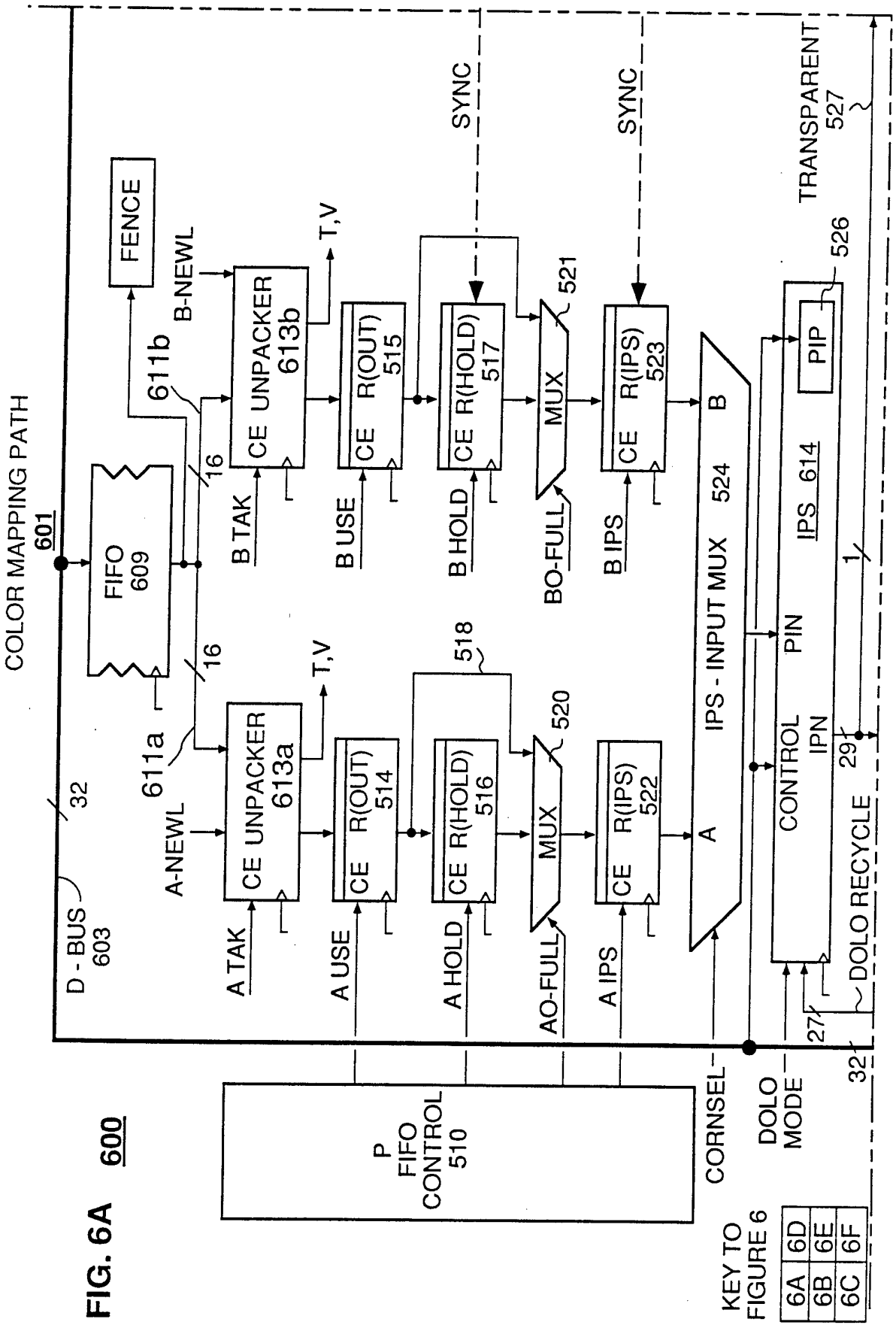

COLOR MAPPING PATH
601
D - BUS
603
32
FIFO
609
16
FENCE
611b
611a
A-NEWL
B-NEWL
CE UNPACKER
613a
CE UNPACKER
613b
T,V
T,V
A TAK
B TAK
A USE
B USE
A HOLD
B HOLD
AO-FULL
BO-FULL
A IPS
B IPS
CE R(OUT)
514
CE R(OUT)
515
CE R(HOLD)
516
CE R(HOLD)
517
518
MUX
520
MUX
521
CE R(IPS)
522
CE R(IPS)
523
SYNC
SYNC
A
B
IPS - INPUT MUX
524
TRANSPARENT
527
PIP
526
CONTROL
PIN
IPS 614
IPN
291
1
DOLO RECYCLE
27
P
FIFO
CONTROL
510
CORNSEL
DOLO
MODE
32
KEY TO
FIGURE 6
6A
6B
6C
6D
6E
6F

FIG. 6A 600

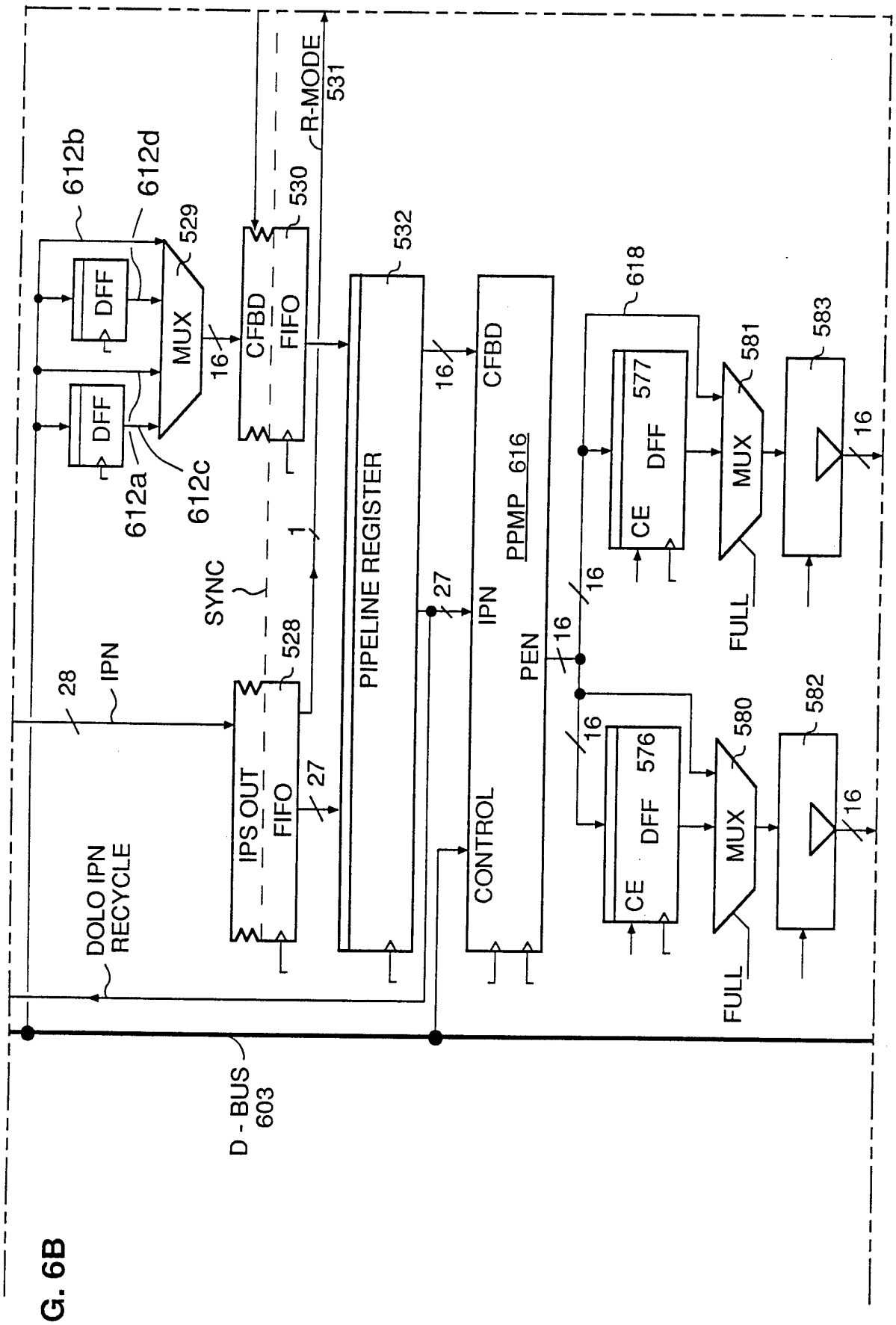

612b
612d
529
DFF
DFF
MUX
612a
612c
16
CFBD
FIFO
530
R-MODE
531
532
PIPELINE REGISTER
16
CFBD
PPMP 616
IPN
PEN
CONTROL
27
SYNC
1
528
IPS OUT
FIFO
27
28
IPN
DOLO IPN
RECYCLE
D - BUS
603
618
577
DFF
CE
581
MUX
FULL
583
16
16
576
DFF
CE
580
MUX
FULL
582
16
16
16

FIG. 6B

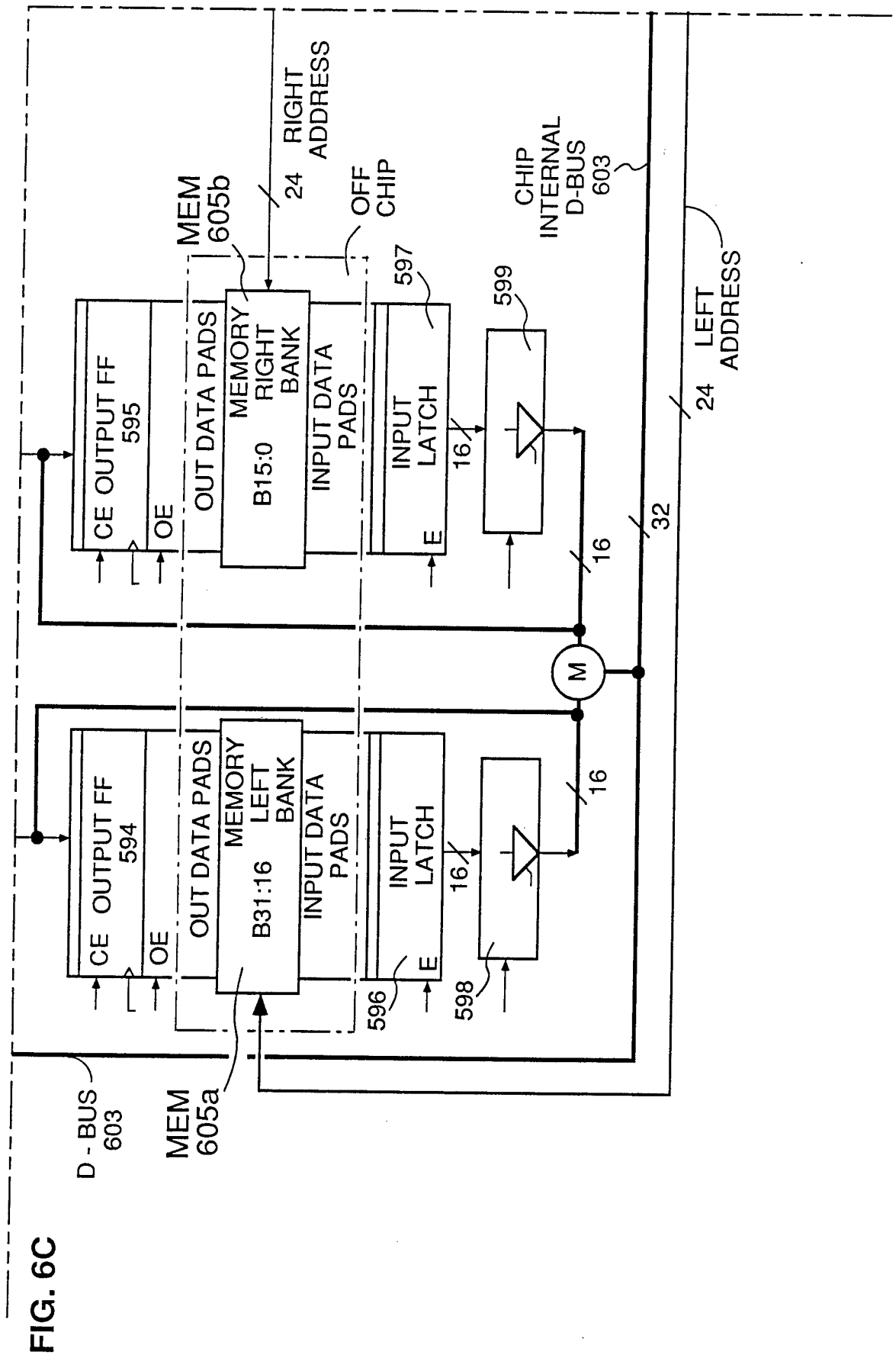

MEM 605b
24 RIGHT ADDRESS
OFF CHIP
CHIP INTERNAL D-BUS 603
24 LEFT ADDRESS
CE OUTPUT FF 595
OE
OUT DATA PADS
MEMORY RIGHT BANK
B15:0
INPUT DATA PADS
597
INPUT LATCH
E
16
599
16
32
M
CE OUTPUT FF 594
OE
OUT DATA PADS
MEMORY LEFT BANK
B31:16
INPUT DATA PADS
596
INPUT LATCH
E
16
598
16
D - BUS 603
MEM 605a

FIG. 6C

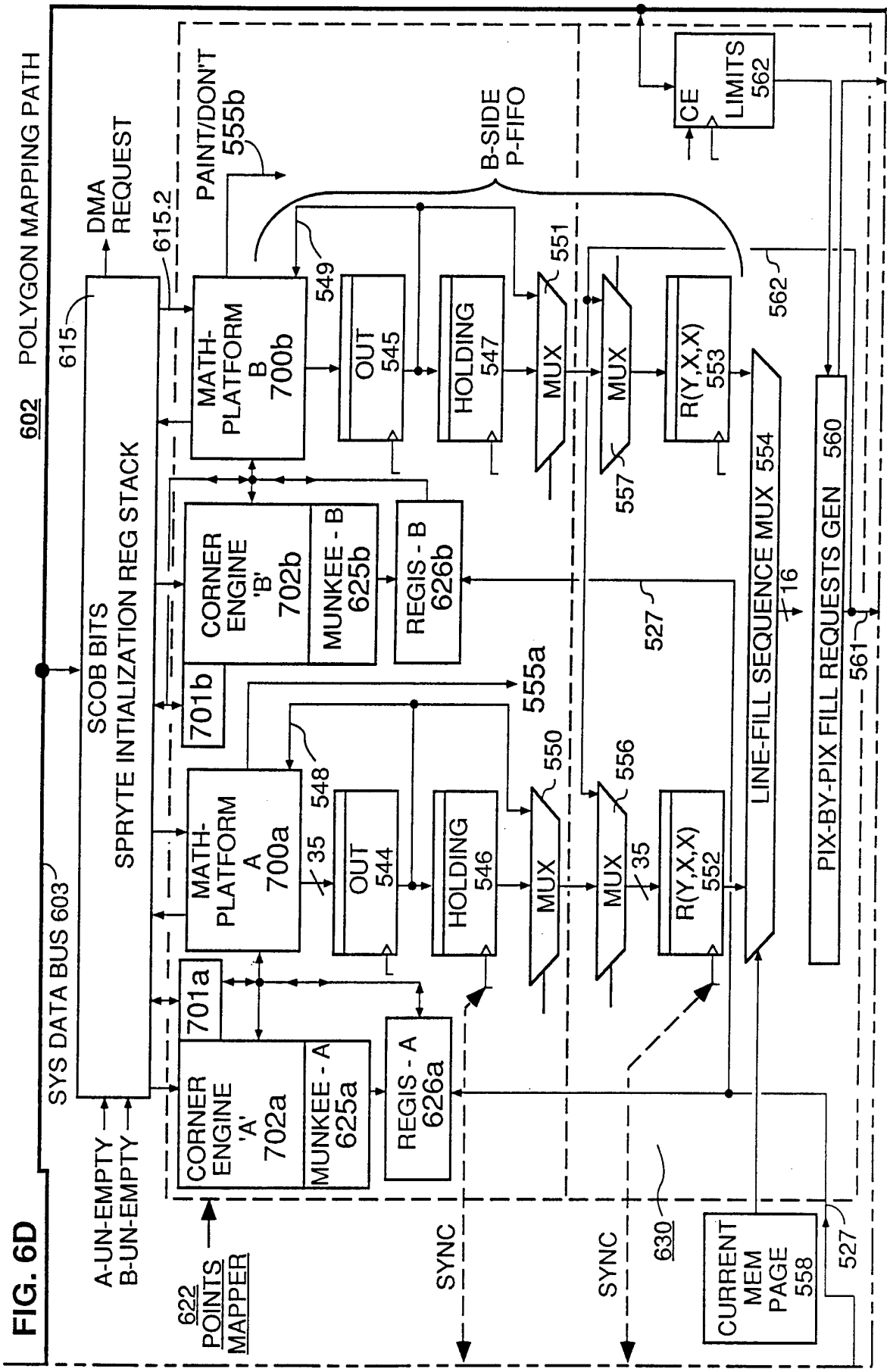

602 POLYGON MAPPING PATH
SYS DATA BUS 603
SCOB BITS
SPRYTE INTIALIZATION REG STACK
615
DMA REQUEST
615.2
PAINT/DON'T 555b
B-SIDE P-FIFO
MATH-PLATFORM B 700b
549
OUT 545
HOLDING 547
MUX 551
MUX
R(Y,X,X) 553
562
CE
LIMITS 562
CORNER ENGINE 'B' 702b
MUNKEE - B 625b
REGIS - B 626b
701b
555a
MATH-PLATFORM A 700a
548
35
OUT 544
HOLDING 546
MUX 550
MUX 556
35
R(Y,X,X) 552
527
LINE-FILL SEQUENCE MUX 554
16
PIX-BY-PIX FILL REQUESTS GEN 560
561
557
701a
CORNER ENGINE 'A' 702a
MUNKEE - A 625a
REGIS - A 626a
A-UN-EMPTY
B-UN-EMPTY
622 POINTS MAPPER
SYNC
SYNC
630
CURRENT MEM PAGE 558
527

FIG. 6D

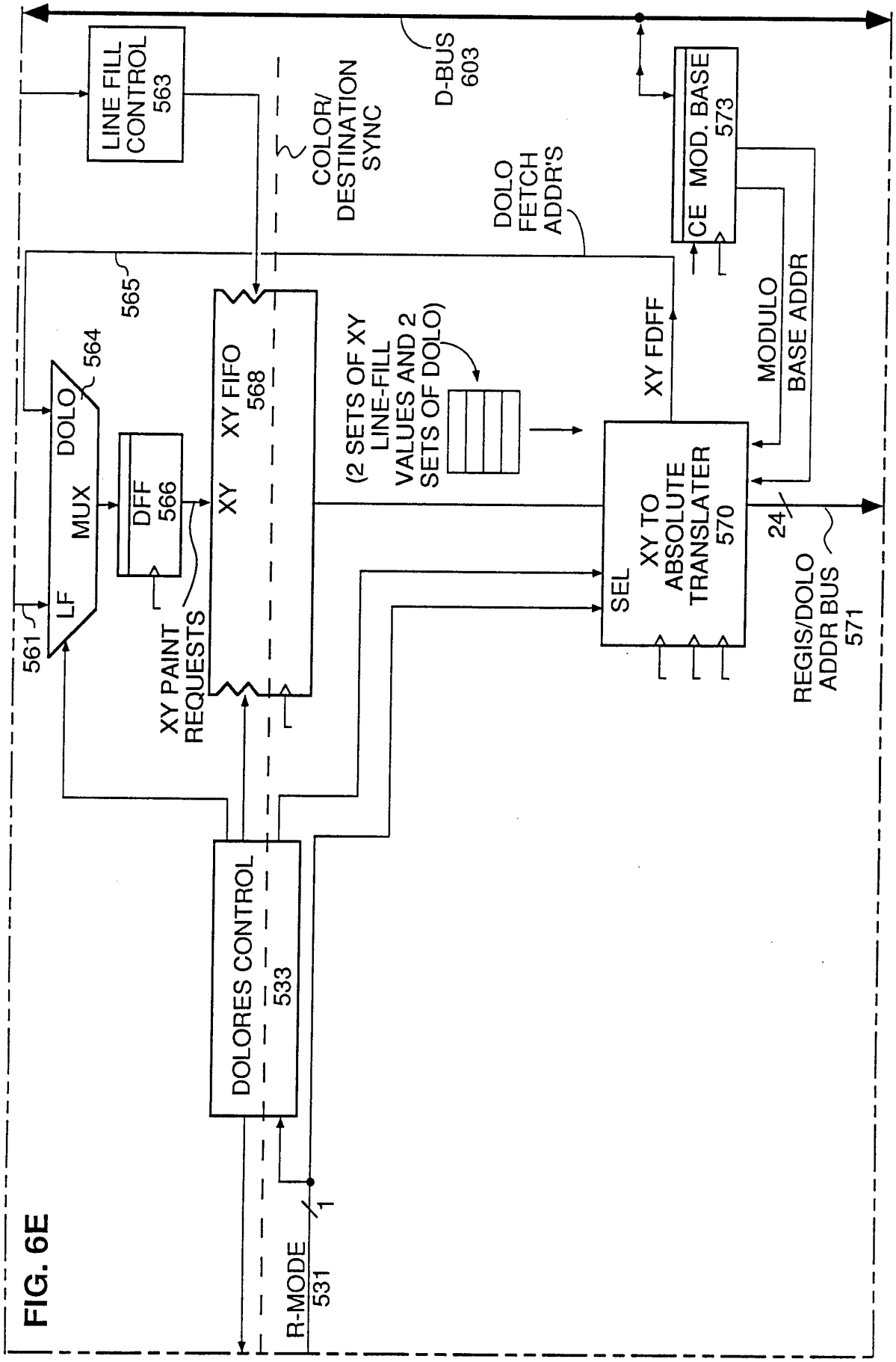

LINE FILL CONTROL 563
COLOR/ DESTINATION SYNC
D-BUS 603
CE MOD. BASE 573
DOLO FETCH ADDR'S
565
564
DOLO
MUX
LF
561
DFF 566
XY PAINT REQUESTS
XY FIFO 568
XY
(2 SETS OF XY LINE-FILL VALUES AND 2 SETS OF DOLO)
XY FDFF
MODULO BASE ADDR
SEL
XY TO ABSOLUTE TRANSLATER 570
24
REGIS/DOLO ADDR BUS 571
DOLORES CONTROL 533
R-MODE 531
1

FIG. 6E

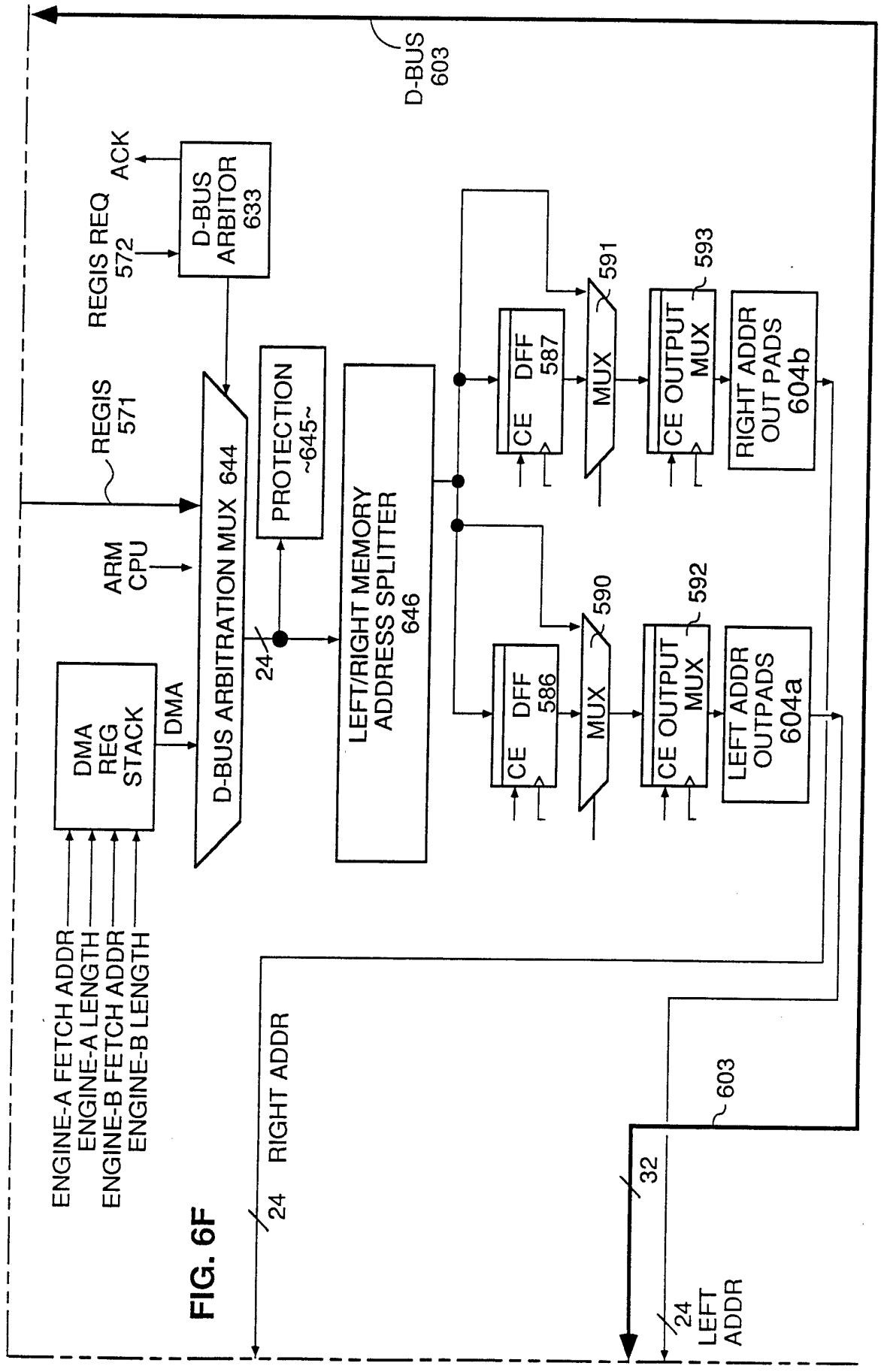

D-BUS
603
ACK
REGIS REQ
572
D-BUS
ARBITOR
633
REGIS
571
PROTECTION
~645~
ARM
CPU
D-BUS ARBITRATION MUX 644
LEFT/RIGHT MEMORY
ADDRESS SPLITTER
646
CE DFF
587
591
MUX
CE OUTPUT
MUX
593
RIGHT ADDR
OUT PADS
604b
CE DFF
586
590
MUX
CE OUTPUT
MUX
592
LEFT ADDR
OUTPADS
604a
DMA
REG
STACK
DMA
24
ENGINE-A FETCH ADDR
ENGINE-A LENGTH
ENGINE-B FETCH ADDR
ENGINE-B LENGTH
24 RIGHT ADDR
603
32
24
LEFT
ADDR

FIG. 6F

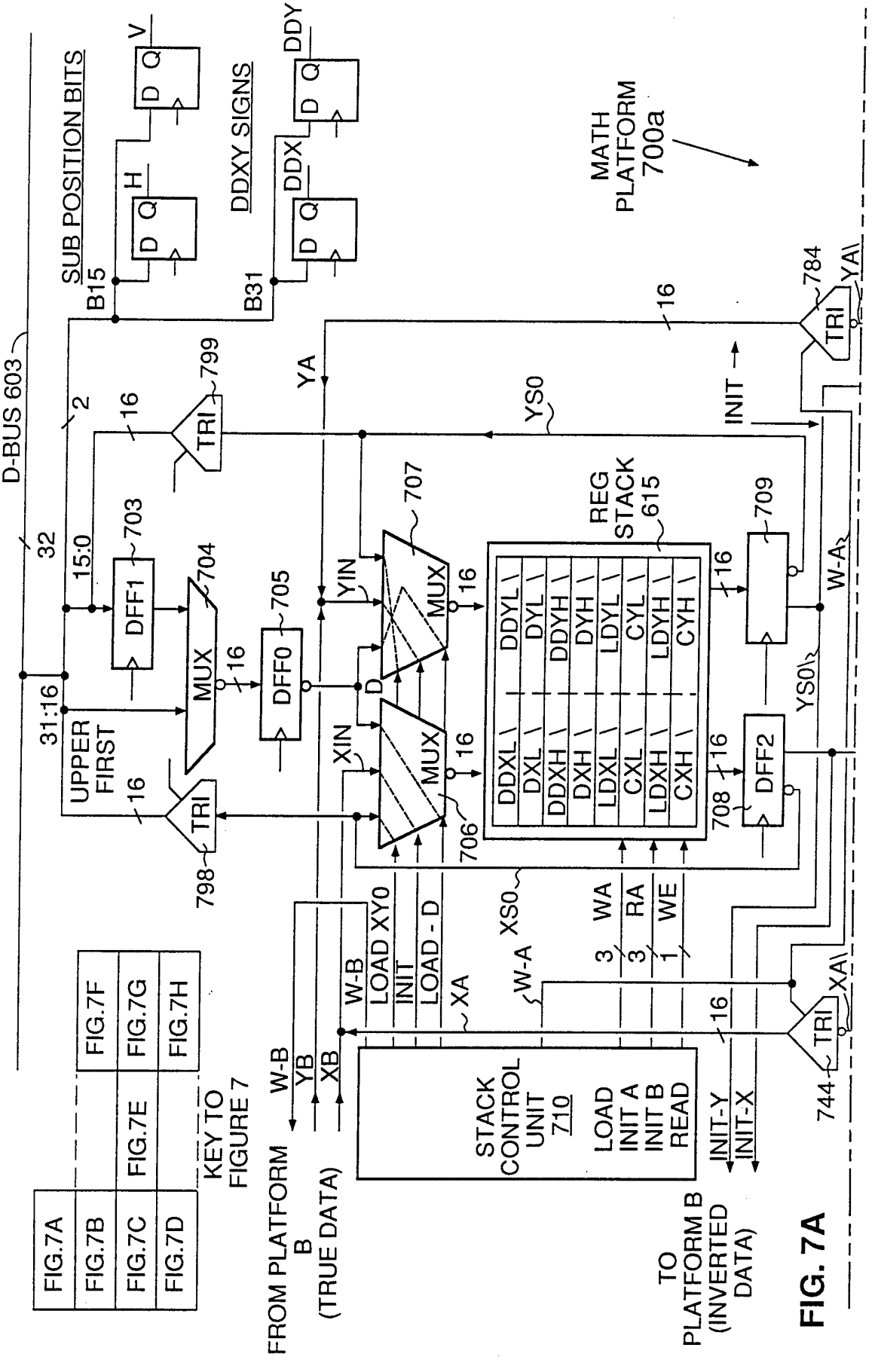

MATH PLATFORM 700a
D-BUS 603
SUB POSITION BITS
DDXY SIGNS
B15
B31
V
H
DDX
DDY
D Q
TRI
DFF1
DFF0
DFF2
MUX
YIN
XIN
YA
YS0
XS0
INIT
W-A
W-B
LOAD XY0
LOAD - D
XA
WA
RA
WE
REG STACK 615
DDXL
DXL
DDXH
DXH
LDXL
CXL
LDXH
CXH
DDYL
DYL
DDYH
DYH
LDYL
CYL
LDYH
CYH
STACK CONTROL UNIT 710
LOAD
INIT A
INIT B
READ
INIT-Y
INIT-X
FROM PLATFORM B (TRUE DATA)
TO PLATFORM B (INVERTED DATA)
KEY TO FIGURE 7
FIG.7A
FIG.7B
FIG.7C
FIG.7D
FIG.7E
FIG.7F
FIG.7G
FIG.7H

FIG. 7A

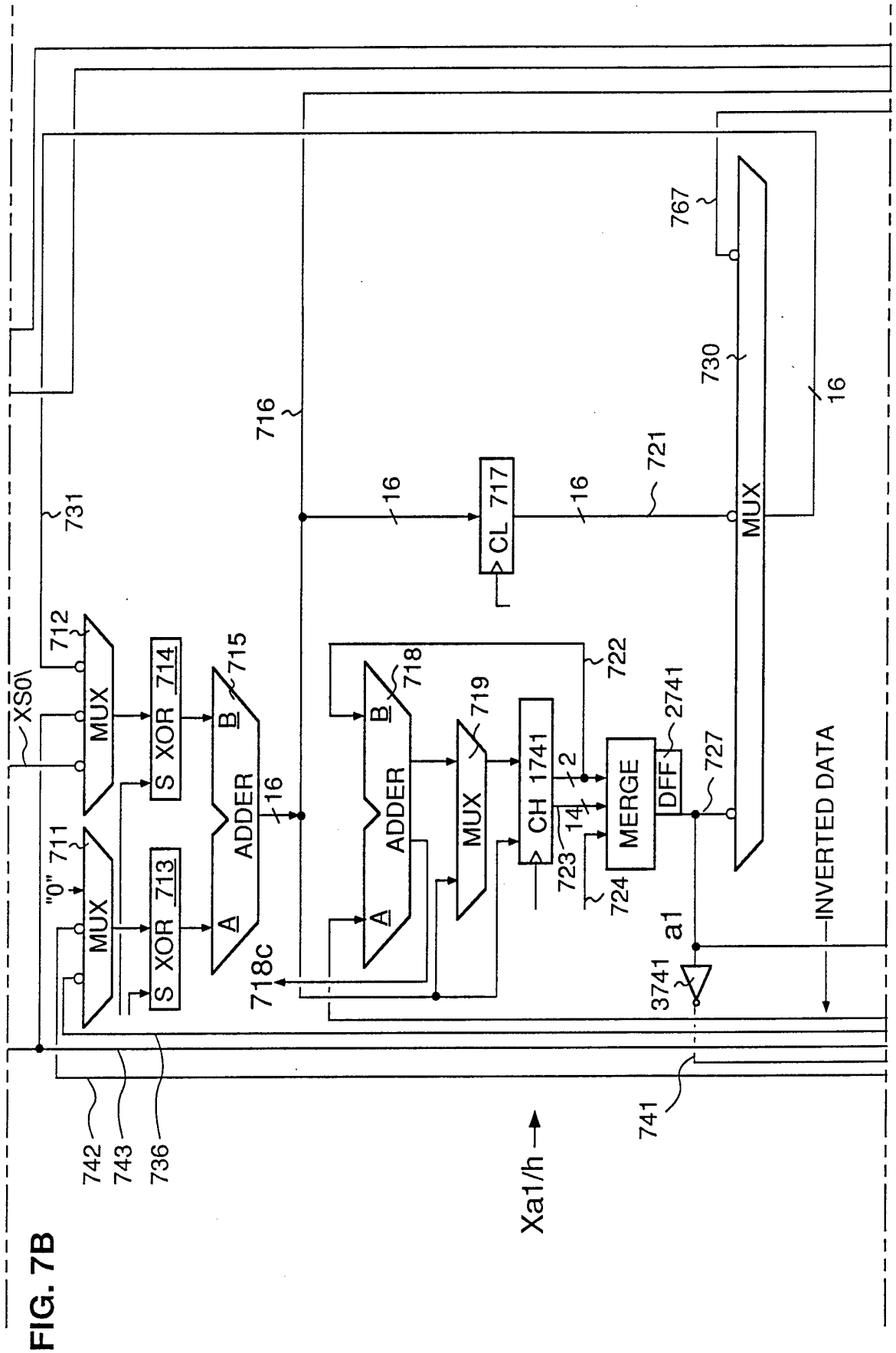

MUX
S XOR 714
S XOR 713
ADDER
A
B
CL 717
CH 1741
MERGE
DFF
INVERTED DATA
a1
3741
2741
"0"
XS0\
Xa1/h
742
743
736
741
731
716
718c
722
723
724
727
730
767
721
712
711
715
718
719
16
14
2

FIG. 7B

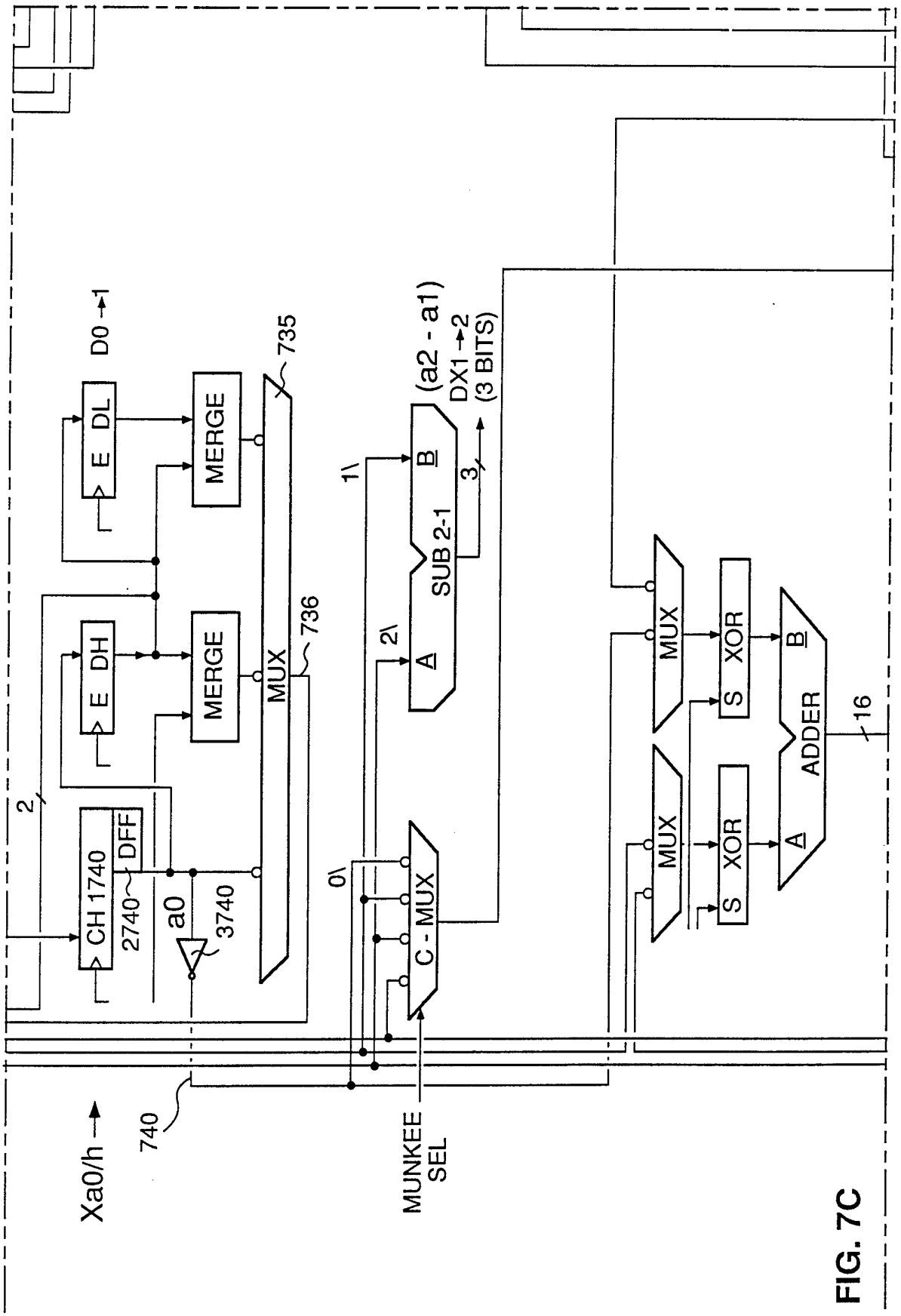

D0 →1
E DL
MERGE
735
E DH
MERGE
MUX
736
2
CH 1740
DFF
2740
a0
3740
740
Xa0/h →
(a2 - a1)
DX1 →2
(3 BITS)
1\
2\
0\
B
A
SUB 2-1
3
C - MUX
MUNKEE
SEL
MUX
MUX
S XOR
S XOR
B
A
ADDER
16

FIG. 7C

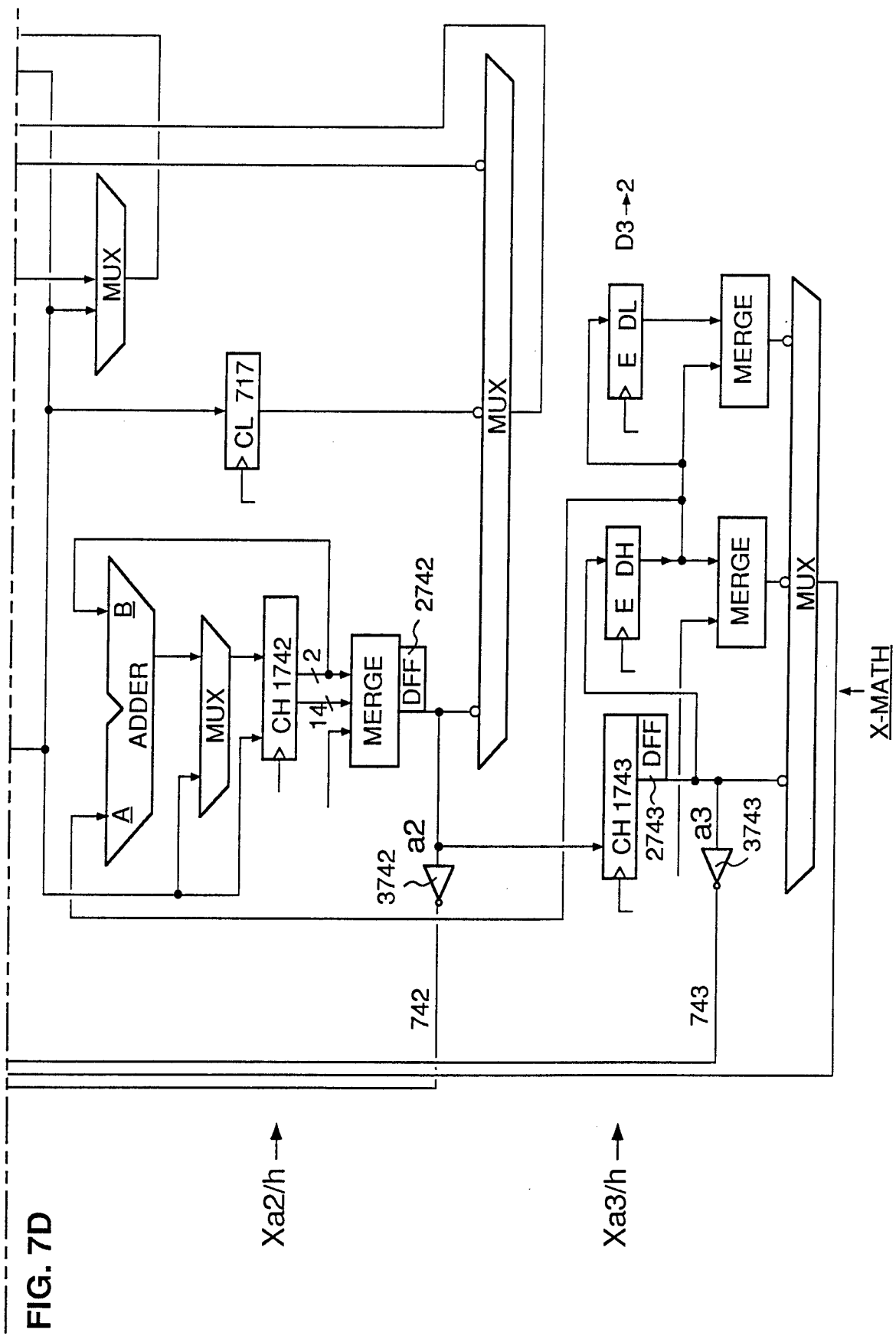

MUX
CL 717
MUX
D3→2
E DL
MERGE
E DH
MERGE
MUX
X-MATH
B
A
ADDER
MUX
CH 1742
2
14
MERGE
DFF
2742
CH 1743
DFF
2743
a3
3743
a2
3742
742
743
Xa2/h
Xa3/h

FIG. 7D

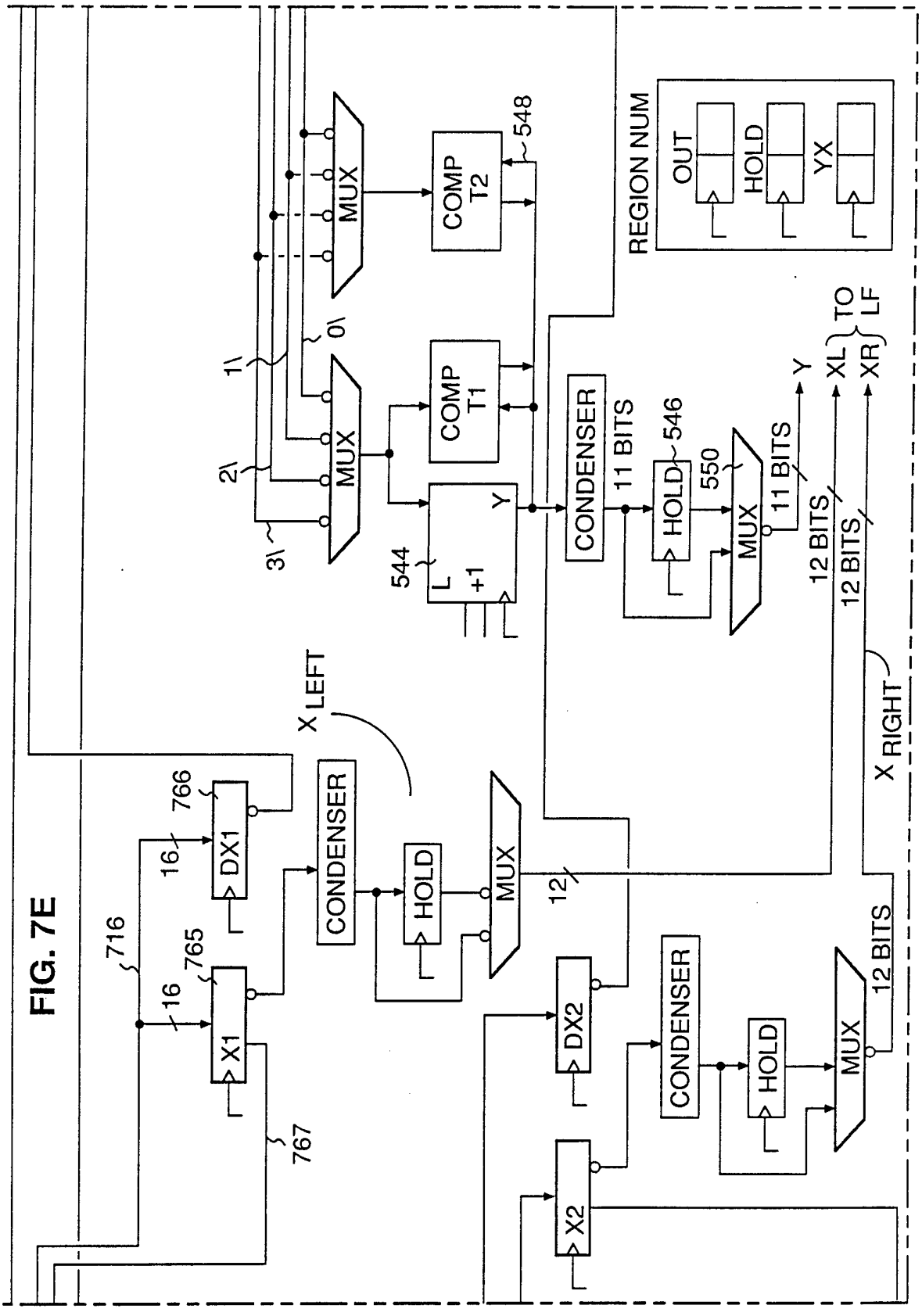

REGION NUM
OUT
HOLD
YX
COMP T2
548
COMP T1
MUX
0\
1\
2\
3\
544
L
+1
Y
CONDENSER
11 BITS
HOLD
546
550
MUX
11 BITS
Y
XL
XR
TO LF
12 BITS
12 BITS
X LEFT
X RIGHT
766
DX1
16
716
765
X1
767
CONDENSER
HOLD
MUX
12
DX2
CONDENSER
HOLD
MUX
12 BITS
X2

FIG. 7E

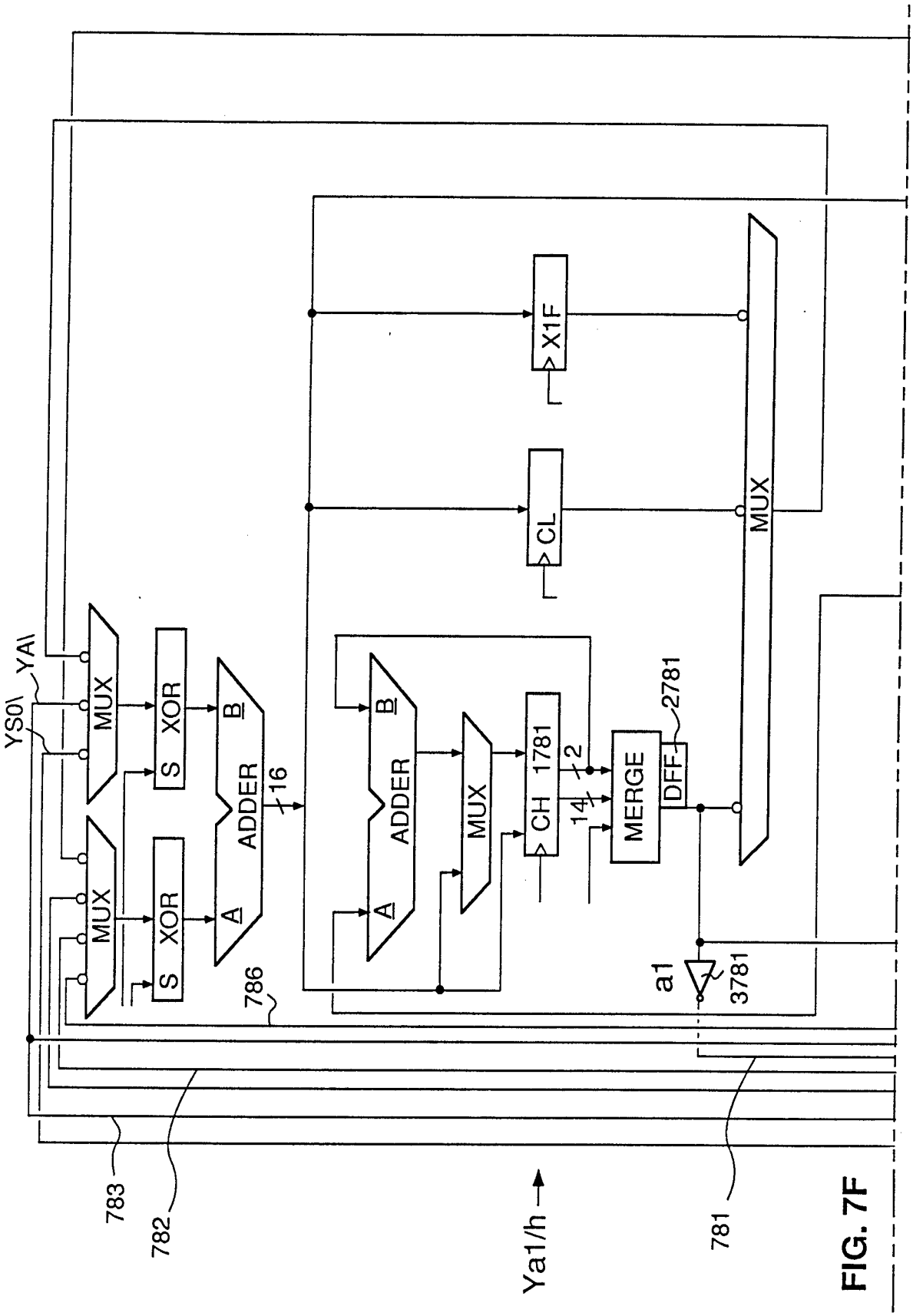

YS0\
YA\
MUX
MUX
S XOR
S XOR
B
A
ADDER
16
786
B
A
ADDER
MUX
CH 1781
2
14
MERGE
DFF
2781
X1F
CL
MUX
a1
3781
783
782
781
Ya1/h

FIG. 7F

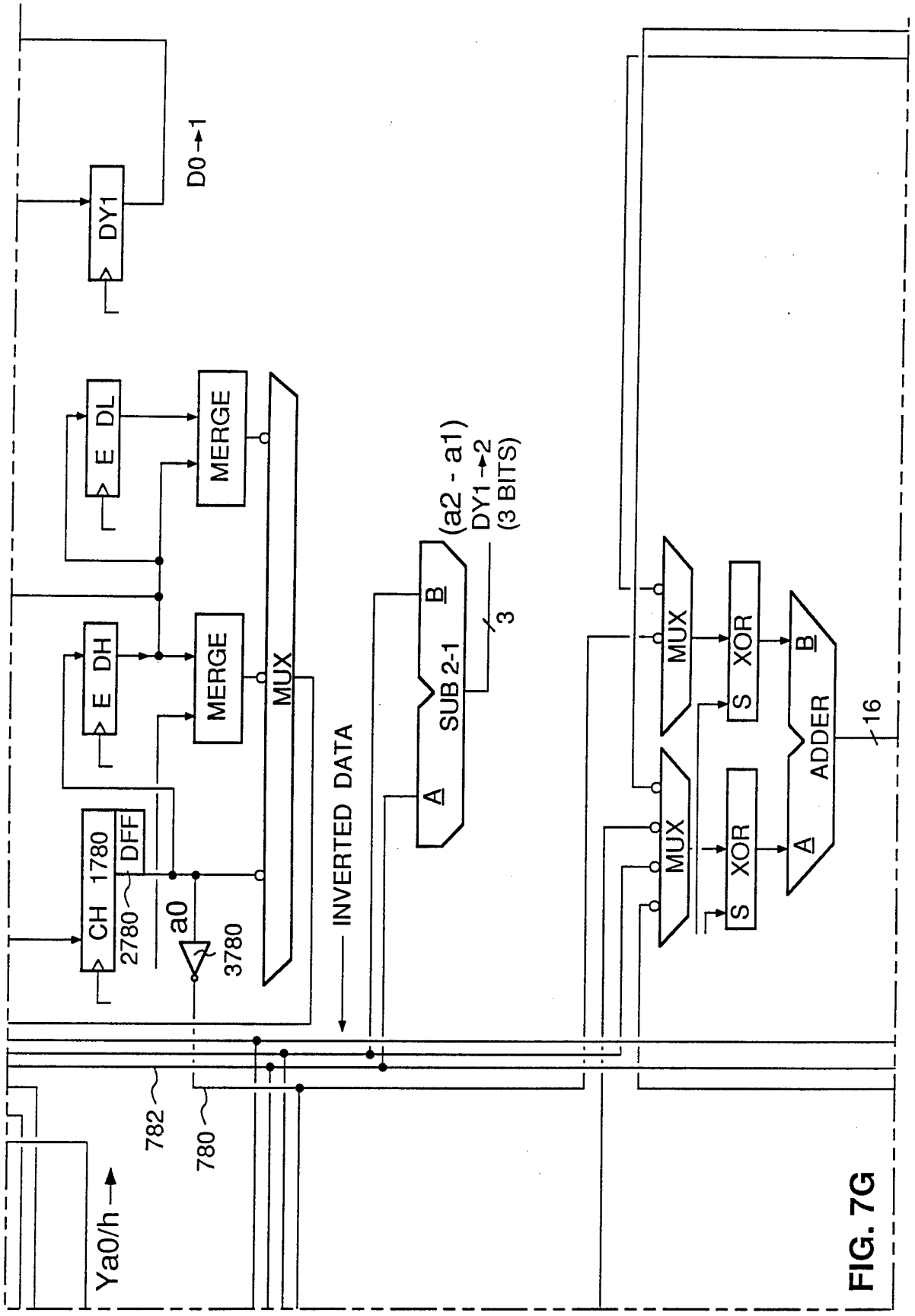

D0→1
DY1
E DL
MERGE
E DH
MERGE
MUX
CH 1780
DFF
2780
a0
3780
INVERTED DATA
(a2 - a1)
DY1→2
(3 BITS)
B
SUB 2-1
A
3
MUX
S XOR
B
MUX
S XOR
A
ADDER
16
782
780
Ya0/h→

FIG. 7G

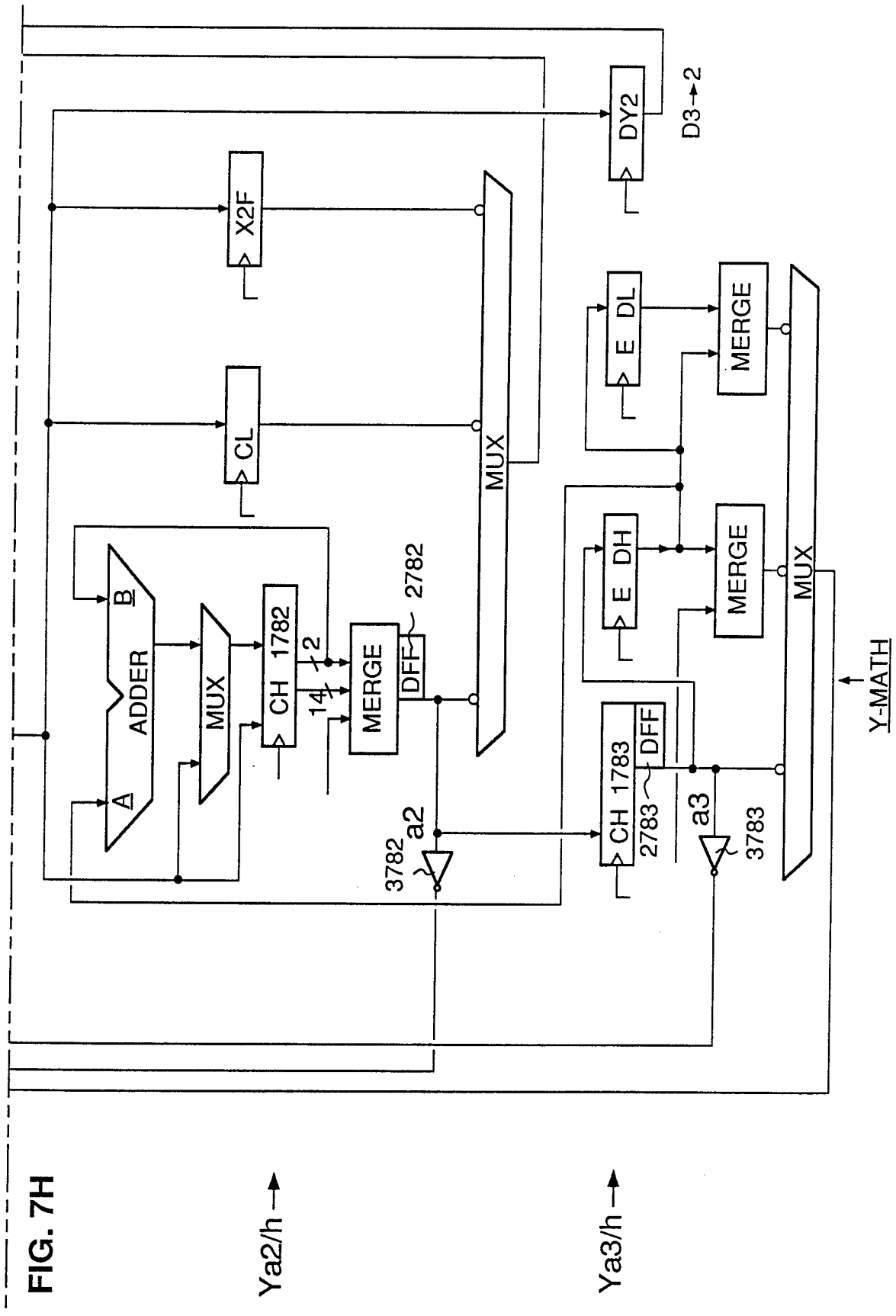

DY2
D3→2
X2F
CL
MUX
E DL
MERGE
E DH
MERGE
MUX
Y-MATH
B
A
ADDER
MUX
CH 1782
14
2
MERGE
DFF
2782
3782
a2
CH 1783
DFF
2783
a3
3783
Ya2/h
Ya3/h

FIG. 7H

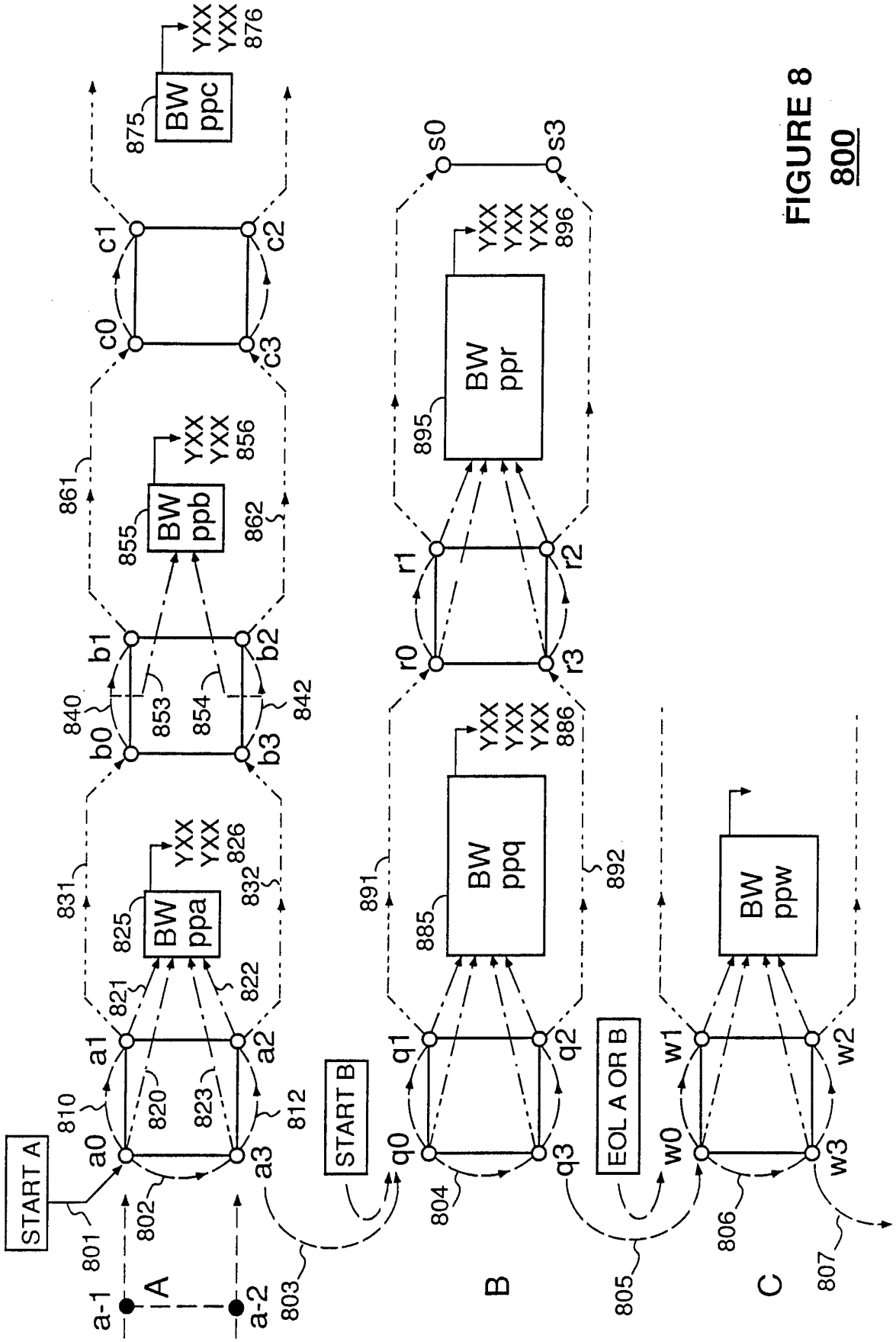

START A
START B
EOL A OR B
A
B
C
BW ppa
BW ppb
BW ppc
BW ppq
BW ppr
BW ppw
YXX YXX
a0
a1
a2
a3
a-1
a-2
b0
b1
b2
b3
c0
c1
c2
c3
q0
q1
q2
q3
r0
r1
r2
r3
s0
s3
w0
w1
w2
w3
801
802
803
804
805
806
807
810
812
820
821
822
823
825
826
831
832
840
842
853
854
855
856
861
862
875
876
885
886
891
892
895
896

FIGURE 8
800

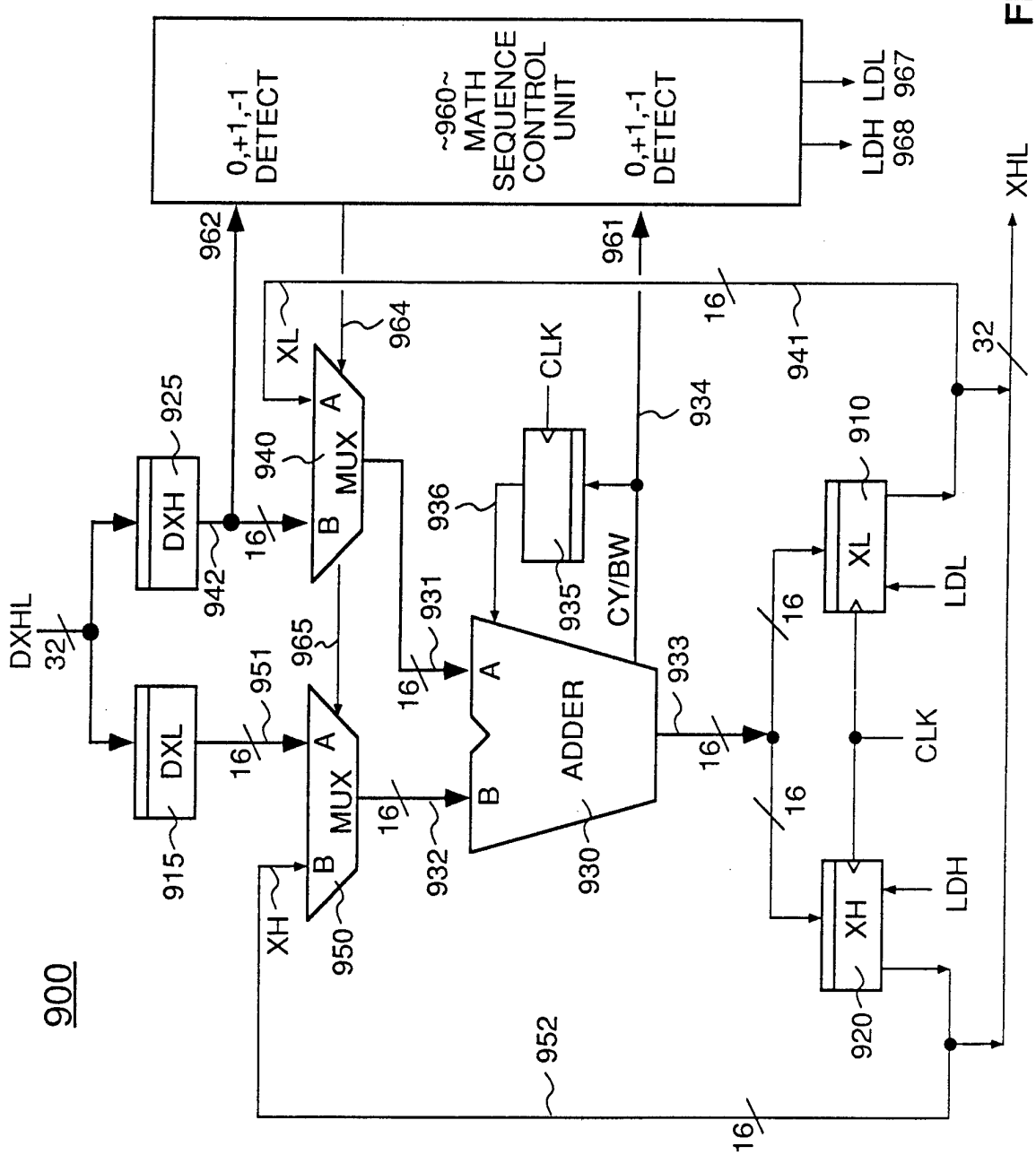

900
0,+1,-1 DETECT
~960~ MATH SEQUENCE CONTROL UNIT
0,+1,-1 DETECT
LDH 968
LDL 967
DXHL
32
DXH
925
DXL
915
942
951
16
MUX
A
B
940
965
950
XH
XL
964
962
961
931
932
ADDER
930
936
935
CLK
CY/BW
934
933
941
910
XL
LDL
CLK
XH
LDH
920
952
XHL
32

FIGURE 9A

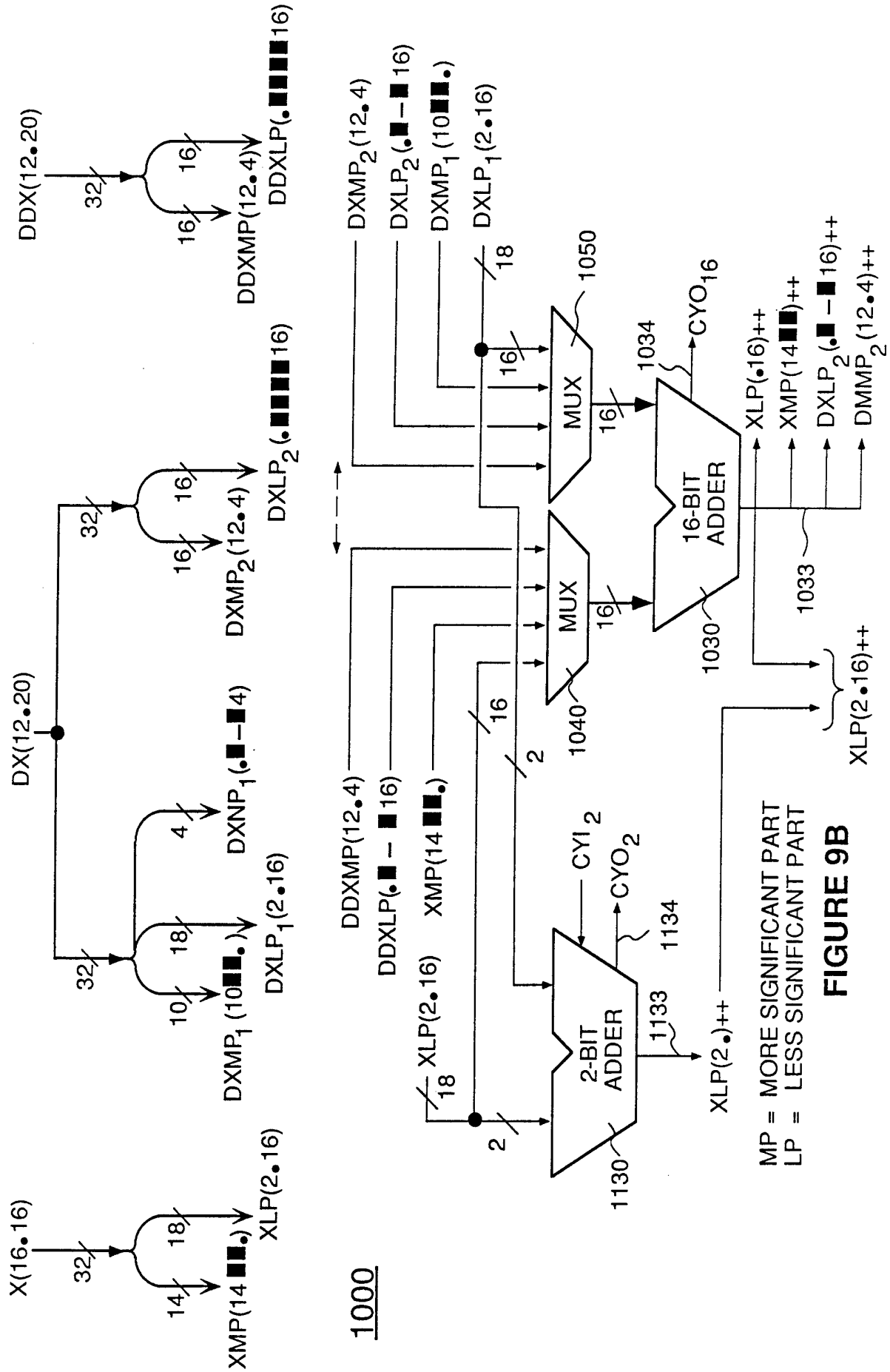

1000
X(16.16)
XMP(14■■.)
XLP(2.16)
DX(12.20)
DXMP1 (10■■.)
DXNP1(.■–■4)
DXLP1(2.16)
DXMP2(12.4)
DXLP2(.■■■■16)
DDX(12.20)
DDXMP(12.4)
DDXLP(.■■■■16)
DDXMP(12.4)
DDXLP(.■ – ■16)
XMP(14■■.)
XLP(2.16)
DXMP2(12.4)
DXLP2(.■–■16)
DXMP1(10■■.)
DXLP1(2.16)
MUX
MUX
16-BIT ADDER
2-BIT ADDER
CYI2
CYO2
CYO16
1030
1033
1034
1040
1050
1130
1133
1134
XLP(2.)++
XLP(2.16)++
XLP(.16)++
XMP(14■■)++
DXLP2(.■ –■16)++
DMMP2(12.4)++
MP = MORE SIGNIFICANT PART
LP = LESS SIGNIFICANT PART

FIGURE 9B

INTERNATIONAL SEARCH REPORT

PCT/US92/09462

A. CLASSIFICATION OF SUBJECT MATTER

IPC(5) :G06F 15/62
US CL :395/152,131; 340/725
According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/152,131; 340/725 395/154,162,164,165,166 340/750,724

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US,A, 4,580,134 (Campbell et al.) 01 April 1986 See Figure 6 and Col. 16-18.	1-13
A	US,A, 4,864,289 (Nishi et al.) 05 September 1989 See Figures 10-11 and Col. 9-10.	1-13
Y	US,A, 4,951,229 (DiNicola et al.) 21 August 1990 See the entire document.	1-13
Y	US,A, 5,131,080 (Fredrickson et al.) 14 July 1992 See Col. 9-12.	1-13
Y,E	US,A, 5,175,815 (Wada) 29 December 1992 See the entire document.	1-13

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be part of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 24 JANUARY 1993	Date of mailing of the international search report 16 FEB 1993
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. NOT APPLICABLE	Authorized officer HEATHER HERNDON Telephone No. (703) 305-9793

Form PCT/ISA/210 (second sheet)(July 1992)*