IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF WISCONSIN

SILICON GRAPHICS, INC. Plaintiff, v. ATI TECHNOLOGIES INC. Defendant.	Civ. Action No. 06-C-0611-C **AMENDED COMPLAINT FOR PATENT INFRINGEMENT**

Plaintiff Silicon Graphics, Inc., for its amended complaint against defendant ATI Technologies Inc., states as follows:

PARTIES

1. Plaintiff Silicon Graphics, Inc. ("SGI") is a Delaware corporation with its corporate offices in Mountain View, California and research and manufacturing facilities in Chippewa Falls, Wisconsin.

2. Defendant ATI Technologies Inc. ("ATI") is a Canadian corporation formed under the Business Corporations Act (Ontario) with its principal and head office at Markham, Ontario, Canada.

JURISDICTION AND VENUE

3. This is an action for patent infringement over which this Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

4. This Court has personal jurisdiction over ATI. ATI has transacted business within this District and specifically performed acts of patent infringement in or directed to this District.

5. Venue properly lies within this District under 28 U.S.C. § 1391 and §1400(b). Defendant ATI has committed acts of patent infringement within this District, by promoting, selling and causing to be sold the accused products in this District.

COUNT I – PATENT INFRINGEMENT OF U.S. PATENT NO. 6,650,327

6. On November 18, 2003, United States Patent No. 6,650,327 ("the '327 patent") entitled "Display System Having Floating Point Rasterization and Floating Point Framebuffering" was duly and legally issued to inventors John M. Airey, Mark S. Peercy, Robert A. Drebin, John Montrym, David L. Dignam, Christopher J. Migdal and Danny D. Loh. SGI is the owner by assignment of the '327 patent. Attached as Appendix A is a true and correct copy of the '327 patent.

7. Defendant ATI has infringed and continues to infringe the '327 patent by making, using, selling and offering to sell infringing ATI Radeon® products for use in desktop, laptop and server/workstation computing. ATI's infringing conduct has caused SGI substantial damages, and, unless enjoined, will cause irreparable injury to SGI, its operations, reputation and good will.

8. SGI has provided legal notice to ATI of its infringing conduct. Notwithstanding this notice, ATI continues to infringe the '327 patent. ATI's infringement is deliberate, willful and wanton, and will continue unless enjoined by this Court.

COUNT II – PATENT INFRINGEMENT OF U.S. PATENT NO. 6,292,200

9. On September 18, 2001, United States Patent No. 6,292,200 ("the '200 patent") entitled "Apparatus And Method For Utilizing Multiple Rendering Pipes For A Single 3-D Display" was duly and legally issued to inventors Andrew Bowen, Dawn Maxon and Gregory

Buchner. SGI is the owner by assignment of the '200 patent. Attached as Appendix B is a true and correct copy of the '200 patent.

10. Defendant ATI has infringed and continues to infringe the '200 patent by making, using, selling and offering to sell infringing ATI CrossFire Systems and Radeon® products compatible for use in CrossFire systems. ATI's infringing conduct has caused SGI substantial damages, and, unless enjoined, will cause irreparable injury to SGI, its operations, reputation and good will.

11. ATI has knowledge of the '200 patent and the infringement thereof. Notwithstanding such knowledge, ATI continues to infringe the '200 patent. Said infringement is deliberate, willful and wanton, and will continue unless enjoined by this Court.

COUNT III – PATENT INFRINGEMENT OF U.S. PATENT NO. 6,885,376

12. On April 26, 2005, United State Patent No. 6,885,376 ("the '376 patent") entitled "System, Method, And Computer Program Product For Near-Real Time Load Balancing Across Multiple Rendering Pipelines" was duly and legally issued to inventors Svend Tang-Petersen and Yair Kurzion. SGI is the owner by assignment of the '376 patent. Attached as Appendix C is a true and correct copy of the '376 patent.

13. Defendant ATI has infringed and continues to infringe the '376 patent by making, using, selling and offering to sell infringing ATI CrossFire Systems and Radeon® products compatible for use in CrossFire systems. ATI's infringing conduct has caused SGI substantial damages, and, unless enjoined, will cause irreparable injury to SGI, its operations, reputation and good will.

14. ATI has knowledge of the '376 patent and the infringement thereof. Notwithstanding such knowledge, ATI continues to infringe the '376 patent. Said infringement is deliberate, willful and wanton, and will continue unless enjoined by this Court.

RELIEF REQUESTED

WHEREFORE, Plaintiff SGI requests that the Court enter a judgment in SGI's favor and against Defendant ATI, and provide SGI the following relief:

A. Order, adjudge and decree that ATI has infringed the '327, '200 and '376 patents in violation of 35 U.S.C. § 271;

B. Issue permanent injunctive relief prohibiting ATI and its respective parents, subsidiaries, principals, officers, directors, agents, attorneys, employees and all others in privity with it from infringing the '327, '200 and '376 patents, pursuant to 35 U.S.C. § 283;

C. Award SGI its damages for patent infringement, and prejudgment interest and costs against ATI pursuant to 35 U.S.C. § 284;

D. Order, adjudge and decree that ATI's infringement of the '327, '200 and '376 patents has been deliberate, willful and wanton;

E. Order, adjudge and decree that ATI's infringement of the '327, '200 and '376 patents has been exceptional under 35 U.S.C. § 285;

F. Trebling said damage award under 35 U.S.C. § 284;

G. Award SGI its reasonable attorneys' fees under 35 U.S.C. § 285; and

H. Award such other and further relief as the Court may deem just and proper.

JURY DEMAND

Plaintiff SGI requests a trial by jury.

Dated: November 30, 2006

Respectfully submitted,

/s/ Edward J. Pardon

Jeffrey S. Ward
Edward J. Pardon
MICHAEL BEST & FRIEDRICH LLP
One South Pinckney Street, Suite 700
Madison, WI 53703-4257
(608) 257-3501
(608) 283-2275 (Fax)

Of counsel:

James M. Bollinger
Philip L. Hirschhorn
Steven Underwood
MORGAN, LEWIS & BOCKIUS LLP
101 Park Avenue
New York, New York 10178
(212) 309-6000
(212) 309-6001 (Fax)

Attorneys for Plaintiff Silicon Graphics, Inc.

CERTIFICATE OF SERVICE

I, Susan Bunge, hereby certify that on the 30th day of November, 2006, a true and correct copy of the Amended Complaint for Patent Infringement was served via e-mail and U.S. Mail addressed to the following counsel of record:

Daniel W. Hildebrand
Joseph A. Ranney
DeWitt, Ross & Stevens S.C.
2 East Mifflin Street, Suite 600
Madison, WI 53703
dwh@dewittross.net
jar@dewittross.net

William H. Manning
Cole M. Fauver
William A. Webb
Brian A. Mayer
Robins, Kaplan, Miller & Ciresi, L.L.P.
800 LaSalle Avenue, Suite 2800
Minneapolis, MN 55402-2015
whmanning@rkmc.com
cmfauver@rkmc.com
wawebb@rkmc.com
bamayer@rkmc.com

/s/ Susan Bunge
Susan Bunge

Appendix A

US006650327B1

(12) **United States Patent**
Airey et al.

(10) **Patent No.: US 6,650,327 B1**
(45) **Date of Patent: Nov. 18, 2003**

(54) **DISPLAY SYSTEM HAVING FLOATING POINT RASTERIZATION AND FLOATING POINT FRAMEBUFFERING**

(75) Inventors: **John M. Airey**, Moutain View, CA (US); **Mark S. Peercy**, Sunnyvale, CA (US); **Robert A. Drebin**, Palo Alto, CA (US); **John Montrym**, Los Altos, CA (US); **David L. Dignam**, Belmont, CA (US); **Christopher J. Migdal**, Mountain View, CA (US); **Danny D. Loh**, Fremont, CA (US)

(73) Assignee: **Silicon Graphics, Inc.**, Mountain View, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/098,041**

(22) Filed: **Jun. 16, 1998**

(Under 37 CFR 1.47)

(51) **Int. Cl.**[7] **G06T 5/391**
(52) **U.S. Cl.** **345/431**; 345/422; 365/189.05
(58) **Field of Search** 345/431, 520, 345/523, 422, 503, 153, 502, 426, 430, 196, 950; 708/606, 512; 365/189.05, 230.08; 326/62; 375/372; 348/419

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,745,125	A *	4/1998	Deering et al.	345/503
5,844,571	A *	12/1998	Narayanaswami	345/422
5,926,406	A *	7/1999	Tucker et al.	708/606
5,995,121	A *	11/1999	Alcorn et al.	345/520
5,995,122	A *	11/1999	Hsich et al.	345/523

OTHER PUBLICATIONS

Larson, G.W. et al., "A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes," *IEEE Transactions On Visualization and Computer Graphics*, vol. 3, No. 4, IEEE, pp. 291–306 (Oct.–Dec. 1997).

Larson, G.W., "LogLuv Encoding for Full–Gamut, High–Dynamic Range Images," *Journal of Graphic Tools*, vol. 3, No. 1, AK Peters, pp. 15–31 (Submitted to Journal on Aug. 26, 1998).

Larson, G.W. and Shakespheare, R., *Rendering with Radiance: The Art and Science of Lighting Visualization*, Morgan Kaufmann Publishers, Entire book submitted (1997).

Lastra, A. et al., "Real–Time Programmable Shading," *Proceedings of the 1995 Symposium of Interactive 3D Graphics*, ACM, pp. 59–66 (1995).

Olano, M. and Lastra, A., "A Shading Language on Graphics Hardware: The PixelFlow Shading System," *Proceedings of SIGGRAPH 98*, pp. 1–10 (Conference Dates: Jul 19–24, 1998).

Rushmeier, H. et al., "Comparing Real and Synthetic Images: Some Ideas About Metrics," *Sixth Eurographics Workshop on Rendering*, Springer–Verlag, pp. 82–91 (Jun. 1995).

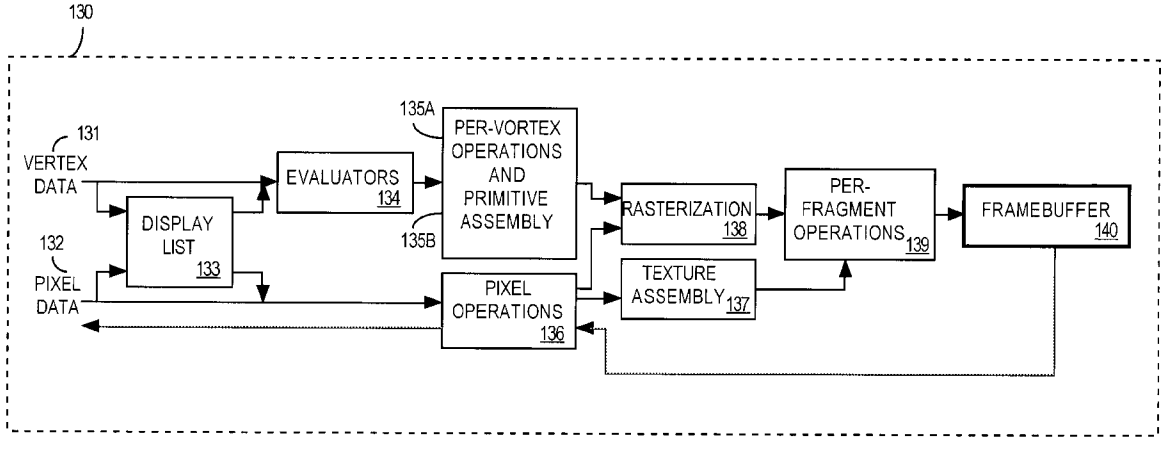
(List continued on next page.)

Primary Examiner—Mark R. Powell
Assistant Examiner—Thu-Thao Havan

(57) **ABSTRACT**

A floating point rasterization and frame buffer in a computer system graphics program. The rasterization, fog, lighting, texturing, blending, and antialiasing processes operate on floating point values. In one embodiment, a 16-bit floating point format consisting of one sign bit, ten mantissa bits, and five exponent bits (s10e5), is used to optimize the range and precision afforded by the 16 available bits of information. In other embodiments, the floating point format can be defined in the manner preferred in order to achieve a desired range and precision of the data stored in the frame buffer. The final floating point values corresponding to pixel attributes are stored in a frame buffer and eventually read and drawn for display. The graphics program can operate directly on the data in the frame buffer without losing any of the desired range and precision of the data.

31 Claims, 7 Drawing Sheets



OTHER PUBLICATIONS

Rushmeier, H.E. and Ward, G.J., "Energy Preserving Non–Linear Filters," *Computer Graphics Proceedings*, Annual Conference Series, ACM, pp. 131–138 (Conference Dates: Jul. 24–29, 1994).

Ward, G., "A Contrast–Based Scalefactor for Luminance Display," *Graphics Gems IV*, Academic Press, Inc., pp. 415–421 (1994).

Ward, G.J. and Rubinstein, F.M., "A New Technique for Computer Simulation of Illuminated Spaces," *Journal of the Illuminating Engineering Society*, vol. 17, No. 1, The Illuminating Engineering Society, pp. 80–91 (Winter 1988).

Ward, G.J. et al., "A Ray Tracing Solution for Diffuse Interreflection," *Computer Graphics*, vol. 22, No. 4, ACM, pp. 85–92 (Aug. 1988).

Ward, G.J. "Adaptive Shadow Testing for Ray Tracing," *Proceedings of the 1991 Eurographics Rendering Workshop*, Springer–Verlag, pp. 11–20 (1991).

Ward, G.J. and Heckbert, P.S., "Irradiance Gradients," *Third Annual Eurographics Workship on Rendering*, Springer–Verlag, pp. 85–98 (May 1992).

Ward, G.J., "Making global illumination user–friendly," *Sixth Eurographics Workshop on Rendering*, Springer–Verlag, pp. 104–114 (Jun. 1995).

Ward, G.J. "Measuring and Modeling Anistrophic Reflection," *Computer Graphics*, vol. 26, No. 2, ACM, pp. 265–272 (Jul.1992).

Ward, G.J. "The RADIANCE Lighting Simulation and Rendering System," Computer Graphics Proceedings, *Annual Conference Series*, pp. 459–472 (Conference Dates: Jul. 24–29, 1994).

Ward, G.J., "Visualization," LD+A (*Lighting Design + Application*), pp. 4–5 & 14–20 (Jun. 1990).

Peercy, M.S. and Hesselink, L., "Dichromatic Color Representations for Complex Display Systems," *Proceedings Visualization '93*, IEEE Computer Society Press, pp. 212–219 and CP–21 (Oct. 25–29, 1993).

Peercy, M. et al., "Efficient Bump Mapping Hardware," *Computer Graphics Proceedings SIGGRAPH 97*, ACM, 4 pages (1997).

Peercy, M. S. et al., "Interactive Full Spectral Rendering," *Proceedings 1995 Symposium on Interactive 3D Graphics*, ACM, pp. 67, 68 and 207 (1995).

Peercy, M.S. et al., "Linear Color Representations for Efficient Image Synthesis," *COLOR research and application*, vol. 21, No. 2, Wiley–Interscience, pp. 129–137 (Apr. 1996).

Peercy, M.S., "Linear Color Representation for Full Spectral Rendering," *Proceedings of SIGGRAPH 20th Annual International Conference on Computer Graphics and Interactive Techniques*, ACM, pp. 191–198 (1993).

Peercy, M.S. and Hesselink, L., "Wavelength selection for color holography," *Practical Holography VIII*, vol. 2176, SPIE, pp. 108–118 (Feb. 7–9, 1994).

Peercy, M.S. and Hesselink, L., "Wavelength selection for true–color holography," *Applied Optics*, vol. 33, No. 29, Optical Society of America, pp. 6811–6817.

* cited by examiner

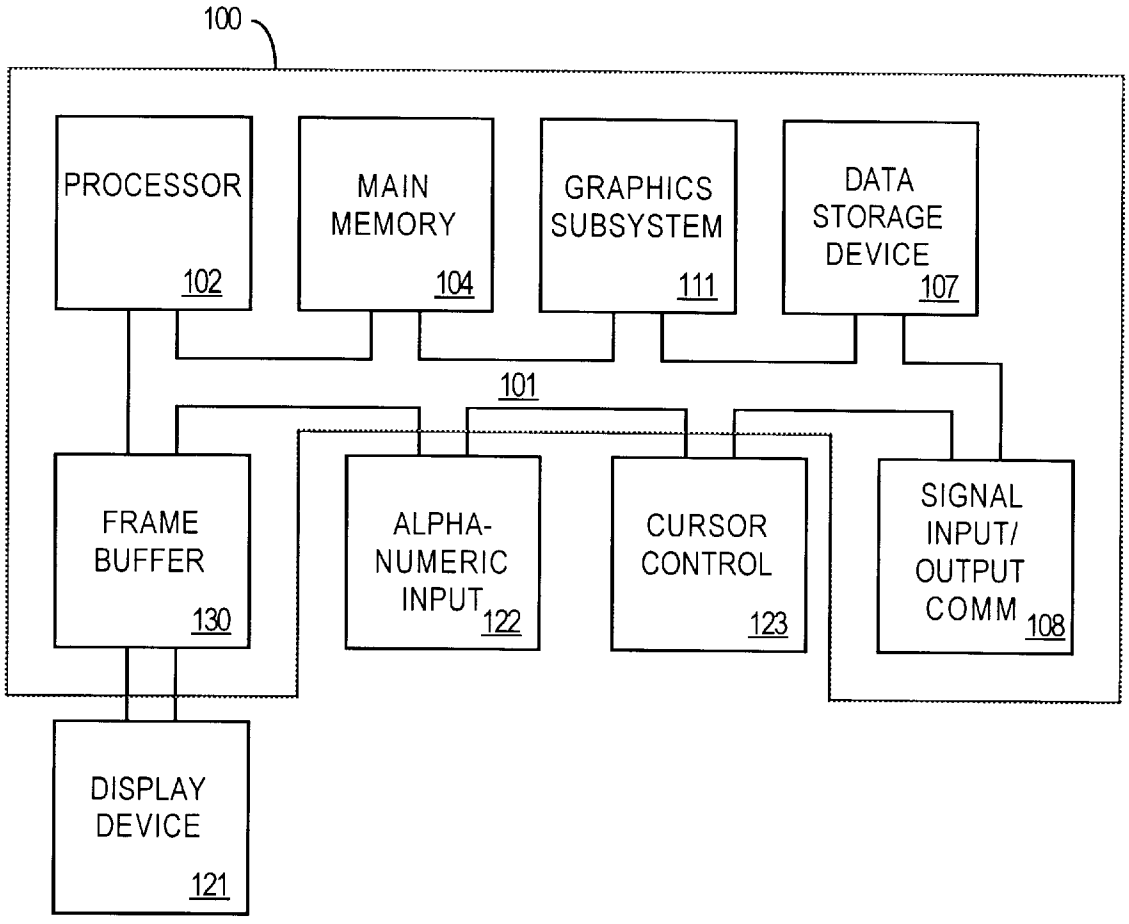

100
PROCESSOR
102
MAIN MEMORY
104
GRAPHICS SUBSYSTEM
111
DATA STORAGE DEVICE
107
101
FRAME BUFFER
130
ALPHA-NUMERIC INPUT
122
CURSOR CONTROL
123
SIGNAL INPUT/ OUTPUT COMM
108
DISPLAY DEVICE
121

FIG. 1

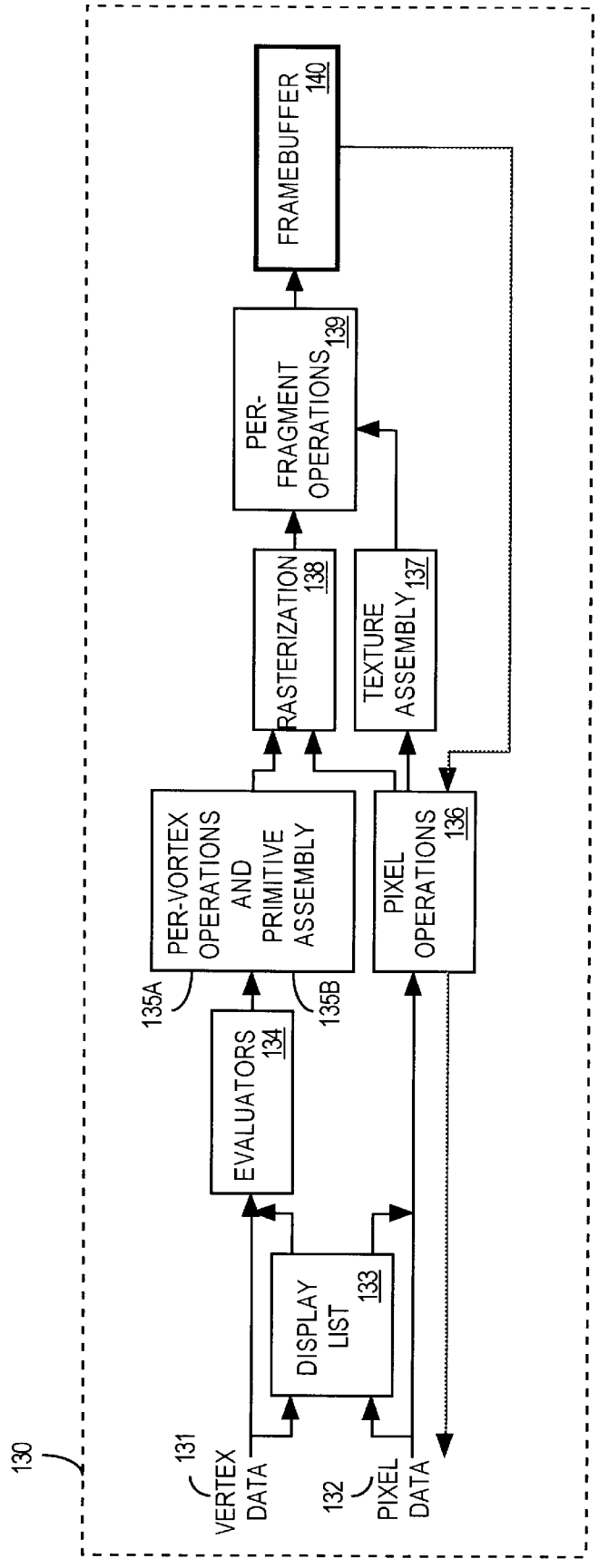

130
131
VERTEX DATA
132
PIXEL DATA
DISPLAY LIST 133
EVALUATORS 134
135A
135B
PER-VORTEX OPERATIONS AND PRIMITIVE ASSEMBLY
PIXEL OPERATIONS 136
TEXTURE ASSEMBLY 137
RASTERIZATION 138
PER-FRAGMENT OPERATIONS 139
FRAMEBUFFER 140

FIG. 2

VALUE	CONDITIONS**
(-1)^sx2^(e-16)x1.m	00000<e<11111
(-1)^sx2^15x1.m	e==11111,m!=1111111111
(-1)^sx2^-16x1.m	e==00000,m!=0000000000
zero	e==00000,s==0,m==0000000000
NaN*	e==00000,s==1,m==0000000000
positive infinity	e==11111,s==0,m==1111111111
negative infinity	e==11111,s==1,m==1111111111

* NaN: "Not a number," which is generated as the result of an invalid operation and also represents the concept of "negative zero"

** Extrapolation to s11e5 is readily achievable

FIG. 3

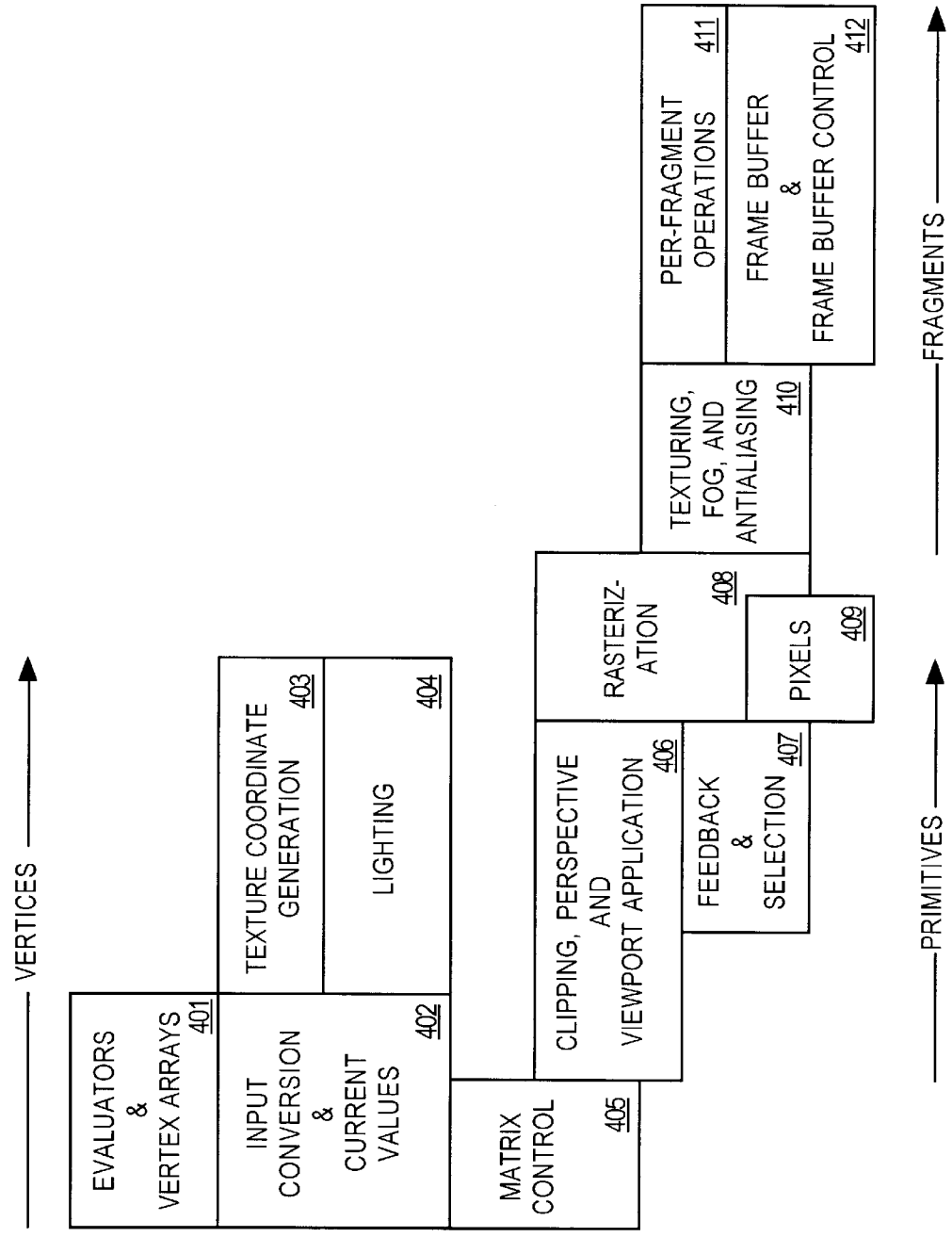

VERTICES
EVALUATORS & VERTEX ARRAYS 401
INPUT CONVERSION & CURRENT VALUES 402
TEXTURE COORDINATE GENERATION 403
LIGHTING 404
MATRIX CONTROL 405
CLIPPING, PERSPECTIVE AND VIEWPORT APPLICATION 406
FEEDBACK & SELECTION 407
RASTERIZ-ATION 408
PIXELS 409
TEXTURING, FOG, AND ANTIALIASING 410
PER-FRAGMENT OPERATIONS 411
FRAME BUFFER & FRAME BUFFER CONTROL 412
PRIMITIVES
FRAGMENTS

FIG. 4

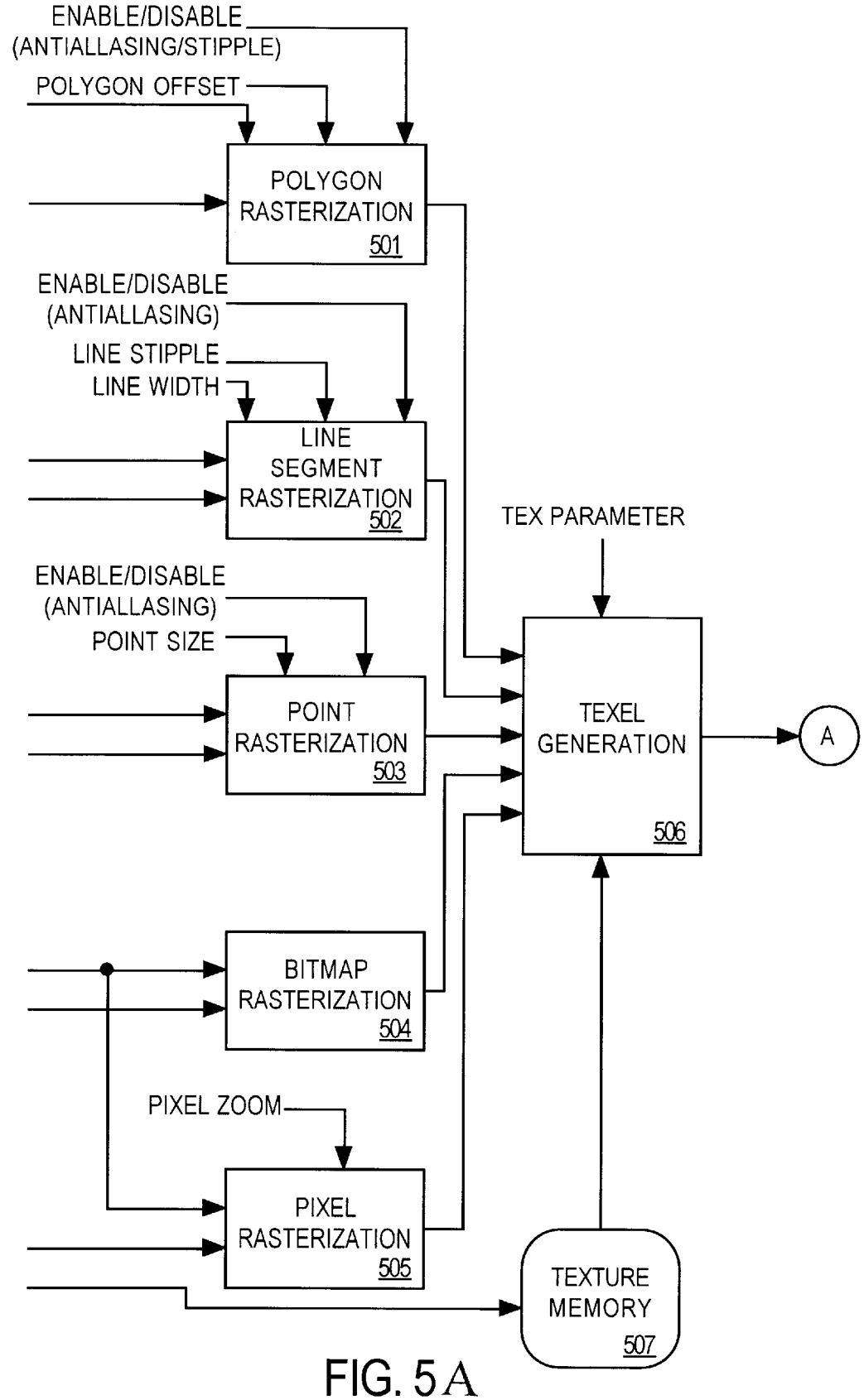

ENABLE/DISABLE
(ANTIALLASING/STIPPLE)
POLYGON OFFSET
POLYGON RASTERIZATION
501
ENABLE/DISABLE
(ANTIALLASING)
LINE STIPPLE
LINE WIDTH
LINE SEGMENT RASTERIZATION
502
TEX PARAMETER
ENABLE/DISABLE
(ANTIALLASING)
POINT SIZE
POINT RASTERIZATION
503
TEXEL GENERATION
506
A
BITMAP RASTERIZATION
504
PIXEL ZOOM
PIXEL RASTERIZATION
505
TEXTURE MEMORY
507

FIG. 5A

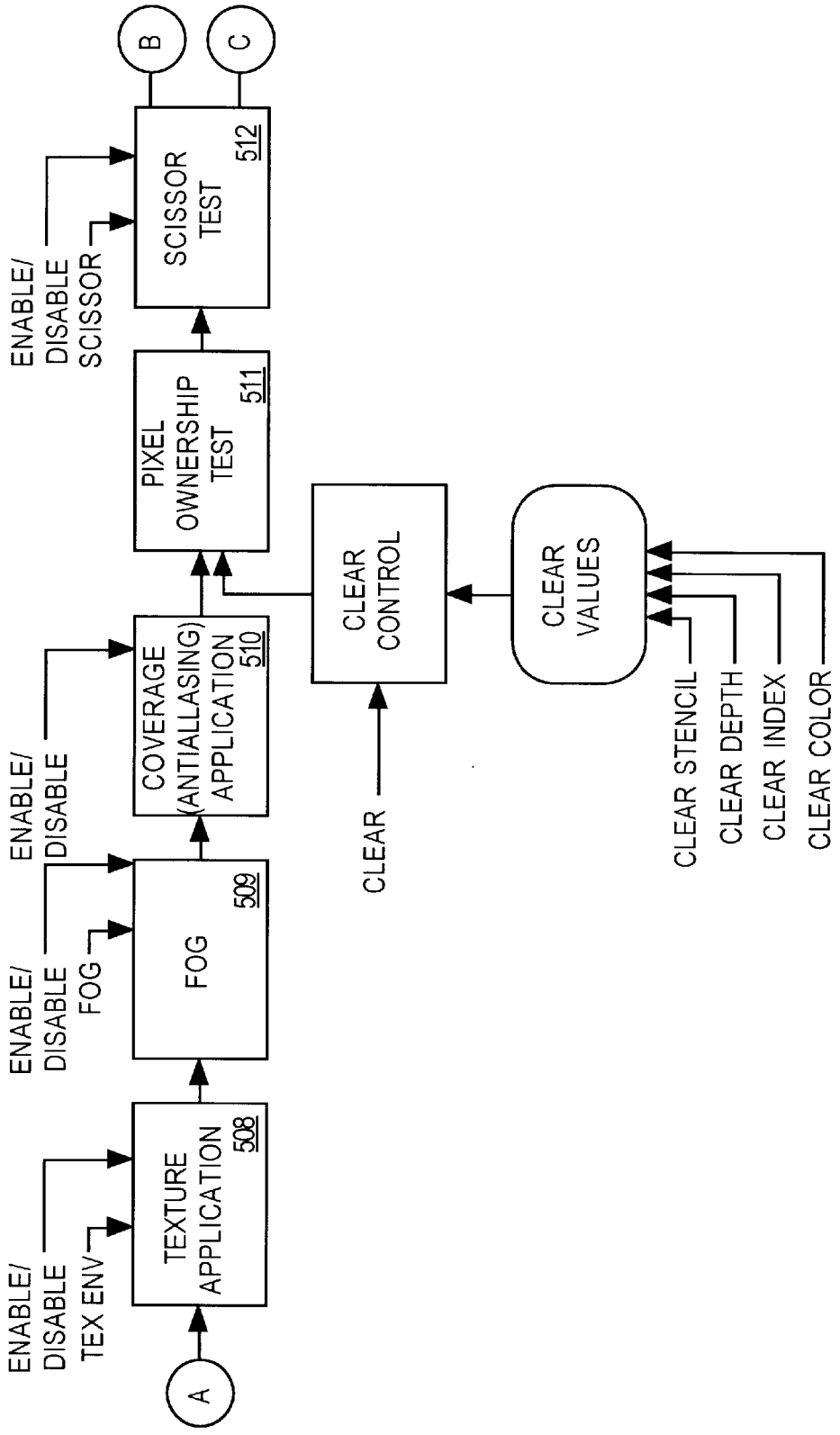

A
B
C
ENABLE/
DISABLE
TEX ENV
TEXTURE
APPLICATION
508
ENABLE/
DISABLE
FOG
FOG
509
ENABLE/
DISABLE
COVERAGE
(ANTIALLASING)
APPLICATION
510
PIXEL
OWNERSHIP
TEST
511
ENABLE/
DISABLE
SCISSOR
SCISSOR
TEST
512
CLEAR
CLEAR
CONTROL
CLEAR
VALUES
CLEAR STENCIL
CLEAR DEPTH
CLEAR INDEX
CLEAR COLOR

FIG. 5 B

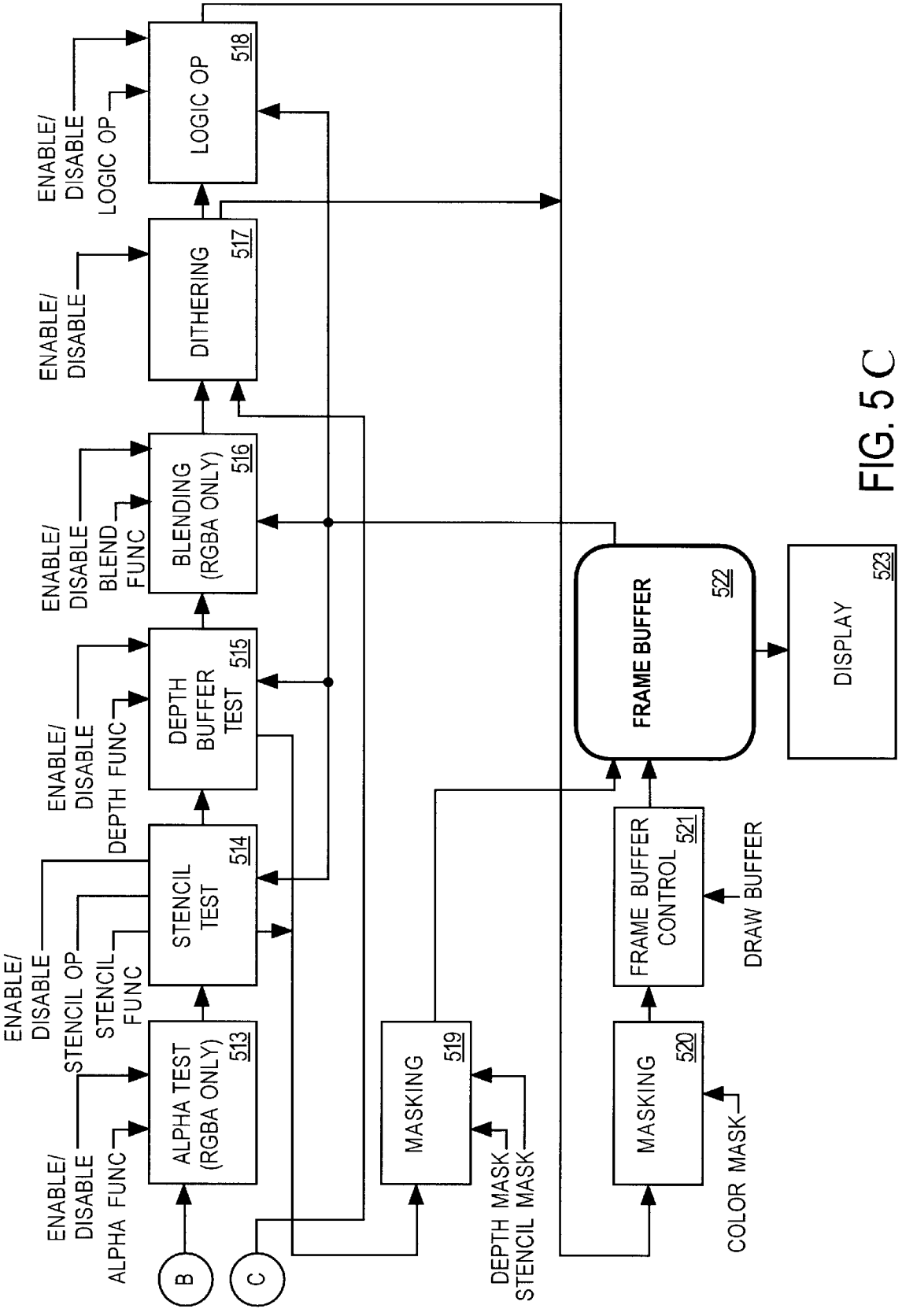

ENABLE/ DISABLE ALPHA FUNC
ENABLE/ DISABLE STENCIL OP
STENCIL FUNC
ENABLE/ DISABLE DEPTH FUNC
ENABLE/ DISABLE BLEND FUNC
ENABLE/ DISABLE
ENABLE/ DISABLE LOGIC OP
ALPHA TEST (RGBA ONLY) 513
STENCIL TEST 514
DEPTH BUFFER TEST 515
BLENDING (RGBA ONLY) 516
DITHERING 517
LOGIC OP 518
MASKING 519
MASKING 520
FRAME BUFFER CONTROL 521
FRAME BUFFER 522
DISPLAY 523
DRAW BUFFER
COLOR MASK
DEPTH MASK
STENCIL MASK
B
C

FIG. 5C

DISPLAY SYSTEM HAVING FLOATING POINT RASTERIZATION AND FLOATING POINT FRAMEBUFFERING

TECHNICAL FIELD

This invention relates to the field of computer graphics. Specifically, the present invention pertains to an apparatus and process relating to floating point rasterization and framebuffering in a graphics display system.

BACKGROUND ART

Graphics software programs are well known in the art. A graphics program consists of commands used to specify the operations needed to produce interactive three-dimensional images. It can be envisioned as a pipeline through which data pass, where the data are used to define the image to be produced and displayed. The user issues a command through the central processing unit of a computer system, and the command is implemented by the graphics program. At various points along the pipeline, various operations specified by the user's commands are carried out, and the data are modified accordingly. In the initial stages of the pipeline, the desired image is framed using geometric shapes such as lines and polygons (usually triangles), referred to in the art as "primitives." The vertices of these primitives define a crude shell of the objects in the scene to be rendered. The derivation and manipulation of the multitudes of vertices in a given scene, entail performing many geometric calculations.

In the next stages, a scan conversion process is performed to specify which picture elements or "pixels" of the display screen, belong to which of the primitives. Many times, portions or "fragments" of a pixel fall into two or more different primitives. Hence, the more sophisticated computer systems process pixels on a per fragment basis. These fragments are assigned attributes such as color, perspective (i.e., depth), and texture. In order to provide even better quality images, effects such as lighting, fog, and shading are added. Furthermore, anti-aliasing and blending functions are used to give the picture a smoother and more realistic appearance. The processes pertaining to scan converting, assigning colors, depth buffering, texturing, lighting, and anti-aliasing are collectively known as rasterization. Today's computer systems often contain specially designed rasterization hardware to accelerate 3-D graphics.

In the final stage, the pixel attributes are stored in a frame buffer memory. Eventually, these pixel values are read from the frame buffer and used to draw the three-dimensional images on the computer screen. One prior art example of a computer architecture which has been successfully used to build 3-D computer imaging systems is the Open GL architecture invented by Silicon Graphics, Inc. of Mountain View, Calif.

Currently, many of the less expensive computer systems use its microprocessor to perform the geometric calculations. The microprocessor contains a unit which performs simple arithmetic functions, such as add and multiply. These arithmetic functions are typically performed in a floating point notation. Basically, in a floating point format, data is represented by the product of a fraction, or mantissa, and a number raised to an exponent; in base 10, for example, the number "n" can be presented by $n=m\times10^e$, where "m" is the mantissa and "e" is the exponent. Hence, the decimal point is allowed to "float." Hence, the unit within the microprocessor for performing arithmetic functions is commonly

referred to as the "floating point unit." This same floating point unit can be used in executing normal microprocessor instructions as well as in performing geometric calculations in support of the rendering process. In order to increase the speed and increase graphics generation capability, some computer systems utilize a specialized geometry engine, which is dedicated to performing nothing but geometric calculations. These geometry engines have taken to handling its calculations on a floating point basis.

Likewise, special hardware have evolved to accelerate the rasterization process. However, the rasterization has been done in a fixed point format rather than a floating point format. In a fixed point format, the location of the decimal point within the data field for a fixed point format is specified and fixed; there is no exponent. The main reason why rasterization is performed on a fixed point format is because it is much easier to implement fixed point operations in hardware. For a given set of operations, a fixed point format requires less logic and circuits to implement in comparison to that of a floating point format. In short, the floating point format permits greater flexibility and accuracy when operating on the data in the pipeline, but requires greater computational resources. Furthermore, fixed point calculations can be executed much faster than an equivalent floating point calculation. As such, the extra computational expenses and time associated with having a floating point rasterization process has been prohibitive when weighed against the advantages conferred.

In an effort to gain the advantages conferred by operating on a floating point basis, some prior art systems have attempted to perform floating point through software emulation, but on a fixed point hardware platform. However, this approach is extremely slow, due to the fact that the software emulation relies upon the use of a general purpose CPU. Furthermore, the prior art software emulation approach lacked a floating point frame buffer and could not be scanned out. Hence, the final result must be converted back to a fixed point format before being drawn for display. Some examples of floating point software emulation on a fixed point hardware platform include Pixar's RenderMan software and software described in the following publications: Olano, Marc and Anselmo Lastra, "A Shading Language on Graphics Hardware: The PixelFlow Shading System," *Proceedings of SIGGRAPH 98, Computer Graphics*, Annual Conference Series, ACM SIGGRAPH, 1998; and Anselmo Lastra, Steve Molnar, Marc Olano, and Yulan Wang, "Real-Time Programmable Shading," Proceedings of the 1995 Symposium of Interactive 3D Graphics (Monterey, Calif., Apr. 9–12, 1995), ACM SIGGRAPH, New York, 1995.

But as advances in semiconductor and computer technology enable greater processing power and faster speeds; as prices drop; and as graphical applications grow in sophistication and precision, it has been discovered by the present inventors that it is now practical to implement some portions or even the entire rasterization process by hardware in a floating point format.

In addition, in the prior art, data is stored in the frame buffer in a fixed point format. This practice was considered acceptable because the accuracy provided by the fixed point format was considered satisfactory for storage purposes. Other considerations in the prior art were the cost of hardware (e.g., memory chips) and the amount of actual physical space available in a computer system, both of which limited the number of chips that could be used and thus, limited the memory available. Thus, in the prior art, it was not cost beneficial to expand the memory needed for the

frame buffer because it was not necessary to increase the accuracy of the data stored therein.

Yet, as memory chips become less expensive, the capability of a computer system to store greater amounts of data increases while remaining cost beneficial. Thus, as memory capacity increases and becomes less expensive, software applications can grow in complexity; and as the complexity of the software increases, hardware and software designs are improved to increase the speed at which the software programs can be run. Hence, due to the improvements in processor speed and other improvements that make it practical to operate on large amounts of data, it is now possible and cost beneficial to utilize the valuable information that can be provided by the frame buffer.

Also, it is preferable to operate directly on the data stored in the frame buffer. Operating directly on the frame buffer data is preferable because it allows changes to be made to the frame buffer data without having to unnecessarily repeat some of the preceding steps in the graphics pipeline. The information stored in the frame buffer is a rich source of data that can be used in subsequent graphics calculations. However, in the prior art, some steps typically need to be repeated to restore the accuracy of the data and allow it to be operated on before it is read back into the frame buffer. In other words, data would need to be read from the frame buffer and input into the graphics program at or near the beginning of the program, so that the data could be recalculated in the floating point format to restore the required precision and range. Thus, a disadvantage to the prior art is that additional steps are necessary to allow direct operation on the frame buffer data, thus increasing the processing time. This in turn can limit other applications of the graphics program; for example, in an image processing application, an image operated on by the graphics program and stored in the frame buffer could be subsequently enhanced through direct operation on the frame buffer data. However, in the prior art, the accuracy necessary to portray the desired detail of the image is lost, or else the accuracy would have to be regenerated by repeated passes through the graphics pipeline.

Another drawback to the prior art is the limited ability to take advantage of hardware design improvements that could be otherwise employed, if direct operation on the frame buffer without the disadvantages identified above was possible. For example, a computer system could be designed with processors dedicated to operating on the frame buffer, resulting in additional improvements in the speed at which graphics calculations are performed.

Consequently, the use of fixed point formatting in the frame buffer is a drawback in the prior art because of the limitations imposed on the range and precision of the data stored in the frame buffer. The range of data in the prior art is limited to 0 to 1, and calculation results that are outside this range must be set equal to either 0 or 1, referred to in the art as "clamping." Also, the prior art does not permit small enough values to be stored, resulting in a loss of precision because smaller values must be rounded off to the smallest value that can be stored. Thus, the accuracy of the data calculated in the graphics pipeline is lost when it is stored in the frame buffer. Moreover, in the prior art, the results that are calculated by operating directly on the data in the frame buffer are not as accurate as they can and need to be. Therefore, a drawback to the prior art is that the user cannot exercise sufficient control over the quality of the frame buffer data in subsequent operations.

Thus, there is a need for a graphical display system which predominately uses floating point throughout the entire geometry, rasterization, and frame buffering processes. The present invention provides one such display system. Furthermore, the display system of the present invention is designed to be compatible to a practical extent with existing computer systems and graphics subsystems.

SUMMARY OF THE INVENTION

The present invention provides a display system and process whereby the geometry, rasterization, and frame buffer predominately operate on a floating point format. Vertex information associated with geometric calculations are specified in a floating point format. Attributes associated with pixels and fragments are defined in a floating point format. In particular, all color values exist as floating point format. Furthermore, certain rasterization processes are performed according to a floating point format. Specifically, the scan conversion process is now handled entirely on a floating point basis. Texturing, fog, and antialiasing all operate on floating point numbers. The texture map stores floating point texel values. The resulting data are read from, operated on, written to and stored in the frame buffer using floating point formats, thereby enabling subsequent graphics operations to be performed directly on the frame buffer data without any loss of accuracy.

Many different types of floating point formats exist and can be used to practice the present invention. However, it has been discovered that one floating point format, known as "s10e5," has been found to be particularly optimal when applied to various aspects of graphical computations. As such, it is used extensively throughout the geometric, rasterization and frame buffer processes of the present invention. To optimize the range and precision of the data in the geometry, rasterization, and frame buffer processes, this particular s10e5 floating point format imposes a 16-bit format which provides one sign bit, ten mantissa bits, and five exponent bits. In another embodiment, a 17-bit floating point format designated as "s11e5" is specified to maintain consistency and ease of use with applications that uses 12 bits of mantissa. Other formats may be used in accordance with the present invention depending on the application and the desired range and precision.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows a computer graphics system upon which the present invention may be practiced.

FIG. **2** is a flow chart illustrating the stages for processing data in a graphics program in accordance with the present invention.

FIG. **3** is a tabulation of the representative values for all possible bit combinations used in the preferred embodiment of the present invention.

FIG. **4** shows a block diagram of the currently preferred embodiment of the display system.

FIG. **5** shows a more detailed layout of a display system for implementing the floating point present invention.

BEST MODE FOR CARRYING OUT THE INVENTION

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings. While the invention will be described in conjunction with the preferred embodiments, it will be understood that they are not intended to limit the invention to these embodiments. On the contrary, the invention is intended to cover alternatives,

modifications and equivalents, which may be included within the spirit and scope of the invention as defined by the appended claims. Furthermore, in the following detailed description of the present invention, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be obvious to one of ordinary skill in the art that the present invention may be practiced without these specific details. In other instances, well known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

Some portions of the detailed descriptions which follow are presented in terms of procedures, logic blocks, processing, and other symbolic representations of operations on data bits within a computer memory. These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. In the present application, a procedure, logic block, process, or the like, is conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, although not necessarily, these quantities take the form of electrical, or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, fragments, pixels, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "processing," "operating," "calculating," "determining," "displaying," or the like, refer to actions and processes of a computer system or similar electronic computing device. The computer system or similar electronic computing device manipulates and transforms data represented as physical (electronic) quantities within the computer system memories, registers or other such information storage, transmission or display devices. The present invention is well suited to the use of other computer systems, such as, for example, optical and mechanical computers.

Referring to FIG. **1**, a computer graphics system upon which the present invention may be practiced is shown as **100**. System **100** can include any computer-controlled graphics systems for generating complex or three-dimensional images. Computer system **100** comprises a bus or other communication means **101** for communicating information, and a processing means **102** coupled with bus **101** for processing information. System **100** further comprises a random access memory (RAM) or other dynamic storage device **104** (referred to as main memory), coupled to bus **101** for storing information and instructions to be executed by processor **102**. Main memory **104** also may be used for storing temporary variables or other intermediate information during execution of instructions by processor **102**. Data storage device **107** is coupled to bus **101** for storing information and instructions. Furthermore, an input/output (I/O) device **108** is used to couple the computer system **100** onto a network.

Computer system **100** can also be coupled via bus **101** to an alphanumeric input device **122**, including alphanumeric and other keys, that is typically coupled to bus **101** for communicating information and command selections to processor **102**. Another type of user input device is cursor control **123**, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor **102** and for controlling cursor movement on, display **121**. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), which allows the device to specify positions in a plane.

Also coupled to bus **101** is a graphics subsystem **111**. Processor **102** provides the graphics subsystem **111** with graphics data such as drawing commands, coordinate vertex data, and other data related to an object's geometric position, color, and surface parameters. The object data are processed by graphics subsystem **111** in the following four pipelined stages: geometry subsystem, scan conversion subsystem, raster subsystem, and a display subsystem. The geometry subsystem converts the graphical data from processor **102** into a screen coordinate system. The scan conversion subsystem then generates pixel data based on the primitives (e.g., points, lines, polygons, and meshes) from the geometry subsystem. The pixel data are sent to the raster subsystem, whereupon z-buffering, blending, texturing, and anti-aliasing functions are performed. The resulting pixel values are stored in a frame buffer **140**. The frame buffer is element **140**, as shown in FIG. **2** of the present application. The display subsystem reads the frame buffer and displays the image on display monitor **121**.

With reference now to FIG. **2**, a series of steps for processing and operating on data in the graphics subsystem **111** of FIG. **1** are shown. The graphics program **130**, also referred to in the art as a state machine or a rendering pipeline, provides a software interface that enables the user to produce interactive three-dimensional applications on different computer systems and processors. The graphics program **130** is exemplified by a system such as OpenGL by Silicon Graphics; however, it is appreciated that the graphics program **130** is exemplary only, and that the present invention can operate within a number of different graphics systems or state machines other than OpenGL.

With reference still to FIG. **2**, graphics program **130** operates on both vertex (or geometric) data **131** and pixel (or image) data **132**. The process steps within the graphics program **130** consist of the display list **133**, evaluators **134**, per-vertex operations and primitive assembly **135**, pixel operations **136**, texture assembly **137**, rasterization **138**, per-fragment operations **139**, and the frame buffer **140**.

Vertex data **131** and pixel data **132** are loaded from the memory of central processor **102** and saved in a display list **133**. When the display list **133** is executed, the evaluators **134** derive the coordinates, or vertices, that are used to describe points, lines, polygons, and the like, referred to in the art as "primitives." From this point in the process, vertex data and pixel data follow a different route through the graphics program as shown in FIG. **2**.

In the per-vertex operations **135**A, vertex data **131** are converted into primitives that are assembled to represent the surfaces to be graphically displayed. Depending on the programming, advanced features such as lighting calculations may also be performed at the per-vertex operations stage. The primitive assembly **135**B then eliminates unnecessary portions of the primitives and adds characteristics such as perspective, texture, color and depth.

In pixel operations **136**, pixel data may be read from the processor **102** or the frame buffer **140**. A pixel map processes the data from the processor to add scaling, for

example, and the results are then either written into texture assembly **137** or sent to the rasterization step **138**. Pixel data read from the frame buffer **140** are similarly processed within pixel operations **136**. There are special pixel operations to copy data in the frame buffer to other parts of the frame buffer or to texture memory. A single pass is made through the pixel operations before the data are written to the texture memory or back to the frame buffer. Additional single passes may be subsequently made as needed to operate on the data until the desired graphics display is realized.

Texture assembly **137** applies texture images—for example, wood grain to a table top—onto the surfaces that are to be graphically displayed. Texture image data are specified from frame buffer memory as well as from processor **102** memory.

Rasterization **138** is the conversion of vertex and pixel data into "fragments." Each fragment corresponds to a single pixel and typically includes data defining color, depth, and texture. Thus, for a single fragment, there are typically multiple pieces of data defining that fragment.

Per-fragment operations **139** consist of additional operations that may be enabled to enhance the detail of the fragments. After completion of these operations, the processing of the fragment is complete and it is written as a pixel to the frame buffer **140**. Thus, there are typically multiple pieces of data defining each pixel.

With reference still to FIG. **2**, the present invention uses floating point formats in the process steps **131** through **139** of graphics program **130**. In other words, the vertex data is given in floating point. Likewise, the pixel data is also given in floating point. The display list **133** and evaluators **134** both operate on floating point values. All pixel operations in block **136** are performed according to a floating point format. Similarly, per-vertex operations and primitive assembly **135**A are performed on a floating point format. The rasterization **138** is performed according to a floating point format. In addition, texturing **137** is done on floating point basis, and the texture values are stored in the texture memory as floating point. All per-fragment operations are performed on a floating point basis. Lastly, the resulting floating point values are stored in the frame buffer **140**. Thereby, the user can operate directly on the frame buffer data.

For example, the maximum value that can be used in the 8-bit fixed point format is 127 (i.e., 2^8-1), which is written as 01111111 in binary, where the first digit represents the sign (positive or negative) and the remaining seven digits represent the number 127 in binary. In the prior art, this value is clamped and stored as 1.0 in the frame buffer. In an 8-bit floating point format, a value "n" is represented by the format n=s_eee_mmmmm, where "s" represents the sign, "e" represents the exponent, and "m" represents the mantissa in the binary formula $n=m\times 2^e$. Thus, in a floating point format, the largest number that can be written is 31×2^7, also written in binary as 01111111. In the present invention, the value is written to and stored in the frame buffer without being clamped or otherwise changed. Thus, use of the floating point format in the frame buffer permits greater flexibility in how a number can be represented, and allows for a larger range of values to be represented by virtue of the use a portion of the data field to specify an exponent.

The present invention uses floating point formats in the frame buffer to increase the range of the data. "Range" is used herein to mean the distance between the most negative valule and the most positive value of the data that can be stored. The present invention permits absolute values much greater than 1.0 to be stored in the frame buffer, thereby enabling the user to generate a greater variety of graphics images. Increased range is particularly advantageous when the user performs operations such as addition, multiplication, or other operations well known and practiced in the art, directly on the data in the frame buffer. Such operations can result in values greater than 1.0, and in the present invention these values can be written to and stored in the frame buffer without clamping. Thus, the present invention results in a substantial increase in the range of data that can be stored in the frame buffer, and preserves the range of data that was determined in steps **131** through **139** of the graphics program illustrated in FIG. **2**.

With reference still to FIG. **2**, the present invention utilizes floating point formats in the frame buffer **140** to maintain the precision of the data calculated in the preceding steps **131** through **139** of the graphics program **130**. "Precision" is used herein to mean the increment between any two consecutive stored values of data. Precision is established by the smallest increment that can be written in the format being used. Increased precision is an important characteristic that permits the present invention to store a greater number of gradations of data relative to the prior art, thereby providing the user with a greater degree of control over the graphics images to be displayed. This characteristic is particularly advantageous when the user performs an operation such as addition, multiplication, or other operations well known and practiced in the art, on the data in the frame buffer. Such operations can result in values that lie close to each other, i.e., data that are approximately but not equal to each other. In the present invention; data typically can be stored without having to be rounded to a value permitted by the precision of the frame buffer. If rounding is needed, the present invention permits the data to be rounded to a value very close to the calculated values. Thus, the present invention results in a substantial increase in the precision of the data that can be stored in the frame buffer relative to the prior art, and preserves the precision of the data that was determined in steps **131** through **139** of the graphics program illustrated in FIG. **2**.

In one embodiment of the present invention, a 16-bit floating point format is utilized in the frame buffer. The 16 bits available are applied so as to optimize the balance between range and precision. The 16-bit floating point format utilized in one embodiment of the present invention is designated using the nomenclature "s10e5", where "s" specifies one (1) sign bit, "10" specifies ten (10) mantissa bits, and "e5" specifies five (5) exponent bits, with an exponent bias of 16. FIG. **3** defines the represented values for all possible bit combinations for the s10e5 format. In this embodiment, the smallest representable number (i.e., precision) is $1.0000\_0000\_00*2^{-16}$ and the range is plus/minus $1.1111\_1111\_10*2^{15}$. (In base 10, the range corresponds to approximately plus/minus 65,000.) In this embodiment, the range and precision provided by this specification are sufficient for operating directly on the frame buffer. The 16-bit format in this embodiment thus represents a cost-effective alternative to the single precision 32-bit IEEE floating point standard.

However, it is appreciated that different sizes other than 16-bit, such as 12-bit, 17-bit or 32-bit, can be used in accordance with the present invention. In addition, other floating point formats may be used in accordance with the present invention by varying the number of bits assigned to the mantissa and to the exponent (a sign bit is typically but not always needed). Thus a floating point format can

be-specified in accordance with the present invention that results in the desired range and precision. For example, if the format specified is "s9e6" (nine mantissa bits and six exponent bits), then relative to the s10e5 format a greater range of data is defined but the precision is reduced. Also, a 17-bit format designated as "s11e5" may be used in accordance with the present invention to preserve 12 bits of information, for consistency and ease of application with programs and users that work with a 12-bit format.

In the present invention,:.the user can apply the same operation to all of the data in the frame buffer, referred to in the art as Single Instruction at Multiple Data (SIMD). For example, with reference back to FIG. **2**, the user may wish to add an image that is coming down the rendering pipeline **130** to an image already stored in the frame buffer **140**. The image coming down the pipeline is in floating point format, and thus in the present invention is directly added to the data already stored in the frame buffer that is also in floating point format. The present invention permits the results determined by this operation to be stored in the frame buffer without a loss of precision. Also, in the present invention the permissible range is greater than 1.0, thereby permitting the results from the operation to be stored without being clamped.

With continued reference to FIG. **2**, in the present invention the data in the frame buffer **140** are directly operated on within the graphics program without having to pass back through the entire graphics program to establish the required range and precision. For example, it is often necessary to copy the data from the texture memory **137** to the frame buffer **140**, then back to the texture memory and back to the frame buffer, and so on until the desired image is reached. In the present invention, such an operation is completed without losing data range and precision, and without the need to pass the data through the entire graphics program **130**.

For example, a graphics program in accordance with the present invention can use multipass graphics algorithms such as those that implement lighting or shading programs to modify the frame buffer data that define the appearance of each pixel. The algorithm approximates the degree of lighting or shading, and the component of the data that specifies each of these characteristics is adjusted accordingly. Multiple passes through the shading/lighting program may be needed before the desired effect is achieved. In the present invention, the results of each pass are accumulated in the present invention frame buffer, and then used for the basis for subsequent passes, without a loss of precision or range. Such an operation requires the use of floating point formats in the frame buffer to increase the speed and accuracy of the calculations.

Also, in the present invention the user of the graphics program is able to enhance a portion of data contained within the frame buffer. For example, such an application will arise when the data loaded into the frame buffer represent an image obtained by a device capable of recording images that will not be visible to the human eye when displayed, such as an image recorded by a video camera in very low light, or an infrared image. The present invention is capable of storing such data in the frame buffer because of the range and precision permitted by the floating point format. The user specifies a lower threshold for that component of the data representing how bright the pixel will be displayed to the viewer. Data falling below the specified threshold are then operated on to enhance them; that is, for each piece of data below the threshold, the component of the data representing brightness is increased by addition, until the brightness is increased sufficiently so that the displayed image can be seen by the human eye. Such an operation is possible because of the precision of the data stored in the frame buffer in the present invention. Other operations involving the manipulation of the data in the frame buffer are also possible using the present invention.

Therefore, in the present invention the data are read from the frame buffer, operated on, then written back into the frame buffer. The use of a floating point frame buffer permits operation on the data stored in the frame buffer without a loss of range and precision. The floating point format is specified to optimize the range and precision required for the desired application. The present invention also allows the data stored in the frame buffer to be operated on and changed without the effort and time needed to process the data through the graphics program **130** of FIG. **2**. As such, the present invention will increase the speed at which operations can be performed, because it is not necessary to perform all the steps of a graphics program to adequately modify the data in the frame buffer. In addition, processing speed is further improved by applying hardware such as processor chips and computer hard drives to work directly on the frame buffer. Thus, application of the present invention provides the foundation upon which related hardware design improvements can be based, which could not be otherwise utilized.

Referring now to FIG. **4**, a block diagram of the currently preferred embodiment of the display system **400** is shown. Display system **400** operates on vertices, primitives, and fragments. It includes an evaluator **401**, which is used to provide a way to specify points on a curve or surface (or part of a surface) using only the control points. The curve or surface can then be rendered at any precision. In addition, normal vectors can be calculated for surfaces automatically. The points generated by an evaluator can be used to draw dots where the surface would be, to draw a wireframe version of the surface, or to draw a fully lighted, shaded, and even textured version. The values and vectors associated with evaluator and vertex arrays **401** are specified in a floating point format. The vertex array contains a block of vertex data which are stored in an array and then used to specify multiple geometric primitives through the execution of a single command. The vertex data, such as vertex coordinates, texture coordinates, surface normals, RGBA colors, and color indices are processed and stored in the vertex arrays in a floating point format. These values are then converted and current values are provided by block **402**. The texture coordinates are generated in block **403**. The lighting process which computes the color of a vertex based on current lights, material properties, and lighting-model modes is performed in block **404**. In the currently preferred embodiment, the lighting is done on a per pixel basis, and the result is a floating point color value. The various matrices are controlled by matrix control block **405**.

Block **406** contains the clipping, perspective, and viewport application. Clipping refers to the elimination of the portion of a geometric primitive that is outside the half-space defined by a clipping plane. The clipping algorithm operates on floating point values. Perspective projection is used to perform foreshortening so that he farther an object is from the viewport, the smaller it appears in the final image. This occurs because the viewing volume for a perspective projection is a frustum of a pyramid. The matrix for a perspective-view frustum is defined by floating point parameters. Selection and feedback modes are provided in block **407**. Selection is a mode of operation that automatically informs the user which objects are drawn inside a specified region of a window. This mechanism is used to determine which object within the region a user is specifying

or picking with the cursor. In feedback mode, the graphics hardware is used to perform the usual rendering calculations. Instead of using the calculated results to draw an image on the screen, however, this drawing information is returned. Both feedback and selection modes support the floating point format.

The actual rasterization is performed in block **408**. Rasterization refers to converting a projected point, line, or polygon, or the pixels of a bitmap or image, to fragments, each corresponding to a pixel in the frame buffer **412**. Note that all primitives are rasterized. This rasterization process is performed exclusively in a floating point format. Pixel information is stored in block **409**. A single pixel (x,y) refers to the bits at location (x,y) of all the bitplanes in the frame buffer **412**. The pixels are all in floating point format. A single block **410** is used to accomplish texturing, fog, and anti-aliasing. Texturing refers to the process of applying an image (i.e., the texture) to a primitive. Texture mapping, texels, texture values, texture matrix, and texture transformation are all specified and performed in floating point. The rendering technique known as fog, which is used to simulate atmospheric effects (e.g., haze, fog, and smog), is performed by fading object colors in floating point to a background floating point color value(s) based on the distance from the viewer. Antialiasing is a rendering technique that assigns floating point pixel colors based on the fraction of the pixel's area that is covered by the primitive being rendered. Antialiased rendering reduces or eliminates the jaggies that result from aliased rendering. In the currently preferred embodiment, blending is used to reduce two floating point color components to one floating point color component. This is accomplished by performing a linear interpolation between the two floating point color components. The resulting floating point values are stored in frame buffer **412**. But before the floating point values are actually stored into the frame buffer **412**, a series of operations are performed by per-fragment operations block **411** that may alter or even throw out fragments. All these operations can be enabled or disabled. It should be noted that although many of these blocks are described above in terms of floating point, one or several of these blocks can be performed in fixed point without departing from the scope of the present invention. The blocks of particular interest with respect to floating point include the rasterization **408**; pixels **409**; texturing fog, and antialiasing **410**, per-fragment operations **411**; and frame buffer and frame buffer control **412** blocks.

FIG. **5** shows a more detailed layout of a display system for implementing the floating point present invention. In the layout, the process flows from left to right. Graphics commands, vertex information, and pixel data generated by previous circuits are input to the polygon rasterization **501**, line segment rasterization **502**, point rasterization **503**, bitmap rasterization **504**, and pixel rasterization **505**. Floating point format can be applied to any and/or all of these five rasterization functions. In particular, the polygons are rasterized according to floating point values. The outputs from these five blocks **501–505** are all fed into the texel generation block **506**. In addition, texture data stored in texture memory **507** is also input to texel generation block **506**. The texture data is stored in the texture memory **507** in a floating point format. Texel values are specified in a floating point format. The texel data is then applied to the texture application block **508**. Thereupon, fog effects are produced by fog block **509**. Fog is achieved by fading floating point object colors to a floating point background color. A coverage application **510** is used to provide antialiasing. The antialiasing algorithm operates on floating point pixels colors. Next, several tests are executed. The pixel ownership test **511** decides whether or not a pixel's stencil, depth, index, and color values are to be cleared. The scissor test **512** determines whether a fragment lies within a specified rectangular portion of a window. The alpha test **513** allows a fragment to be accepted or rejected based on its alpha value. The stencil test **514** compares a reference value with the value stored at a pixel in the stencil buffer. Depending on the result of the test, the value in the stencil buffer is modified. A depth buffer test **515** is used to determine whether an incoming depth value is in front of a pre-existing depth value. If the depth test passes, the incoming depth value replaces the depth value already in the depth buffer. Optionally, masking operations **519** and **520** can be applied to data before it is written into the enabled color, depth, or stencil buffers. A bitwise logical AND function is performed with each mask and the corresponding data to be written.

Blending **516** is performed on floating point RGBA values. Color resolution can be improved at the expense of spatial resolution by dithering **517** the color in the image. The final operation on a fragment is the logical operation **518**, such as an OR, XOR, or INVERT, which is applied to the incoming fragment values and/or those currently in the color buffer. The resulting floating point values are stored in the frame buffer **522** under control of **521**. Eventually, these floating point values are read out and drawn for display on monitor **523**. Again, it should be noted that one or more of the above blocks can be implemented in a fixed point format without departing from the scope of the present invention. However, the blocks of particular importance for implementation in a floating point format include the polygon rasterization **501**, texel generation **506**, texture memory **507**, fog **509**, blending **516**, and frame buffer **522**.

In the currently preferred embodiment, the processor for performing geometric calculations, the rasterization circuit, and the frame buffer all reside on a single semiconductor chip. The processor for performing geometric calculations, the rasterization circuit, and the frame buffer can all have the same substrate on that chip. Furthermore, there may be other units and/or circuits which can be incorporated onto this single chip. For instance, portions or the entirety of the functional blocks shown in FIGS. **4** and **5** can be fabricated onto a single semiconductor chip. This reduces pin count, increases bandwidth, consolidates the circuit board area, reduces power consumption, minimizes wiring requirements, and eases timing constraints. In general, the design goal is to combine more components onto a single chip.

The preferred embodiment of the present invention, a floating point frame buffer, is thus described. While the present invention has been described in particular embodiments, it should be appreciated that the present invention should not be construed as limited by such embodiments, but rather construed according to the following claims.

What is claimed is:

1. A computer system, comprising:

a processor for performing geometric calculations on a plurality of vertices of a primitive;

a rasterization circuit coupled to the processor that rasterizes the primitive according to a rasterization process which operates on a floating point format;

a frame buffer coupled to the rasterization circuit for storing a plurality of color values; and

a display screen coupled to the frame buffer for displaying an image according to the color values stored in the frame buffer;

wherein the rasterization circuit performs scan conversion on vertices having floating point color values.

2. A computer system, comprising:

a processor for performing geometric calculations on a plurality of vertices of a primitive;

a rasterization circuit coupled to the processor that rasterizes the primitive according to a rasterization process which operates on a floating point format;

a frame buffer coupled to the rasterization circuit for storing a plurality of color values;

a display screen coupled to the frame buffer for displaying an image according to the color values stored in the frame buffer;

a texture circuit coupled to the rasterization circuit that applies a texture to the primitive, wherein the texture is specified by floating point values; and

a texture memory coupled to the texture circuit that stores a plurality of textures in floating point values.

3. A computer system, comprising:

a processor for performing geometric calculations on a plurality of vertices of a primitive;

a rasterization circuit coupled to the processor that rasterizes the primitive according to a rasterization process which operates on a floating point format;

a frame buffer coupled to the rasterization circuit for storing a plurality of color values; and

a display screen coupled to the frame buffer for displaying an image according to the color values stored in the frame buffer;

wherein the floating point format is comprised of sixteen bits in a s10e5 format.

4. A computer system, comprising:

a processor for performing geometric calculations on a plurality of vertices of a primitive;

a rasterization circuit coupled to the processor that rasterizes the primitive according to a rasterization process which operates on a floating point format;

a frame buffer coupled to the rasterization circuit for storing a plurality of color values;

a display screen coupled to the frame buffer for displaying an image according to the color values stored in the frame buffer; and

a fog circuit coupled to the rasterization circuit for performing a fog function, wherein the fog function operates on floating point color values.

5. A computer system, comprising:

a processor for performing geometric calculations on a plurality of vertices of a primitive;

a rasterization circuit coupled to the processor that rasterizes the primitive according to a rasterization process which operates on a floating point format;

a frame buffer coupled to the rasterization circuit for storing a plurality of color values;

a display screen coupled to the frame buffer for displaying an image according to the color values stored in the frame buffer; and

a blender coupled to the rasterization circuit which blends floating point color values.

6. A computer system, comprising:

a processor for performing geometric calculations on a plurality of vertices of a primitive;

a rasterization circuit coupled to the processor that rasterizes the primitive according to a rasterization process which operates on a floating point format;

a frame buffer coupled to the rasterization circuit for storing a plurality of color values;

a display screen coupled to the frame buffer for displaying an image according to the color values stored in the frame buffer; and

logic coupled to the rasterization circuit which performs per-fragment operations on floating point color values.

7. A computer system, comprising:

a processor for performing geometric calculations on a plurality of vertices of a primitive;

a rasterization circuit coupled to the processor that rasterizes the primitive according to a rasterization process which operates on a floating point format;

a frame buffer coupled to the rasterization circuit for storing a plurality of color values; and

a display screen coupled to the frame buffer for displaying an image according to the color values stored in the frame buffer;

wherein the processor, the rasterization circuit, and the frame buffer are on a single semiconductor chip.

8. The computer system of claim **7**, wherein the processor, the rasterization circuit, and the frame buffer reside on a same substrate of the single semiconductor chip.

9. In a computer system, a method for rendering a three-dimensional image for display, comprising the steps of:

performing geometric calculations on a plurality of vertices of a plurality of polygons;

scan converting a plurality of pixels according to the vertices, wherein scan conversion is performed on floating point color values;

applying a texture to the image by reading floating point texture values stored in a texture memory;

simulating fog effects, wherein fog is simulated by modifying floating point color values;

drawing the image for display on a display screen coupled to the computer system.

10. The method of claim **9**, wherein the floating point values are comprised of sixteen bits.

11. The method of claim **10**, wherein the floating point values are specified by a s10e5 format.

12. The method of claim **10** further comprising the step of storing the floating point color values in a frame buffer.

13. The method of claim **10** further comprising the step of blending at least two floating point color values.

14. The method of claim **10** further comprising the step of performing antialiasing on floating point color values.

15. The method of claim **10** further comprising the steps of:

reading data from the frame buffer;

modifying the data;

writing modified data back to the frame buffer.

16. The method of claim **10** further comprising the step of modifying color values for lighting, wherein lighting calculations operate on floating point color values.

17. In a computer system, a method for operating on data stored in a frame buffer, comprised of:

storing the data in the frame buffer in a floating point format;

reading the data from the frame buffer in the floating point format;

operating directly on the data in the floating point format; and

writing the data to the frame buffer in the floating point format;

wherein the steps of writing, storing, and reading the data in the frame buffer in the floating point format are further comprised of a specification of the floating point format, wherein the specification corresponds to a level of range and precision.

18. The method of claim **17** wherein the specification is comprised of 16 bits of data and the data are comprised of one sign bit, ten mantissa bits, and five exponent bits.

19. The method of claim **17** wherein the specification is comprised of 17 bits of data and the data are comprised of one sign bit, 11 mantissa bits, and five exponent bits.

20. The method of claim **17** wherein the specification is comprised of 16 bits of data and the data are comprised of ten mantissa bits, and six exponent bits.

21. The method of claim **17** wherein the specification is comprised of 32 bits of data and the data are comprised of one sign bit, 23 mantissa bits, and eight exponent bits.

22. A computer system having a floating point frame buffer for storing a plurality of floating point color values;

wherein the floating point color values are written to, read from, and stored in the frame buffer using a specification of the floating point color values that corresponds to a level of range and precision.

23. The computer system of claim **22**, wherein the floating point color values are comprised of 16 bits of data and the data are comprised of one sign bit, ten mantissa bits, and five exponent bits.

24. The computer system of claim **22**, wherein the floating point color values are comprised of 17 bits of data and the data are comprised of one sign bit, 11 mantissa bits, and five exponent bits.

25. A computer system, comprising:

a processor for performing geometric calculations on a plurality of vertices of a primitive;

a rasterization circuit coupled to the processor that rasterizes the primitive according to a rasterization process which operates on an s10e5 floating point format;

a frame buffer coupled to the rasterization circuit for storing a plurality of s10e5 floating point color values;

a display screen coupled to the frame buffer for displaying an image according to the s10e5 color values stored in the frame buffer.

26. The computer system of claim **25** further comprising:

a texture circuit coupled to the rasterization circuit that applies a texture to the primitive, wherein the texture is specified by s10e5 floating point values.

27. The computer system of claim **25** further comprising a lighting circuit coupled to the rasterization circuit for performing a lighting function, wherein the lighting function executes on s10e5 floating point color values.

28. The computer system of claim **25** further comprising a fog circuit coupled to the rasterization circuit for performing a fog function, wherein the fog function operates on s10e5 floating point color values.

29. The computer system of claim **25** further comprising an antialiasing circuit coupled to the rasterization circuit which performs an antialiasing algorithm on s10e5 floating point color values.

30. The computer system of claim **25** further comprising a blender coupled to the rasterization circuit which blends s10e5 floating point color values.

31. The computer system of claim **25** further comprising logic coupled to the rasterization circuit which performs per-fragment operations on s10e5 floating point color values.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,650,327 B1
DATED : November 18, 2003
INVENTOR(S) : Airey et al.

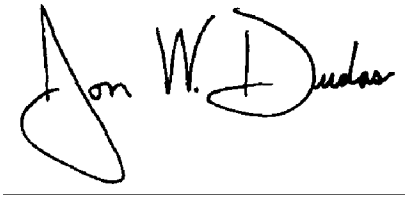
It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 14,
Line 36, please replace “values;” with -- values; and --.
Line 52, please replace “data;” with -- data; and --.

Column 16,
Line 5, please replace “values;” with -- values; and --.

Signed and Sealed this

Twentieth Day of April, 2004



JON W. DUDAS
Acting Director of the United States Patent and Trademark Office

Appendix B

US006292200B1

(12) **United States Patent**
Bowen et al.

(10) **Patent No.: US 6,292,200 B1**
(45) **Date of Patent: Sep. 18, 2001**

(54) **APPARATUS AND METHOD FOR UTILIZING MULTIPLE RENDERING PIPES FOR A SINGLE 3-D DISPLAY**

(75) Inventors: **Andrew Bowen**, San Jose; **Dawn Maxon**, Belmont; **Gregory Buchner**, Los Altos, all of CA (US)

(73) Assignee: **Silicon Graphics, Inc.**, Mountain View, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/177,911**

(22) Filed: **Oct. 23, 1998**

(51) **Int. Cl.**[7] **G06T 1/20**
(52) **U.S. Cl.** **345/506**; 345/520; 709/251
(58) **Field of Search** 345/501–506, 345/520, 507–509, 419, 530, 545, 541, 544; 709/251, 238

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,841,444 * 11/1998 Mun et al. 345/506

OTHER PUBLICATIONS

"PixelFlow: The Realization", Eyles et al, 1997 SIGGRAPH/Eurographics Workshop, ACM digital Library, pp. 57–68, Aug. 3–4, 1997.*

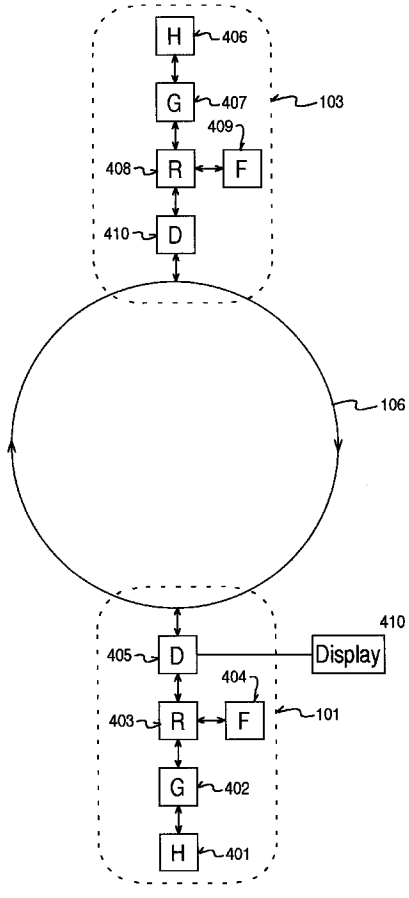
* cited by examiner

Primary Examiner—Kee M. Tung
(74) *Attorney, Agent, or Firm*—Wagner, Murabito & Hao LLP

(57) **ABSTRACT**

A computer graphics system having a hyperpipe architecture. Multiple rendering pipes are coupled together through a hyperpipe network scheme. Each of the rendering pipes are capable of rendering primitives for an entire frame or portions thereof. This enables multiple rendering pipes to process graphics data at the same time. A controller coordinates the multiple rendering pipes by sending requests to the appropriate rendering pipes to retrieve the pixel data generated by that particular pipe. It then merges the pixel data received from the various rendering pipes. A single driver then draws the three-dimensional image out for display.

18 Claims, 6 Drawing Sheets



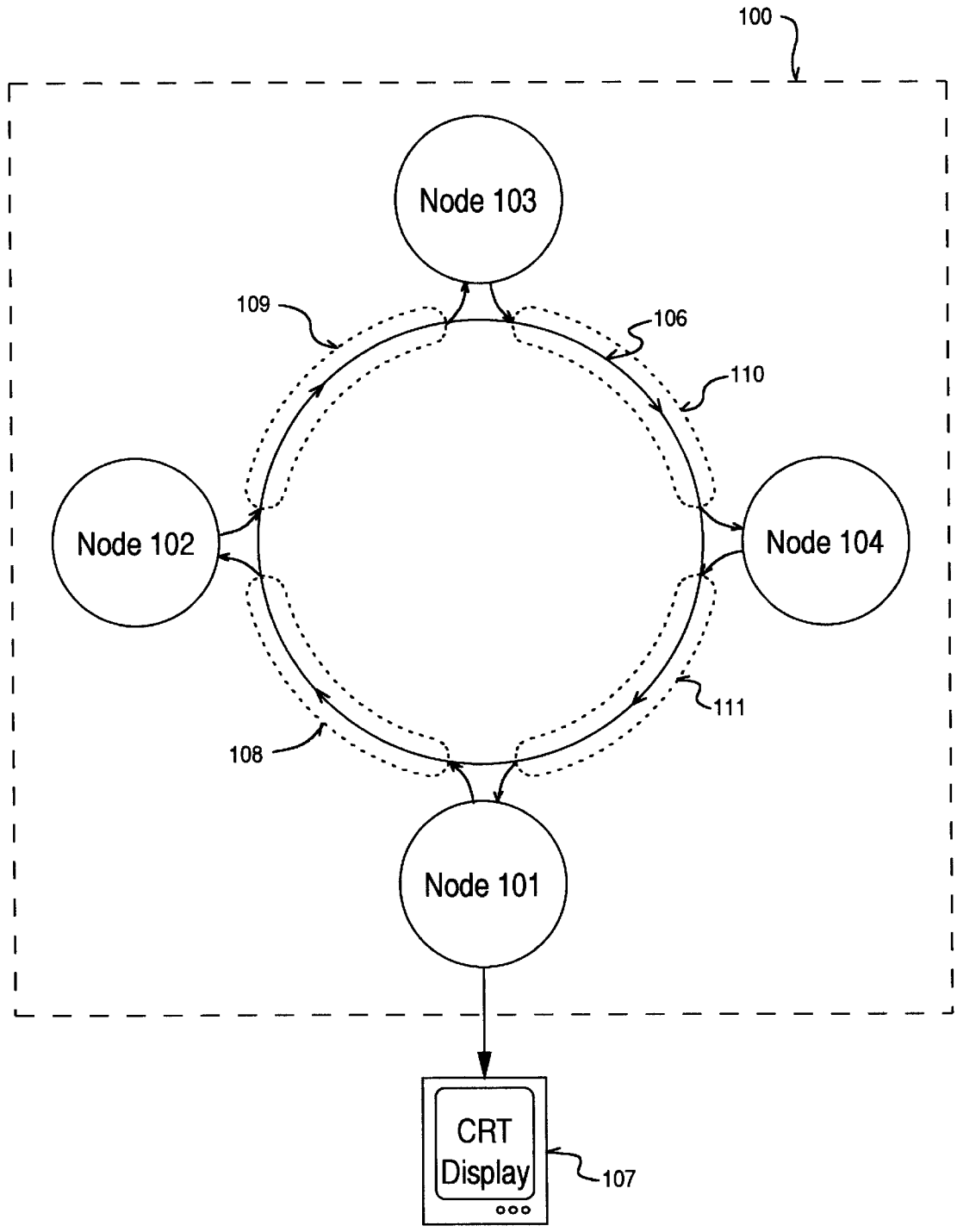

100
Node 103
109
106
110
Node 102
Node 104
111
108
Node 101
CRT
Display
107

Figure 1

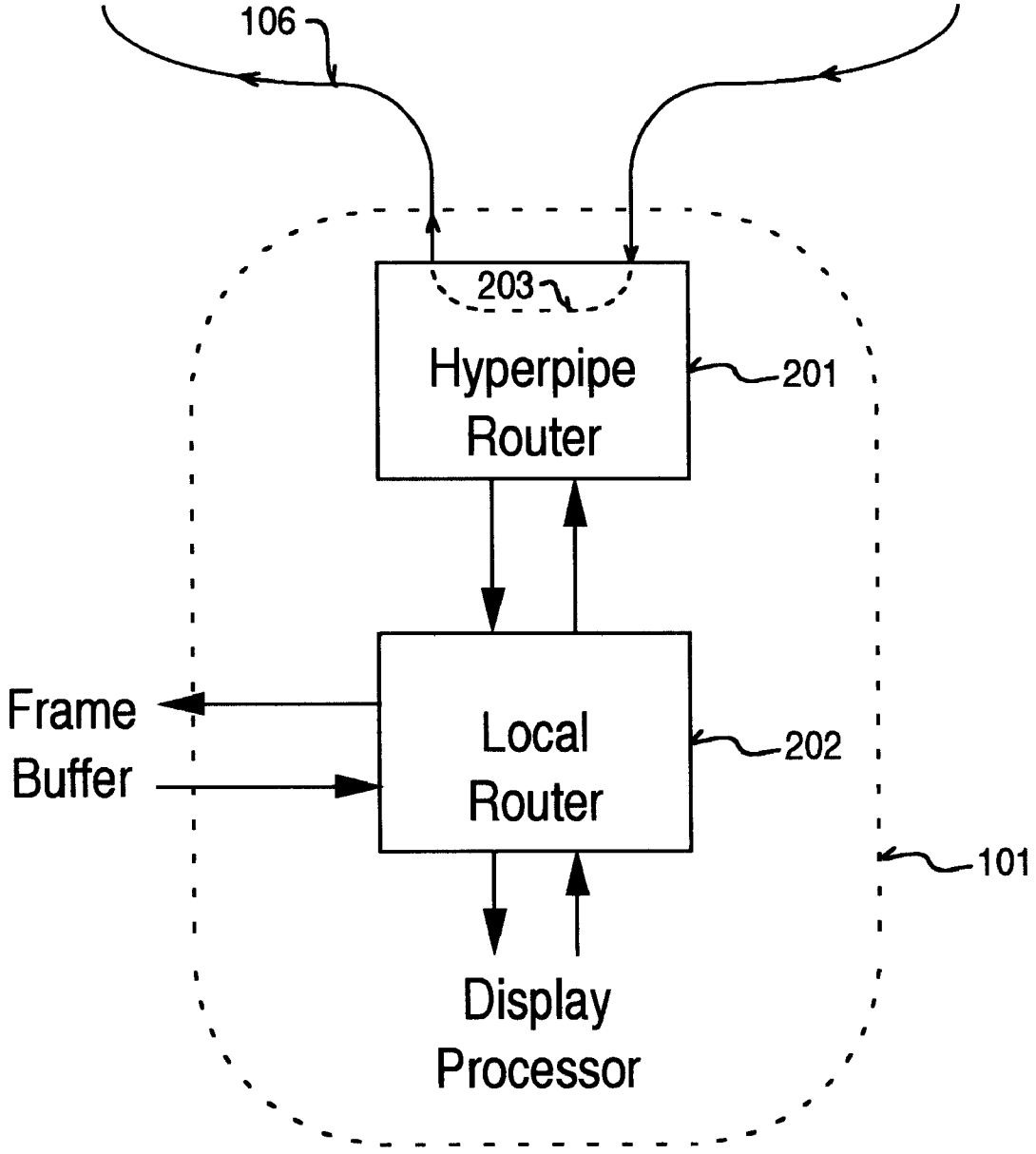

106
203
Hyperpipe Router
201
Frame Buffer
Local Router
202
101
Display Processor

Figure 2

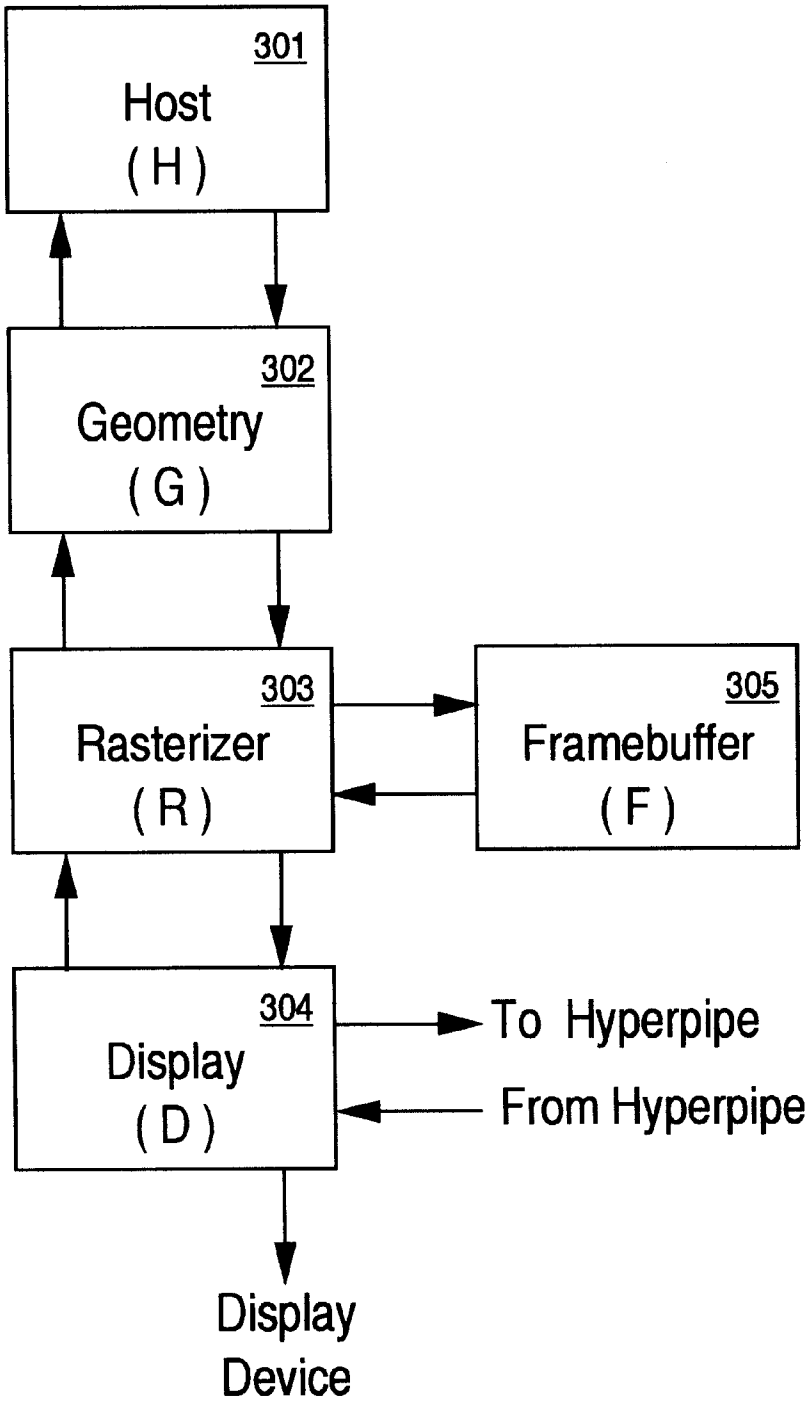

301
Host
(H)
302
Geometry
(G)
303
Rasterizer
(R)
305
Framebuffer
(F)
304
Display
(D)
To Hyperpipe
From Hyperpipe
Display
Device

Figure 3

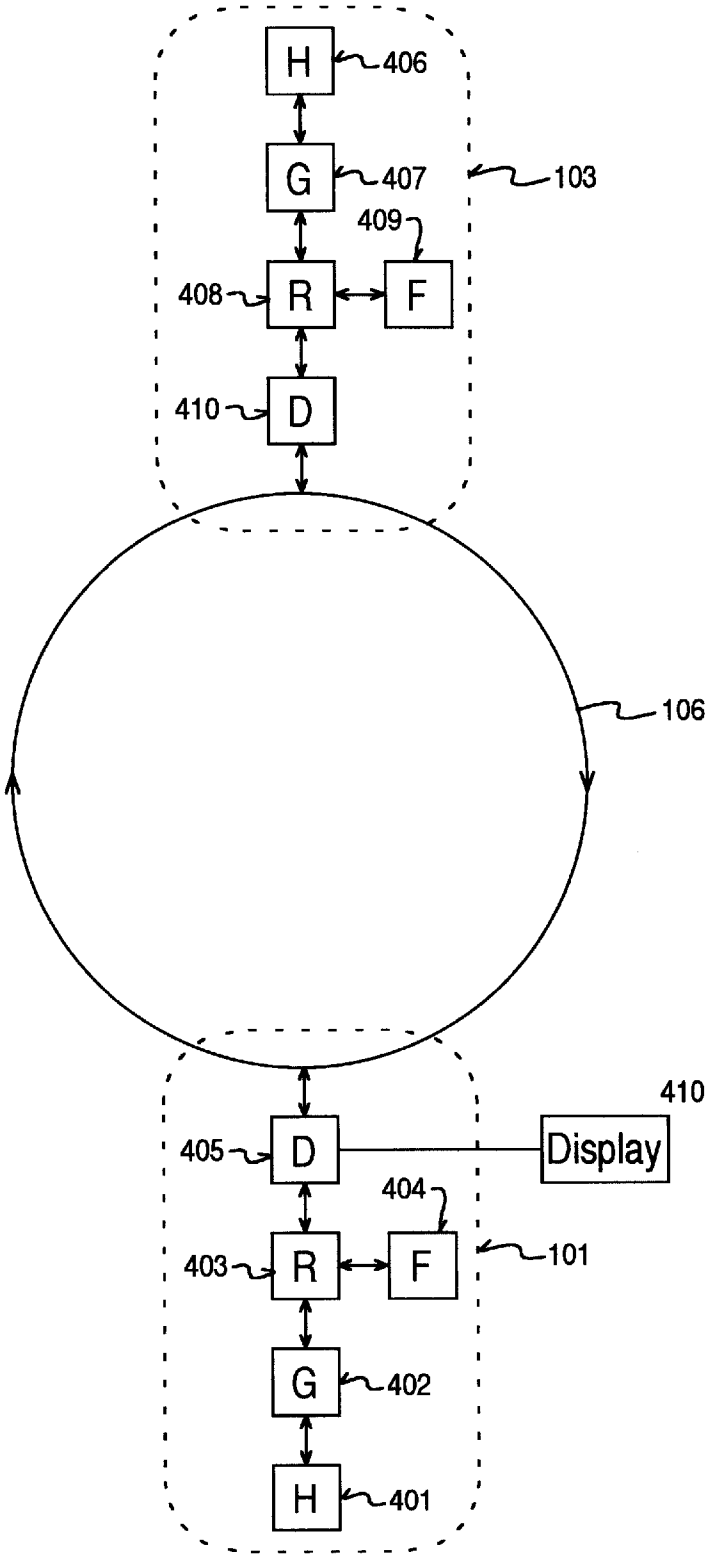

H
406
G
407
103
409
408
R
F
410
D
106
410
405
D
Display
404
403
R
F
101
G
402
H
401

FIGURE 4

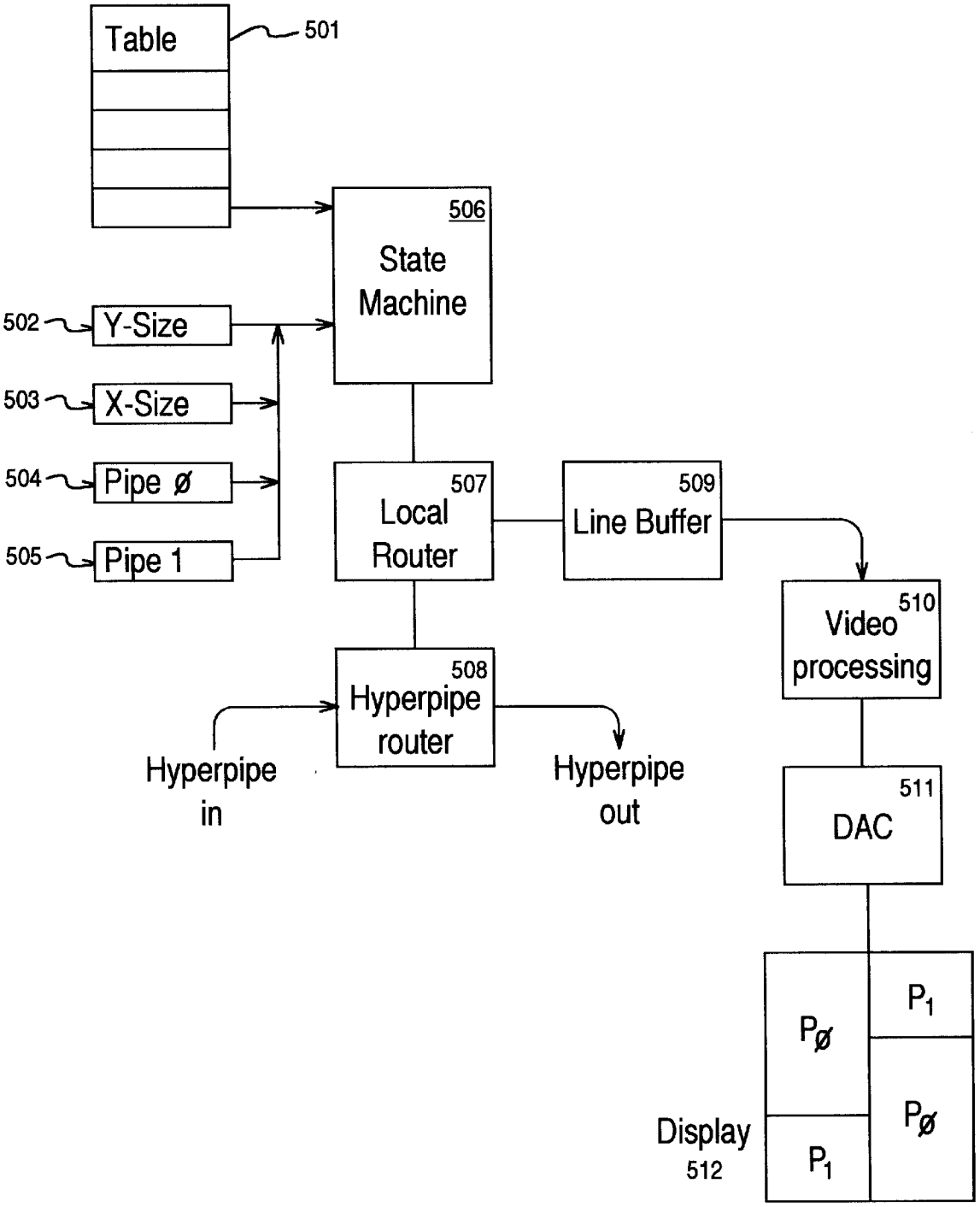


Table
501
506
State Machine
502
Y-Size
503
X-Size
504
Pipe ø
505
Pipe 1
507
Local Router
509
Line Buffer
510
Video processing
508
Hyperpipe router
Hyperpipe in
Hyperpipe out
511
DAC
Pø
P1
Pø
P1
Display
512

Figure 5

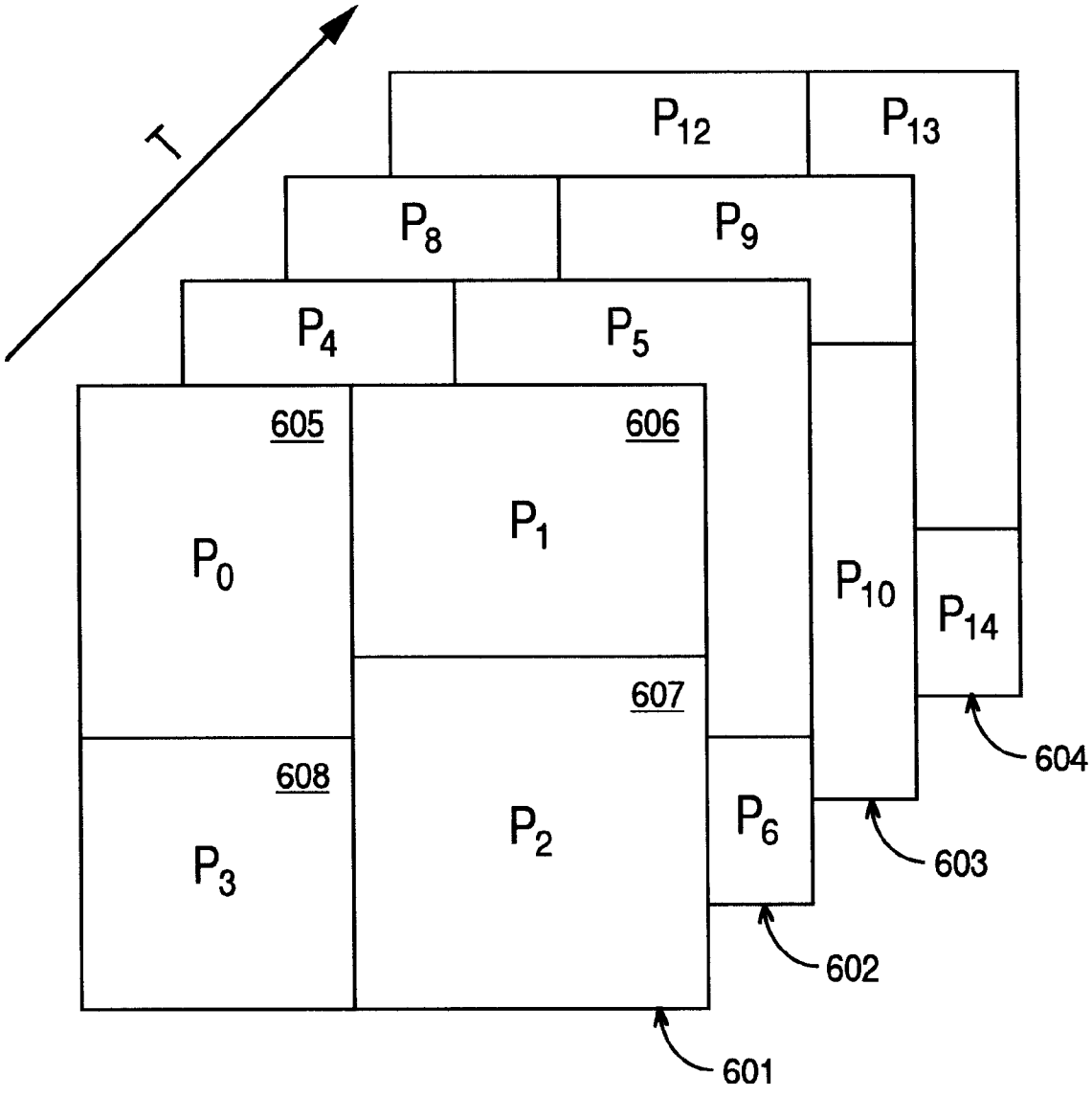

T
P12
P13
P8
P9
P4
P5
605
606
P0
P1
P10
P14
607
608
P3
P2
P6
604
603
602
601

Figure 6

APPARATUS AND METHOD FOR UTILIZING MULTIPLE RENDERING PIPES FOR A SINGLE 3-D DISPLAY

FIELD OF THIE INVENTION

The present invention relates the field of computer graphics. More particularly, the present invention pertains to an apparatus and method for utilizing multiple rendering pipes for the generation of a single 3-D display.

BACKGROUND OF THE INVENTION

Today, computer graphics is used in a wide variety of applications, such as in business, science, animation, simulation, computer-aided design, process control, electronic publication, etc. In an effort to portray a more realistic real-world representation, three dimensional objects are transformed into models having the illusion of depth for display onto a two-dimensional computer screen. This is accomplished by using a number of polygons to represent a three-dimensional object. Complex three-dimensional objects may require upwards of hundreds of polygons in order to form an accurate model. Hence, a three-dimensional object can be readily manipulated (e.g., displayed in a different location, rotated, scaled, etc.) by processing the individual respective polygons corresponding to that object. Next, a scan conversion process is used to determine which pixels of a computer display fall within each of the specified polygons. Thereupon, texture is applied to only those pixels residing within specified polygons. In addition, hidden or obscured surfaces, which are normally not visible, are eliminated from view. Hence, displaying a three dimensional object on a computer system is a rather complicated task and can require a tremendous amount of processing power.

This is especially true for those cases involving dynamic computer graphics for displaying three-dimensional objects that are in motion. In order to simulate smooth motion, the computer system should have a frame rate of at least 30 hertz. In other words, new images should be updated, redrawn and displayed at least thirty times a second. This imposes a heavy processing and computational burden on the computer system. Indeed, even more processing power is required for interactive computer graphics, where displayed images change in response to a user input and where there are multiple objects in a richly detailed scene.

However, each extra object that is added into a scene needs to be modeled, scan converted, textured, Z-buffered for depth, etc., all of which, adds to the amount of processing resources that is required. In addition, it would be highly preferable if lghting, shadowing, shading, and fog could be included as part of the 3-D scene. Generating these special effects, again, consumes valuable processing resources. Hence, a major problem associated with producing realistic three-dimensional scenes is that it requires such a tremendous amount of processing power. The "richer" and more realistic a scene becomes, the more processing power that is required to render that scene. Moreover, speed becomes a major limiting factor as the computer must render millions of pixels in order to produce these amazingly complex scenes in less than one thirtieth (1/30) of a second.

Even though the processing power of computer systems continues to improve, there exists whole markets which demand even greater and greater processing power. Certain purchasers (e.g., drug companies, oil exploration, medical imaging, film studios, etc.) will pay a premium to obtain even faster and more powerful computer for rendering 3-D images.

In the past, there have been attempts to utilize several rendering engines in a single computer system in order to perform parallel processing. Each of these rendering engines is used to render one particular frame of image. While one rendering engine is in the process of generating one frame's worth of image data, another separate rendering engine is simultaneously generating the next frame's worth of image data. Meanwhile, other rendering engines can simultaneously be processing subsequent frames, etc. The digital-to-analog (DAC) outputs of each of these rendering engines are wired together to drive the cathode ray tube (CRT) display screen. By rendering multiple frames's worth of data at the same time with multiple rendering engines, the computer's overall processing speed is increased.

Unfortunately, however, there are some drawbacks to this way of ganging together multiple rendering engines. First, since there are multiple DACs driving the same CRT screen, there tends to be some scintillation between frames as DACs are switched from frame to frame. Furthermore, there are serious synchronization problems in order to properly coordinate the activities amongst all the rendering engines and their respective DACs.

Thus, there exists a need for some apparatus or method which increases the rendering power and speed of a 3-D computer system without sacrificing picture quality or increasing programming complexity in an unacceptable way. The present invention provides a novel solution by having one output controller which requests and receives data from multiple rendering engines. This effectively resolves virtually all problems associated with using multiple rendering engines. Furthermore, with the present invention, multiple rendering engines can now contribute to the generation of one single frame. The end result is that processing power and speed is dramatically improved with minimal or no discernible degradation to the displayed images.

SUMMARY OF THE INVENTION

The present invention pertains to a computer graphics system having a hyperpipe architecture. The hyperpipe architecture includes multiple rendering pipes. Each of the rendering pipes is capable of rendering pixels for an entire frame or portions thereof. This enables multiple rendering pipes to process graphics data at the same time. The pixel data generated by a rendering pipe is stored in its local memory. The multiple rendering pipes are coupled together through a hyperpipe network scheme. A controller coordinates the multiple rendering pipes by sending requests to the appropriate rendering pipes to retrieve the pixel data generated by that particular pipe. It then merges the pixel data received from the various rendering pipes into a frame's worth of data. A single driver is then used to draw that frame out for display. Thereby, rather than having just one rendering circuit working on a frame, multiple rendering circuits can operate in parallel on generating a frame's worth of pixel data. In the meantime, other rendering pipes can optionally be used to generate subsequent frames. This increases the system's overall rendering power and speed. By simply adding additional rendering pipes onto the hyperpipe network, the computer system's rendering capabilities can be readily scaled up to meet cost and graphics demands.

In the currently preferred embodiment of the present invention, a uni-directional, point-to-point ring topology is used. The hyperpipe network consists of a broad band packetized protocol with error correction. This scheme is preferred because of its relatively fixed and predictable

latency. A fixed latency is desired as it allows the controller to send out requests ahead of when the pixel data will actually be used.

In one embodiment of the present invention, each of the rendering pipes includes a host processor, a geometry engine, a rasterizer, a frame buffer, and a display unit. A graphics application runs on the host processor and issues high-level commands and graphics data. The geometry engine performs arithmetic operations involving geometric calculations on the vertices of primitives used to render the threedimensional images. The rasterizer then fills the primitives and stores the resulting pixel data in its local frame buffer memory. A display unit can either request and receive video data packets from it's local pipe, or make similar requests over the hyperpipe. Other display units on the remote pipes can receive these requests and route the responses back on to the hyperpipe network. The master, or consumer, pipe then performs all the video backend processing on the data, e.g., color mapping, resizing, color space conversion, and gamma correction, and sends it to the output display device as a coherent video stream. A single controller designated as the controller issues the requests the rendering pipes and merges the received pixel data. A single driver then takes the merged data and drives a display monitor.

BRIEF DESCRIPTON OF THE DRAWINGS

The operation of this invention can be best visualized by reference to the drawings.

FIG. **1** shows a high-level diagram of a computer architecture upon which the present invention may be practiced.

FIG. **2** shows a diagram depicting the flow of packets on the hyperpipe bus/network into, through, and out from an exemplary node/rendering pipe.

FIG. **3** shows a block diagram of the currently preferred embodiment of a node or rendering pipe.

FIG. **4** shows a hyperpipe computer system having two nodes.

FIG. **5** shows a block diagram of a display controller for a consumer node.

FIG. **6** is a diagram showing how multiple frames of images are rendered by multiple rendering pipes in a hyperpipe architecture.

DETAILED DESCRIPTION

An apparatus and method for utilizing multiple rendering pipes for the generation of a single 3-D display is described. In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be obvious, however, to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid obscuring the present invention.

Referring now to FIG. **1**, a high-level diagram of a computer architecture upon which the present invention may be practiced is shown. The computer system **100** includes multiple processing nodes. Although four such nodes **101–104** are shown, any number of such nodes can be implemented. In general, adding more nodes proportionally increases the speed and processing power of the computer system **100**. Each processing node is capable of performing rendering operations independently of the other nodes. The term "pipe" has been coined to refer to a rendering node. In the currently preferred embodiment of the present invention, each of these rendering nodes or pipes **101–104** is the same as any other node or pipe. The only difference is that a single one of the pipes is designated as being the "consumer." Since all rendering pipes are the same, it does not matter which particular pipe becomes the consumer. For example, node **101** can be selected as being the consumer.

It is the function of the consumer to gather the requisite data from all the appropriate pipes in order to draw the image for display on CRT display screen **107**. The consumer **101** gets the requisite data by generating requests and sending the requests onto a bus network interconnect **106**. Bus/network **106** can be any high-bandwidth bus or network for transmission of digital data (e.g., ethernet, ATM, SONET, FDDI, etc.). In the currently preferred embodiment, bus/network **106** consists of a high-speed, high-bandwidth, unidirectional ring topology having a packet-based protocol. This bus/network establishes a point-to-point connection. The term "hyperpipe" has been coined to represent the digital backbone connecting all the rendering pipes. The requests are sent sequentially through bus/network **106** to each of the nodes. When a node receives a request, it examines that request to determine whether it is the one which has been designated to service that request In other words, the request contains information specifying which of the nodes contains the desired data. For example, if the desired data were generated by and stored in node **103**, then consumer **101** would generate a request. This request is then sent over bus **106** in the format of a packet. Assuming a unidirectional clockwise transmission over bus **106**, node **102** would be the first node to receive the request packet. Node **102** quickly checks the packet to determine whether the associated request was designated for it. A designation (e.g., an address) specifying the appropriate node can be contained in the header of the request packet. Since node **102** is not the intended recipient of that request, node **102** simply ignores the packet. Thereupon, the packet proceeds back through the bus **106** to the next downstream node **103**. Node **103** examines the packet and determines that the request is destined for it. Thereupon, node **103** retrieves the requested data from its local memory (e.g., frame buffer) and sends the data in the form of packets over bus **106** destined to consumer **101**. The request packet is then sent to node **104**, which checks the request packet and ignores it. Consumer **101** continuously generates requests for data. As the data for a frame is received, consumer **101** draws the image out display (CRT) screen **107**.

It should be noted that there are latencies associated with transmitting packets over various segments of bus **106**. The latencies between nodes may also vary. Further latencies are incurred locally by nodes checking received requests, processing requests, and sending data over bus **106** to the consumer. It is known that high latencies do not pose a major problem because the penalty associated with the high latency is paid once at the beginning. Thereupon, no additional delays will be incurred. All processing is essentially shifted time-wise by the latency. However, if the latency is not constant, then one must add FIFO (first-in-first-out) buffers to temporarily hold the data because, due to the variable latencies, it is not known exactly when the data might be received. Larger FIFO buffers must be used to account for greater variances in the latencies. In order to minimize the overhead and costs associated with having large FIFO's, it is a design criteria to keep the latencies as constant as possible. The present invention accomplishes this by using a unidirectional ring topology. This ensures that all requests/data response packets incur approximately

the same fixed latency. In other words, each request/data response packet will incur the fixed latency associated with a full loop around bus **106** plus the local node latency for processing the request. For example, a request from consumer node **101** designated for node **102** will incur a latency associated with traversing span **108** of bus **106** to node **102**. A local latency associated with node **102** for processing the request is then incurred. Additional latencies are incurred by the data packets traversing through spans **109–111** of bus **106** back to consumer node **101**. Likewise, if the request were intended for node **104**, the same approximate latency would be incurred. In this case, the request would incur latencies for traversing through spans **108** and **109**, local latency associated with node **104**, and latency for the data traversing through span **111**. A request for data which happens to reside with the consumer node, nonetheless is required to be routed all the way through the entire bus loop to ensure that it too incurs relatively the same latency. Hence, all request/data packets, regardless of node designations, experience the same bus loop (e.g., spans **10–111**) latency plus local latency.

Furthermore, by implementing a unidirectional loop topology, all the latencies are known. This allows the system to issue requests in advance of when the data is actually needed. For example, suppose that the latency has been measured or quantitatively determined to be X number of clock cycles. Suppose also that consumer node **101** desires data from node **103** at a particular point in time Y. Consumer node **101** would issue a request for this data X number of clock cycles before time Y. Thereby, the data would arrive just in time. There may be tolerances within the system which might skew the results several clock cycles in either direction. A small FIFO is implemented to store data in case the data comes early. By predicting, anticipating, and generating requests early, the effects of the latencies are minimized.

The advantages conferred by the present invention are several fold in that it eliminates the need to switch drivers (i.e., no scintillation's). Also, since there is just one controller for driving the display, the colors and intensities are well balanced. Furthermore, multiple nodes/pipes can be merged together to help in rendering a single frame, thereby allowing parallel processing of complex images. In addition, this architecture is adapted to be easily scaled up or down according to cost, speed, and rendering constraints.

FIG. **2** shows a diagram depicting the flow of packets on the hyperpipe bus/network **106** into, through, and out from an exemplary node/rendering pipe **101**. A packet on hyperpipe bus **106** is input to node **101**. The packet is examined by a hyperpipe router **201**. Hyperpipe router **201** examines the address in the packet's header to determine whether that packet is intended for node **101**. If the packet is not intended for node **101**, it is immediately forwarded back onto the hyperpipe bus **106** through path **203**. However, if the packet was intended for node **101**, it is routed to a local router **202** which directs the packet to the appropriate circuit within node **101** (e.g., the rasterizer). Packets originating from node **101** (e.g., request packets or data packets), are forwarded from local router **202** to hyperpipe router **201** for transmission onto hyperpipe bus **106**.

FIG. **3** shows a block diagram of the currently preferred embodiment of a node or rendering pipe. An application program running on host processor (H) **301** directs the rendering process. The application program provides the high-level instructions and data to be used in the rendering process. This information is passed on to a geometry engine (G) **302**, which performs the arithmetic operations on vertices. The vertices are then filled by a rasterizer block (R) **303**. Rasterizer **303** performs color, blending, anti-aliasing, scan-conversion, depth, texture, lighting, and fog functions. The final pixel values are stored in framebuffer (F) **305**. When requested, the appropriate pixel values are read from framebuffer **305** by display block (D) **304** and put out onto the hyperpipe bus or drawn out for display onto a CRT screen. It should be noted that nodes and hyperpipes can have many different types of configurations. Any standard type of 3-D graphics subsystem can be adapted to be used in conjunction with the present invention.

FIG. **4** shows a hyperpipe computer system having two nodes **101** and **103**. Node **101** consists of a host **401** coupled to a geometry engine **402**. The geometry engine **402** is coupled to rasterizer **403**. Pixels generated by rasterizer **403** are stored in frame buffer **404**. A display block **405** controls the movement of packets to/from hyperpipe bus **106**. Furthermore, if node **101** is a consumer node, then display block issues requests and draws images out to a display screen. Likewise, node **103** is essentially the same, hardware-wise, as node **101**. Node **103** consists of a host **406** coupled to a geometry engine **407**. The geometry engine **407** is coupled to rasterizer **408**. Pixels generated by rasterizer **408** are stored in frame buffer **409**. A display block **410** controls the movement of packets to/from hyperpipe bus **106**. When a request packet destined for node **103** is received, display block **410** reads the requested pixel data from its local frame buffer **409**, packetizes the data, and sends it onto hyperpipe bus **106** for transmission to node **101**. The display block **405** of node **101** takes this packetized data, processes it, and sends it to display device **410**. Additional nodes, identical to node **103**, can be added and coupled to hyperpipe bus **106** to get attain even greater and faster rendering capabilities.

FIG. **5** shows a block diagram of a display controller for a consumer node. The display controller consists of a table **501**, several registers **502–505**, and a state machine **506**. A small local memory is used to store table **501** which contains parameters for video formats which change from field to field or from frame to frame (e.g., interlace, interlace stereo, field sequential, stereo field sequential, etc.). A node may be instructed to contribute in the rendering of a portion of a frame. The portion of the frame is specified according to an X, Y coordinate system. Thereby, register **502** contains the Y-size coordinate, and register **503** contains the X-size coordinate. Registers **504** and **505** are small buffers for temporarily storing data from the various rendering pipes (e.g., pipe **0** and pipe **1**). The information contained in table **501** and registers **502–505** are fed into a state machine **506** for processing. State machine **506** generates requests to the appropriate pipes by sending requests through local rough **507** to hyperpipe route **508**. Responses arrive either over the hyperpipe route **508**, or from the local pipe route **507**. Data is merged in line buffer **509**, processed in **510** and passed to an output device **511**(e.g. A DAC). Note that the frame can have separate sections rendered by different nodes/rendering pipes. For example, for a two node/rendering pipe system, the display surface **512** is subdivided into four sections. Pipe **0** renders two sections, and pipe **1** renders two sections.

FIG. **6** is a diagram showing how multiple frames of images are rendered by multiple rendering pipes in a hyperpipe architecture. Four frames **601–604** are shown. The frames are rendered at a standard 60 hertz rate (i.e., each frame is rendered every 1/60th of a second). A single frame can have one or more rendering pipes rendering pixels for that frame. For instance, frame **601** can have four rendering pipes P**0**–P**3** rendering pixel data in unison. In other words,

pipe P0 is rendering section 605 while pipe P1 is rendering section 606 and while pipes P2 and P3 are rendering sections 607 and 608. The pixels are then merged and displayed at the same time. Note that a frame need not be subdivided into equal portions. Instead, it is more efficient to subdivide the frame so that each rendering pipe shares approximately the same graphics rendering burden. Each rendering pipe should approximately render the same number of primitives. Meanwhile, one or more other rendering pipes can be rendering subsequent whole frames or subsequent portions of frames. For example, pipes P4–P7 can be rendering frame 602 while frame 601 is being rendered. It can be seen that frames can be rendered faster by adding additional rendering pipes.

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.

What is claimed is:

1. A computer system comprising:

a plurality of rendering pipes for rendering pixels of an image, wherein each of the rendering pipes comprises a host processor having an application program issuing graphics commands, a geometry circuit coupled to the host processor for processing primitives, a rasterizer coupled to the geometry circuit for generating pixel data, a frame buffer coupled to the rasterizer which stores the pixel data, an interface coupled to the rasterizer that accepts requests from the transmission medium and outputs pixel data;

a transmission medium coupling together each of the plurality of rendering pipes;

a controller coupled to one of the rendering pipes which coordinates pixel information of the image between each of the plurality of rendering pipes, wherein each of the rendering pipes is capable of rendering pixels for an entire frame or portions thereof;

a memory coupled to the controller for storing the pixel information;

a display coupled to the memory for displaying the image.

2. The computer system of claim 1, wherein the transmission medium comprises a uni-directional ring topology.

3. The computer system of claim 2, wherein the transmission medium comprises a point-to-point connection.

4. The computer system of claim 1, wherein the rendering circuit includes a local memory for storing pixel data generated locally.

5. The computer system of claim 4, wherein the controller requests the pixel data stored in the local memory.

6. The computer system of claim 5, wherein the controller merges pixel data received from a plurality of rendering circuits before drawing the image for display.

7. The computer system of claim 1 wherein the rendering circuit is further comprised of a router which examines packets from the transmission medium and routes the packets according to address information contained in the packets.

8. The computer system of claim 1 further comprising a single display driver which drives the display.

9. The computer system of claim 1, wherein the controller generates requests a pre-determined amount of clock cycles ahead of when pixel data is actually needed.

10. The computer system of claim 9, wherein the pre-determined amount of clock cycles is approximately equal to a fixed latency.

11. In a computer system, a method of rendering a three-dimensional image for display comprising the computer-implemented steps of:

rendering pixels of a three-dimensional image, wherein a plurality of rendering circuits are used to render portions of a single frame and each of the rendering pipes is capable of rendering pixels for an entire frame or portions thereof;

executing an application program on a host processor which issues graphics commands;

processing vertices by a geometry circuit coupled to the host processor;

generating pixel data through a rasterizer coupled to the geometry circuit;

storing the pixel data in a frame buffer coupled to the rasterizer;

accepting requests from the transmission medium for the pixel data;

outputting the pixel data onto the transmission medium;

storing pixel data in a plurality of memories, each rendering circuit storing pixel data generated in a local memory;

transmitting a request through a transmission medium coupling together each of the plurality of rendering circuits;

transmitting pixel data from one of the rendering circuits through the transmission medium to a frame buffer in response to the request;

merging pixel data received from a plurality of the rendering circuits into a frame;

driving a display coupled to the frame buffer to display the three-dimensional image.

12. The method of claim 11, wherein the transmission medium comprises a uni-directional ring topology.

13. The method of claim 12, wherein the transmission medium comprises a point-to-point connection.

14. The method of claim 11, wherein each of the rendering circuits performs the executing, processing, generating, storing, accepting, and outputting steps.

15. The method of claim 11, further comprising the step of routing packets from the transmission medium according to address information contained in the packets.

16. The method of claim 11 further comprising the step of driving the display with a single driver.

17. The method of Claim 11 futher comprising the step of generating requests at a pre-determined number of clock cycles ahead of when pixel data is actually needed.

18. The method of claim 17, wherein the pre-determined number of clock cycles is approximately equal to a fixed latency corresponding to the computer system.

* * * * *

Appendix C

US006885376B2

(12) **United States Patent**
Tang-Petersen et al.

(10) **Patent No.: US 6,885,376 B2**
(45) **Date of Patent: Apr. 26, 2005**

(54) **SYSTEM, METHOD, AND COMPUTER PROGRAM PRODUCT FOR NEAR-REAL TIME LOAD BALANCING ACROSS MULTIPLE RENDERING PIPELINES**

(75) Inventors: **Svend Tang-Petersen**, Mountain View, CA (US); **Yair Kurzion**, San Jose, CA (US)

(73) Assignee: **Silicon Graphics, Inc.**, Mountain View, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 80 days.

(21) Appl. No.: **10/330,217**

(22) Filed: **Dec. 30, 2002**

(65) **Prior Publication Data**

US 2004/0125111 A1 Jul. 1, 2004

(51) **Int. Cl.**[7] **G06T 1/20**

(52) **U.S. Cl.** **345/506**; 345/502; 345/505; 712/28; 712/32

(58) **Field of Search** 345/501, 502, 345/503, 504, 505, 506; 712/28, 31, 32

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,920,487 A	*	4/1990	Baffes	718/105
6,191,800 B1	*	2/2001	Arenburg et al.	345/505
6,683,614 B2	*	1/2004	Walls et al.	345/506
2003/0005100 A1	*	1/2003	Barnard et al.	709/223
2003/0164832 A1	*	9/2003	Alcorn	345/505
2003/0169269 A1	*	9/2003	Sasaki et al.	345/581

OTHER PUBLICATIONS

Schneider, B., "Parallel Polygon Rendering" [online], [Retrieved on Jun. 9, 2003]. Retrieved from the Internet: <URL:http://www.gris.uni-tuebingen.de/~bartz/tutorials/vis2000course/s5.pdf> (7 pages).

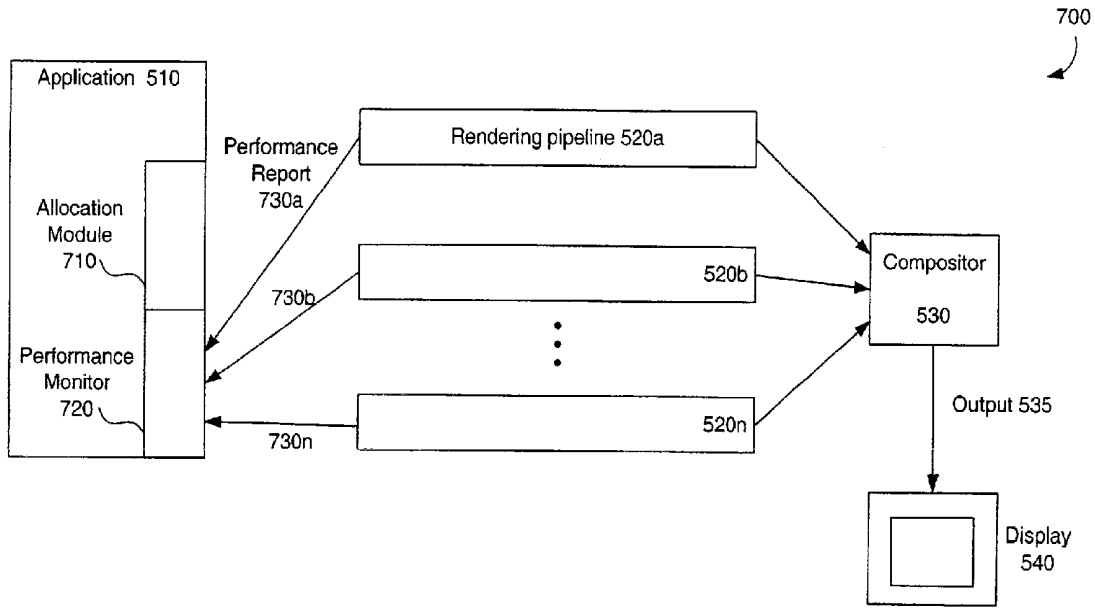
* cited by examiner

Primary Examiner—Matthew C. Bella
Assistant Examiner—Mackly Monestime
(74) *Attorney, Agent, or Firm*—Sterne, Kessler, Goldstein & Fox PLLC

(57) **ABSTRACT**

A system, method, and computer program product for creating a sequence of computer graphics frames, using a plurality of rendering pipelines. For each frame, each rendering pipeline receives a subset of the total amount of graphics data for the particular frame. At the completion of a frame, each rendering pipeline sends a performance report to a performance monitor. The performance monitor determines whether or not there was a significant disparity in the time required by the respective rendering pipelines to render their tiles. If a disparity is detected, and if the disparity is determined to be greater than some threshold, an allocation module resizes the tiles for the next frame. This serves to balance the load across rendering pipelines for each frame.

29 Claims, 18 Drawing Sheets



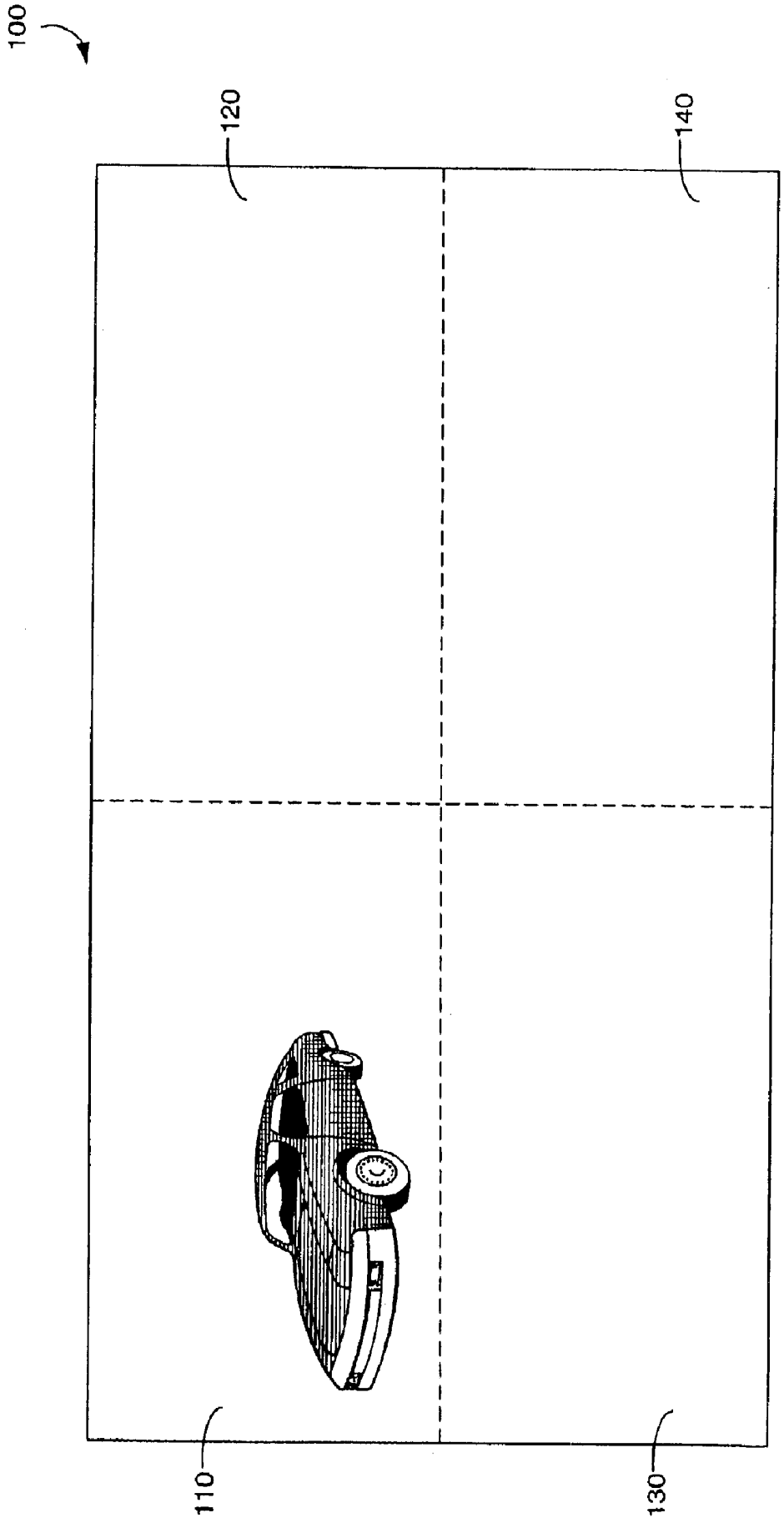


FIG. 1

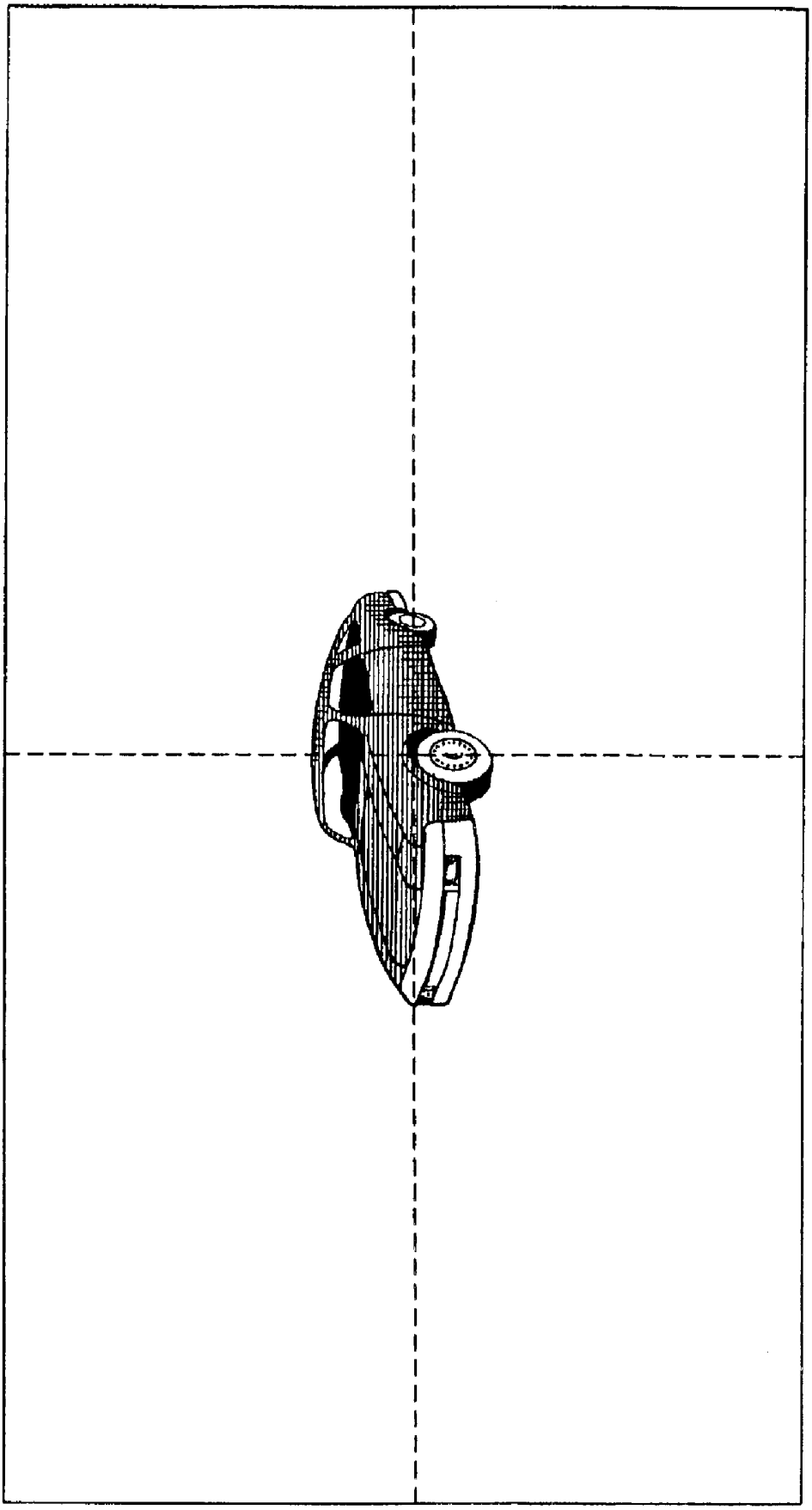


FIG. 2

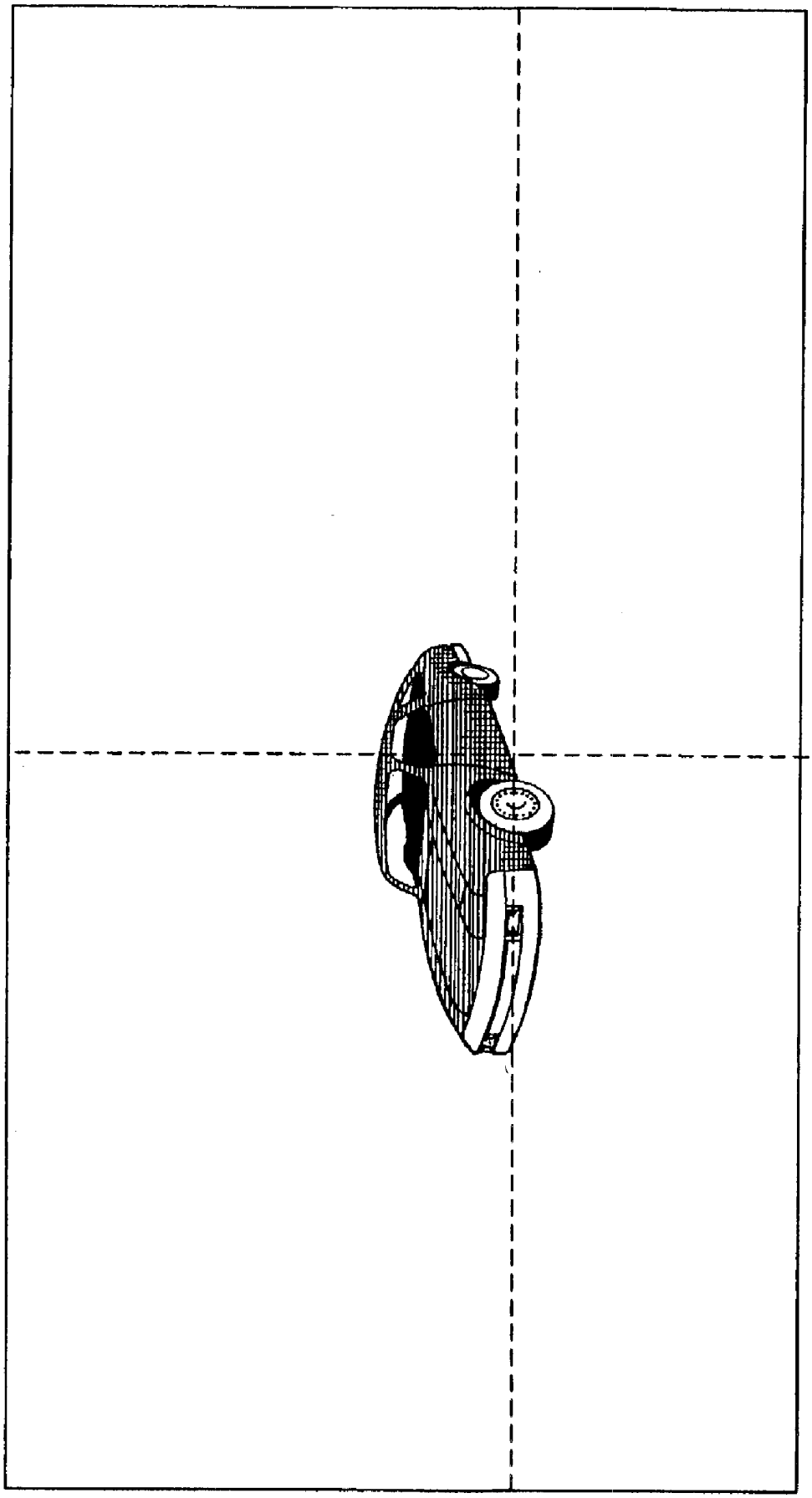


FIG. 3

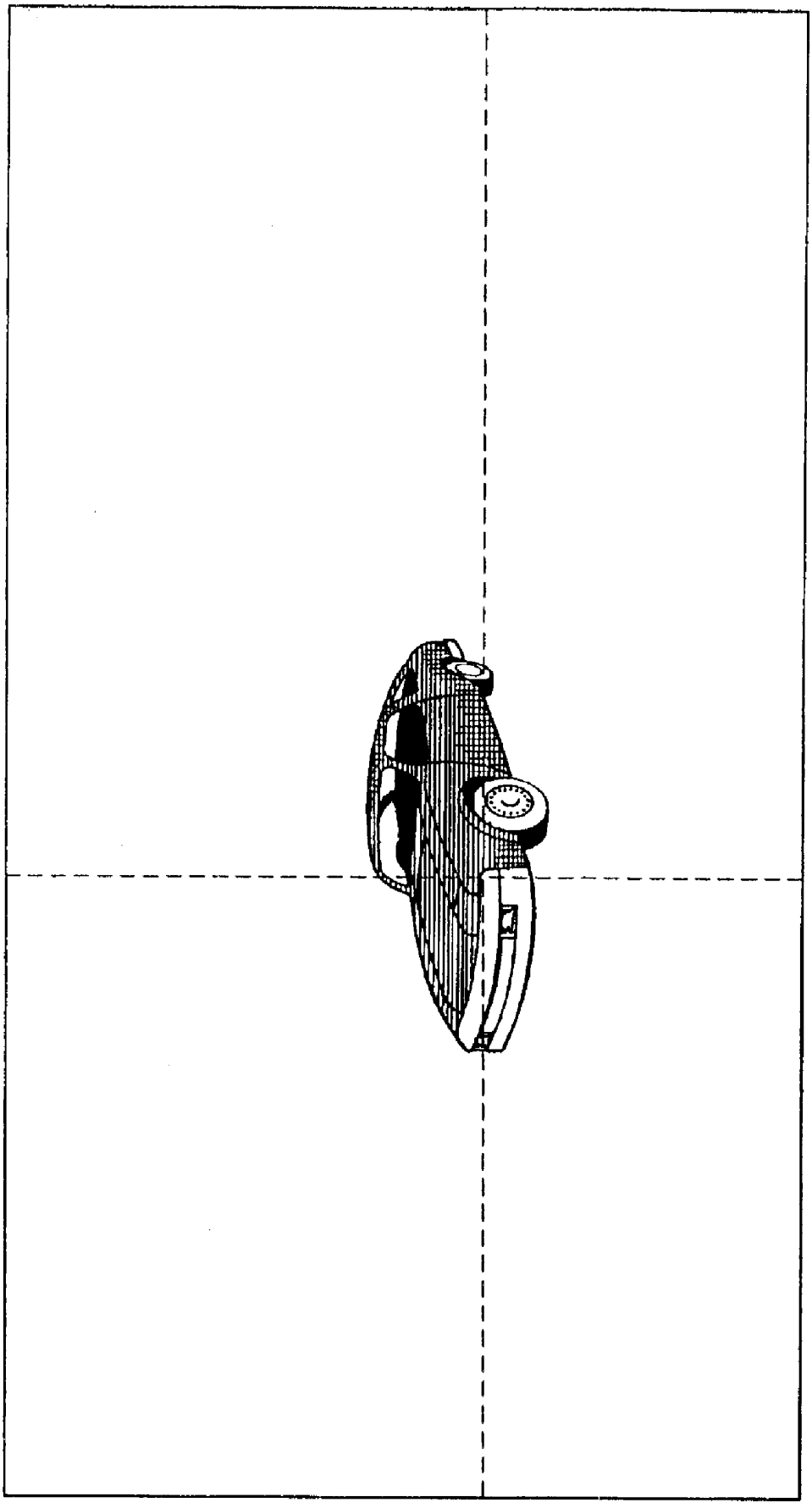


FIG. 4

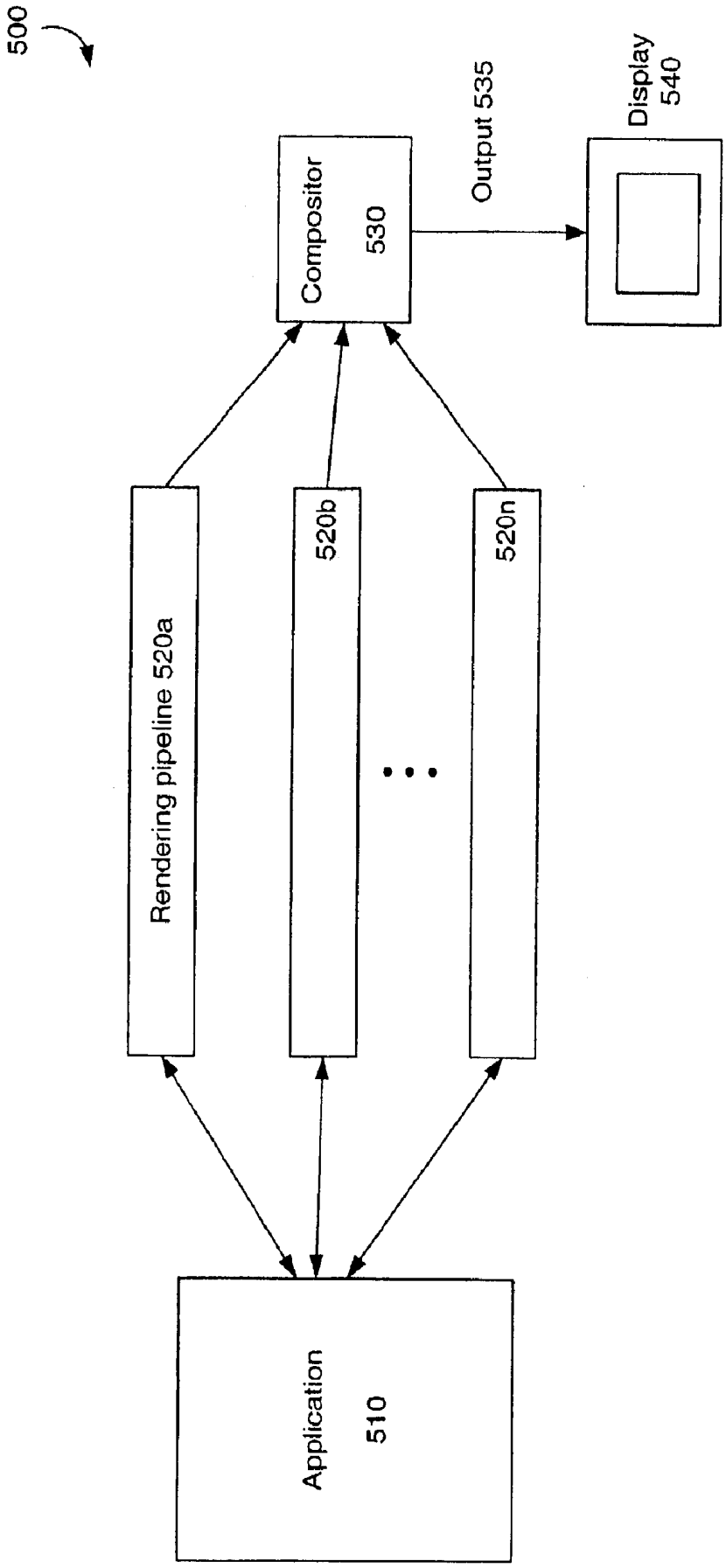

500
Display 540
Output 535
Compositor 530
Rendering pipeline 520a
520b
520n
Application 510

FIG. 5

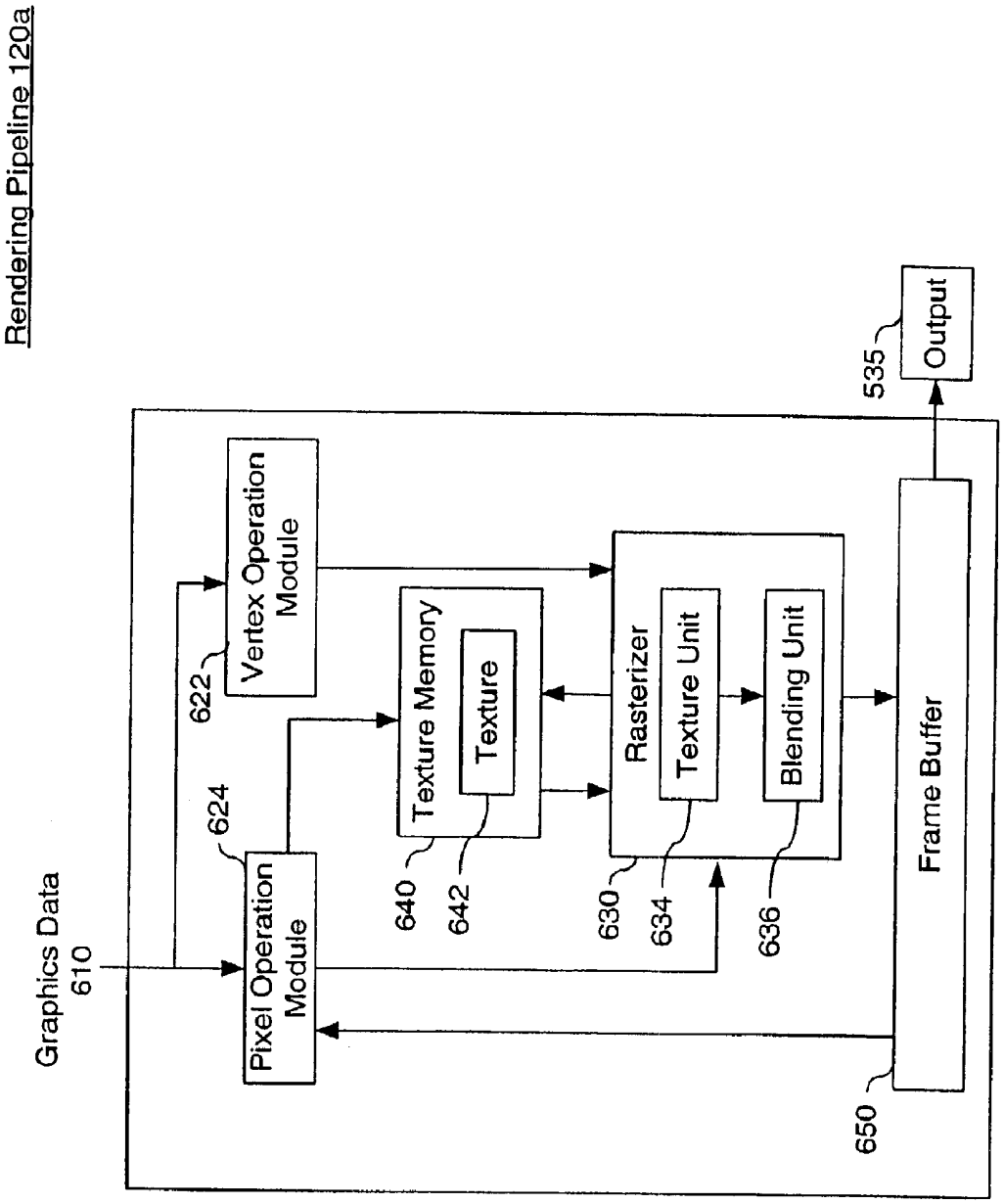

Rendering Pipeline 120a
Graphics Data
610
Pixel Operation Module
624
Vertex Operation Module
622
Texture Memory
Texture
640
642
Rasterizer
Texture Unit
Blending Unit
630
634
636
Frame Buffer
650
Output
535

FIG. 6

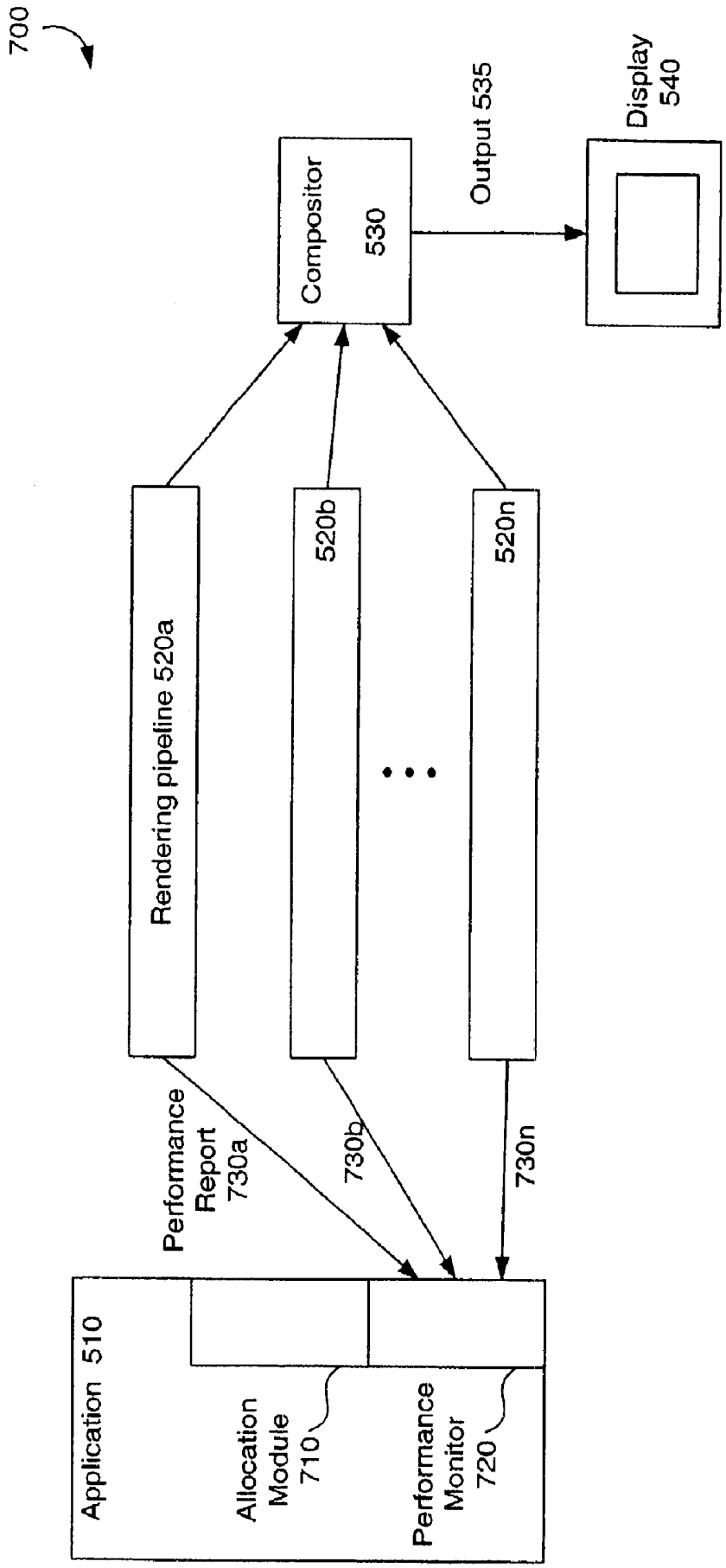

700
Compositor 530
Output 535
Display 540
Rendering pipeline 520a
520b
520n
Performance Report 730a
730b
730n
Application 510
Allocation Module 710
Performance Monitor 720

FIG. 7

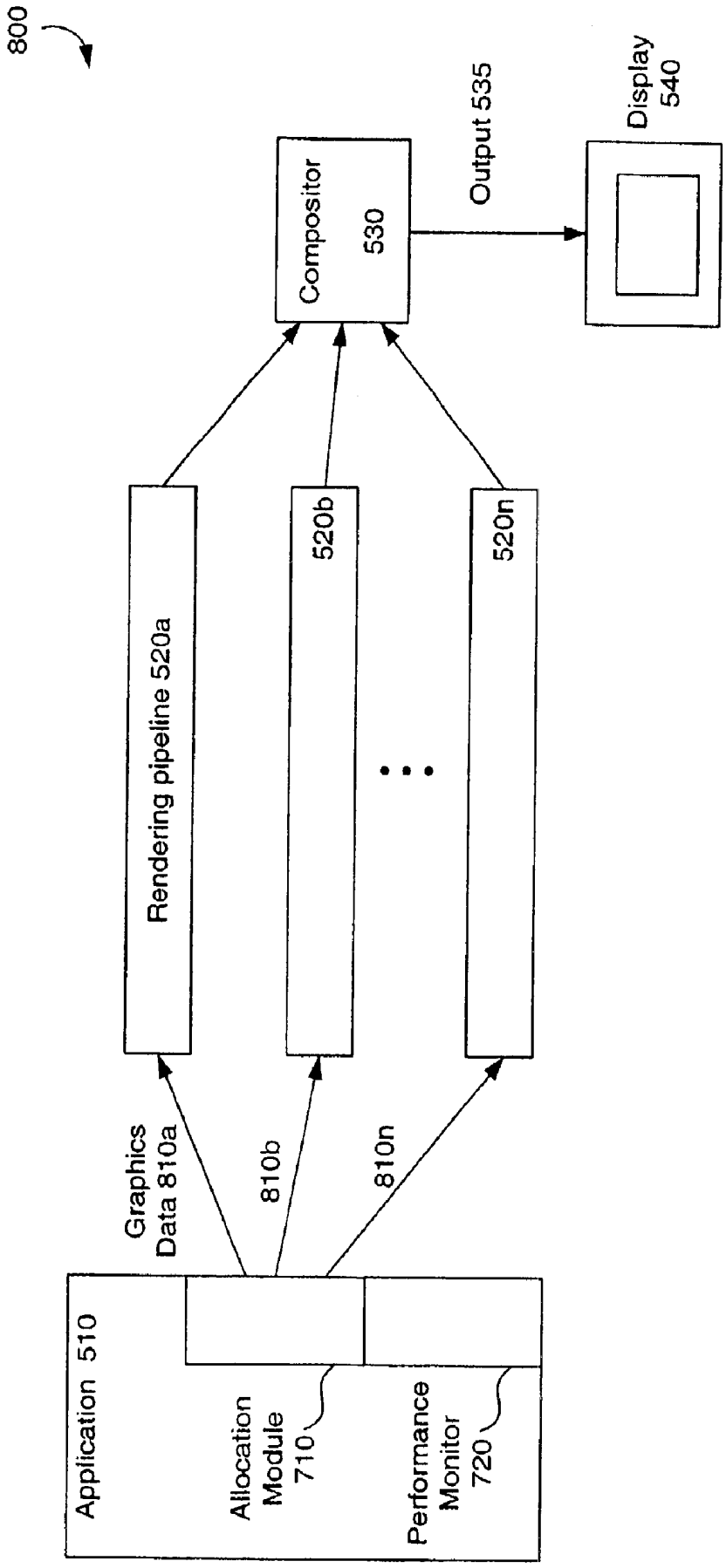

800
Display 540
Output 535
Compositor 530
Rendering pipeline 520a
520b
520n
Graphics Data 810a
810b
810n
Application 510
Allocation Module 710
Performance Monitor 720

FIG. 8

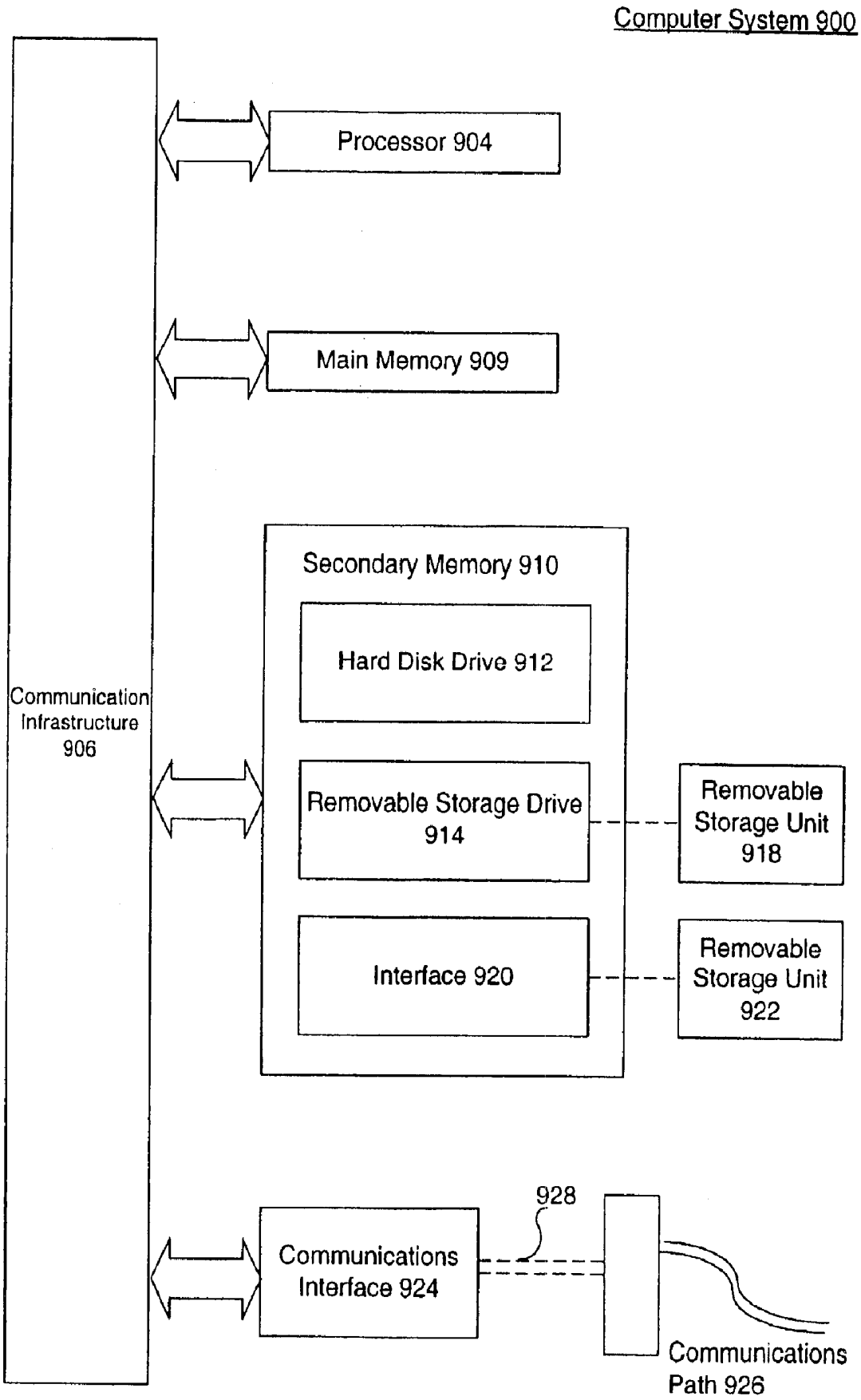

Computer System 900
Processor 904
Main Memory 909
Secondary Memory 910
Hard Disk Drive 912
Removable Storage Drive 914
Interface 920
Removable Storage Unit 918
Removable Storage Unit 922
Communication Infrastructure 906
Communications Interface 924
928
Communications Path 926

FIG. 9

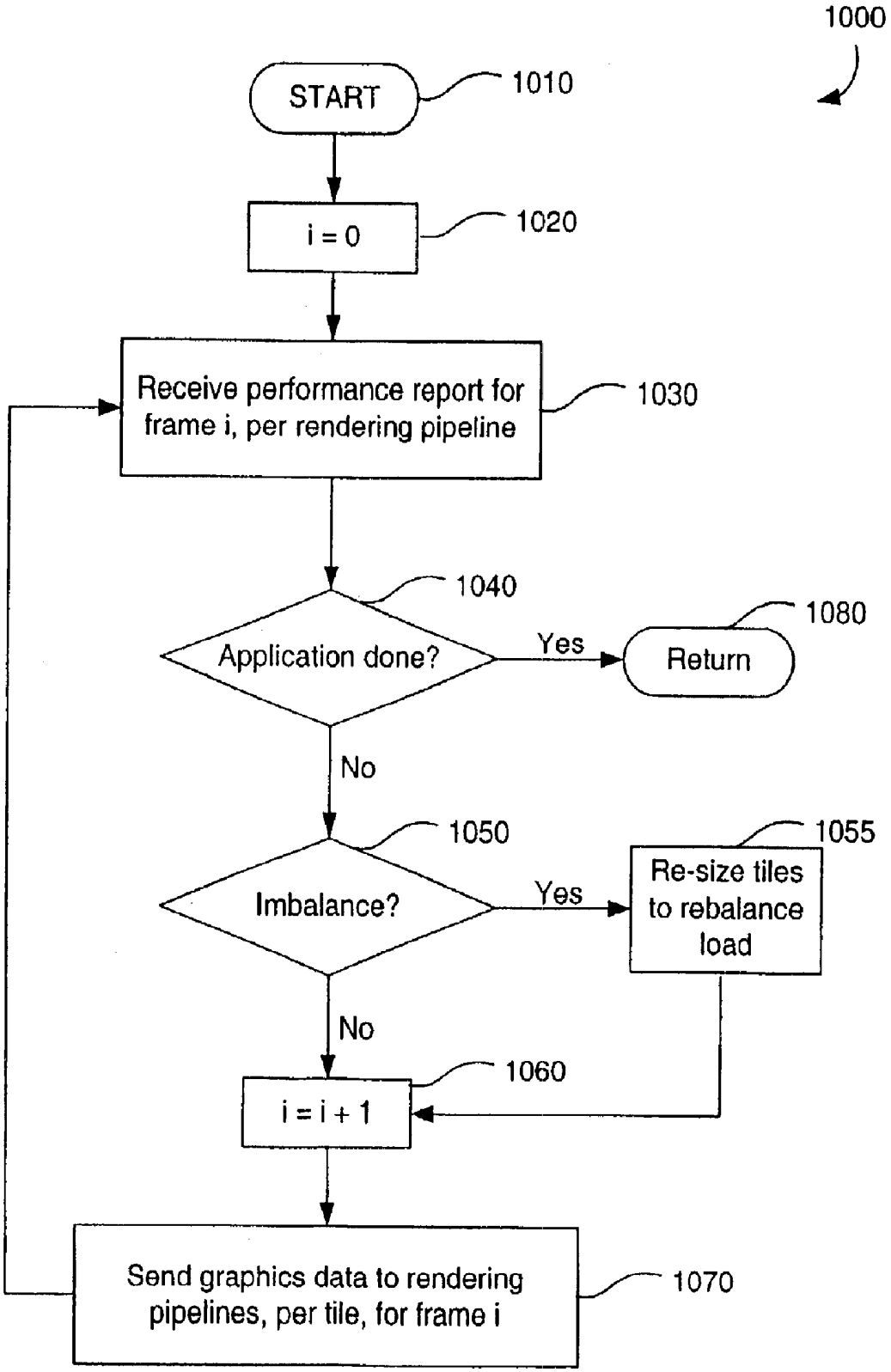

1000
START
1010
i = 0
1020
Receive performance report for frame i, per rendering pipeline
1030
1040
Application done?
Yes
1080
Return
No
1050
Imbalance?
Yes
1055
Re-size tiles to rebalance load
No
1060
i = i + 1
Send graphics data to rendering pipelines, per tile, for frame i
1070

FIG. 10

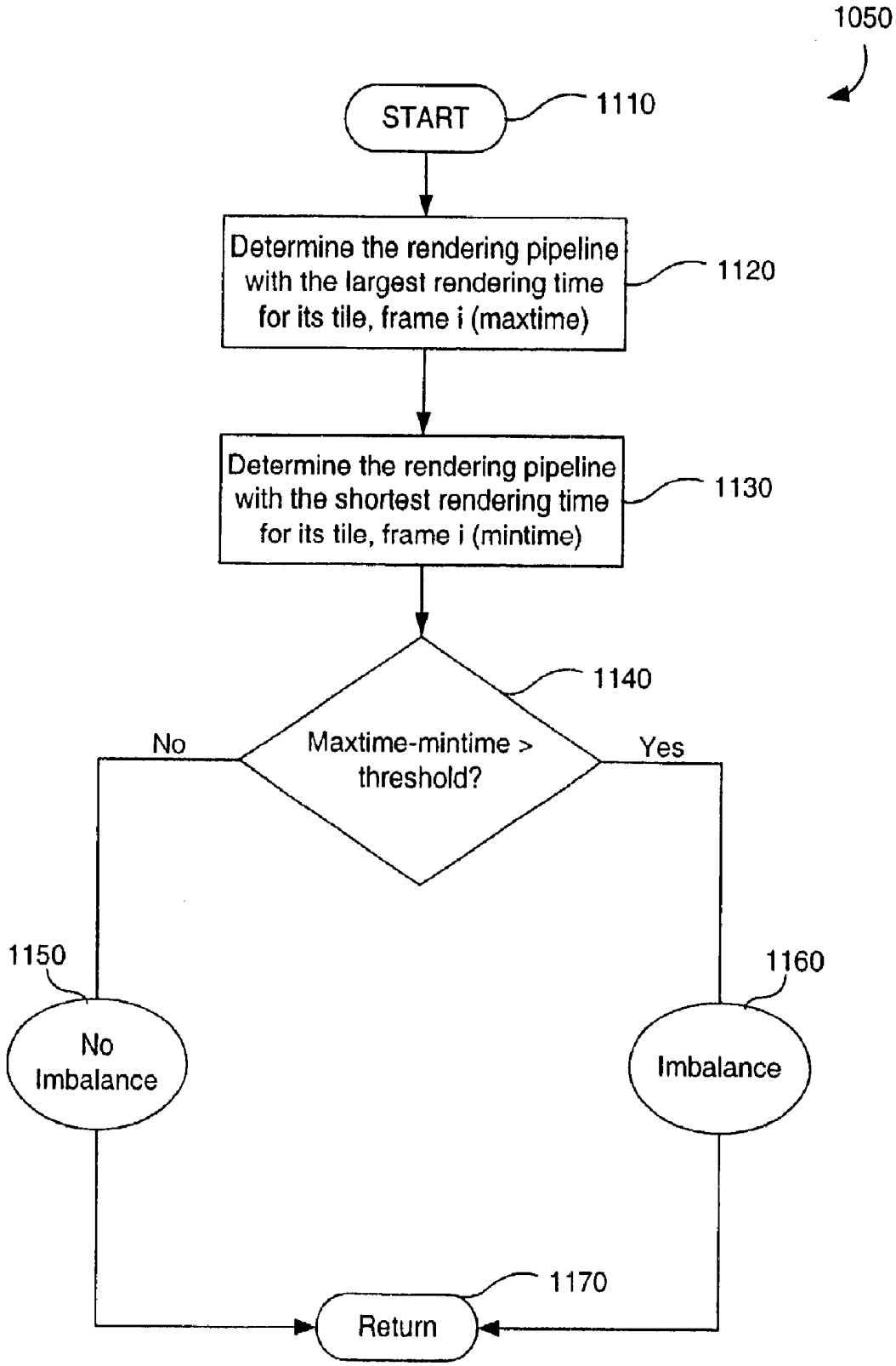

1050
START
1110
Determine the rendering pipeline with the largest rendering time for its tile, frame i (maxtime)
1120
Determine the rendering pipeline with the shortest rendering time for its tile, frame i (mintime)
1130
1140
Maxtime-mintime > threshold?
No
Yes
1150
No Imbalance
1160
Imbalance
1170
Return

FIG. 11

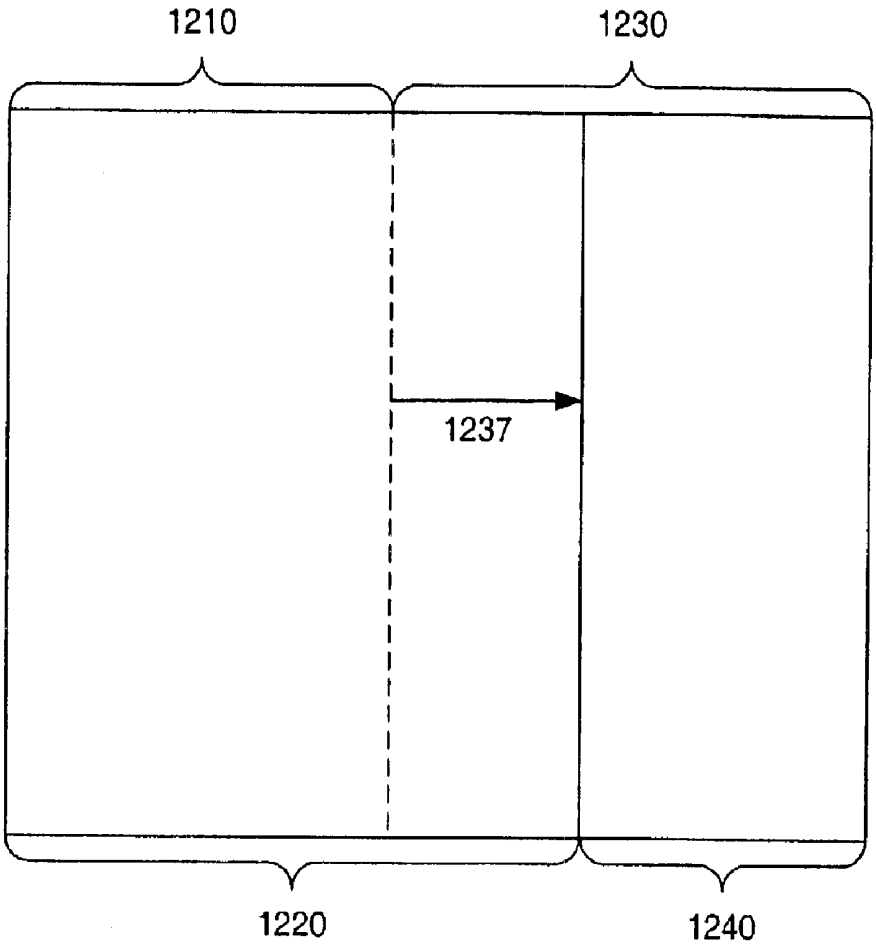


FIG. 12A

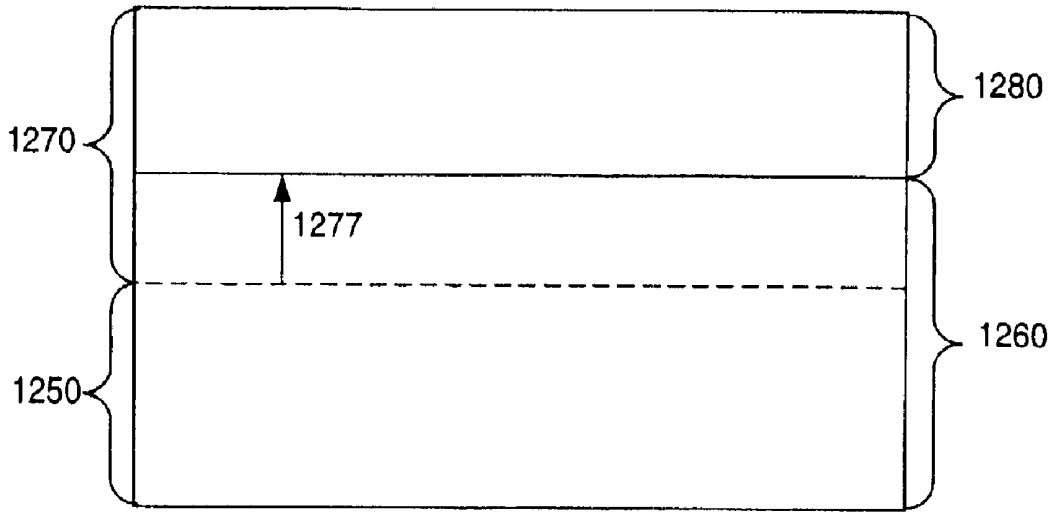


FIG. 12B

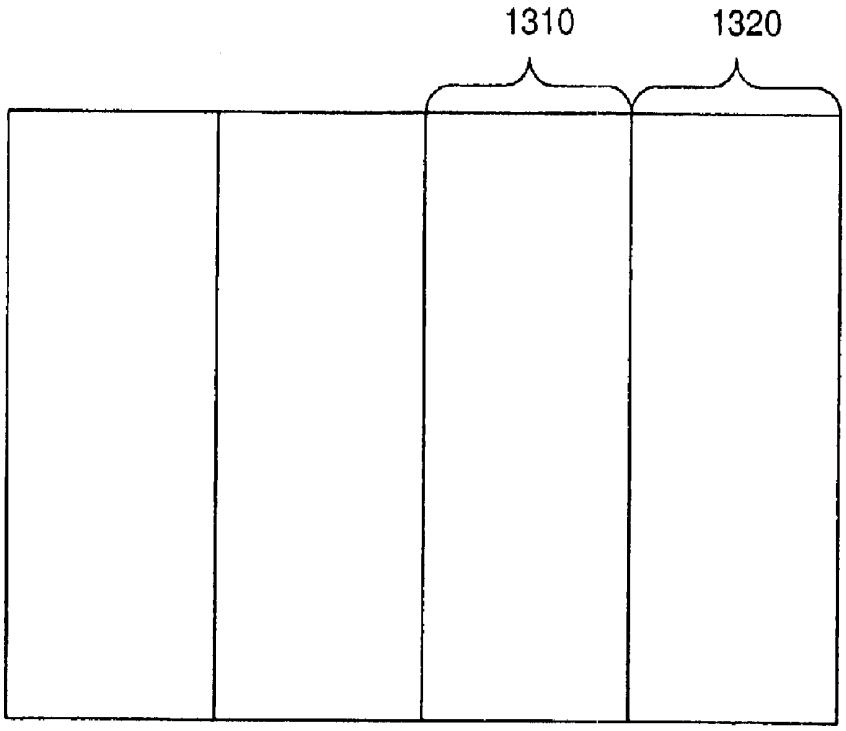


FIG. 13A

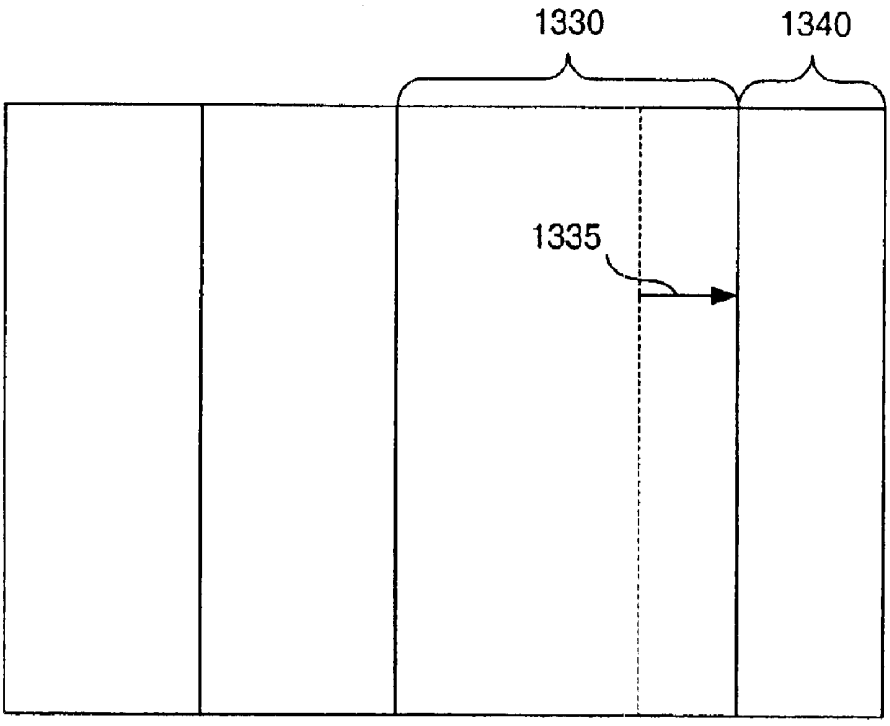


FIG. 13B

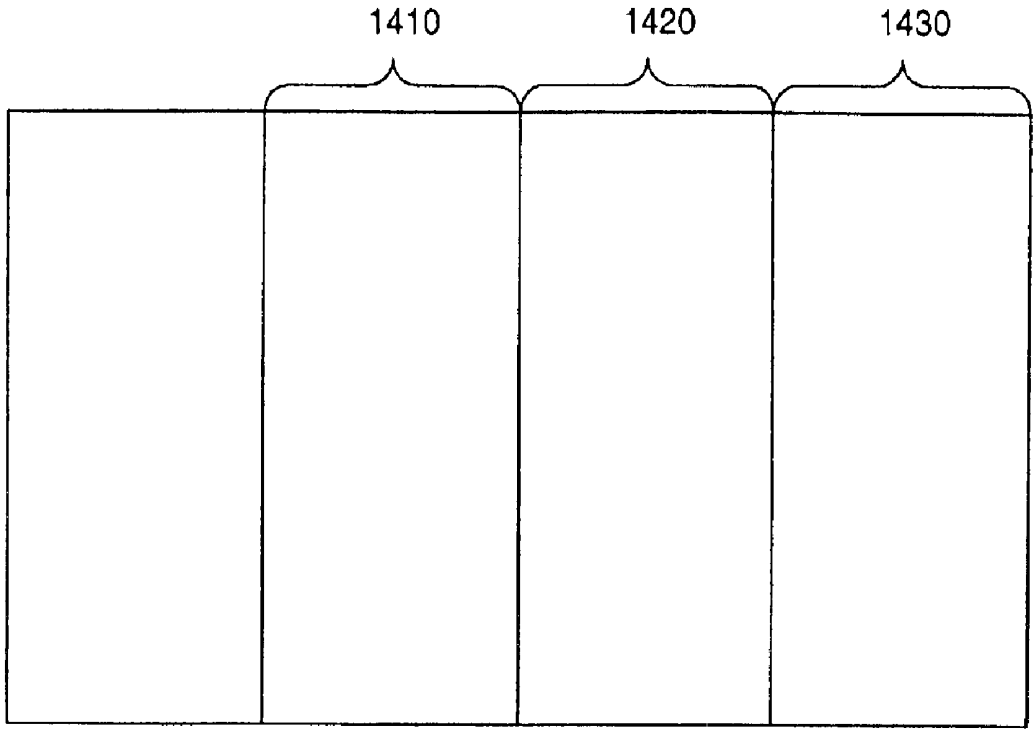


FIG. 14A

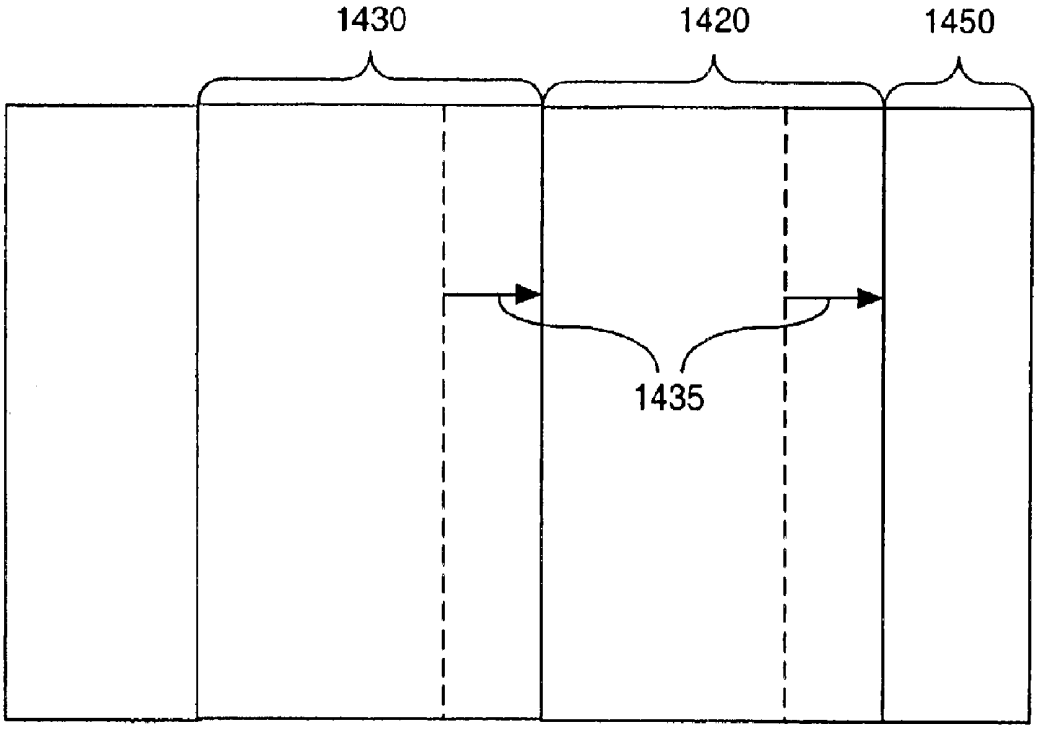


FIG. 14B

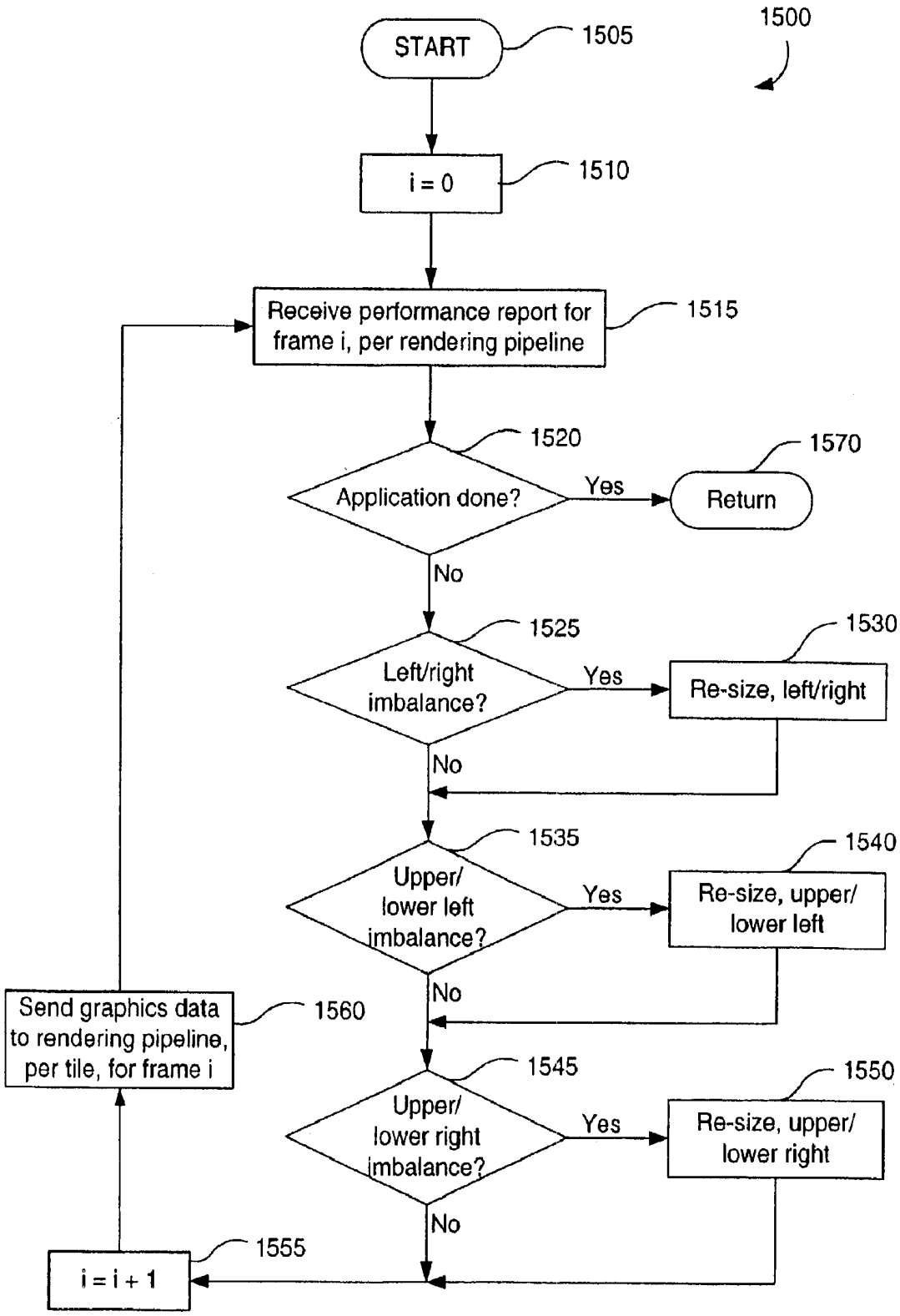

1500
START
1505
i = 0
1510
Receive performance report for frame i, per rendering pipeline
1515
1520
Application done?
Yes
1570
Return
No
1525
Left/right imbalance?
Yes
1530
Re-size, left/right
No
1535
Upper/ lower left imbalance?
Yes
1540
Re-size, upper/ lower left
No
1545
Upper/ lower right imbalance?
Yes
1550
Re-size, upper/ lower right
No
Send graphics data to rendering pipeline, per tile, for frame i
1560
i = i + 1
1555

FIG. 15

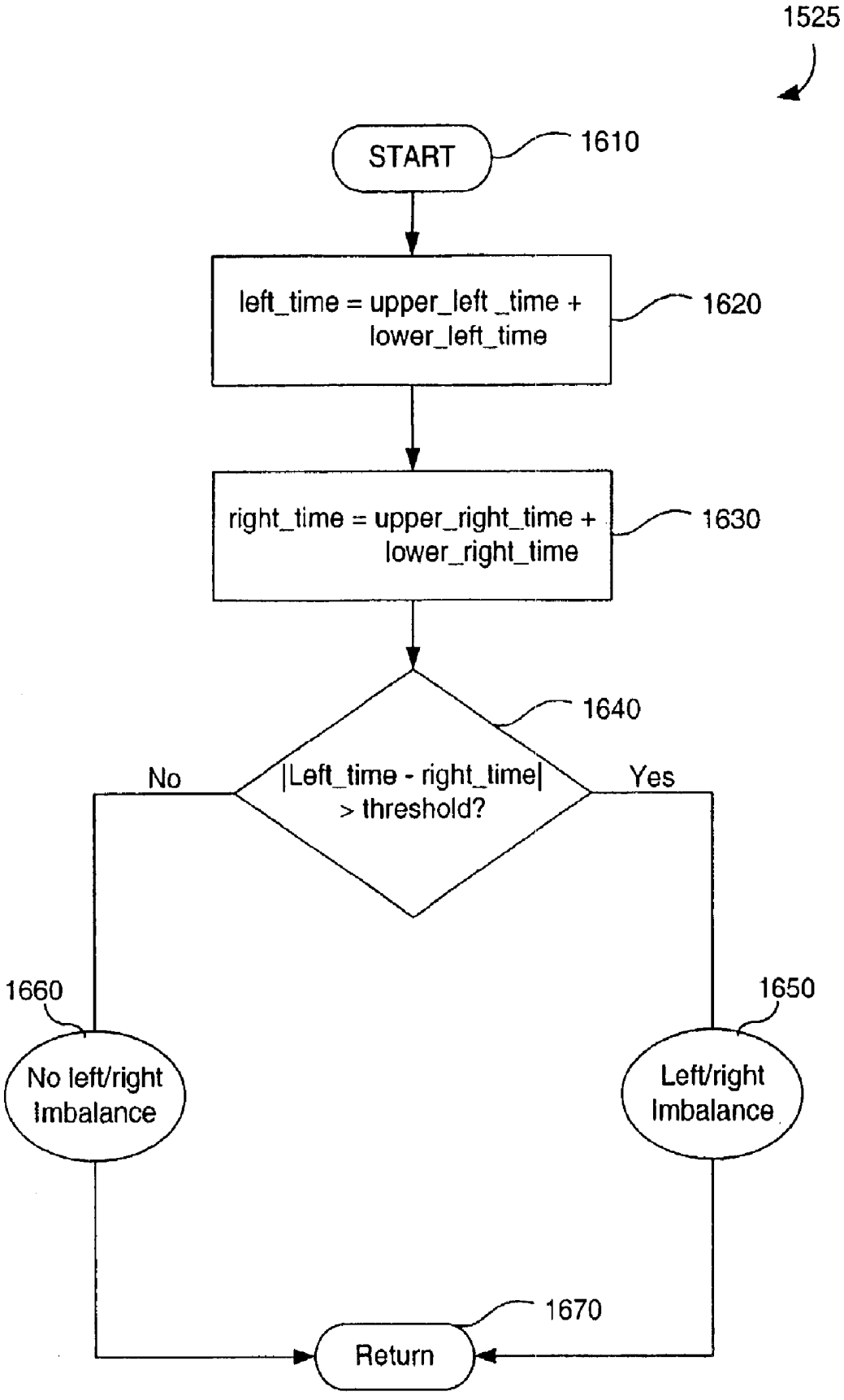

1525
START
1610
left_time = upper_left _time + lower_left_time
1620
right_time = upper_right_time + lower_right_time
1630
1640
|Left_time - right_time| > threshold?
No
Yes
1660
No left/right Imbalance
1650
Left/right Imbalance
1670
Return

FIG. 16

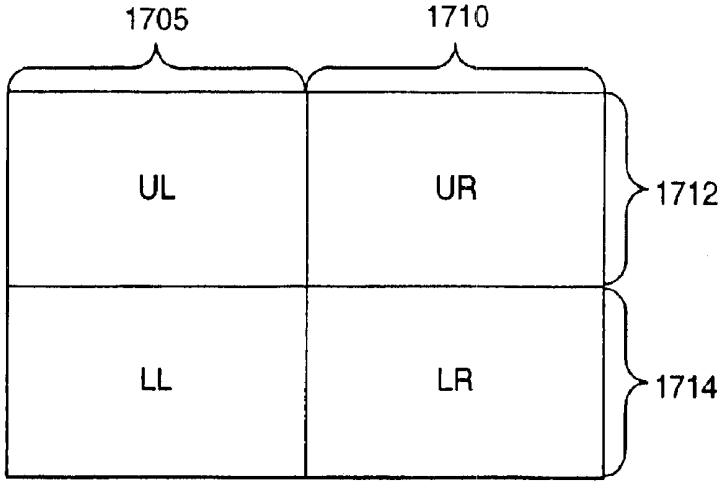


FIG. 17A

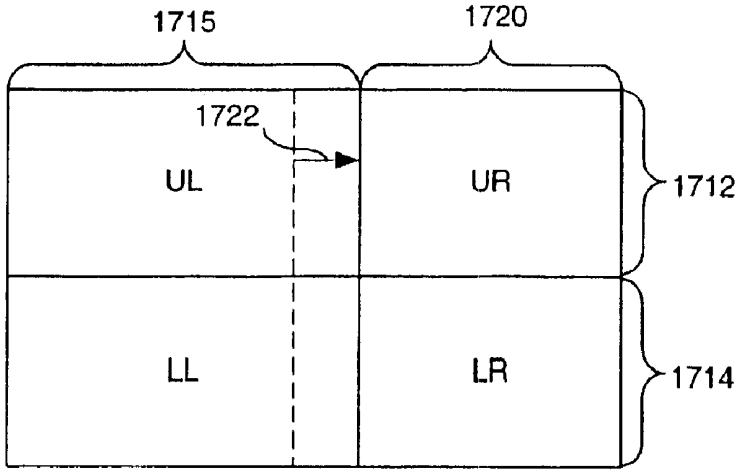


FIG. 17B

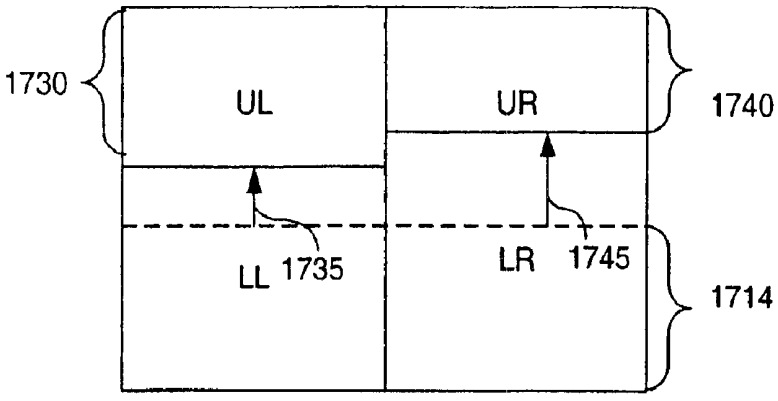


FIG. 17C

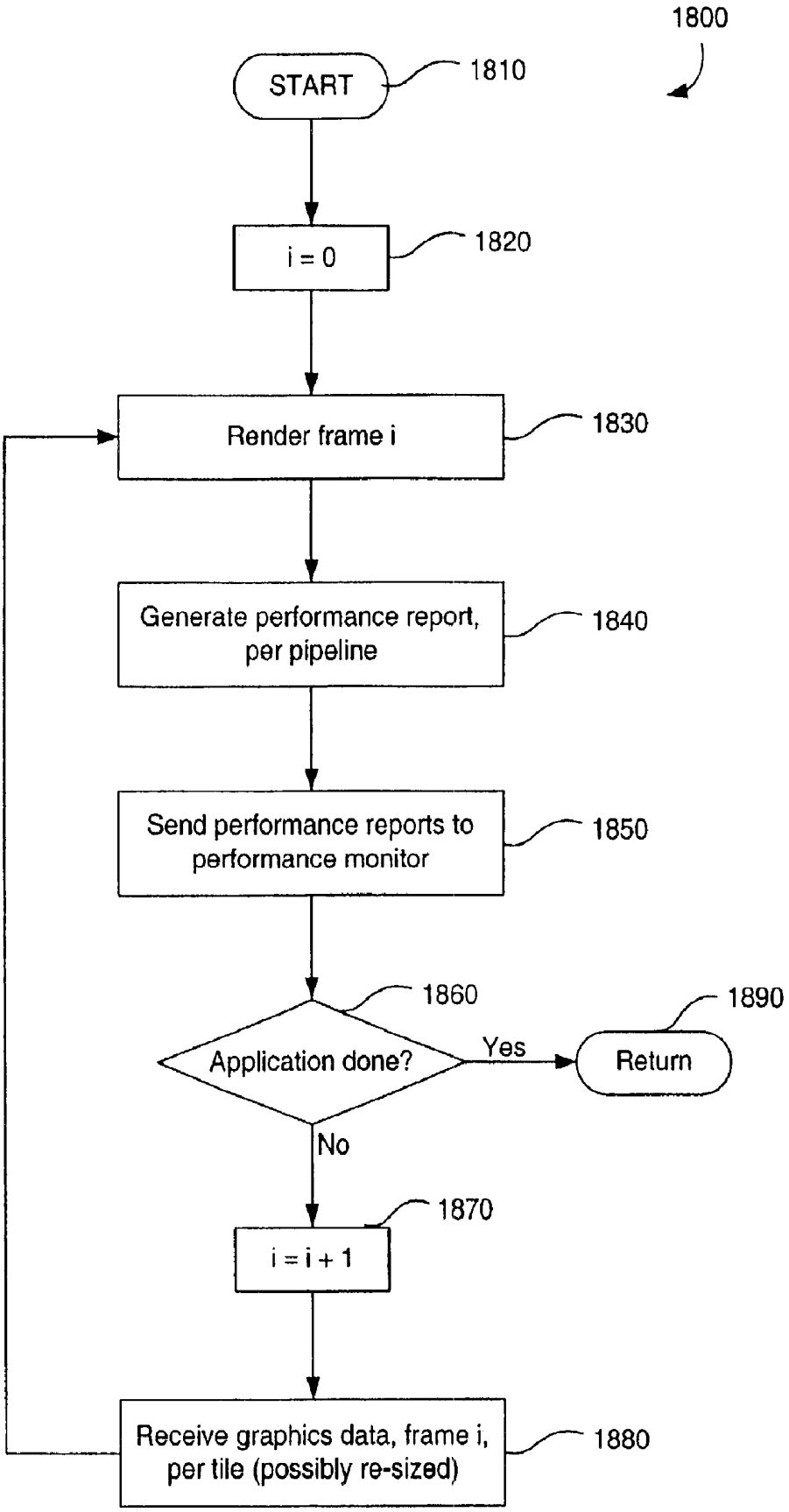

1800
START
1810
i = 0
1820
Render frame i
1830
Generate performance report, per pipeline
1840
Send performance reports to performance monitor
1850
1860
Application done?
Yes
1890
Return
No
1870
i = i + 1
Receive graphics data, frame i, per tile (possibly re-sized)
1880

FIG. 18

SYSTEM, METHOD, AND COMPUTER PROGRAM PRODUCT FOR NEAR-REAL TIME LOAD BALANCING ACROSS MULTIPLE RENDERING PIPELINES

CROSS-REFERENCE TO RELATED APPLICATIONS

Not applicable.

STATEMENT REGARDING FEDERALLY-SPONSORED RESEARCH AND DEVELOPMENT

Not applicable.

REFERENCE TO MICROFICHE APPENDIX/SEQUENCE LISTING/TABLE/COMPUTER PROGRAM LISTING APPENDIX (submitted on a compact disc and an incorporation-by-reference of the material on the compact disc)

Not applicable.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention described herein relates to computer graphics system performance.

2. Background Art

Computer graphics systems sometimes use parallelism in order to enhance performance. In particular, a graphics system may use more than one rendering pipeline in order to create an image. In such an architecture, each pipeline is responsible for rendering some portion of a frame. When rendering is completed, the outputs of the respective rendering pipelines are combined by a compositor to produce the complete frame. Such an arrangement can significantly increase throughput. If, for example, four equivalent rendering pipelines are used, then the time necessary to render a particular frame is, on average, one fourth of the time that would be required if a single rendering pipeline were used.

This is only an average however. Such a performance enhancement is only possible if the required processing is distributed evenly across all rendering pipelines for each frame. This is typically not the case. If, for example, four rendering pipelines are used, wherein each pipeline is responsible for rendering a particular quadrant of a frame, some quadrants may require more rendering than others. If this is the case, then rendering the frame can only proceed as quickly as the slowest rendering pipeline. The frame will only be completed when the slowest pipeline is finished. An example is shown in FIG. **1**. Here, a frame **100** is divided into four quadrants, **110**, **120**, **130** and **140**. If each quadrant is assigned to a particular rendering pipeline, it is apparent that the pipeline associated with quadrant **110** will have more processing to perform, compared to the other quadrants. As a result, rendering of this frame will only be completed when the rendering pipeline associated with quadrant **110** has finished. While this example may be an extreme case, it shows that even given multiple rendering pipelines, in some situations the performance of a multiple pipeline computer graphics system may not be much better than the performance of a single pipeline computer graphics system.

Hence, there is a need for a system and method by which parallelism inherent in a computer graphics system having multiple rendering pipelines is more fully exploited. In particular, the advantages of having multiple rendering pipelines need to be realized under all circumstances.

BRIEF SUMMARY OF THE INVENTION

The invention described herein is a system, method, and computer program product for creating a sequence of computer graphics frames using a plurality of rendering pipelines. For each frame, each rendering pipeline renders a subset of the total amount of graphics data. The output of each rendering pipeline represents a portion of the frame. In an embodiment of the invention, each portion of the frame is rectangular. Each rectangle is referred to hereinafter as a tile. Each rendering pipeline is therefore responsible for the rendering of its own particular tile in a given frame. After completion of a frame, each rendering pipeline produces a performance report. The performance report states the amount of time that was required to render a tile in the current frame.

At the completion of a frame, each rendering pipeline sends its performance report to a performance monitor. The performance monitor determines whether or not there was a significant disparity between the times required by the rendering pipelines to render their tiles. If a disparity is detected, and if the disparity is determined to be significant (i.e., greater than some threshold), then an allocation module resizes the tiles for the subsequent frame. If this is the case, the rendering pipeline bearing the largest processing load will have its tile reduced in size for purposes of the subsequent frame. This reduces the load of this pipeline. The rendering pipeline having the lowest processing load, as evidenced by its performance report, will then have its tile increased in size. The load on this pipeline is therefore increased. The latter pipeline will therefore have more processing to perform for purposes of the subsequent frame, while the former pipeline will have less rendering to perform. This serves to balance the load across rendering pipelines for the subsequent frame. This represents a near real time load balancing after each frame is rendered, allowing optimal use of the parallel architecture.

Further embodiments, features, and advantages of the present inventions, as well as the structure and operation of the various embodiments of the present invention, are described in detail below with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS/FIGURES

FIG. **1** illustrates a frame in which significantly more rendering is required for one tile, compared to the other tiles.

FIG. **2** illustrates a frame in which rendering is required for all tiles.

FIG. **3** illustrates a frame in which tiles have been resized to adjust the processing required in the respective rendering pipelines.

FIG. **4** illustrates a frame in which tiles are further resized, to further reallocate the processing burden across rendering pipelines.

FIG. **5** illustrates the overall system according to an embodiment of the invention.

FIG. **6** is a block diagram illustrating a rendering pipeline in greater detail, according to an embodiment of the invention.

FIG. **7** illustrates an embodiment of the invention, wherein the performance monitor and allocation module are incorporated in the graphics application.

FIG. **8** illustrates graphics data being distributed to the rendering pipelines, according to an embodiment of the invention.

FIG. **9** is an illustration of the computing environment of an embodiment of the invention.

FIG. **10** is a flowchart illustrating the processing of an embodiment of the invention when the tiles of a frame are arranged in a single row or column.

FIG. **11** is a flowchart illustrating in greater detail the determination of whether or not an imbalance exists across a set of rendering pipelines, according to an embodiment of the invention.

FIGS. **12**A and **12**B illustrate the resizing of tiles, wherein a given frame is composed of two tiles.

FIGS. **13**A and **13**B illustrate the resizing of tiles, wherein the frame consists of a single row of four tiles.

FIGS. **14**A and **14**B represent the resizing of tiles, wherein the frame consists of a row of four tiles and wherein the tile that is increasing in size is not adjacent to the tile that is decreasing in size.

FIG. **15** is a flowchart illustrating the processing of an embodiment of the invention, wherein the frame is composed of four tiles, arranged in two columns and two rows.

FIG. **16** is a flowchart illustrating the determination of whether an imbalance exists with respect to tiles on the left and tiles on the right in a frame composed of four tiles in a 2×2 arrangement, according to an embodiment of the invention.

FIGS. **17**A through **17**C represent the resizing of tiles in a frame in which tiles are initially configured in a 2×2 arrangement.

FIG. **18** is a flowchart illustrating the processing of an embodiment of the invention from the perspective of a rendering pipeline.

DETAILED DESCRIPTION OF THE INVENTION

I. Overview

The invention described herein is a system, method, and computer program product for creating a sequence of computer graphics frames using a plurality of rendering pipelines. For each frame, each rendering pipeline renders a subset of the total amount of graphics data. The output of each rendering pipeline represents a portion of the frame. In an embodiment of the invention, each portion of the frame is rectangular. Each rectangle is referred to hereinafter as a tile. Each rendering pipeline is therefore responsible for the rendering of its own particular tile in a given frame. After completion of a frame, each rendering pipeline then produces a performance report. The performance report states the amount of time that was required to render a tile in the current frame. At the completion of a frame, each rendering pipeline sends its performance report to a performance monitor. The performance monitor determines whether or not there was a significant disparity between the times required by the rendering pipelines to render their tiles. If a disparity is detected, and if the disparity is determined to be significant (i.e., greater than some threshold), then an allocation module resizes the tiles for the subsequent frame. If this is the case, the rendering pipeline bearing the largest processing load will have its tile reduced in size for purposes of the subsequent frame. This reduces the load of this pipeline. The rendering pipeline having the lowest processing load, as evidenced by its performance report, will then have its tile increased in size. The load on this pipeline is therefore increased. The latter pipeline will therefore have more processing to perform for purposes of the subsequent frame, while the former pipeline will have less rendering to perform. This serves to balance the load across rendering pipelines for the subsequent frame such that relatively little latency is experienced. This represents a near real time load balancing after each frame is rendered, allowing optimal use of the parallel architecture.

FIG. **2** illustrates a frame that has been subdivided into four tiles. A separate rendering pipeline is responsible for each tile. Hence, a first rendering pipeline renders the upper left quadrant of the frame of FIG. **2**; a second rendering pipeline renders the upper right quadrant of the frame; etc. When each rendering pipeline has completed it processing, the four resulting tiles are combined to form the frame of FIG. **2**.

After rendering is completed for the current frame, each rendering pipeline constructs and submits a performance report to the performance monitor. The performance monitor determines whether there is a disparity in the processing burdens of the respective rendering pipelines. If it is determined, for example, that one or both of the upper tiles took significantly longer to render than one or both of the lower tiles, then the processing load is not balanced evenly among the rendering pipelines. The tiles will be resized to reallocate the processing burden among the rendering pipelines. One possible result is shown in FIG. **3**. Here, the horizontal boundary separating the upper and lower tiles has been shifted down. Both upper quadrants are now larger; both lower quadrants are now smaller. As a result, each of the two rendering pipelines responsible for the upper tiles are responsible for rendering more of the frame. Moreover, the two rendering pipelines responsible for rendering the two lower tiles are responsible for less of the frame.

FIG. **4** illustrates a case where the horizontal boundary is lowered and the vertical boundary is moved to the left, relative to the frame of FIG. **2**. Again, this adjusts the processing workload of each of the four rendering pipelines. The rendering pipeline associated with the upper right tile has the greatest increase in rendering workload. The rendering pipeline responsible for the lower left tile of the frame has the largest decrease in rendering workload.

II. System

The system of the invention is illustrated generally in FIG. **5**. Computer graphics system **500** includes a graphics application program **510**. Application **510** is in communication with each of rendering pipelines **520***a*–**520***n*. This allows the distribution of graphics data from application **510** to each of the rendering pipelines **520***a*–**520***n*. At the completion of rendering, each rendering pipeline sends a performance report to a performance monitor (not shown). In an embodiment of the invention, the performance monitor is implemented as part of application **510**. Each pipeline's performance report indicates the amount of time required by that pipeline to render its tile of the current frame. Moreover, each of rendering pipelines **520***a*–**520***n* sends rendered data associated with its tile to a compositor **530**. Compositor **530** then combines the rendered data, i.e., the tiles, to produce output **535**, which can then be displayed at display **540**. In alternative embodiments of the invention, output **535** can be sent to a different form of input/output (I/O) device, such as a printer or a memory medium.

FIG. **6** illustrates a rendering pipeline in greater detail. Rendering pipeline **600** is illustrative and not intended to limit the scope of the present invention. Other types of rendering pipelines can be used as would be apparent to a person skilled in the art, given this description. Therefore, while rendering pipelines **520***a* through **520***n* can have the

structure shown in FIG. **6**, other embodiments of rendering pipelines can be used. Moreover, rendering pipelines **520***a* through **520***n* need not be identical.

Rendering pipeline **600** comprises a vertex operation module **622**, a pixel operation module **624**, a rasterizer **630**, a texture memory **640**, and a frame buffer **650**. Rendering pipeline receives graphics data **610**, which is initially routed to vertex operation module **622** and a pixel operation module **624**. Texture memory **640** can store one or more textures or images, such as texture **642**. Texture memory **640** is connected to a texture unit **634** by a bus (not shown). Rasterizer **630** comprises texture unit **634** and a blending unit **636**. Texture unit **634** and blending unit **636** can be implemented separately or together as part of a graphics processor. The operation of these features of rendering pipeline **600** would be known to a person skilled in the relevant art given the description herein.

In embodiments of the present invention, texture unit **634** can obtain either a point sample or a filtered texture sample from textures and/or images **642** stored in texture memory **640**. Blending unit **636** blends texels and/or pixel values according to weighting values to produce a single texel or pixel. The output of texture unit **638** and/or blending unit **636** is stored in frame buffer **650**. The contents of frame buffer **650** can then be read out as output **670**.

FIG. **7** illustrates the system of the invention in greater detail during its performance reporting operation. Each of the rendering pipelines **520***a*–**520***n* send a performance report, labeled **730***a*–**730***n*, respectively, to a performance monitor **720**. In the illustrated embodiment, performance monitor **720** is incorporated in application **510**. Performance reporting takes place after rendering each frame. Performance monitor **720** determines whether the performance reports indicate any disparity in the workloads of the respective rendering pipelines **520***a* through **520***n*. In an embodiment of the invention, performance monitor **720** identifies the rendering pipeline that required the greatest amount of time to render its tile for the current frame, and identifies the rendering pipeline that required the least amount of time to render its tile for the current frame. If the difference in the two times exceeds a threshold value, a conclusion is reached that a significant disparity exists. Such a conclusion is then passed to allocation module **710**. Note that in an alternative embodiment of the invention, the workload of a rendering pipeline, as given in a performance report, is stated in terms of clock cycles.

In FIG. **8**, allocation module **710** sends graphics data to each of rendering pipelines **520***a*–**520***n*. Each rendering pipeline receives a distinct subset of the total graphics data required for the subsequent frame. Hence, rendering pipeline **520***a* receives graphics data **810***a*. Likewise, rendering pipeline **520***b* receives graphics data **810***b*, etc. The graphics data sent to each rendering pipeline reflects any changes that may have been made to the sizes and shapes of the tiles associated, respectively, with rendering pipelines **520***a*–**520***n*. The process of resizing tiles is performed by allocation module **710** and is described in greater detail below.

Note that in the embodiment illustrated in FIGS. **7** and **8**, allocation module **710** and performance monitor **720** are shown as components of graphics application **510**. In an alternative embodiment of the invention, these modules can reside external to application **510**. These modules may be implemented as software, hardware, or firmware, or as some combination thereof.

The allocation module **710** and performance monitor **720** of the present invention may be implemented using hardware, software or a combination thereof. In an embodiment of the invention, they are implemented in software as part of application program **510**, which is executed on a computer system or other processing system. An example of such a computer system **900** is shown in FIG. **9**. The computer system **900** includes one or more processors, such as processor **904**. The processor **904** is connected to a communication infrastructure **906**, such as a bus or network. After reading this description, it will become apparent to a person skilled in the relevant art how to implement the invention using other computer systems and/or computer architectures.

Computer system **900** also includes a main memory **908**, preferably random access memory (RAM), and may also include a secondary memory **910**. The secondary memory **910** may include, for example, a hard disk drive **912** and/or a removable storage drive **914**. The removable storage drive **914** reads from and/or writes to a removable storage unit **918** in a well known manner. Removable storage unit **918** represents a floppy disk, magnetic tape, optical disk, or other storage medium which is read by and written to by removable storage drive **914**. The removable storage unit **918** includes a computer usable storage medium having stored therein computer software and/or data.

In alternative implementations, secondary memory **910** may include other means for allowing computer programs or other instructions to be loaded into computer system **900**. Such means may include, for example, a removable storage unit **922** and an interface **920**. Examples of such means may include a removable memory chip (such as an EPROM, or PROM) and associated socket, and other removable storage units **922** and interfaces **920** which allow software and data to be transferred from the removable storage unit **922** to computer system **900**.

Computer system **900** may also include a communications interface **924**. Communications interface **924** allows software and data to be transferred between computer system **900** and external devices. Examples of communications interface **924** may include a modem, a network interface (such as an Ethernet card), a communications port, a PCMCIA slot and card, etc. Software and data transferred via communications interface **924** are in the form of signals **928** which may be electronic, electromagnetic, optical or other signals capable of being received by communications interface **924**. These signals **928** are provided to communications interface **924** via a communications path (i.e., channel) **926**. This channel **926** carries signals **928** and may be implemented using wire or cable, fiber optics, a phone line, a cellular phone link, an RF link and other communications channels. In an embodiment of the invention, signals **928** comprise performance reports **730***a* through **730***n*, received for processing by performance monitor **720**. Information representing graphics data **810***a* through **810***n* can also be sent in the form of signals **928** from processor **904** to rendering pipelines **520***a* through **520***n*.

In this document, the terms "computer program medium" and "computer usable medium" are used to generally refer to media such as removable storage units **918** and **922**, a hard disk installed in hard disk drive **912**, and signals **928**. These computer program products are means for providing software to computer system **900**.

Computer programs (also called computer control logic) are stored in main memory **908** and/or secondary memory **910**. Computer programs may also be received via communications interface **924**. Such computer programs, when executed, enable the computer system **900** to implement the present invention as discussed herein. In particular, the

computer programs, when executed, enable the processor **904** to implement the present invention. Accordingly, such computer programs represent controllers of the computer system **900**. Where the invention is implemented using software, the software may be stored in a computer program product and loaded into computer system **900** using removable storage drive **914**, hard drive **912** or communications interface **924**.

III. Method

The method of the invention according to one embodiment is illustrated in FIG. **10**. This embodiment addresses processing when the tiles of a frame are configured in a single row or a single column (i.e., 1 x n or n x 1). The process begins at step **1010**. In step **1020**, an index value i is initialized to zero. In step **1030**, a performance monitor receives a performance report for frame i, for each rendering pipeline. In step **1040**, a determination is made as to whether the application has completed running. If so, the process concludes in step **1080**. If the application is not yet finished, the process continues at step **1050**. Here a determination is made as to whether an imbalance exists among the rendering pipelines as to the amount of time that was required to render their respective tiles in the current frame. This determination is illustrated in greater detail below.

If an imbalance is detected, then the process continues at step **1055**. Here, the tiles are resized so as to rebalance the load among rendering pipelines for purposes of rendering the subsequent frame. For example, if the tiles of a frame are arrayed as a single row, and the imbalance exists between two rendering pipelines that correspond to adjacent tiles, the resizing can be expressed numerically as follows:

$$\text{pixelshift}=0.5*(\text{maxtime}-\text{mintime})*(\text{width of maxtile})/\text{maxtime}.$$

This equation describes the amount by which the vertical boundary between the two adjacent tiles needs to be shifted. Maxtime refers to the amount of time required by the rendering pipeline that took the longest to render its tile. Similarly, mintime refers to the time required by the rendering pipeline that took the shortest time to render its tile. Width of maxtile is the width of the tile that took the longest to render.

Note that if the tiles corresponding to maxtime and mintime are not immediately adjacent, then the intervening tile or tiles maintain their current width and are repositioned in the direction of the reduced tile. Also, if the tiles are arranged in a single column instead of a single row, then the boundaries between tiles are horizontal, and the shift of boundaries is vertical. Hence, width of maxtile above is replaced by the height of maxtile. These variations are described in greater detail below.

In step **1060**, the index i is incremented by one in order to process the subsequent frame. In step **1070**, graphics data is sent by the allocation module to the rendering pipelines. Allocation of graphics data is done on a per tile basis. All graphics data associated with a particular tile is sent to a particular rendering pipeline. If necessary, the tiles will have been resized according to the process described with respect to step **1055** above. Processing returns to step **1030** once rendering of the subsequent frame is completed.

Step **1050** above, the determination of whether or not an imbalance exists among rendering pipelines, is illustrated in greater detail in FIG. **11**. The process starts at step **1110**. In step **1120**, the maxtime rendering pipeline is determined, i.e., the pipeline with the longest rendering time for its tile in the current frame. In step **1130**, the mintime rendering pipeline is determined, i.e., the pipeline with the shortest rendering time for its tile in the current frame. In step **1140**, a determination is made as to whether the difference between maxtime and mintime exceeds the threshold value. If so, then an imbalance is detected (condition **1160**); if not, then no imbalance is detected (condition **1150**). The process is concluded at step **1170**.

In an embodiment of the invention, the threshold is defined to be a fixed percentage of maxtime. For example, the threshold can be defined to be 10% of maxtime. In this case, if the difference between maxtime and mintime exceeds 10% of maxtime, then an imbalance is detected. Depending on the size of the dataset being rendered, a different percentage may be appropriate. For some datasets, 10% may be appropriate. However, 1 or 2% may be more appropriate for a larger dataset since, for a large dataset, 1 or 2% of maxtime can be a significant disparity.

The resizing of tiles, described above with respect to step **1055**, is illustrated in FIGS. **12**A and **12**B. The frame of FIG. **12**A is composed of two tiles. Their widths in the current frame are labelled **1210** and **1230**. In this example, an imbalance has been detected in the time required by the respective rendering pipelines, such that the tile having width **1230** has taken significantly longer to render than the tile having width **1210**. As a result, the boundary between the two tiles is shifted by distance **1237**. Hence, for the next frame, the tiles have widths of **1220** and **1240** respectively. This resizing alters the amount of graphics data that must be rendered by each rendering pipeline, so that the respective workloads are more balanced for the next frame.

FIG. **12**B shows a frame consisting of two tiles arranged vertically. In the current frame, the tiles have heights **1250** and **1270**. A determination is then made that a significant imbalance exists in the rendering times for the two tiles. In particular, the tile having width **1270** has taken significantly longer to render than the tile having width **1250**. As a result, the boundary between the two tiles is shifted upward by a distance **1277**. In the next frame, therefore, one tile has a height **1280**, while the other has a height **1260**. Hence, for the next frame, one rendering pipeline now has a smaller tile to render, while the other rendering pipeline has a larger tile to render compared to the previous frame. The respective workloads are now more balanced for the next frame.

FIGS. **13**A and **13**B illustrate a frame composed of four tiles arranged as a single row. In the current frame, shown in FIG. **13**A, two adjacent frames have widths **1310** and **1320**. A determination is then made that, of the four rendering pipelines, the rendering pipeline associated with the tile having width **1310** required the shortest amount of time to render its tile (mintime), while the rendering pipeline associated with the tile having width **1320** took the longest to render its tile (maxtime). Moreover, it is determined that the difference in these two rendering times is so significant as to be an imbalance. Hence, as shown in FIG. **13**B, the tile that required the shortest amount of time to render is increased in width by a distance **1335**. The tile that had taken the longest amount of time to render is shrunk by a corresponding amount. In the next frame, therefore, these two tiles have widths **1330** and **1340** respectively, and their associated rendering pipelines have accordingly altered workloads. The other two tiles remain unchanged in width.

FIGS. **14**A and **14**B represent the situation where an imbalance has been detected, but the tile that required the longest amount of time to render and the tile requiring the shortest amount of time to render are separated by one or more intervening tiles. In FIG. **14**A, the tile having width **1410** is determined to have taken the shortest amount of time to render, while the tile having width **1430** is determined to have taken the longest amount of time to render. As a result,

the tile boundaries shift as shown in FIG. **14**B. In particular, the right boundary of the frame that took the shortest amount of time to render is shifted to the right by a distance **1435**. Also, the left boundary of the frame that required the longest amount of time to render is shifted to the right by the same distance. Hence, for purposes of the next frame, these two tiles have widths **1430** and **1450** respectively. Note, that the intervening tile having width **1420** has been repositioned to the right, but retains its original width. Hence, this tile has been repositioned but not resized.

Another embodiment of the process of the invention is illustrated in FIG. **15**. This embodiment addresses the situation where a frame is divided into four tiles, two rows by two columns. The process begins with step **1505**. In step **1510**, an index value is initialized to zero. In step **1515**, the performance monitor receives a performance report for the current frame from each rendering pipeline. In step **1520**, a determination is made as to whether the application has finished running. If so, the process concludes at step **1570**. Otherwise, the process continues at step **1525**. Here, a determination is made as to whether an imbalance exists as to rendering of the two half frames (i.e., the two left tiles, taken collectively, in comparison to the two right tiles, taken collectively). This determination is described in greater detail below.

If such an imbalance exists, then processing continues at step **1530**. Here, the vertical boundary is shifted, either left or right, thereby resizing all four tiles. The magnitude of this shift is determined according to the process of step **1055** of FIG. **10**. In this case, the left and right half frames are treated as single tiles. The time for rendering the left half tile is the sum of the rendering times for the upper and lower left tiles. The time for rendering the right half frame is calculated similarly. The above equation for the pixelshift is then applied to determine the extent of the boundary shift.

In step **1535**, a decision is made as to whether an imbalance exists between the upper and lower tiles of the left half. If so, processing continues at step **1540**. Here, the upper left (UL) and lower left (LL) tiles are resized. In step **1545**, a determination is made as to whether an imbalance exists between rendering of the upper and lower right tiles UR and LR. If so, then resizing of the upper right and lower right tiles is performed in step **1550**. The resizing operations of steps **1540** and **1550** are performed according to the above equation for pixelshift, where the width of maxtile is replaced by the height of maxtile.

In step **1555**, the index value is incremented by one so that the next frame can be rendered. In step **1560**, graphics data for the next frame is sent to each rendering pipeline. The graphics data sent to a given rendering pipeline depends on its potentially re-sized tile. The processing then returns to step **1515**.

Step **1525** above, the step of determining whether an imbalance exists between the left and right half frames, is illustrated in greater detail in FIG. **16**. The process begins with step **1610**. In step **1620**, the total rendering time required for the left tiles is determined by summing the time required to render the upper left (UL) tile ($upper_{13}left_{13}time$) and the time required to render the lower left(LL) tile ($lower_{13}left_{13}time$). For convenience, the total is referred to as $left_{13}time$.

In step **1630**, the analogous operation is performed with respect to the right tiles, upper right UR and lower right LR. The total rendering time required for the right tiles is determined by summing the time required to render the upper right (UR) tile ($upper_{13}right_{13}time$) and the time required to render the lower right (LR) tile ($lower_{13}right_{13}time$). For convenience, the total is referred to as $right_{13}time$

In step **1640**, a determination is made as to whether the magnitude of the difference between $left_{13}time$ and $right_{13}time$ exceeds a threshold value. If so, an imbalance is detected between left and right half frames (state **1650**). If not, then no significant imbalance is detected (state **1660**). The process concludes at step **1670**.

The results of the processing of FIG. **15** are illustrated in FIGS. **17**A-**17**C. FIG. **17**A illustrates a frame composed of a 2×2 array of tiles. The width of the left half frame is shown as width **1705**. The width of the right half frame is shown as width **1710**. The height of the upper tiles is shown as height **1712**, and the height of the lower tiles is shown as height **1714**. In FIG. **17**B, a determination has been made that right time exceeds left time by a significant margin. As a result, the vertical boundary is shifted to the right by a distance of **1722**. As a result, the left half frame now has a width of **1715**, while the right half frame has width **1720**. At this point, the heights of the upper and lower tiles are unchanged. In FIG. **17**C, the left and right sides of the frame are considered independently. With respect to the left side, a determination is made that tile UL took significantly longer to render than tile LL in the current frame. Consequently, the horizontal boundary on the left side is raised, thereby increasing the height of the tile LL by a distance **1735**. As a result, tile UL now has a height **1730**. On the right side, a determination is made that tile UR took significantly longer to render than the tile LR in the current frame. As a result, the horizontal boundary on the right side is raised by a distance **1745**. The height of tile UR, for purposes of the next frame, is now **1740**.

In an alternative embodiment of the method of the invention, a 2×2 frame can first be processed as upper and lower half frames. In such an embodiment, a determination is made as to whether either upper or lower half frame takes significantly longer to render than the other. For either half frame, the time required to render the half frame is the sum of the rendering times for its left and right tiles. If either upper or lower half frame takes significantly longer to render than the other, the horizontal boundary is shifted by an amount determined by the above pixelshift equation for purposes of the next frame. The left and right tiles of each of these half frames can then be considered. For each half frame, a determination is made as to whether the left or right tile has taken significantly longer to render than the other. If so, the vertical boundary for that half frame is shifted according to the above pixelshift equation for purposes of the next frame.

The above methods can be applied to a tiling scheme other than the 2×2, 1×n, and n×1 cases described above, provided that the tiling scheme can be decomposed into such cases. For example, a tiling scheme having two rows of eight tiles can be decomposed into two half tiles, each 1×8. The horizontal boundary can first be shifted if the difference in rendering times between the two half tiles is significant, as described above with respect to FIGS. **10**, **11**, and **12**B. Within each half frame, re-sizing can be performed as described above with respect to FIGS. **10**, **11**, and **13**B or **14**B.

The processing of the invention from the perspective of the rendering pipelines is illustrated in FIG. **18**. The process begins at step **1810**. In step **1820**, an index value is initialized to zero. In step **1830**, frame i is rendered. In step **1840**, each rendering pipeline generates a performance report stating the length of time required to render its respective tile in the current frame. In step **1850**, the performance reports

are sent to the performance monitor. In step **1860**, the determination is made as to whether the application has completed running. If so, the process concludes at step **1890**. Otherwise, the process continues at step **1870** where the index value is incremented by 1. In step **1880**, after any imbalances have been identified and any tile resizing has been performed, the rendering pipelines receive graphics data for the next frame. As in previous frames, each rendering pipeline receives the graphics data associated with a particular tile. The process then continues at step **1830**, wherein the next frame is rendered.

What is claimed is:

1. A system for generating a sequence of computer graphics frames, the system comprising:

a plurality of rendering pipelines that each receive a distinct subset of graphics data for a respective current frame in the sequence of frames, render said distinct subset of graphics data, and produce a performance report regarding the workload incurred by each respective rendering pipeline during said rendering;

a performance monitor that receives said performance report from each rendering pipeline and determines whether a disparity in the workloads of the respective rendering pipelines exceeds a threshold to thereby identify a load imbalance; and

an allocation module that reallocates graphics data for a next frame to said rendering pipelines, wherein reallocation depends on said load imbalance and seeks to reduce any subsequent load imbalance associated with rendering said next frame.

2. The system of claim **1**, further comprising a graphics application, wherein said graphics application comprises said performance monitor.

3. The system of claim **2**, wherein said graphics application further comprises said allocation module.

4. The system of claim **1**, further comprising a compositor that receives rendered graphics data from each said rendering pipeline and composits said rendered graphics data to form each of said frames.

5. The system of claim **1**, wherein each said distinct subset of graphics data corresponds to one of a plurality of tiles of said current frame.

6. The system of claim **5**, wherein said allocation module reallocates graphics data to said rendering pipelines for said next frame by resizing tiles of said next frame relative to said tiles of said current framer.

7. A method of rendering successive frames using a plurality of rendering pipelines, the method comprising the steps of:

(a) rendering a current frame, wherein each rendering pipeline renders a tile of the current frame;

(b) generating a performance report for each rendering pipeline, each performance report indicating the workload incurred by the respective rendering pipeline during said rendering;

(c) sending the performance reports to a performance monitor; and

(d) at each rendering pipeline, receiving graphics data associated with a tile of a next frame, wherein a plurality of the tiles of the next frame have been resized relative to the corresponding tiles of the current frame if the difference between the performance reports are above a threshold.

8. A method of controlling the rendering of successive frames, wherein the rendering is performed using a plurality of rendering pipelines, the method comprising the steps of:

(a) receiving a performance report for each rendering pipeline, each performance report indicating the workload incurred by the respective rendering pipeline during rendering of a current frame;

(b) determining whether the performance reports indicate a significant load imbalance among the rendering pipelines, wherein said significant load imbalance indicates that the difference between the performance reports is above a threshold;

(c) if a significant load balance is indicated, resizing at least one tile of the next frame relative to a corresponding tile of the current frame; and

(d) sending graphics data associated with the next frame to the rendering pipelines, wherein the graphics data sent to a given rendering pipeline is associated with a tile of the next frame.

9. The method of claim **8**, wherein said step b) comprises the steps of:

(i) determining the rendering pipeline with the longest rendering time for its tile in the current frame;

(ii) determining the rendering pipeline with the shortest rendering time for its tile in the current frame; and

(iii) determining if the difference between the longest and shortest rendering times exceeds a threshold value, thereby indicating a significant load imbalance.

10. The method of claim **9**, wherein the threshold value is a percentage of the longest rendering time.

11. The method of claim **9**, wherein said step c) comprises the steps of:

(i) with respect to the next frame, increasing the size of the tile corresponding to the rendering pipeline with the shortest rendering time, by an amount proportional to the difference between the longest and shortest rendering times; and

(ii) with respect to the next frame, decreasing the size of the tile corresponding to the rendering pipeline with the longest rendering time by the same amount.

12. The method of claim **8**, wherein a subset of tiles in the current frame constitutes a first half frame, and the remaining tiles in the current frame constitute a second half frame wherein said step b) comprises:

(i) summing the rendering times for all tiles in the first half frame of the current frame;

(ii) summing the rendering times for all tiles in the second half frame of the current frame;

(iii) determining if the difference between the two sums exceeds a threshold value, thereby indicating a significant load imbalance.

13. The method of claim **12**, wherein said step c) comprises:

(i) with respect to the next frame, increasing the size of at least one tile of the half frame having the lesser sum; and

(ii) with respect to the next frame, decreasing the size of at least one tile of the half frame having the greater sum,

wherein the size of the half frame having the lesser sum is increased by an amount proportional to the difference between the two sums and the size of the frame half having the greater sum is decreased by the same amount.

14. The method of claim **13**, wherein said step c) further comprises the steps of:

(iii) determining the rendering pipeline with the longest rendering time for its tile in the first half frame of the current frame;

(iv) determining the rendering pipeline with the shortest rendering time for its tile in the first half frame of the current frame;

(v) determining the difference between the longest and shortest rendering times in the first half frame of the current frame;

(vi) determining if the difference between the longest and shortest rendering times in the first half frame of the current frame exceeds the threshold value; and

(vii) if the difference exceeds the threshold value,

with respect to the next frame, increasing the size of the tile corresponding to the rendering pipeline with the shortest rendering time in the first half frame of the current frame, by an amount proportional to the difference between the longest and shortest rendering times in the first half frame of the current frame and,

with respect to the next frame, decreasing the size of the tile corresponding to the rendering pipeline with the longest rendering time in the first half frame of the current frame, by the same amount.

15. The method of claim **13**, wherein said step c) further comprises the steps of:

(iii) determining the rendering pipeline with the longest rendering time for its tile in the second half frame of the current frame;

(iv) determining the rendering pipeline with the shortest rendering time for its tile in the second half frame of the current frame;

(v) determining the difference between the longest and shortest rendering times in the second half frame of the current frame;

(vi) determining if the difference between the longest and shortest rendering times in the second half frame of the current frame exceeds the threshold value; and

(vii) if the difference exceeds the threshold value,

with respect to the next frame, increasing the size of the tile corresponding to the rendering pipeline with the shortest rendering time in the second half frame of the current frame by an amount proportional to the difference between the longest and shortest rendering times in the second half frame of the current frame, and

with respect to the next frame, decreasing the size of the tile corresponding to the rendering pipeline with the longest rendering time in the second half frame of the current frame, by the same amount.

16. A computer program product comprising a computer useable medium having control logic stored therein for causing a computer to render successive frames using a plurality of rendering pipelines, the computer control logic comprising:

a first computer readable program code means for causing the computer to render a current frame, wherein each rendering pipeline renders a tile of the current frame;

a second computer readable program code means for causing the computer to generate a performance report for each rendering pipeline, each performance report indicating the workload incurred by the respective rendering pipeline during said rendering;

a third computer readable program code means for causing the computer to send the performance reports to a performance monitor; and

a fourth computer readable program code means for causing the computer to allow receipt of graphics data at each rendering pipeline, wherein graphics data received at each rendering pipeline is associated with a tile of a next frame, and wherein a plurality of the tiles of the next frame have been resized relative to the corresponding tiles of the current frame if the difference between the performance reports are above a threshold.

17. A computer program product comprising a computer useable medium having control logic stored therein for causing a computer to control the rendering of successive frames, wherein the rendering is performed using a plurality of rendering pipelines, the computer control logic comprising:

a first computer readable program code means for causing the computer to receive a performance report for each rendering pipeline, each performance report indicating the workload incurred by the respective rendering pipeline during rendering of a current frame;

a second computer readable program code means for causing the computer to determine whether the performance reports indicate a significant load imbalance among the rendering pipelines, wherein the significant load imbalance indicates that the difference between the performance reports is above a threshold;

a third computer readable program code means for causing the computer to rebalance the workload to be incurred by the rendering pipelines during rendering of a next frame, if a significant load balance is indicated, by resizing at least one tile of the next frame relative to a corresponding tile of the current frame; and

a fourth computer readable program code means for causing the computer to send graphics data associated with the next frame to the rendering pipelines, wherein the graphics data sent to a given rendering pipeline is associated with a tile of the next frame.

18. The computer program product of claim **17**, wherein said second computer readable program code means comprises:

(i) computer readable program code means for causing the computer to determine the rendering pipeline with the longest rendering time for its tile in the current frame;

(ii) computer readable program code means for causing the computer to determine the rendering pipeline with the shortest rendering time for its tile in the current frame; and

(iii) computer readable program code means for causing the computer to determine if the difference between the longest and shortest rendering times exceeds a threshold value, thereby indicating a significant load imbalance.

19. The computer program product of claim **17**, wherein the threshold value is a percentage of the longest rendering time.

20. The computer program product of claim **17**, wherein said third computer readable program code means comprises:

(i) computer readable program code means for causing the computer to increase the size of the tile corresponding to the rendering pipeline with the shortest rendering time, by an amount proportional to the difference between the longest and shortest rendering times, for purposes of the next frame;

(ii) computer readable program code means for causing the computer to decrease the size of the tile corresponding to the rendering pipeline with the longest rendering time by the same amount, for purposes of the next frame.

21. A method of controlling the rendering of successive frames, wherein the rendering is performed using a plurality of rendering pipelines, the method comprising the steps of:

(a) receiving a performance report for each rendering pipeline, each performance report indicating the workload incurred by the respective rendering pipeline during rendering of a current frame;

(b) determining whether the performance reports indicate a significant load imbalance among the rendering pipelines, wherein said step b) comprises:

(i) determining the rendering pipeline with the longest rendering time for its tile in the current frame;

(ii) determining the rendering pipeline with the shortest rendering time for its tile in the current frame; and

(iii) determining if the difference between the longest and shortest rendering times exceeds a threshold value, thereby indicating a significant load imbalance;

(c) if a significant load balance is indicated, resizing at least one tile of the next frame relative to a corresponding tile of the current frame; and

(d) sending graphics data associated with the next frame to the rendering pipelines, wherein the graphics data sent to a given rendering pipeline is associated with a tile of the next frame.

22. The method of claim 21, wherein the threshold value is a percentage of the longest rendering time.

23. The method of claim 21, wherein said step c) comprises the steps of:

(i) with respect to the next frame, increasing the size of the tile corresponding to the rendering pipeline with the shortest rendering time, by an amount proportional to the difference between the longest and shortest rendering times; and

(ii) with respect to the next frame, decreasing the size of the tile corresponding to the rendering pipeline with the longest rendering time by the same amount.

24. A method of controlling the rendering of successive frames, wherein the rendering is performed using a plurality of rendering pipelines, the method comprising the steps of:

(a) receiving a performance report for each rendering pipeline, each performance report indicating the workload incurred by the respective rendering pipeline during rendering of a current frame;

(b) determining whether the performance reports indicate a significant load imbalance among the rendering pipelines, wherein a subset of tiles in the current frame constitutes a first half frame, and the remaining tiles in the current frame constitute a second half frame wherein said step b) comprises:

(i) summing the rendering times for all tiles in the first half frame of the current frame;

(ii) summing the rendering times for all tiles in the second half frame of the current frame;

(iii) determining if the difference between the two sums exceeds a threshold value, thereby indicating a significant load imbalance;

(c) if a significant load balance is indicated, resizing at least one tile of the next frame relative to a corresponding tile of the current frame; and

(d) sending graphics data associated with the next frame to the rendering pipelines, wherein the graphics data sent to a given rendering pipeline is associated with a tile of the next frame.

25. The method of claim 24, wherein said step c) comprises:

(i) with respect to the next frame, increasing the size of at least one tile of the half frame having the lesser sum; and

(ii) with respect to the next frame, decreasing the size of at least one tile of the half frame having the greater sum,

wherein the size of the half frame having the lesser sum is increased by an amount proportional to the difference between the two sums and the size of the frame half having the greater sum is decreased by the same amount.

26. The method of claim 25, wherein said step c) further comprises the steps of:

(iii) determining the rendering pipeline with the longest rendering time for its tile in the first half frame of the current frame;

(iv) determining the rendering pipeline with the shortest rendering time for its tile in the first half frame of the current frame;

(v) determining the difference between the longest and shortest rendering times in the first half frame of the current frame;

(vi) determining if the difference between the longest and shortest rendering times in the first half frame of the current frame exceeds the threshold value; and

(vii) if the difference exceeds the threshold value,

with respect to the next frame, increasing the size of the tile corresponding to the rendering pipeline with the shortest rendering time in the first half frame of the current frame, by an amount proportional to the difference between the longest and shortest rendering times in the first half frame of the current frame and,

with respect to the next frame, decreasing the size of the tile corresponding to the rendering pipeline with the longest rendering time in the first half frame of the current frame, by the same amount.

27. The method of claim 25, wherein said step c) further comprises the steps of:

(iii) determining the rendering pipeline with the longest rendering time for its tile in the second half frame of the current frame;

(iv) determining the rendering pipeline with the shortest rendering time for its tile in the second half frame of the current frame;

(v) determining the difference between the longest and shortest rendering times in the second half frame of the current frame;

(vi) determining if the difference between the longest and shortest rendering times in the second half frame of the current frame exceeds the threshold value; and

(vii) if the difference exceeds the threshold value,

with respect to the next frame, increasing the size of the tile corresponding to the rendering pipeline with the shortest rendering time in the second half frame of the current frame by an amount proportional to the difference between the longest and shortest rendering times in the second half frame of the current frame, and

with respect to the next frame, decreasing the size of the tile corresponding to the rendering pipeline with the longest rendering time in the second half frame of the current frame, by the same amount.

28. A computer program product comprising a computer useable medium having control logic stored therein for causing a computer to control the rendering of successive

frames, wherein the rendering is performed using a plurality of rendering pipelines, the computer control logic comprising:

a first computer readable program code means for causing the computer to receive a performance report for each rendering pipeline, each performance report indicating the workload incurred by the respective rendering pipeline during rendering of a current frame;

a second computer readable program code means for causing the computer to determine whether the performance reports indicate a significant load imbalance among the rendering pipelines, wherein said second computer readable program code means comprises:

(i) computer readable program code means for causing the computer to determine the rendering pipeline with the longest rendering time for its tile in the current frame;

(ii) computer readable program code means for causing the computer to determine the rendering pipeline with the shortest rendering time for its tile in the current frame; and

(iii) computer readable program code means for causing the computer to determine if the difference between the longest and shortest rendering times exceeds a threshold value, thereby indicating a significant load imbalance;

a third computer readable program code means for causing the computer to rebalance the workload to be incurred by the rendering pipelines during rendering of a next frame, if a significant load balance is indicated, by resizing at least one tile of the next frame relative to a corresponding tile of the current frame; and

a fourth computer readable program code means for causing the computer to send graphics data associated with the next frame to the rendering pipelines, wherein the graphics data sent to a given rendering pipeline is associated with a tile of the next frame.

29. A computer program product comprising a computer useable medium having control logic stored therein for causing a computer to control the rendering of successive frames, wherein the rendering is performed using a plurality of rendering pipelines, the computer control logic comprising:

a first computer readable program code means for causing the computer to receive a performance report for each rendering pipeline, each performance report indicating the workload incurred by the respective rendering pipeline during rendering of a current frame;

a second computer readable program code means for causing the computer to determine whether the performance reports indicate a significant load imbalance among the rendering pipelines;

a third computer readable program code means for causing the computer to rebalance the workload to be incurred by the rendering pipelines during rendering of a next frame, if a significant load balance is indicated, by resizing at least one tile of the next frame relative to a corresponding tile of the current frame, wherein said third computer readable program code means comprises:

(i) computer readable program code means for causing the computer to increase the size of the tile corresponding to the rendering pipeline with the shortest rendering time, by an amount proportional to the difference between the longest arid shortest rendering times, for purposes of the next frame;

(ii) computer readable program code means for causing the computer to decrease the size of the tile corresponding to the rendering pipeline with the longest rendering time by the same amount, for purposes of the next frame; and

a fourth computer readable program code means for causing the computer to send graphics data associated with the next frame to the rendering pipelines, wherein the graphics data sent to a given rendering pipeline is associated with a tile of the next frame.

* * * * *