US 20080284786A1

(19) **United States**

(12) **Patent Application Publication**

Airey et al.

(10) **Pub. No.: US 2008/0284786 A1**

(43) **Pub. Date: Nov. 20, 2008**

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

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

Correspondence Address:
Bromberg & Sunstein LLP
125 Summer Street
Boston, MA 02110-1618 (US)

(73) Assignee: **SILICON GRAPHICS, INC.**, Sunnyvale, CA (US)

(21) Appl. No.: **12/168,578**

(22) Filed: **Jul. 7, 2008**

Related U.S. Application Data

(63) Continuation of application No. 09/614,363, filed on Jul. 12, 2000, which is a continuation of application No. 09/098,041, filed on Jun. 16, 1998, now Pat. No. 6,650,327.

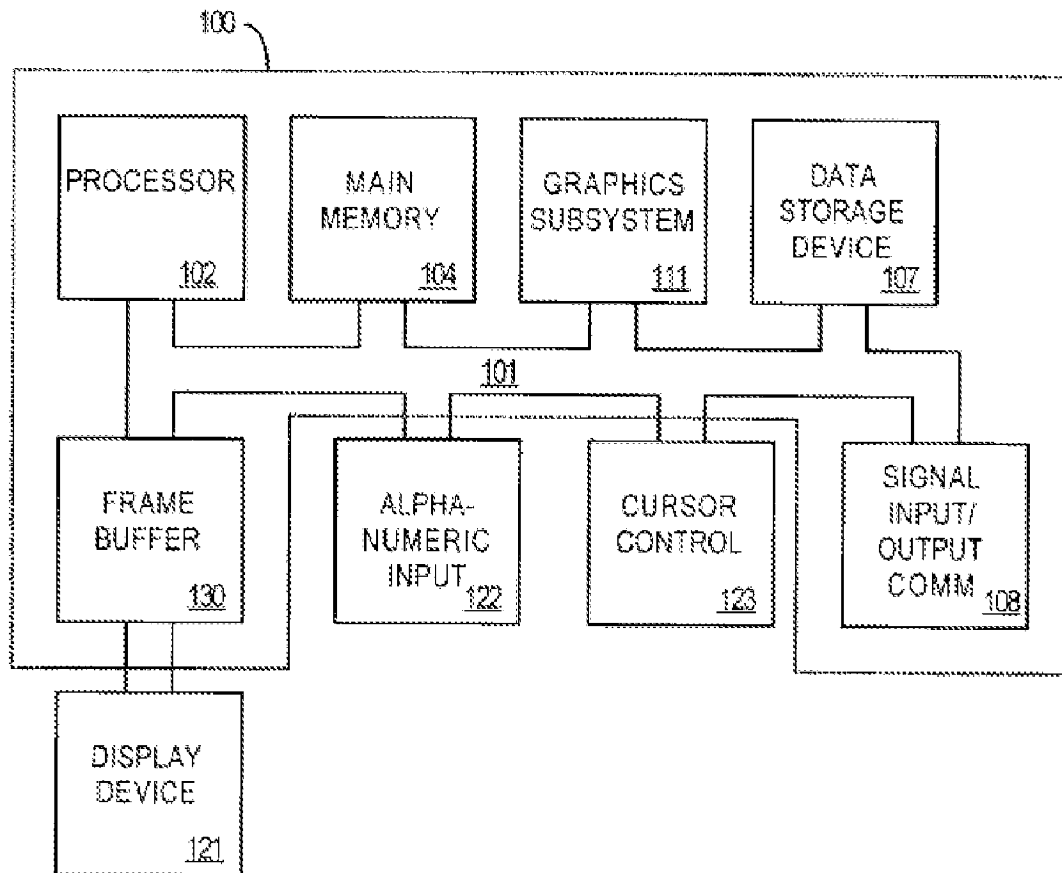
Publication Classification

(51) **Int. Cl.**
G09G 5/36 (2006.01)

(52) **U.S. Cl.** **345/545**

(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.



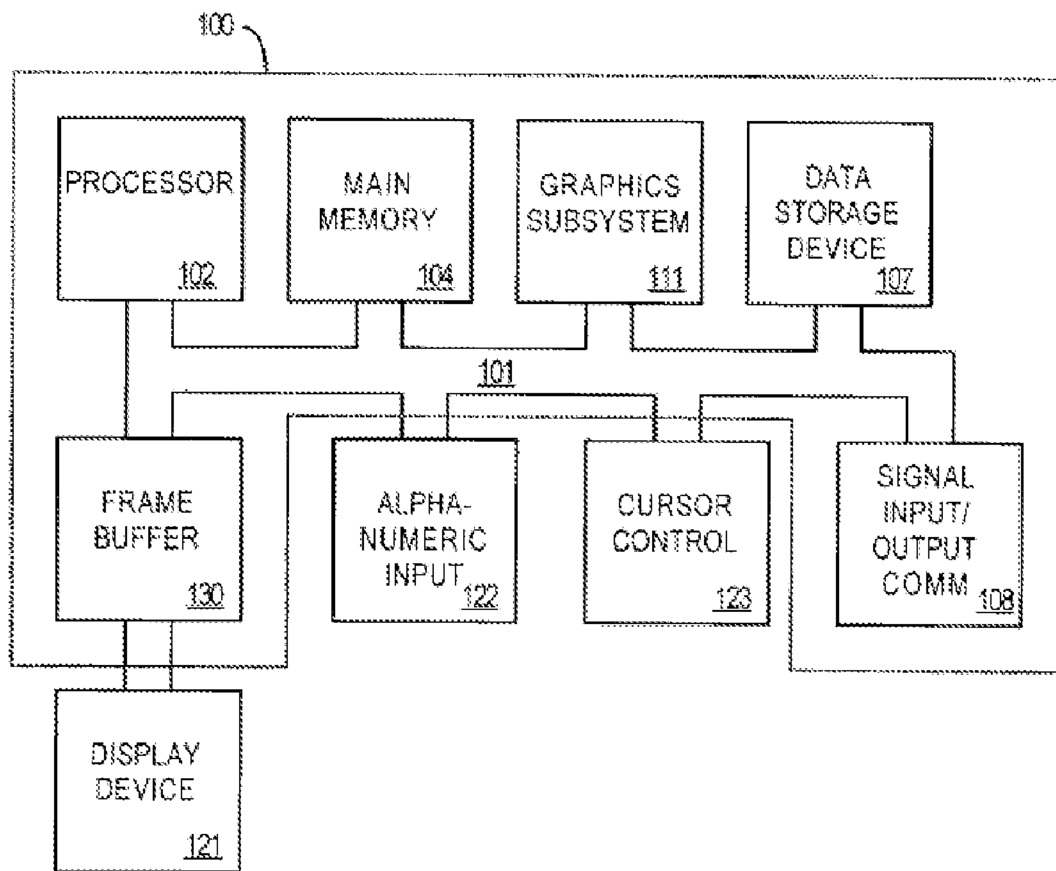


FIG. 1

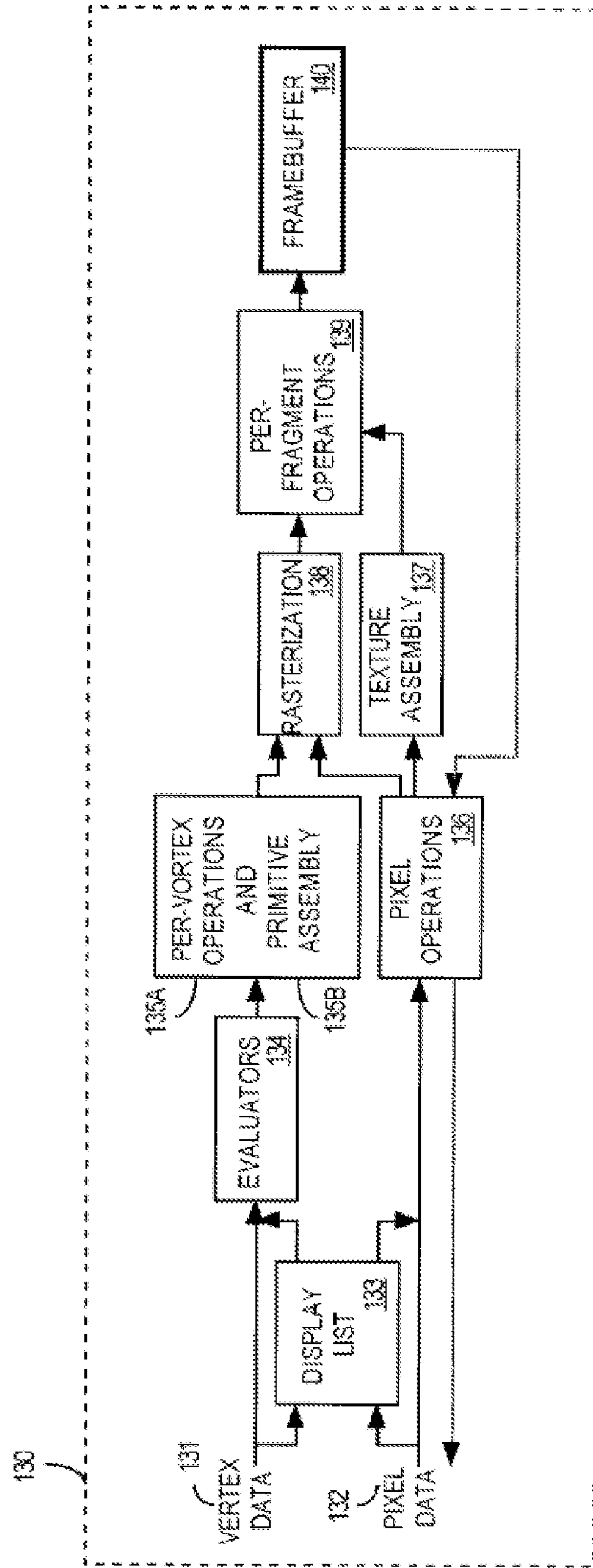


FIG. 2

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 s1e5 is readily achievable

FIG. 3

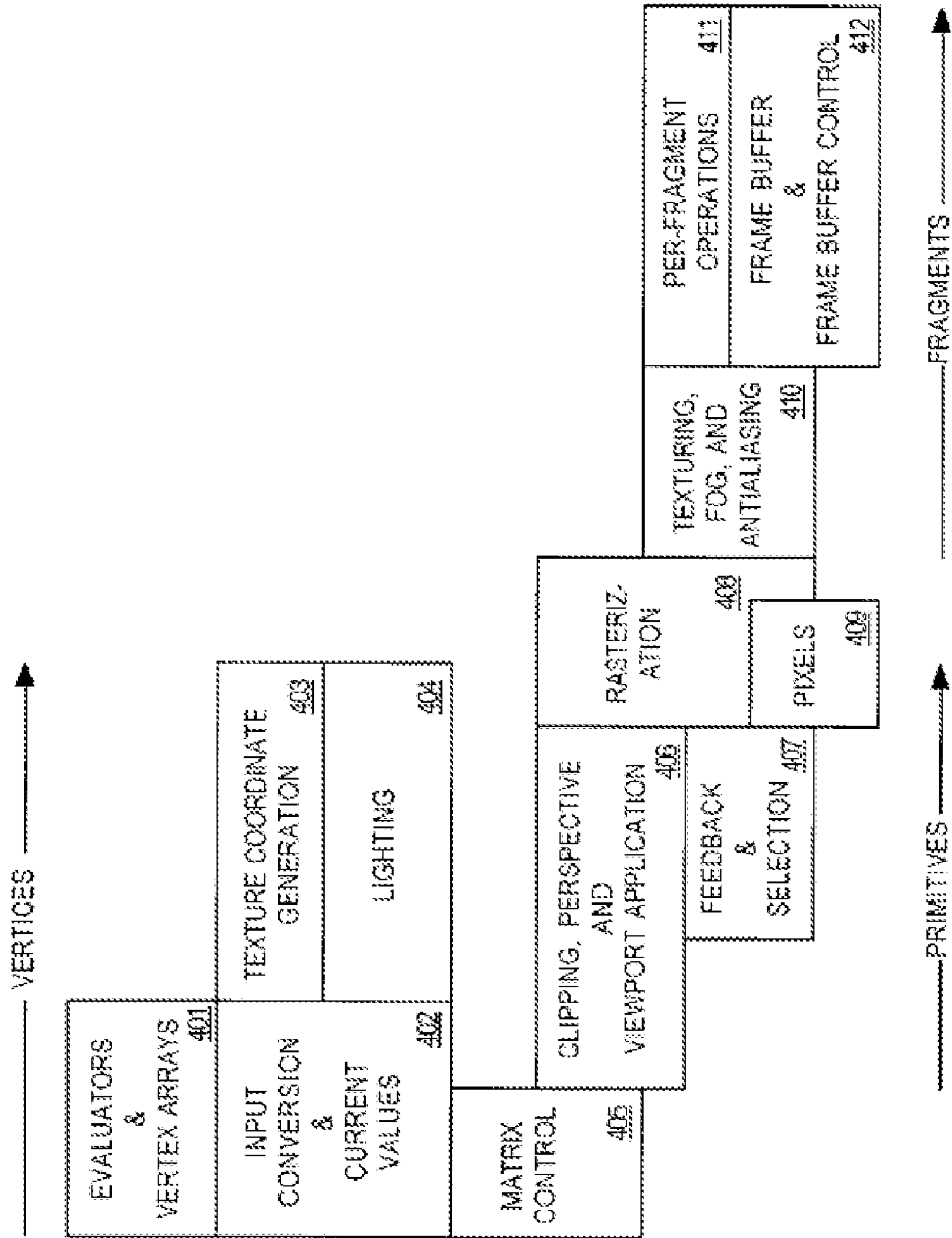


FIG. 4

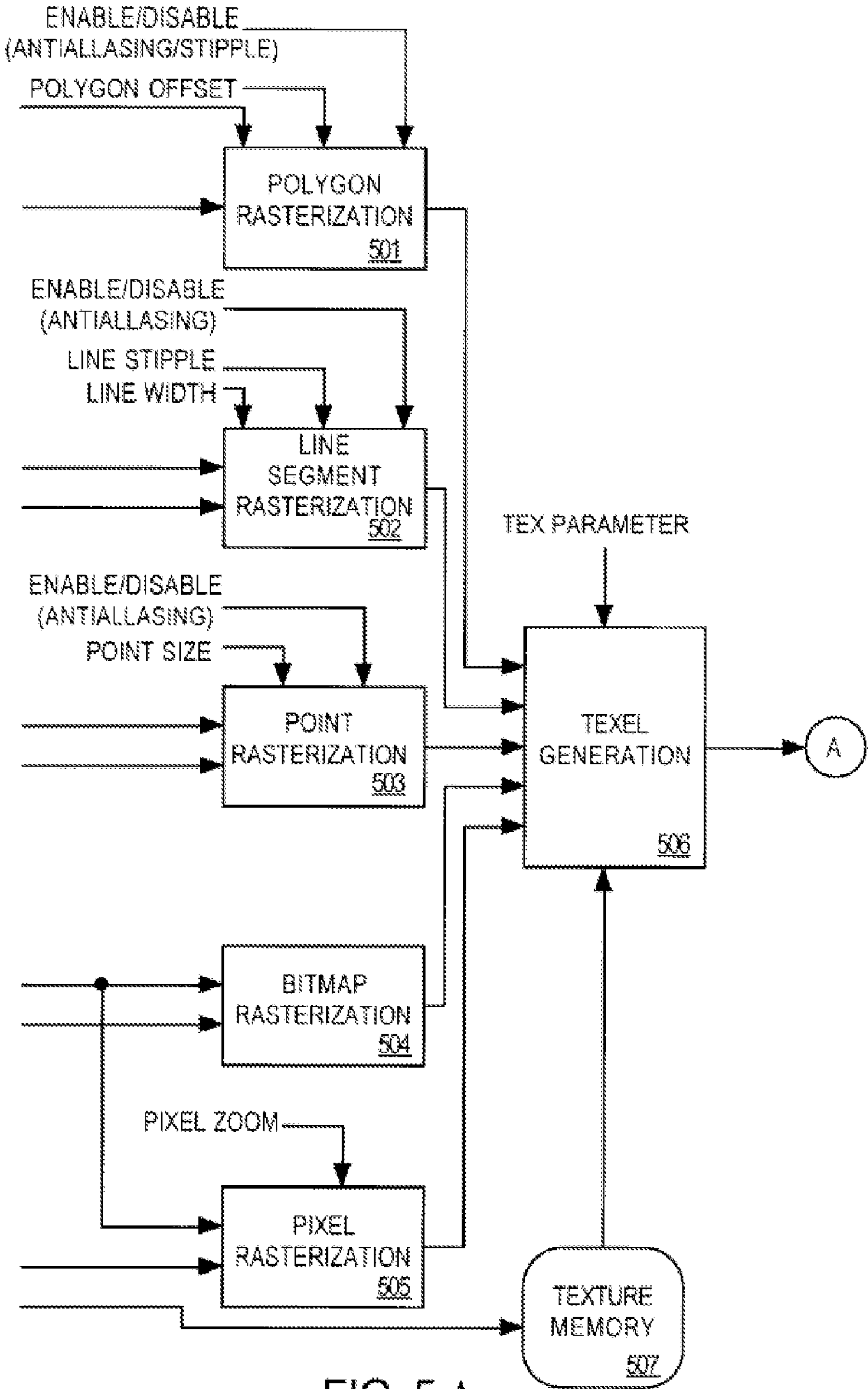


FIG. 5A

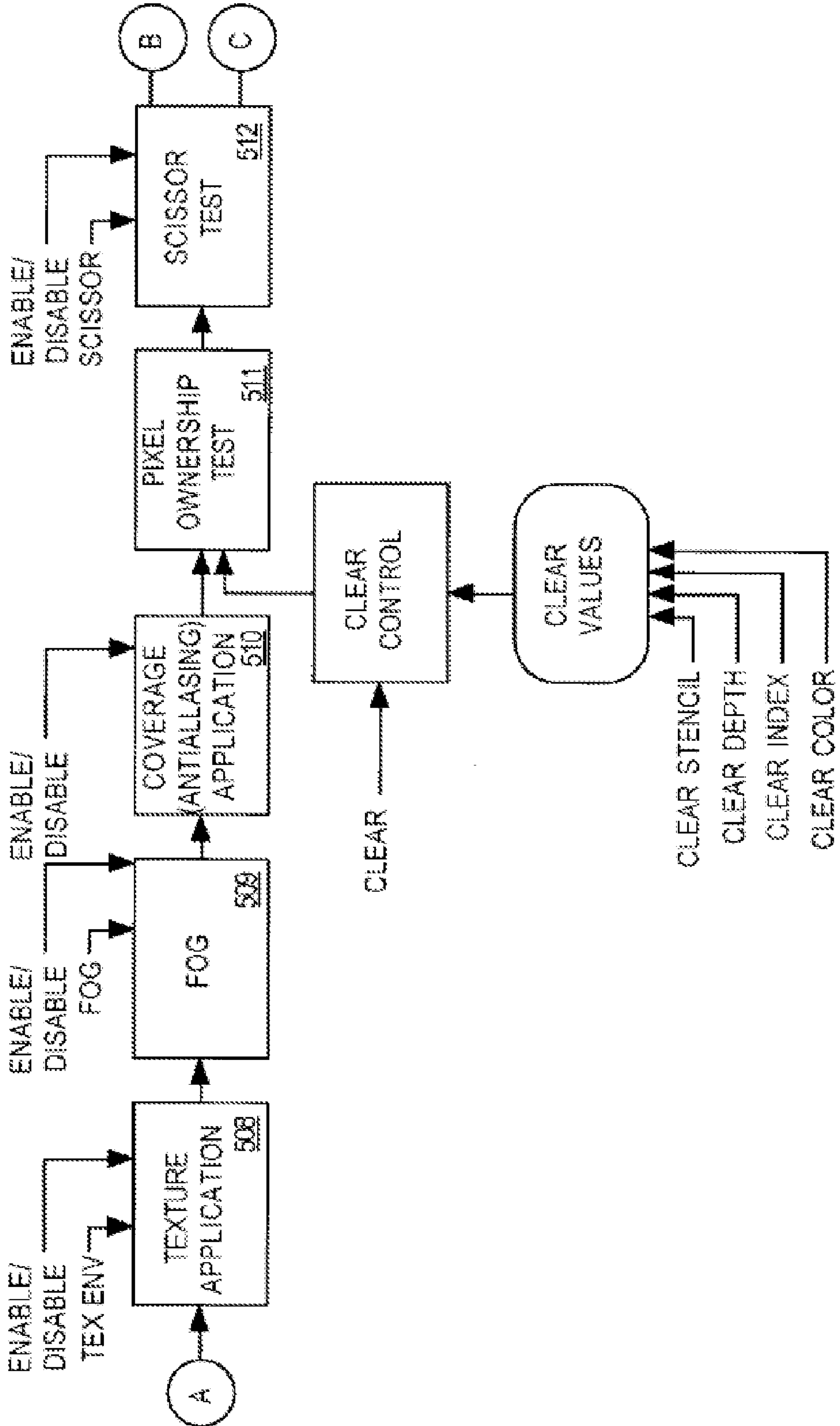


FIG. 5 B

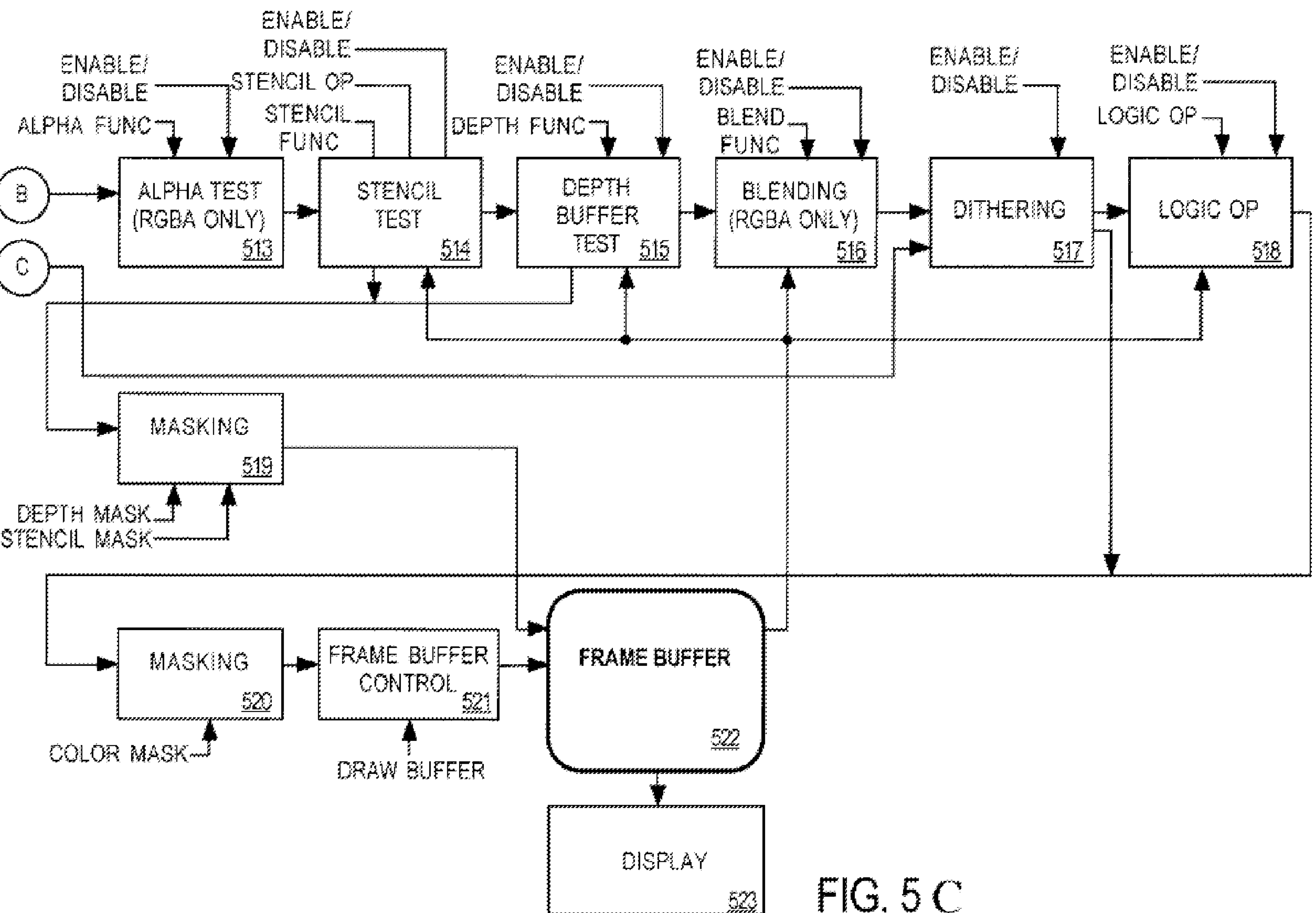


FIG. 5 C

DISPLAY SYSTEM HAVING FLOATING POINT RASTERIZATION AND FLOATING POINT FRAMEBUFFERING

PRIORITY

[0001] This patent application is a continuation of co-pending U.S. patent application Ser. No. 09/614,363, filed Jul. 12, 2000, and entitled, "DISPLAY SYSTEM HAVING FLOATING POINT RASTERIZATION AND FLOATING POINT FRAMEBUFFERING," which in turn is a continuation of U.S. patent application Ser. No. 09/098,041, now U.S. Pat. No. 6,650,327, filed Jun. 16, 1998, and entitled, "DISPLAY SYSTEM HAVING FLOATING POINT RASTERIZATION AND FLOATING POINT FRAMEBUFFERING," the disclosures of which are incorporated herein, in their entireties, by reference.

TECHNICAL FIELD

[0002] 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

[0003] 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.

[0004] 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.

[0005] 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.

[0006] 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.times.10.sup.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.

[0007] 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.

[0008] 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.

[0009] 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.

[0010] 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.

[0011] 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.

[0012] 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.

[0013] 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.

[0014] 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.

[0015] 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

[0016] 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.

[0017] 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

[0018] FIG. 1 shows a computer graphics system upon which the present invention may be practiced.

[0019] FIG. **2** is a flow chart illustrating the stages for processing data in a graphics program in accordance with the present invention.

[0020] FIG. **3** is a tabulation of the representative values for all possible bit combinations used in the preferred embodiment of the present invention.

[0021] FIG. **4** shows a block diagram of the currently preferred embodiment of the display system.

[0022] 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

[0023] 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.

[0024] 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.

[0025] 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.

[0026] 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.

[0027] 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.

[0028] 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**.

[0029] 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.

[0030] 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**.

[0031] 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**.

[0032] 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.

[0033] 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.

[0034] 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.

[0035] 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.

[0036] 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.

[0037] 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.

[0038] 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 $n=m\times2^e$. 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.

[0039] 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 value 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**.

[0040] 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 pre-

serves the precision of the data that was determined in steps **131** through **139** of the graphics program illustrated in FIG. **2**.

[0041] 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 $1.0000\text{-}0000_00*2^{-16}$ 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.

[0042] 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.

[0043] 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.

[0044] 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**.

[0045] 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.

[0046] 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.

[0047] 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.

[0048] 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**.

[0049] 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.

[0050] 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.

[0051] 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.

[0052] 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 raster-

ization **501**, texel generation **506**, texture memory **507**, fog **509**, blending **516**, and frame buffer **522**.

[0053] 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.

[0054] 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 rendering circuit comprising:

a geometry processor;

a rasterizer coupled to the geometry processor, the rasterizer comprising a scan converter having an input and an output, the scan converter being configured to scan convert data received at the input, at least a portion of the data received at the input being in floating point format, the scan converter being configured to output data from the output, at least a portion of the data from the output being floating point data; and

a frame buffer coupled to the rasterizer for storing a plurality of color values in floating point format.

2. The rendering circuit as defined by claim **1** wherein the scan converter is configured to scan convert on an entirely floating point basis.

3. The rendering circuit as defined by claim **1** wherein the data received at the input comprises color data.

4. The rendering circuit as defined by claim **1** wherein the rasterizer further includes a floating point texture circuit.

5. The rendering circuit as defined by claim **1** wherein the rasterizer further includes a floating point texture memory.

6. The rendering circuit as defined by claim **1** wherein the rasterizer further includes a floating point fog circuit.

7. The rendering circuit as defined by claim **1** wherein the rasterizer further includes a floating point blender.

8. The rendering circuit as defined by claim **1** wherein the rasterizer further includes a floating point lighting circuit.

9. The rendering circuit as defined by claim **1** wherein the rasterizer operates entirely on a floating point basis.

10. The rendering circuit as defined by claim **1** further comprising a circuit board coupled with the geometry processor, rasterizer, and frame buffer.

11. A rendering circuit comprising:

a rasterizer for performing a rasterization process, at least a portion of the rasterization process performed in a floating point format; and

a floating point frame buffer coupled to the rasterizer for storing a plurality of floating point color values.

12. The rendering circuit as defined by claim **11** wherein the floating point color values are read out from the frame buffer in the floating point format for display.

13. The rendering circuit as defined by claim **11** wherein the rasterization process is performed on an entirely floating point basis.

14. The rendering circuit as defined by claim **11** wherein the rasterizer comprises an input and an output, the rasterizer configured to process floating point data received at the input, the rasterizer configured to output floating point data at the output.

15. The rendering circuit as defined by claim **11** wherein the rasterizer includes a floating point texture circuit.

16. The rendering circuit as defined by claim **11** wherein the rasterizer includes a floating point texture memory.

17. The rendering circuit as defined by claim **11** wherein the rasterizer includes a floating point fog circuit.

18. The rendering circuit as defined by claim **11** wherein the rasterizer includes a floating point blender.

19. The rendering circuit as defined by claim **11** wherein the rasterizer includes a floating point lighting circuit.

20. The rendering circuit as defined by claim **11** wherein the rasterizer includes a floating point scan converter.

21. The rendering circuit as defined by claim **11** further comprising a circuit board coupled with the rasterizer and frame buffer.

22. A rendering circuit comprising:

a rasterizer including at least one of a floating point fog subsystem and a floating point texture subsystem; and

a frame buffer coupled to the rasterizer for storing a plurality of color values in the floating point format.

23. The rendering circuit as defined by claim **22** wherein the rasterizer comprises both the floating point fog subsystem and the floating point texture subsystem.

24. The rendering circuit as defined by claim **22** wherein the at least one floating point fog subsystem is configured to perform fog operations on an entirely floating point basis.

25. The rendering circuit as defined by claim **22** wherein the at least one floating point texture subsystem is configured to perform texture operations on an entirely floating point basis.

26. The rendering circuit as defined by claim **22** wherein the rasterizer comprises an input and an output, the rasterizer configured to process floating point data received at the input, the rasterizer configured to output floating point data at the output.

27. The rendering circuit as defined by claim **22** wherein the rasterizer further comprises at least one of a floating point blending subsystem and a floating point antialiasing subsystem.

28. The rendering circuit as defined by claim **22** further comprising texture memory configured to store texture data on a floating point basis.

29. A rendering circuit comprising:

a geometry processor configured to perform geometric calculations on a plurality of vertices of a three-dimensional primitive;

a rasterizer coupled with the geometry processor, the rasterizer comprising a first raster portion and a second raster portion, the first raster portion configured to translate three-dimensional primitives into a set of corresponding fragments or pixels, the second raster portion configured to fill-in the set of fragments or pixels, at least a portion of the second raster portion operating on a floating point basis; and

a floating point frame buffer coupled to the rasterizer for storing a plurality of floating point color values.

30. The rendering circuit as defined by claim **29** wherein the first raster portion operates on an entirely floating point basis.

31. The rendering circuit as defined by claim **29** wherein the second raster portion includes a floating point texture circuit.

32. The rendering circuit as defined by claim **29** wherein the second raster portion includes a floating point texture memory.

33. The rendering circuit as defined by claim **29** wherein the second raster portion includes a floating point fog circuit.

34. The rendering circuit as defined by claim **29** wherein the second raster portion includes a floating point blender.

35. The rendering circuit as defined by claim **29** wherein the second raster portion includes a floating point lighting circuit.

36. The rendering circuit as defined by claim **29** where the second raster portion operates entirely on a floating point basis.

37. The rendering circuit as defined by claim **29** where the second raster portion operates at least in part on a fixed point basis

38. A rendering circuit comprising:

means for performing geometric calculations on a plurality of vertices of a primitive; and

buffer means for storing a plurality of color values in the floating point format.

39. The rendering circuit as defined by claim **38** further comprising means for rasterizing the primitive at least partially on a floating point basis.

40. The rendering circuit as defined by claim **39** wherein the rasterizing means comprises means for filling fragments and pixels on an entirely floating point basis.

41. The rendering circuit as defined by claim **39** wherein the rasterizing means comprises means for scan converting on an entirely floating point basis.

42. A rendering circuit comprising:

a floating point frame buffer for storing a plurality of floating point color values, the floating point color values being 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.

43. A rendering circuit comprising:

a floating point frame buffer for storing a plurality of floating point color values, the frame buffer being configured so that floating point color values can be 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.

44. The rendering circuit as defined by claim **43** further comprising a read block configured to read floating point color values from the frame buffer.

45. The rendering circuit as defined by claim **43** further comprising a write block configured to write floating point color values to the frame buffer.

46. The rendering circuit as defined by claim **45** further comprising:

a read block configured to read floating point color values from the frame buffer; and

a operation block configured to operate directly on floating point color values read from the frame buffer.

47. The rendering circuit as defined by claim **43** further comprising a rasterizer that at least partially operates on a floating point basis.

48. In a rendering circuit, a method for operating on data stored in a frame buffer, the method comprising:

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

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

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

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

49. The method as defined by claim **48** further comprising operating directly on the data in the floating point format.

50. The method as defined by claim **49** wherein storing is performed before reading, reading is performed before operating, and operating is performed before writing.

51. The method as defined by claim **48** wherein storing is performed before reading, reading being performed before writing.

52. A rendering circuit comprising:

a rasterizer for performing a rasterization process; and

a hardware floating point frame buffer coupled to the rasterizer for storing a plurality of floating point color values.

53. The rendering circuit as defined by claim **52** wherein the frame buffer and rasterizer are implemented on a single chip.

54. The rendering circuit as defined by claim **53** wherein at least a portion of the rasterization process is performed in a floating point format.

55. The rendering circuit as defined by claim **52** wherein the rasterizer and frame buffer form a dedicated rendering pipeline.

56. The rendering circuit as defined by claim **52** further comprising a block configured to read floating point color values from the frame buffer.

57. The rendering circuit as defined by claim **56** wherein the floating point color values are processed to produce processed floating point color values, the circuit further comprising a store block configured to store processed floating point color values in the frame buffer.

* * * * *