A Distributed Frame Buffer within a Window-Oriented High Performance Graphics System

Thomas Haaker and Harald Selzer
Technische Hochschule Darmstadt
FG Graphisch-Interaktive Systeme

Hans Joseph
Fraunhofer-Arbeitsgruppe für Graphische Datenverarbeitung

Wilhelminenstr. 7, D-6100 Darmstadt, FRG

Today's workstation users demand high computational performance combined with powerful graphics and a comfortable window system. Existing and forthcoming standards like GKS-3D, PHIGS/PHIGS+, X Window System, and PEX have to be supported optimally.

This paper presents the architecture of a graphics engine designed to meet the above requirements. Utilizing a distributed frame buffer pixel access with a high bandwidth is achieved. Several functions of a window management system like clipping at arbitrarily shaped window boundaries, fast copying of windows and performing Bit-Block-Transfer operations (BitBlT) are performed by hardware. Finally, a homogeneous and load-adaptive multiprocessor configuration for geometry and rendering calculation is described.

1. Introduction and Background

Computer Graphics is an area of research and industrial exploitation with a large and still increasing number of applications. The human visual perception is the most effective method to perceive a large amount of information. Therefore graphics methods for displaying data including efficient man-machine interfaces enhance the user's productivity. The photorealistic rendering of complex scenes and fast computations for the visualization of technical, physical, chemical, or medical data requires innovative workstation architectures. Speed improvements will be achieved by the design of specialized hardware to meet the requirements for the display of large amounts of scientific data within time constraints. Moreover the user asks for an environment that allows the simultaneous display of computational results of different applications. This has to be performed in parallel to the access and manipulation of data by several user processes. This task is best performed by the use of workstations. They provide a window environment as an interface to the user. To meet the requirements for concurrency and image quality with high interactivity, a substantial hardware support for window-oriented workstations is mandatory.

The low price of raster scan monitors and the rapid decrease in cost of semiconductor memories introduced the raster displays as the only important technique in the mass market of workstations. In the workstation area the raster scan displays are assisted by a frame buffer - often associated with the term 'image memory'. The frame buffer contains the image to be displayed on a pixel-by-pixel basis and refreshes the raster scan display continuously. This implies a memory architecture with two functional ports. One port is used by the rasterization process which computes all the pixels from a geometric description. The other port is controlled by a circuit which routes the color values of the stored pixels to the CRT display [14,17]. The advantage of a frame buffer is based on the fact that the screen refresh process is independent of the image generation and rasterization process, i.e. the refresh frequency for the monitor can be chosen independently of the frame buffer update rate achieved by the rasterization process. Therefore the frame buffer manages to display an arbitrary image with no limits to the complexity and the number of graphics objects.

On the other hand, a lot of pixels have to be updated in order to obtain major image modifications. As a consequence of today's very high resolution monitors (up to 2k x 2k pixel), frame buffers are usually implemented using slow memory chips with a very high density. Before Video RAM (VRAM) chips were introduced, only single ported dynamic RAM chips could be used to realize a frame buffer. Therefore the screen refresh consumed a significant fraction of the frame buffer access bandwidth, it became a bottleneck for the pixel update.

In general, modern applications are constructed of VRAM chips that provide an additional serial port [14,17]. Using this port for image refresh to the monitor, the random access port is nearly all of the time available for the pixel update (only a small amount of time is required for RAM refresh). So the rasterization process can be accelerated by simply using VRAM chips instead of DRAM chips.

Nevertheless, Computer Graphics users have an ever-increasing desire for higher performance, more functionality and better picture quality. This is constantly driven by rapid improvements in semiconductor technology and more sophisticated rasterization techniques [7]. Therefore the principal frame buffer functionality of storing pixels and refreshing the screen must be extended. Additional issues and their impacts on the overall performance have to be considered.

- Bringing parallelism to the frame buffer is an effective method to speed up image generation. On the pixel level the components of the pixel representation (e. g. color-, Z-value, window identifier) and on the primitive level several picture elements referring to the same graphics primitive (e. g. several spans of a triangle) can be computed and accessed in parallel. Additionally, the manipulation of geometric data may be performed by a parallel multiprocessor approach.

  In order to achieve this parallelism, various methods and techniques have been investigated [7]. As an example, the image memory can be organized into subsequent scan lines or square areas, and the memory chips can be arranged in an interleaved mode to reduce the influence of the memory cycle time. Furthermore, some frame buffer designs are based on custom designed VLSI chips which combine memory and processor capability on a single chip. The most prominent representatives are the Scan Line Access Memory (SLAM) [5], the Super Buffer [8] and Pixel-Planes [6]. They all are capable to cause a high speed-up for special rasterization algorithms. But problems of data distribution, flexibility, programmability, and mass production arise when systems with a very high pixel resolution have to be implemented, as it is required for modern workstations. Moreover, such graphics systems are mostly restricted to a fixed rasterization algorithm with the consequence that they cannot be adjusted to new and possibly more sophisticated techniques. As a result, the Distributed Frame Buffer is targeted as a state-of-the-art architecture [1,2,9,10]. It provides a high bandwidth between rendering processors and the frame buffer. This is achieved by enabling the pixel accesses of multiple rasterization units in parallel. Even in applications with a very high pixel resolution, an efficient design can be done by using off-the-shelf and hence cheap VRAM chips. A major part of this paper will focus on the distributed frame buffer concept as a basis for a high performance graphics system.

- In currently available workstations, a window management system is essential. Therefore, the appropriate frame buffer architecture should not only be optimized for fast primitive rendering within the whole screen area, but also within a screen portion - the window. This means that the throughput of processed graphics primitives should not decrease if the image generation is limited to the boundaries of a window. It should be kept in mind, that modern workstations even have to provide arbitrarily shaped windows [1,2].

  In summary the frame buffer architecture has to support efficiently fast window updates and movements, creating and deleting pop-up menus and handling arbitrarily shaped windows. In a multi-tasking environment these requirements have to be managed in conjunction with a fast context switch, because different window contents are usually processed by different tasks [11].

- The integration of video input sources into a graphics system by the use of a frame grabber hardware extension is a very reasonable feature for animation and image synthesis applications. The video input can then be manipulated by image processing algorithms and displayed like computer generated images. Nowadays, designers of electronic publishing systems investigate on the integration of moving images into electronic documents. Therefore a frame buffer has to provide a frame grabber mechanism. The windows containing video sequences should be controlled and manipulated in the same manner as a text or graphics window.

Keeping in mind, that the frame buffer must be tightly coupled to the rasterization processors, the strong influence of the frame buffer architecture on the hardware and software design of the overall system is becoming apparent. On the other hand, image generation, manipulation, and rasterization comprises many different tasks and complex computations. The stages of the image generation output pipeline can be qualified by a simple scheme as shown in figure 1. It is the basis for most of the graphics system architectures [2,3,9,16].

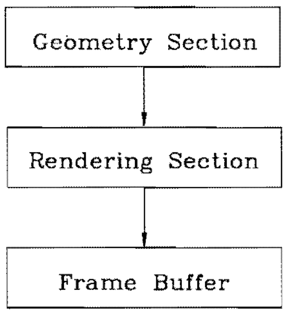

Figure 1: Image generation scheme

The geometry section comprises modelling and viewing transformations, the perspective projection and the clipping of the graphics primitives. The rendering section consists of the evaluation of an illumination model, of the rasterization process, shading and hidden surface removal. At the end of the rendering process the pixels are stored into the frame buffer and are displayed on the monitor.

A system architecture is needed where many processors perform all these tasks in order to achieve fast image generation and manipulation. With a parallel architecture the designer splits up the computational burden. Partitioning of the complete graphics process is state-of-the-art, and as a consequence there are a lot of proposals to design a graphics system with several processors working in parallel and executing different tasks.

One characteristic implementation is shown in [4] by developing hardware to speed up graphical algorithms of the geometry computation. It is the realization of a pipeline structure with all inherent disadvantages. It is almost impossible to balance the workload of the pipeline stages. Additionally the hardware pipeline is fixed for a given algorithm even if the requirements are changing.

Avoiding these problems Torborg, [15] presented a parallel architecture for the geometric section. The computational task is distributed to a configurable number of MIMD geometry processors. But even this solution presents a two stage pipeline consisting of the geometry and rendering section with possible difficulties of work-load balancing.

Therefore it is reasonable to apply Torborg's approach to the geometry and the rendering section. This leads to a well balanced system of several processors working in parallel, which takes advantage of the distributed frame buffer concept. It is realized in the 3DGRP - the 3D Geometry and Rendering Pipeline [13] which is also presented in this paper.

2. The 3DGRP Frame Buffer Architecture

The basic idea of the 3DGRP frame buffer architecture is to support a multiprocessor approach to speed up rendering algorithms in general. Although several rendering processors have to access the frame buffer in parallel to achieve the desired speed up a bottleneck must be avoided at the interface between the rendering processors and the memory chips of the frame buffer. This leads to the concept of the distributed frame buffer which is state of the art in modern workstations. For instance the raster subsystem of the Silicon Graphics' Superworkstation [1,2] is organized into five memory banks that are attached to one span processor respectively. The frame buffer in the AT&T Pixelmachine [9,10] is divided into an array of pixel nodes. Each pixel node contains a fraction of the whole memory tightly coupled to a MIMD processor.

For the 3DGRP frame buffer design three conditions for a resonable partition of the entire storage have to be taken into account:

- the required display resolution of 1280 x 1024 pixel
- the organization of modern VRAM-chips (256k x 4 bit)
- the number of necessary VRAM-chips, which should be minimal.

This leads to a distributed frame buffer consisting of five equally sized memory banks. Every bank contains 1280 : 5 = 256 columns of the whole screen. The arrangement of these columns is shown in figure 2.

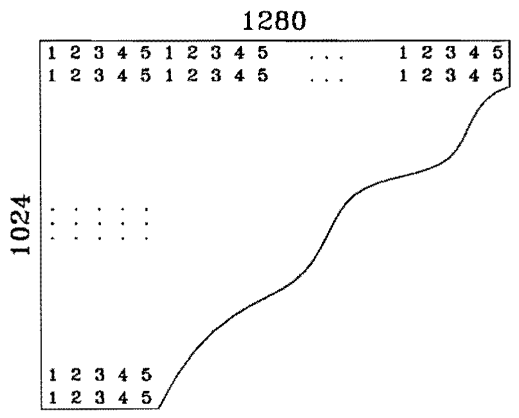

Figure 2: Frame buffer interleaving

Every memory bank can be accessed exclusively by a tightly coupled rendering processor, i.e. one processor need not to compete with others for memory access.

The global frame buffer architecture is represented by figure 3. The on-screen memory contains all the pixels visible on the screen. The resolution is 1280x1024 pixels. Each pixel consists of 24 bit color data twice (double buffer), 24 bit depth value (Z-buffer), 8 bit transparency, 8 bit window identifier, and up to 15 bit cursor overlay planes. The color representation can individually be chosen between true color and indexed color for every pixel. In case of true color the 24 bit color value will be splitted into the R-, G- and B-channel with 8 bit each. In the indexed color mode only 8 bits of the 24 bit color value are used as the input of the color look up table.

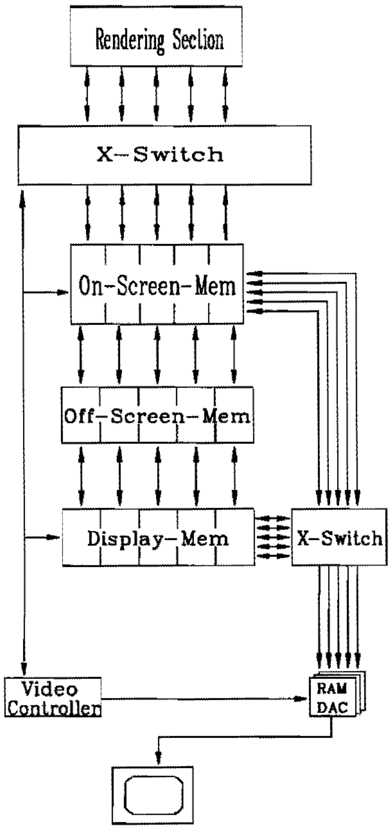

Figure 3: Frame buffer architecture

The off-screen-memory provides additional memory space for pixel data. It can be used by a window manager for the implementation of a backing store or as a font cache. The off-screen memory contains nearly two million pixels. The use of the display memory is described later on. All of these memories can be accessed by the rendering processors.

The display memory is used as temporary storage for all pixels within one or more window boundaries. It is needed for the hardware supported window copying mechanism described later on. All memory sections can be read and written by the rendering processors using the random-access ports of the memory chips, but the display memory cannot be seen by the user of the 3DGRP.

In order to realize BitBlT operations within the distributed frame buffer every processor needs to access all the frame buffer banks. Therefore a crossbar switch (X-switch) connects all memory banks with the rendering processors.

Every pixel has a 8 bit window identifier (window-ID) which is used to index a window look up table (WLUT). The window-ID controls write accesses of the rendering processors to the frame buffer as well as it presents basic information for the video controller. Figure 4 illustrates the video controller's evaluation mechanism of the window-ID.

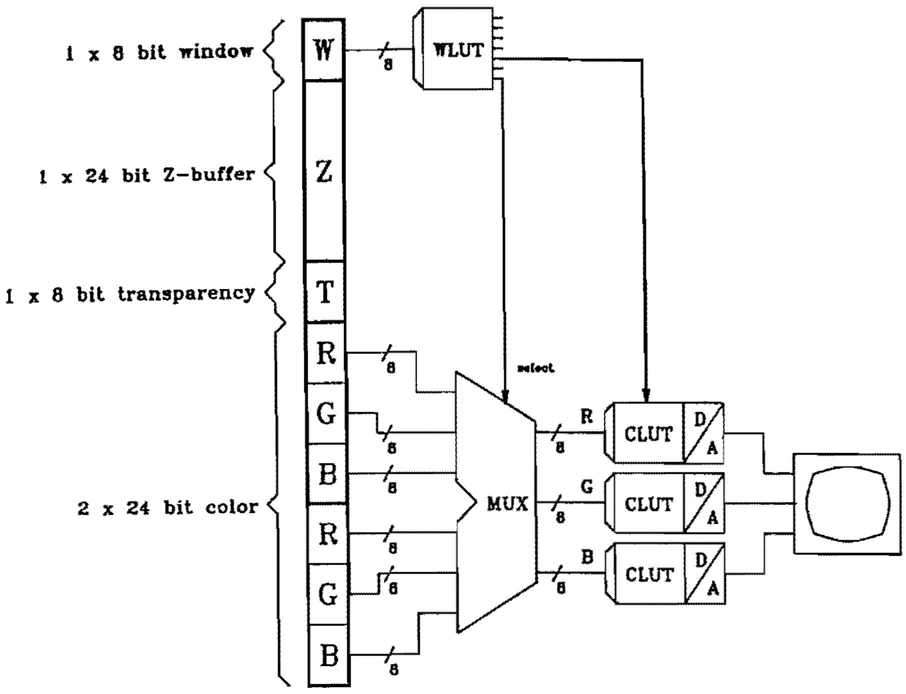

Figure 4: Window-ID evaluation

- Controlling the rendering processors' write accesses allows to handle arbitrarily shaped windows. Due to this for every write operation the rendering processors have to compare the window-ID of the new pixel with the stored one. In case of difference the write operation is cancelled. This mechanism is also used for off-screen memory access.
- The video controller utilizes the output of the WLUT to select for every window individually the source of pixel data which are to be displayed on the monitor Therefore double buffering is individually selected for every window.
- The window-ID manages to select one out of five colour look up tables (CLUT) for every window individually.
- The video controller can fill a window of any size and shape with a constant colour during one frame cycle by evaluating the WLUT output.
- Finally, advantage can be taken of the window-ID to support very efficiently the moving of multiple windows across the screen. Therefore the video controller is able to copy these windows without intervention of the processors. Before starting the copy function the window manager has to provide a shift vector, the identifiers of the windows to move and all identifiers of those windows which are allowed to cover the ones to move. While the copying is done by the video controller the rendering processors can already start to restore the formerly overlapped screen areas. The hardware copy function handles the visible part of the double buffer (color data), the window-IDs and optionally the cursor data.

The serial input/output ports of the VRAM chips of the on-screen and the display memory and the input ports of the RAMDACs are linked via the second crossbar switch. Normally the SAM ports of the on-screen memory are switched directly to the RAMDACs for a conventional screen refresh and to the SAM ports of the display memory to store all the pixel information of the present frame. This is a necessary precondition (figure 5a) before starting the hardware supported window copying described by the following example.

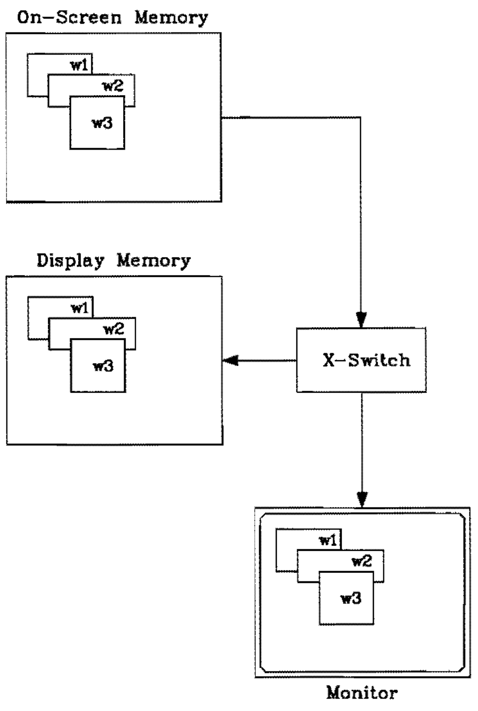

Figure 5a

A screen image including three windows W1, W2, W3 which is displayed on the monitor is stored in the on-screen and display memory. The joint windows W1, W2 are to be moved from the upper left corner (see figure 5b) in direction to the lower right corner. For this example the window W3 can occlude the windows W1 and W2 and the window W2 covers W1. The mechanism of the hardware supported window copying is illustrated by figure 5b to 5e.

The windows to be moved (W1, W2) are copied selectively into the display memory within one frame cycle. Simultaneously the contents of the on-screen memory is displayed on the monitor (figure 5b). Therefore the on-screen memory is read as in normal video refresh mode. The serial input stream to the display memory is controlled by two parameters: The window identifier related to every pixel determines (via the WLUT) if a pixel is to be overwritten in the display memory. The second parameter is a starting address of the display memory, which is calculated of a shift vector given by the window manager.

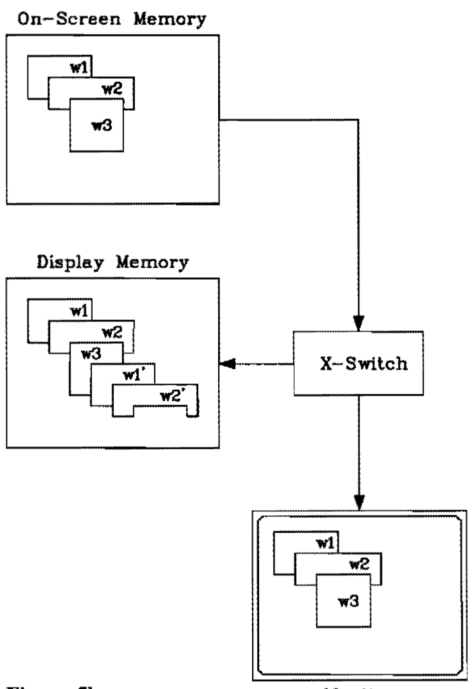

Figure 5b

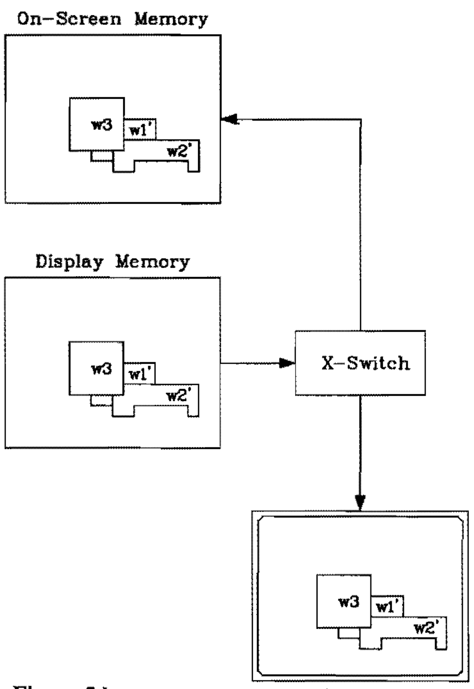

Figure 5d

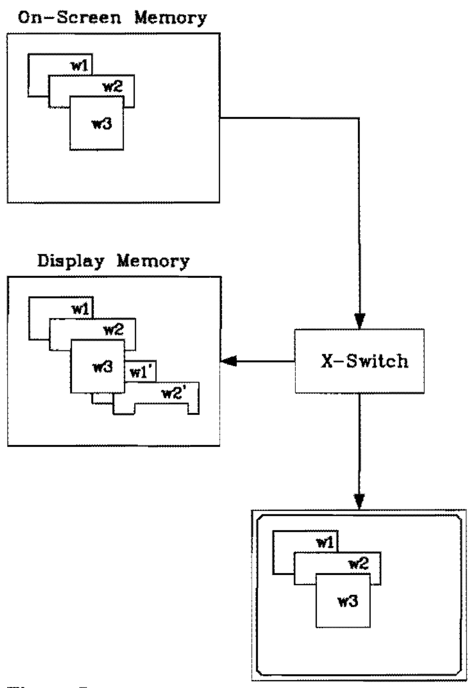

Figure 5c

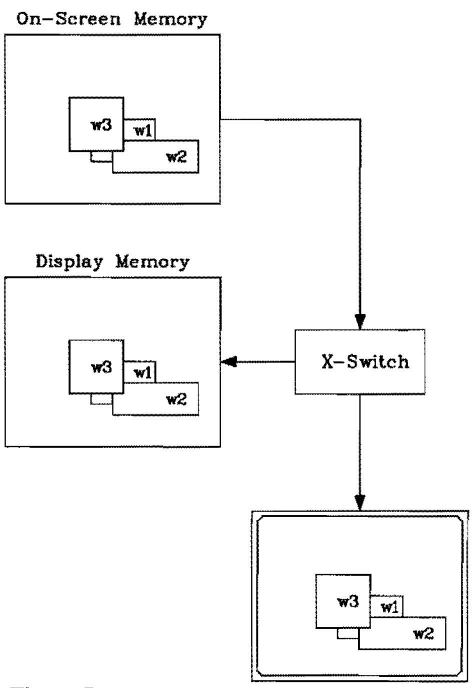

Figure 5e

In the following step all windows which may cover the ones to be moved (window W3 for this example) are copied into the display memory without regarding the shift vector(figure 5c). That means, the start addresses of the on-screen and display memory are the same. The selective copying is done as described in the previous step and again it lasts the time of one frame. The monitor input is still generated by the contents of the on-screen memory.

In the next frame, the video refresh is done by the display memory output. Simultaneously all pixels of the display memory are transferred via the serial links to the same location in the on-screen memory (figure 5d).

As a result of the copying procedure described above, there may be regions visible, which have to be regenerated by the rendering processors (e.g. background formerly covered by W1,W2). The rendering processors are free to perform this regeneration during all steps of the copying procedure which are illustrated by figure 5b to 5d. The final result is shown in figure 5e.

3. The 3DGRP Multiprocessor System

Realizing the disadvantages of existing architectures mentioned in the introduction, the goals for the 3DGRP are as follows:

- Highly parallel
  To achieve the required system performance.
- Homogeneous
  Only one type of processor shall be installed, in order to minimize the efforts for hardware and software development.
- Algorithm-independent
  Changing of algorithms must not require changes in the architecture.
- Application-independent
  Every application shall be performed with full system performance.
- Automatic load balancing
  No processor will enter a period of being idle, thus achieving the highest possible effectiveness.
- Easy to extend
  If a performance upgrade is needed, the hardware can be extended with minimal effort.

Figure 6 presents the global structure of the 3DGRP. There is a number of identical processor moduls GRi working in parallel and accessing the frame buffer. Each modul contains a processor, its memory and an interface to the geometry and rendering bus and to the frame buffer.

As in Torborg's approach, the objects are distributed to the processor moduls GRi via the geometry bus, which is implemented as multiprocessor bus as well as a broadcast bus. If a GR processor fetches a new geometric primitive to be processed, the geometry bus acts as a normal multiprocessor bus. Enabling the geometry bus in the broadcast mode it is a means to speed up the initialization phase and to shorten the overall transfer time. This is useful while updating context data within the memory of the GR modules.

Each processor performs the geometry processing for one object completely. The outcoming data are fed to the processors of the system via the rendering bus, which is similar to the geometry bus a broadcast bus as well as a normal multiprocessor bus. Then the processors render the objects for all those pixels being situated in the attached frame buffer bank. That means, the geometry processing of one object is done by only one processor. However, the rendering task of this object is done by all the processors but with reduced pixel amount.

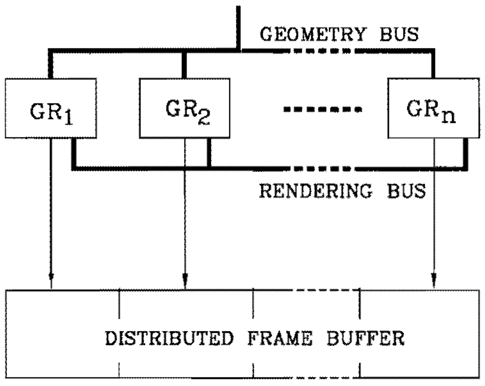

Figure 6: Global structure of the 3DGRP

The logical interface between the geometry and rendering calculations transfers pixels, vectors, triangles and trapeziums with edges parallel to the y-axis as rendering primitives to the GR modules. Due to their being preprocessed e. g. triangle data are calculated really fast. The start values for a span are computed in less than 500 ns and a pixel within a span is delivered every 1 us. Note that these values are valid for true color, z-buffered Gouraud shading, and all the calculations are done with full floating point precision, thus reducing computational inaccuracy to a minimum.

The effort of computation for processing the incoming data is dependent on the size and position of the geometric primitives. Small triangles or short vectors in parallel to the x- or y-axis require only a small number of rendering operations. The greater burden of computation relative to the resulting number of pixels of that primitive is in the geometry section. If there are very big triangles, much more processing power for rendering calculations is needed than for geometric calculations.

The peak performance of this architecture is limited by the number of processors involved and is delivered when nearly all processing power is exploited for rendering computations. The above mentioned data dependencies cause shifts of processing power requirements between the geometry and the rendering section. This architecture will overcome this difficulty within its area of performance.

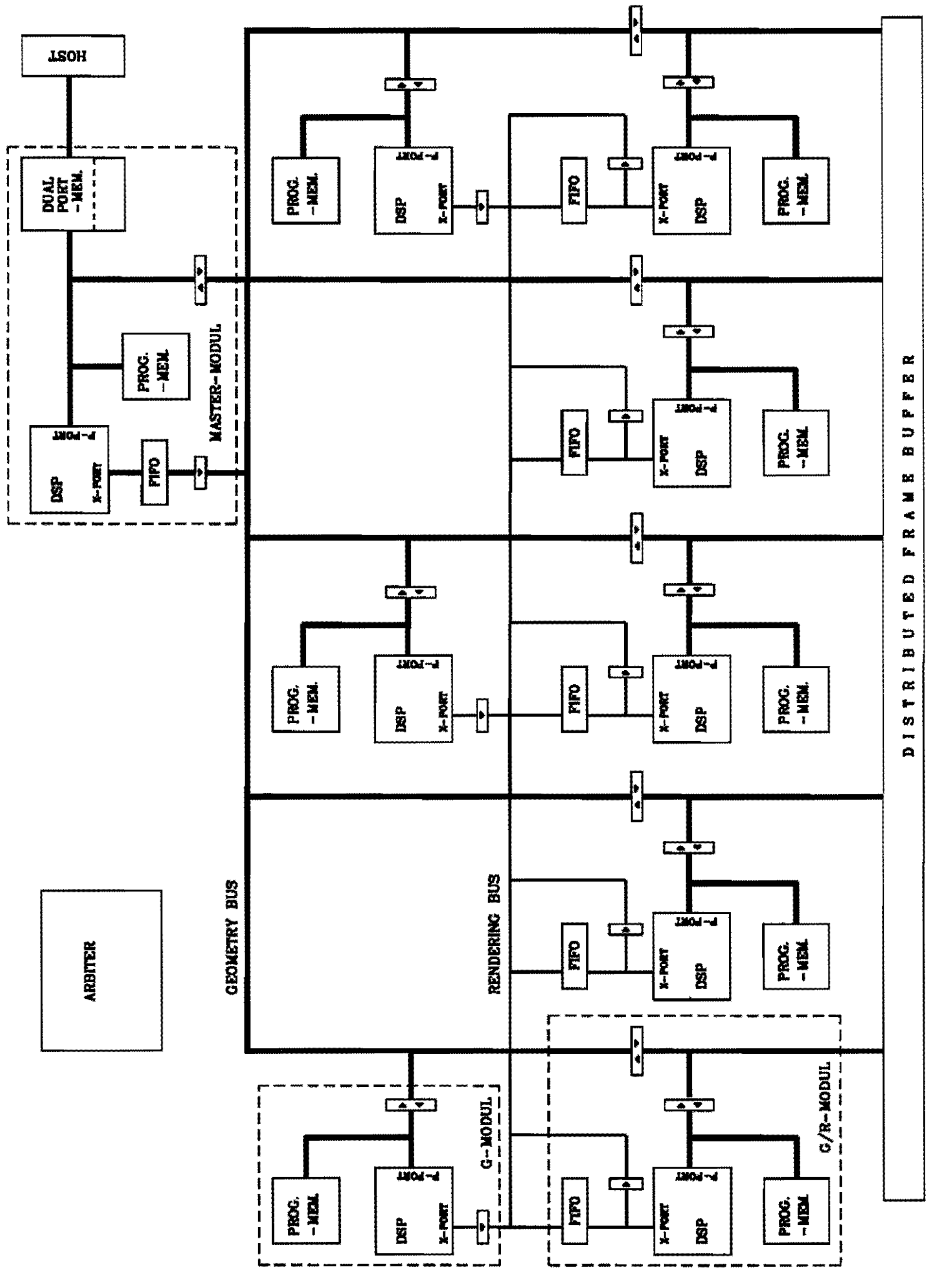

Figure 7: The 3DGRP architecture

Since any processor may run idle after rendering an object, if there is no rendering data in his input buffer, it will request new unprocessed objects and continue geometry calculations. In this way an automatic load-balancing is achieved across all the processors. When several GR moduls are doing geometric calculation, the overall rendering performance is reduced in favour of geometry processing power, but no computational power is going to be wasted by a GR processor starting out to run an idle state.

For this switching mechanism an algorithm is required, that changes quickly the context of computation. The internal structure of the processors (single cycle instructions, internal memory) provide a fast task switching within 2 us or less thus supporting sustained processor performance and avoiding the advances of automatic load balancing to be eaten up by switching time.

In figure 7 a realization example of 3DRP is given, utilizing five GR-modules. As processor the digital signal processor (DSP) Texas TMS 320C30, the fastest microprocessor available at present, has been chosen. This processor provides two memory ports, so it can be connected easily to the geometry and rendering bus.

For graphics applications where the computation burden within the geometry section is much higher than within the rendering section the basic structure shown in figure 6 can be augmented with several geometry moduls (G modul). In figure 7 there are three G moduls presented. The processor of a G modul performs only the geometry tasks of the graphics application. Therefore the G modul is a copy of the GR modul, but a FIFO and a frame buffer interface are not needed.

The master DSP within the master modul (fig. 7) communicates with the host via a dual-ported memory. The host transfers the graphics data of the application into the dual-ported memory. The master DSP interprets and preprocesses the data and moves them via the extension port into the master FIFO. All processors (of the G and GR moduls) can read the geometric data from the master FIFO. The arbiter in the background is scheduling the geometry bus, so that only one processor at a time can access the geometry bus.

The performance of this realization of a 3DGRP architecture is estimated as nearly 400,000 Gouraud-shaded 10 pixel vectors per second and nearly 50,000 Gouraud-shaded 100 pixel triangles per second.

References

1. Akely, K., Jermoluk, T.: High Performance Polygon Rendering. Proceedings of SIGGRAPH, 22(4): pp. 239-246, August 1988
2. Akely, K.: The Silicon Graphics 4D/240GTX Superworkstation. IEEE Computer Graphics & Applications, 7: pp. 71-83, July 1989
3. Apgar, B., Bersack, B. and Mammen, A.: A Display System for the Stellar Graphics Supercomputer Model GS1000. Proceedings of SIGGRAPH, 22(4): pp. 255-268, August 1988
4. Clark, J.: The Geometry Engine: A VLSI Geometry System for Graphics. Computer Graphics, 16(3): pp. 127-133, July 1982
5. Demetrescu, S.: High Speed Image Rasterization Using Scan Line Access Memories. Proc. 1985 Chapel Hill Conference on VLSI, pp. 221-243, Computer Science Press, 1985
6. Fuchs, H. and Poulton, J.: Pixel Planes: A VLSI-Oriented Design for a Raster Graphics Engine. VLSI Design, 2(3): pp. 20-28, 3rd. Quater 1981
7. Gharachorloo, N., Gupta, S., Sproull, R.F. and Sutherland, I.E.: A Characterization of Ten Rasterization Techniques. Proceedings of SIGGRAPH, 23(3): pp. 355-368, July 1989
8. Gharachorloo, N. and Pottle, C.: SUPER BUFFER: A Systolic VLSI Graphics Engine for Real Time Raster Image Generation. Proc. 1985 Chapel Hill Conference on VLSI, pp. 285-305, Computer Science Press, 1985
9. Potmesil, M. and Hoffert, E.M.:The Pixel Machine: A Parallel Image Computer. Proceedings of SIGGRAPH, 23(3): pp. 69-78, July 1989
10. Potmesil, M., McMillan, L., Hoffert, E.M. et al.: A Parallel Image Computer with a Distributed Frame Buffer: System Architecture and Programming. Proceedings of EUROGRAPHIGS '89, Hamburg, FRG: pp. 197-208, September 1989
11. Rhoden, D., Wilcox, C.:Hardware Acceleration for Window Systems. Proceedings of SIGGRAPH, 23(3): pp. 61-67, July 1989
12. Rogers, D.F.: Procedural Elements for Computer Graphics. MacGraw-Hill, 1985
13. Selzer, H., Haaker, T., Joseph, H.: 3DGRP - A High Performance Graphics System. Proceedings of The 1989 IFIP WG5.10 International Working Conference on Workstations for Experiments, Lowell, USA, July 1989
14. Sproull, R.F.: Frame Buffer Display Architectures. Annual Review of Computer Sience, 1: pp. 19-46, Annual Reviews Inc., 1986
15. Torborg, J.: A Parallel Processor Architecture for Graphics Arithmetic Operations. Computer Graphics, 21(4): pp. 197-204, 1987
16. Tunick, D.: Powerful display system revs up image and graphics processing. Electronic Design, July 23, 1987
17. Whitton, M.C.: Memory Design for Raster Graphics Displays. Computer Graphics & Applications, 4(3): pp. 48-65, March 84