4 Dynamic Load Balancing within a High Performance Graphics System

Harald Selzer

ABSTRACT Interactive 3D graphics applications require significant arithmetic processing to meet the ever-inreasing desire for higher image complexity and higher resolution in displayed images.
This paper describes a graphics processor architecture with a high degree of parallelism connected to a distributed frame buffer. The architecture can be configured with an arbitrary number of identical, high level programmable processors operating in parallel.
Within the architecture an automatic load balancing mechanism is presented which distributes the processing load between geometry and rendering section.
After the unique features of the architecture are described the load balancing mechanism is analyzed and the increase of performance is demonstrated.

4.1 Introduction

Since human visual perception is the most effective method to perceive a lot of informations in a short time, the photorealistic rendering for the visualization of medical, physical or technical data requires speed improvements and demands for developing innovative architectures.

Modern workstations with a state of the art graphics platform incorporate some form of hardware support for graphics applications to release the CPU from the burden of visualization tasks. Sophisticated user interfaces within any CAX application in conjunction with high interactivity and realistic images require to split and parallelise the system to distribute overall computing load.

This paper describes considerations made within the work for the GRACE project [1], a development which tries to satisfy the requirements of a graphics processor architecture.

4.2 Background

4.2.1 Contemporary Architectures

Common to all raster display systems is the frame buffer, which stores the image on a pixel by pixel basis and decouples image generation and video refresh process. The design of the frame buffer with its partitioning related to object or screen space and the degree of parallel access possibilities are a keyfeature to systems merit [16].

Attempting to satisfy the demands of increased calculation rates a lot of architectures

[1] This project was funded by the Commission of the European Community in the ESPRIT-II-Program, Project-No 2569 (EuroWorkStation)

with different basic concepts have been developed [9] showing the demands of integrating more system functionality on a single chip.

A well known approach making extensive use of full custom VLSI devices is the design of the Pixel Planes [15],[8].

To achieve higher rendering performance and to overcome the frame buffer bottleneck the rasterization processor and the frame buffer memory are integrated on the same chip. Similar approaches were proposed in the Scan Line Access Memory [7] and the Smart Image Memory [4].

Other architectures try to parallelize functional modules of the image generation process e.g. by mapping the geometry section to a multistage pipeline of customized VLSI devices [5]. This design was enhanced and is now available as a full parallelized state of the art workstation [2],[3].

Another more general architecture is the Pixel Machine, a MIMD computer based on an array of asynchronous processor nodes with parallel access to a large frame buffer [14]. The advantage of this approach is the homogeneous structure and the programmability which allows all algorithms to be implemented in software.

4.2.2 Goals

4.2.2.1 Principal Considerations

1. Frame Buffer

The memory in which an image is stored on a pixel by pixel basis is called the frame buffer or image memory. This memory is accessed on the one hand by the rendering processor, which writes data into the memory and on the other hand by the video refresh controller, which reads from the memory and conveys pixel data to the video output circuitry and the display monitor.

The image memory built up with conventional DRAMs can bother image generation process at rendering processor side as well as at video refresh side. Using todays available video RAMs (VRAMs) improves the speed of frame buffer access dramatically (Whit84). Nevertheless a certain level of performance implies the need of parallelism within the frame buffer. A resolution of 1024x1280 visible pixels with a 60 Hz refresh rate (noninterlaced) requires a pixel frequency of about 110 MHz or equivalent 330 Mbyte/s transferrate for full colour representation with 24 bits/pixel. A monolithic frame buffer can not achieve that. The maximum clock frequency of the VRAM shift register measures 30-40 MHz and is therefore limited to resolutions 640x480 pixels with 60 Hz video refresh rate (noninterlaced) or equivalent.

On the other side display processors with 25ns cycle times have to compete for the random access port of a VRAM with a normal cycle time of about 150ns (no page, nibble or static column mode is taken into account) slowing down image generation .

The solution appears to be found in writing multiple pixels into the memory in parallel, the basic concept of the distributed frame buffer. The frame buffer could be divided into rows, columns, or arrays [9] [16] and each of these parts is attached to a separate rendering processor thus overcoming the memory access bottleneck.

2. Floating Point Versus Fixpoint Calculation

While designing a system- architecture the central question is arising which processor is the most suitable for that design.

Graphics applications are very arithmetic intensiv tasks and therefore need processing devices with very powerfull arithmetic and logic units (ALUs).

State of the art processors include a floating point unit on chip. But nevertheless only a few of these processors incorporate sufficient powerfull arithmetic units to perform floating point operations as fast as integer operations. Especially if the system signed with application specific integrated circuits (ASICS) it is worth while to consider which accuracy for mathematical calculations in a graphics system is needed. Numerical intensive operations are performed in the geometry and the rendering section. Typical tasks within the geometry section requiring floating point calculation with high accuracy are the following:

- Transforming objects with world coordinates to image space,
- Interpolating vertex normals (Phong shading)
- Normalizing interpolated vertex normals (Phong shading)
-Performing the lighting calculations (Phong and Gouraud shading).

Using a single precision floating point number results in a maximum inaccuracy of 2exp-150 (decimal equivalent: 7*10exp-46) per operation [13]. This is a sufficient precision for the operations mentioned above without visible effects. The rendering section comprises operations like

- colour interpolation (Gouraud shading)
- z-value interpolation for z-buffering
- transparency calculation
- algorithms for image processing.

For an image with a limited resolution most of this operations could be done with fixpoint arithmetic in an appropriate precision.

Suggesting a resolution of 2048 x 2048 pixel and a fixpoint representation with a fractional part consistng of 16 part consisting of 16 bits, a RGB model colour interpolation over a whole scanline would incorporate a binary error of 2exp-6 - a deviation not perceptible for the human eye on todays monitors.

This shows that for the mathematical calculation in the geometry section floating point units are necessary but in the rendering section the mathematical computations could be done with fixpoint precision. Therefore, if a straightforward architecture for a specific application is implemented with no parallelism on board or module level, fixpoint arithmetic may suite well - an approach that was realized and tested well for a fast Gouraud triangle shader [1].

In the system-architecture discussed in this paper the processors should be able to perform rendering tasks as well as geometry calculations. This argue mainly led to the decision to incorporate digital signal processors (DSPs) which have a floating point unit on chip and were at the time of system design the fastest processors available on the market.

4.2.2.2 System Characteristics and Design Goals

The architecture to be realized should be capable of generating high quality images within a moderate time. That means the hardware should be as fast as possible but not as big as possible. The employed computatiomal power should be used very effectively. Other characteristics are:

- A flexible system, high level programmable to enable the implementation of all graphics functions necessary and various algorithms for image generation.
- Parallelism should be implemented wherever possible
- Homogene. To become familiar with an off-the-shelf VLSI device needs some time. To become familiar with a few different such devices needs a lot of time. Therefore the number of different off- the-shelf VLSI components had to be reduced to a minimum to ease system use and shorten software development time.
- The arithmetical and logical units (ALUs) should be available off-the-shelf.
- The frame buffer design should overcome the access bottleneck on the generation side as well as on the video side and incorporate hardware support for fast window handling.
- The frame buffer resolution is 1280 * 1024 pixel with a video refresh rate of 60 Hz (noninterlaced). Every pixel has 24 bit colour and is double buffered as well as z-buffered.
- The frame buffer should provide double buffering in order to accomodate dynamics and z-buffering too.

4.3 The Architecture

4.3.1 Overview

Taking into account the demands of the different tasks within the image generation process the mapping of the functional sections to hardware suggested the splitting into units as shown in Figure 4.1.

Above of the frame buffer there are three different units handling the image generation process:

- The Master Module
- The Geometry Module
- The Rendering Module.

The master module is the systems supervisor, handles the communication to the host processor and is responsible for start-up and synchronising activities.

The geometry module transforms and clips the graphic primitives, subdivides biparametric patches and the lighting calculations that are necessary and tasks like this.

The rendering module performs the shading algorithms and transfers pixel data to the frame buffer. The rendering module also supports too all functions of the geometry module (Figure 4.1).

All modules contain a digital signal processor (DSP) with up to 256k * 32 bit wide, fast static memory for instruction and data storage. This type of processor was chosen because of its 60ns instruction cycles, the on-chip cache and the floating point unit and the two independent, parallel bus interfaces [10].

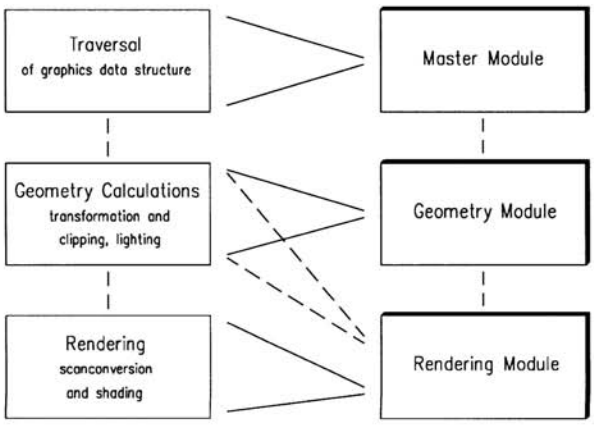

FIGURE 4.1. Mapping functional sections to hardware

4.3.1.1 The Master Modul

The communication to the host processor is handled over a 256k * 32 bit dual ported memory allowing to transfer and process data in parallel. The interface is asynchronous and interrupt driven for fast response and transfers data up to 20 Mbyte/s.

The master module traverses the graphics data structure and feeds graphics data to a special first-in-first-out memory (FIFO) for delivering to the appropriate processors.

In the case of synchronizing or updating (e.g. graphics context, colour lookup tables, etc.) the master takes over system control and bypasses the pipeline with a direct access to the appropriate resource.

4.3.1.2 The Geometry Module

Graphics data are transferred to the geometry modules by a rate of 33 Mbyte/s. The geometry module performs the transformation, clipping, polygon and patch subdivision, normal interpolation and renormalisation and lighting operations in an appropriate manner and delivers the processed graphical primitives to the rendering module data FIFOs.

4.3.1.3 The Rendering Module

The structure of the rendering module is similar to that of the geometry module. For rendering calculations like shading and scan conversion the processor fetches data from its data FIFO and conveys the calculated pixel values to the frame buffer. For image processing purposes data are read from the frame buffer, manipulated and written back.

Because the rendering module can act as a geometry module too, it can also directly fetch graphics data from the master data FIFO and deliver processed data to the FIFOs of the appropriate rendering modules (see Section 4.5).

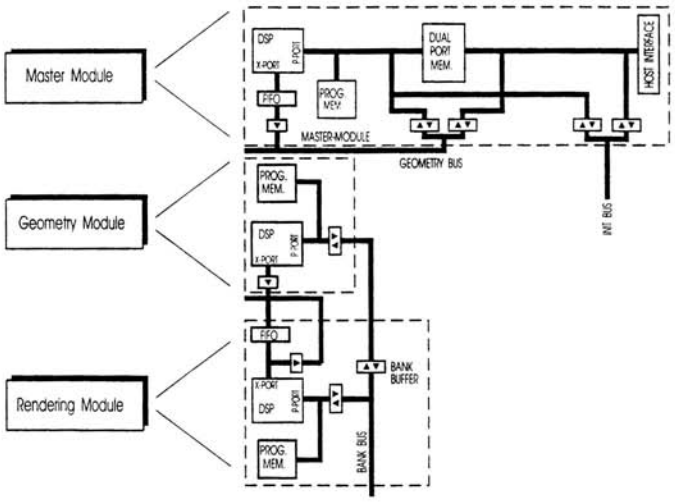

FIGURE 4.2. Basic module structure

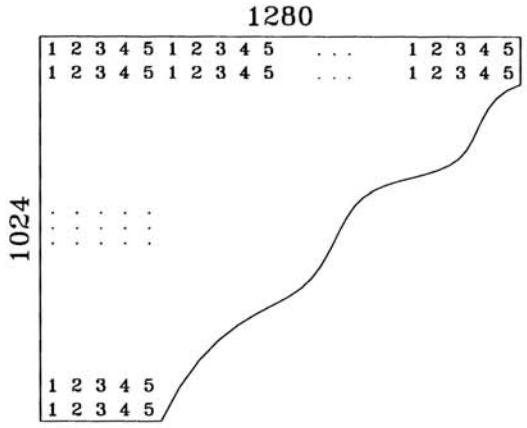

FIGURE 4.3. Frame buffer interleaving

4.3.1.4 The Frame Buffer

The frame buffer is distributed and divided into 5 parts with an overall resolution of 1280x1024 pixels with 88 bits per pixel (2x24 bit colour, 24 bit z-buffer, 8 bit transparency, 8 bit window identifier) with a video refresh rate of 60 Hz (noninterlaced).

Clipping at arbitrarily shaped windows is supported by hardware as well as fast copying of windows and bit block operations at high data transfer rates [11].

4.3.2 Overall Architecture

The system is organized as a pipeline with additional parallelism on functional level by multiplying geometry and rendering modules (see Figure 4.4). The number of the rendering modules is fixed to a multiple of five due to technical reasons, whereas the geometry modules can be multiplied theoretically unlimited. The current configuration comprises three geometry and five rendering modules.

Three independent busses enable parallel data transfer to and from multiple resources of the system.

All modules are connected to the geometry bus which acts as the system bus. All system resources are accessable by the master. System, graphics or update data are transferred in single or broadcast mode with 33 Mbyte/s.

The rendering bus is designated to convey only rendering primitives to the data FIFOs on the rendering modules. For speed reasons data are transferred synchronously with up to 132 Mbyte/s.

Theinit bus allows a direct acccess to the video and cursor planes used for fast update of the colour look up tables (CLUT) and generating the cursor in separate cursor planes.

Each rendering processor writes data with 33 Mbytes/s to the frame buffer bank attached to it which results in a total transfer rate of 165 Mbytes/s.

The frame buffer and the video/cursor plane memories can be accessed also by the host processor in order to get a possibility to bypass the graphics pipeline. This supports e.g. the handling of pixel maps if the host processor wants to transfer pixel values to or read back from the image memory.

4.4 Dataflow

In the entire system graphics data are processed simultaneous and transferred to the subsequent modules in parallel. From stage to stage the number of elements per object increases as the content of information per element decreases (Figure 4.5).

The master module traverses the graphics data structure and puts the high order primitives like splines, polygons, meshes or triangles into the data FIFO. If a geometry module has finished the last task, it accesses the geometry bus and fetches the next primitiv or task automatically. All geometry calculations are done within a single module.

The logical interface between the geometry and rendering calculations transfers triangles, vectors, pixel and trapeziums with edges parallel to the screen y axis columns. The data structure incorporates processor specific data (due to the distributed frame buffer) and common data. The latter ones are broadcasted to the rendering modules.

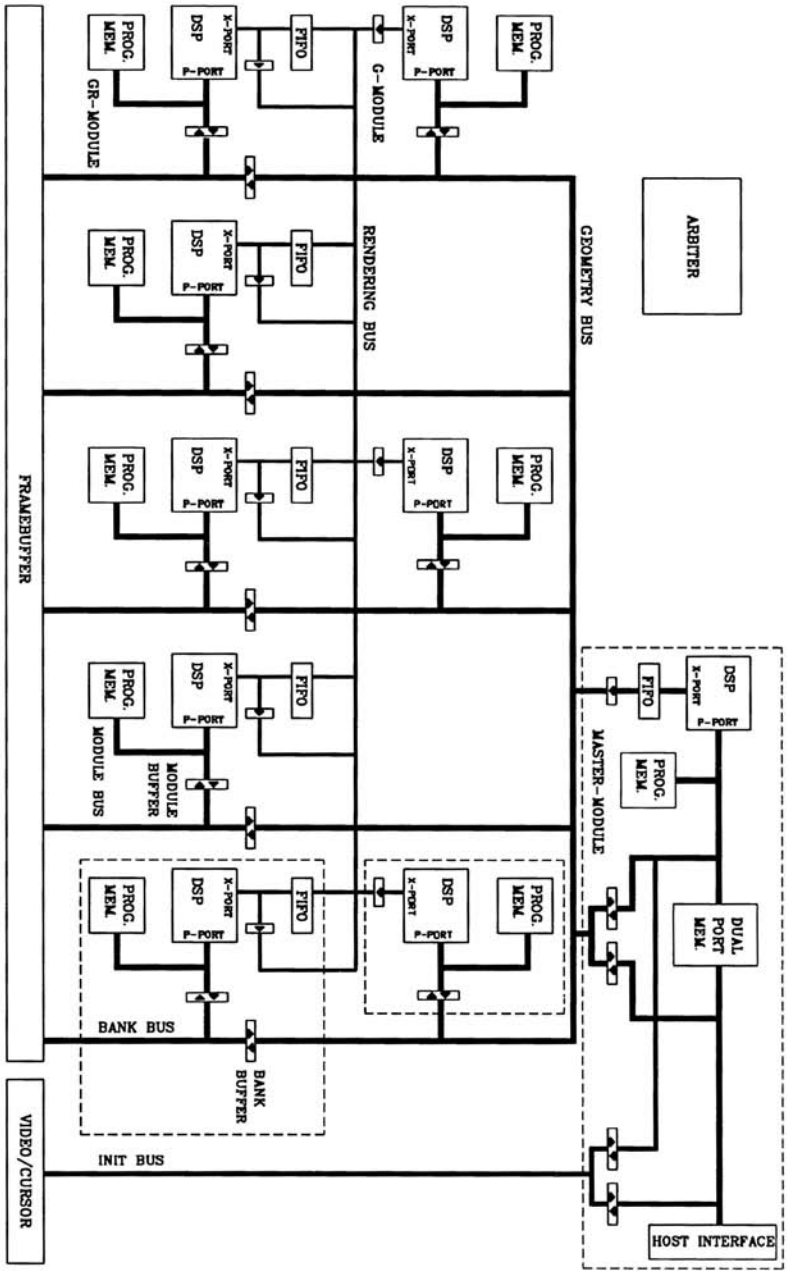

FIGURE 4.4. System architecture

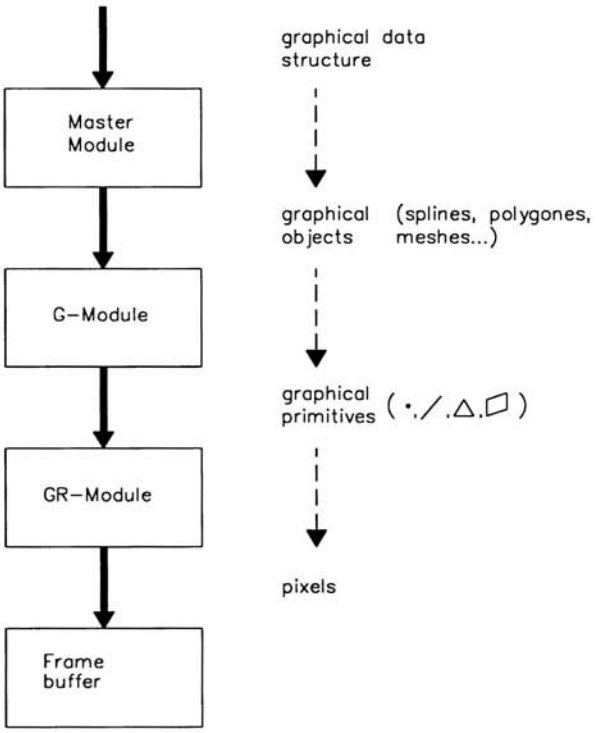

FIGURE 4.5. Graphics data processing

4.5 Load Balancing

4.5.1 Automatical Regulation

The effort of computation in the geometry and rendering section depends on size and position of the geometrical objects. Small triangles or short vectors parallel to the x or y axis require only a small number of rendering operations. In fact the time consumed to initialize the rendering processor for primitives producing only a few pixels is greater than the rendering time itself. On the other hand the number of geometric calculations for interpolating shading methods is independent from the resulting size of the primitive.

An anlysis of scene complexity has shown, that in most cases the image is generated from a lot of small triangles (1-10 pixels), a number of medium sized (11-100 pixels) and a few large ones (101-1000 pixels) [6].

Further investigations with less complex scenes (no more than 5000 triangles) have shown a more extrem distribution of the size of triangles incorporated (s. statistics shown below). The pictures are shown at the end of this paper.

The reason is the way of modeling a scene i. e. things of interest are generated with a lot

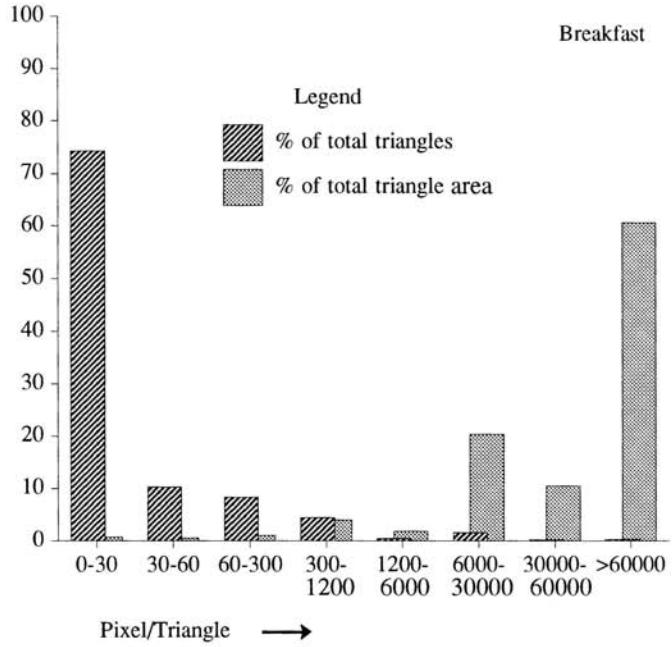

FIGURE 4.6. Image statistics for "breakfast"

of primitives to get a fine grained surface. The rest of the scene especially the background is defined with only a few but very large primitives (triangles).

The pictures analyzed in the statistics below were rendered with a solution of 1024 x 1280 pixels.

The chessman figure is an example for a picture defined without background.

Additionally the future increase in graphics performance will be used to display more complex scenes rather than displaying the same number of objects faster. Those images will comprise a lot of very small triangles shifting the load of computation to the geometry section.

Nevertheless the size and the number of triangles an image consists of may vary from scene to scene or even from view to view. This will cause idle states within a fixed balanced architecture. With the intention of exploiting all the distributed computational power of the system, the processing units have to be able to adapt their activities to the actual processing requirements of the scene.

To enabel such a dynamic load balancing and to speed up geometry calculation dynamically if required, the rendering modules are capable of performing all the geometry

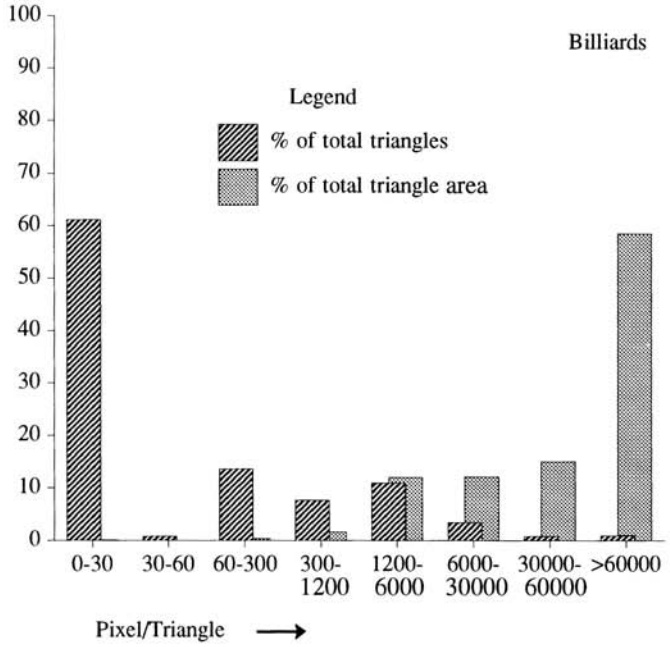

FIGURE 4.7. Image statistics for "billiards"

calculations too.

After rendering an object any processor may run into an idle state if there is no rendering data in his input buffer. Performing a task switch it will request new unprocessed geometry objects and continue geometry calculations. In this way an automatical load balancing is achieved across all the processors. When several rendering modules are doing geometrical calculation the overall rendering performance is reduced in favour of geometry processing power. Doing so the exploitation of the processing power incorporated encounters more than 95% and no computational power is going to be wasted by a rendering module starting out to run an idle state.

4.5.2 Task Switching

The capability of automatically distributing the work load between geometry and rendering modules means inherently task switching between two jobs within the same application. Supported by the large local memory (up to 256k x 32bit) the switching is reduced to the saving and restoring of all processor registers, processing interrupt control and in-

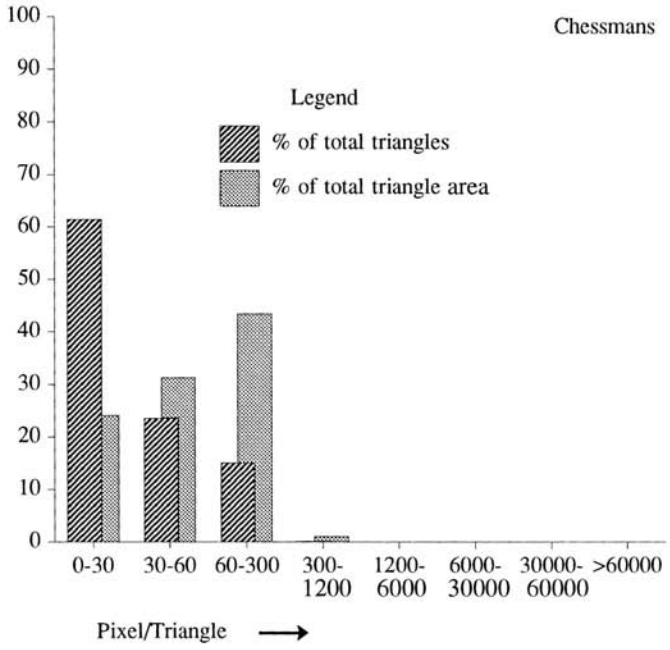

FIGURE 4.8. Image statistics for "chessmen"

specting the rendering FIFO status. This is performed in less than 5 us. Since this time is very small with respect to the time consumed for geometry processing the self balancing is even useful for scenes comprising only very small triangles (see below).

4.5.3 Performance Increase by Task Switching

The task switiching capability of the rendering processors accelerates the geometry calculation but as mentioned above the switching itself consumes time. How much calculation time is eaten up by task switching and by which factor geometry calculations are accelerated if a rendering processor switches to geometry processing is evaluated in this chapter.

The peak performance of this architecture is delivered when all rendering processor power is exploited for rendering calculations and the rendering are working continuously.

If the balancing mechanism is activated the peak performance is not achieved, but the processing power of the rendering modules is used to speed up geometry calculation. This has two effects:

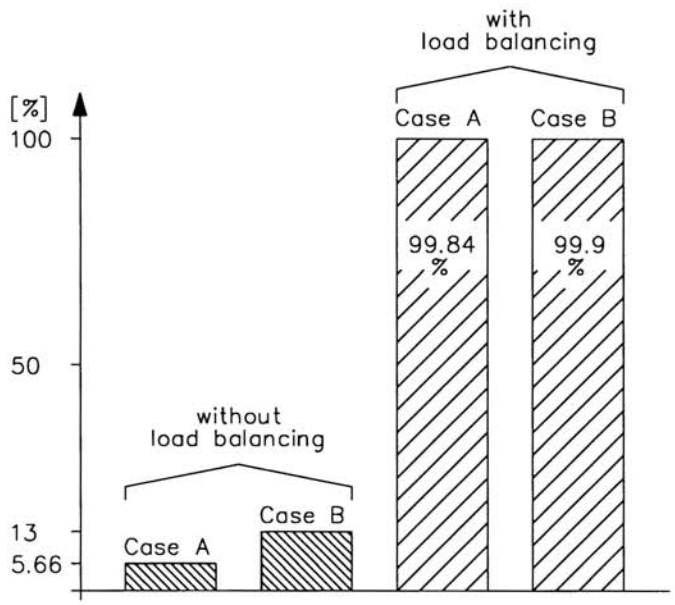

FIGURE 4.9. Exploitation of the rendering processor computation power

- it prevents the rendering modules from running into an idle state and
- speeds up image generation by supporting substantially geometry processing.

For simulation of the architecture the DARENDER graphics software was chosen [12]. This is a functional implementation for PHIGS-PLUS/PEX. It is fully written in and incorporates no optimazations in form of assembler routines or similar. The hardware for simulation was a 32 MHz application board of the Digital Signal Processor development toolkit. The scene used for exploitation measurement shows several goblets in 3D space. The goblets were fed into the architecture in a b-spline representation with different parametrizations: -Case A: The goblets were tesselated into 10082 triangles covering 81389 pixels (about 8 pixel/triangle). -Case B: The goblets were tesselated into 800 triangles covering 79713 pixels (about 100 pixels/triangle).

Backface culling was disabled, so all the traingles were rendered with Gouraud shading and z-buffered.

The hardware architecture was simulated with one geometry processing and five rendering processing modules. Figure 4.9 shows important results of the simulation.

First, for the goblet scene only 5.6 % of computation time in case A and 13 % in case B were needed for rendering.

Second, without load balancing the rendering processors are going to waste a lot of their computational power within an idle state.

The time for all task switchings and for data transfer was taken into account. Figure 4.10

	image generation time		
	without load balancing	with load balancing	speed up
billiards	3,69s	1,82s	2,03
breakfast	11,18s	3,37s	3,32
chessman	11,02s	2,46s	4,48

FIGURE 4.10. Image generation speed up

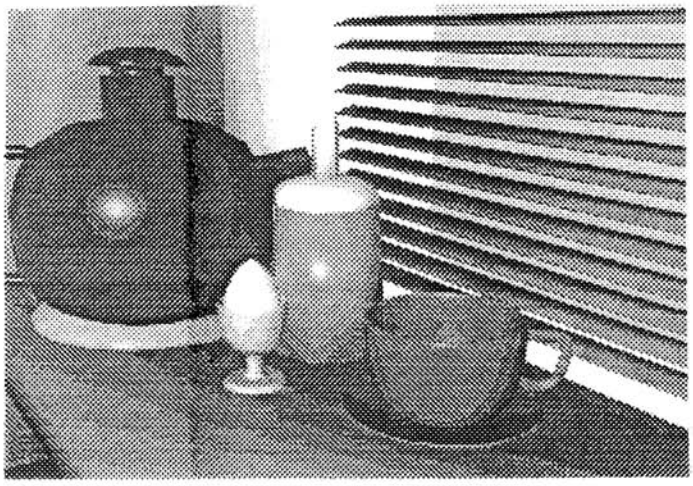

FIGURE 4.11. Breakfast

shows the resulting speed up using dynamic load balancing for the images shown below. The configuration choosen incorporated one geometry module and five rendering modules.

The simulations done so far are very appropriate for this architecture in the way a scene was rendered, which burdens a big computational load to the geometry modules and threrefore shows the load balancing mechanism working very effectively. Further simulations will be accomplished with less complex primitive representation and different scene definitions in order to evaluate the effectiveness of the dynamic load balancing mechanism in more detail.

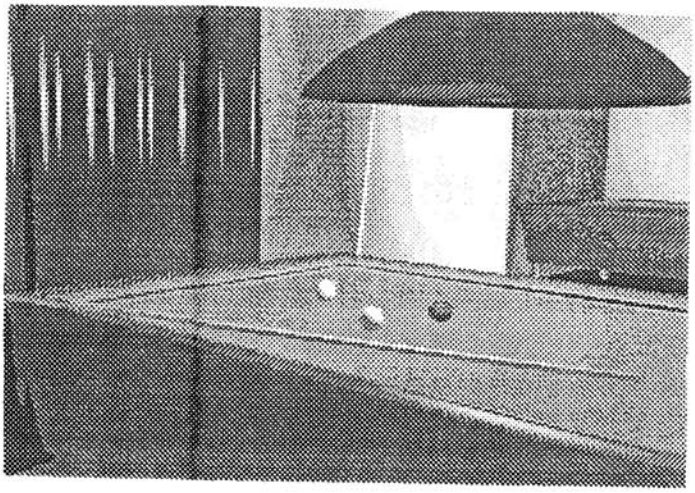

FIGURE 4.12. Billiards

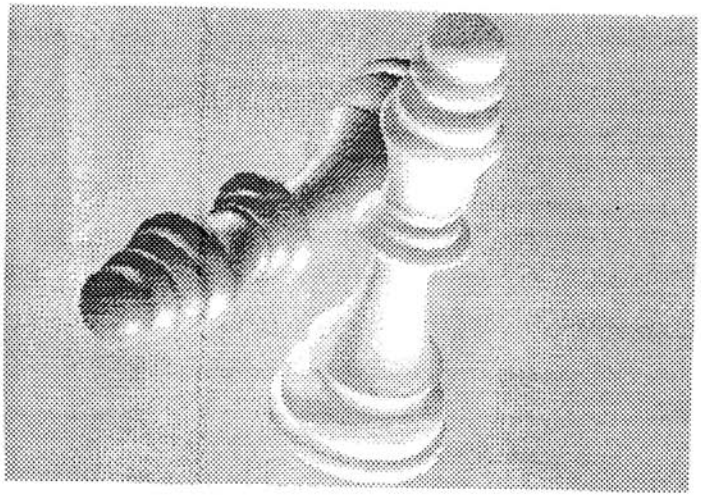

FIGURE 4.13. Chessmen

4.6 Summary

The presented architecture has a high degree of parallelism and overcomes the bottleneck of frame buffer access with its distributed frame buffer as well as overcoming the bottleneck of data dependencies with its dynamical regulation mechanism for load balancing. The high level programmability and homogeous structure make it easy to handle and designated for future enhancements.

4.7 References

[1] H.-J. Ackermann, C. Hornung, "The Triangle Shading Engine", In: R.L.Grimsdale, A. Kaufman (Eds.):*Advances in Computer Graphics Hardware V*,Eurographics Seminars. Springer-Verlag, Berlin, 1992, p.3-13.

[2] K. Akely, T. Jermoluk, "High Performance Polygon Rendering", ACM Computer Graphics, Volume 22 Number 4, August 1988, pp. 239-246.

[3] K. Akely, "The Silicon Graphics 4D/240 GTX Superworkstation", IEEE Computer Graphics and Applications, July 1989, pp. 71-83.

[4] J. Clark, M. Hannah, "Distributed Processing in a High Performance Smart Image Memory", Lambda 1, 3 , pp. 40-45.

[5] J. Clark, "The Geometry Engine, A VLSI Geometry System for Graphics", ACM Computer Graphics, Volume 16, Number 3, July 1982.

[6] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, Neil Hunt, "The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics", ACM Computer Graphics, Volume 22, Number 4, August 1988.

[7] S. Demetrescu, "High Speed Image Rasterization Using Scan Line Access Memories", Proceedings of Chapel Hill Conference on VLSI, 1985.

[8] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, L. Israel, "Pixel Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced.Memories", ACM Computer Graphics, Volume 23, Number 3, July 1989, pp. 79- 88.

[9] N. Gharachorloo, S. Gupta, R. F. Sproull, I. E. Sutherland, "A Characterization of Ten Rasterization Techniques", ACM Computer Graphics, Volume 23, Number 3, July 1989, pp. 355-368.

[10] ESPRIT II - Project 2569, "3DGRP - 3D Geometry and Rendering.

[11] T. Haaker, H. Joseph, H. Selzer, "A Distributed Frame Buffer within a Window-Oriented High Performance Graphics System", In: R.L.Grimsdale, W. Strasser (Eds.):*Advances in Computer Graphics Hardware IV*,Eurographics Seminars. Springer-Verlag, Berlin, 1991, p.261-274.

[12] Christoph Hornung, "DaRender: The Darmstadt Renderer for PHIGS-PLUS/PEX", Technische Hochschule Darmstadt, Institut für Graphische Interaktive Systeme, Jahresbericht 1990.

[13] International Electrotechnical Commission, "Binary floating point arithmetic for microprocessor systems", July 1986.

[14] M. Potmesil, E. M. Hoffert, "The Pixel Machine: A Parallel Image Computer", ACM Computer Graphics, Volume 23, Number 3, July 1989, pp. 69-78.

[15] J. Poulton, H. Fuchs, J. D. Austin, J. G. Eyles, J. Heinecke, Cheng-Hong H., J. Goldfeather, J. Hultquist, S. Spach, "Pixel Planes: Building a VLSI-Based Graphics System", Proceedings of Chapel Hill Conference on VLSI, 1985.

[16] M. C. Whitton, "Memory Design for Raster Graphics Displays", IEEE Computer Graphics and Applications, Volume 4, Number 3, March 1984, pp. 48-65.