Volume Rendering FPGA Hardware

J. Hesser, A. Kugel, H. Singpiel, B. Vettermann, R. Männer

Lehrstuhl für Informatik V, Universität Mannheim
B6, 26, D-68131 Mannheim
{jhesser@rumms.uni-mannheim.de}

Abstract—

This paper discusses the implementation of volume rendering on FPGA hardware. The FPGA board is based on a 2x2 matrix of FPGA devices that are reprogrammed for different data processing tasks that occur during rendering. Moreover, it is shown how branch and data hazards are overcome in the pipelined system by multithreading.

Keywords— **FPGA, PCI, Volume Rendering, Visualization, Reconfigurable Computing**

I. Introduction

Volume rendering is today a standard rendering approach for displaying complex three or higher dimensional data, e.g. computerized tomography (CT), magnetic resonance imaging (MRI), or ultrasound (US). The basic idea of this sort of rendering is the simulation of the interaction of light with matter including reflection and absorption.

Imaging systems like CT produce stacks of two-dimensional slices that are interpreted as a three dimensional volume. Each volume element (a pixel of a slice) is denoted a voxel.

Volume rendering simulates how light passes through the volume. Part of the light is reflected to the viewer. Along the way the reflected light it is partially absorbed until it reaches the viewing plane. There the contributions of all reflected light rays are accumulated and generate the image that is displayed.

Typical data sets contain between 10-1000 voxels which corresponds to the rendering time on modern workstations that lie in the range of 10-1000 seconds. Thus interactive or even real-time rendering is not possible. Therefore accelerating the rendering process is imperative. There are two approaches to accomplish this task, hardware acceleration and/or algorithmic optimizations. The latter prevents to process voxels that do not contribute to the final image.

Volume rendering hardware for accelerating this type of visualization has been proposed over the last decade in several papers [1,2,3]. However, only a few prototypes have been built yet. Among them are the VIRIM system [LEV 88], VIZZARD [GUE 94], and the implementation of Cube-4 on HPs Teramac FPGA-processor [BRA 97]. Currently Mitsubishi integrates the Cube-4 architecture on the VG500 chip [KNI 94].

The second approach to speed-up rendering is to use algorithmic optimizations. These optimizations skip the processing of transparent voxels that have no interaction with light. This is called space-leaping. Furthermore, it suppresses the processing of all voxels whose reflected light does not contribute to the final image since nearly all light is absorbed during its way to the viewing plane. This technique is called early-ray termination. The advantage of these two improvements is that the complexity of rendering decreases for a volume of N^3 voxels from $O(N^3)$ to $O(N^2)$.

Thus a combination of both algorithmic optimization and special purpose hardware architecture should be an ideal solution. Nevertheless, only recently [LAC 94] an approach to solve this problem has been proposed. The new architecture is based on classical distance coding information for space-leaping and uses multithreading for hiding data hazards. While in paper [LAC 94] only the principle hardware architecture has been proposed we concentrate here on the implementation of the architecture on a multipurpose FPGA processor system called ATLANTIS.

II. Algorithm and Optimizations

In Figure 1 a principle standard volume rendering pipeline without optimization is shown. It consists of several pipeline operations:

From each pixel of the final image a ray is cast into the volume. For each ray, at regular intervals sample points are generated beginning with sample points nearest to the viewing plane. Each sample point processing then involves the following operations:

- The position of the sample point in the data volume is calculated. The 8 neighboring voxels are read out and a trilinear interpolation is performed. As result the gray value (interpreted as opacity) of the sample point is obtained.
- A gradient is calculated by determining the gray value change between neighboring sample points. This gradient is the normal of a possible plane in the volume.
- Using the plane normal, shading is performed, i.e., to calculate how much light is reflected to the viewer direction by diffuse and specular reflection.
- It is calculated how much of the reflected light intensity reaches the viewing plane due to absorption. Each contribution of a ray is accumulated

and represents the brightness of the considered pixel. This last stage is called compositing.

In this solution the data flows from the address generation stage where the position of the sample point is determined over the memory read-out, interpolation/gradient, and shading to the compositing stage. One sample point after the other of the considered ray is processed in the volume rendering pipeline. Because there is no feedback in the data flow it can easily be implemented in hardware.

With larger data sets, however, an excessive amount of hardware is required to achive real-time rendering rates [LAC 94]. Algorithmic optimizations mitigate this problem significantly, reducing the amount of hardware needed by at least one order of magnitude.

The optimization techniques are mainly space-leaping and early-ray termination. Space-leaping skips empty regions in the data set. Early-ray termination stops further calculation when the intensity of a ray is below a user definable threshold.

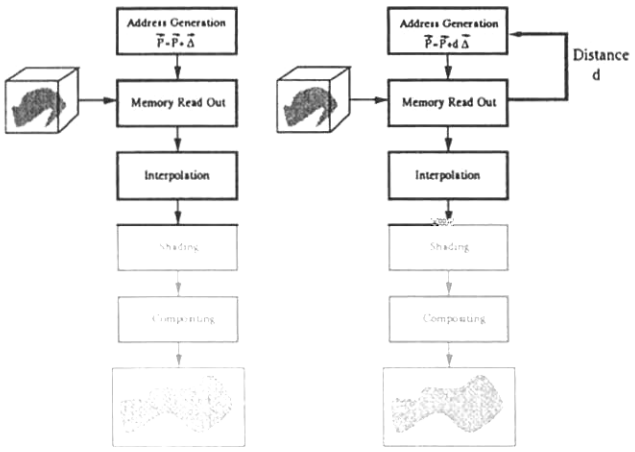

Fig. 1 Left: Standard volume rendering pipeline. Right: Volume rendering pipeline with algorithmic optimization.

The implementation of these optimization techniques in the standard volume-rendering pipeline would introduce major problems in the pipeline flowthat reduce the pipeline speed drastically. This problem described next (see Figure 1).

Space-leaping assumes that the data set is preprocessed. For each voxel the minimal distance to the next non-transparent voxel is stored in a distance data set. During rendering along with the sample point's neighbors the corresponding distance value is read out. It represents the number of voxels that can be skipped before the next non-transparent voxel can be encountered.

However, the sample points of a ray are processed sequentially in the pipeline. The subsequent sample point address can only be determined when the distance value of the current sample point is already read from the volume memory. Consequently, there is a feedback in the rendering pipeline (data hazard). Further address calculation has to stop until the distance information is available. With modern fast memory interfaces like SDRAMs there is a delay of 3 clock cycles from addressing to data out. An additional delay originates from the calculation of the next sample point address. Both delays reduce the pipeline utilization to less than 10%.

The top of Figure 2 illustrates the problem. In this example a delay of three clock cycles is assumed. The hardware stages are drawn in the vertical direction, the elapsed time in clock cycles is represented by the abscissa. Dark squares indicate that the respective hardware stage is used during the corresponding clock cycle. As can be seen, due to the dependency of the operations 1 and 3 the pipeline is used only at one third of the time.

The bottom of Figure 2 shows our solution (see [LAC 94] for details). Instead of inserting no-operations into the pipeline, sample point positions of other rays are calculated. This approach corresponds to multithreading. Each ray is considered a thread and after each sample point there is a context switch to the next thread (ray)[1].

In our solution, up to 64 rays are processed in a round robin order. Thus there is enough time to calculate the next sample point position of a ray before this result is required.

Figure 3 shows an implementation of the rendering pipeline with multithreading. The main modification to the earlier pipeline is the ray queue. It is realized as a shift register where each slot stores the parameters of a single ray. When the new sample point of a ray is calculated the ray is added at the end of the ray queue.

Moreover, early-ray termination deletes a ray from the queue whose intensity is below a user defined threshold.

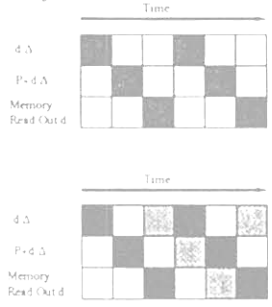

Fig. 2 Pipeline utilisation.
Top: Without multithreading
Bottom: With multithreading, different gray values indicate different rays.

[1] This approach is possible since the rays are processed independently.

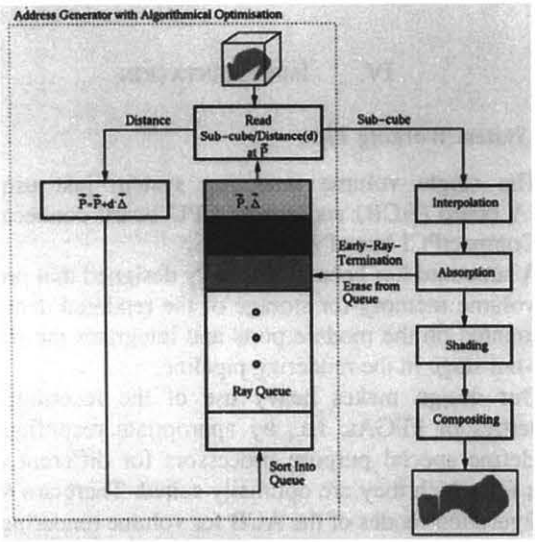

Fig. 3 Volume rendering pipeline with multithreading.

III. HARDWARE

For our implementation we use a multi-purpose FPGA and RISC based computing system called ATLANTIS [KUG 99]. It realizes a hybrid[2] system with a close coupling of RISC and FPGAs. CompactPCI provides the basic communication mechanism. Applications besides volume rendering are pattern recognition tasks like particle track recognition in high energy and heavy ion physics and n-body calculation in astronomy. The high-performance hardware is complemented by CHDL, a FPGA design tool with special support for hybrid systems.

A. *ATLANTIS system*

The basic idea behind ATLANTIS is to provide a modular multi-purpose computing platform for a variety of applications. Dedicated FPGA boards for computing and I/O, a high-end host CPU plus a private backplane bus system for up to 1 GB/s data rate support flexibility and scalability. A highly successful approach to adjust such a hybrid system to different applications is modularity. ATLANTIS implements modularity at different levels. First of all there are the main entities host CPU and FPGA processor which allow to partition an application into modules tailored for either target.

To satisfy the requirements of different applications concerning computing power and I/O bandwidth, computing and I/O resources are implemented on separate FPGA boards. Depending on a specific application a certain amount and combination of computing and I/O boards together with the host CPU form the particular system. For this a backplane based interconnect system (private bus) provides scalability.

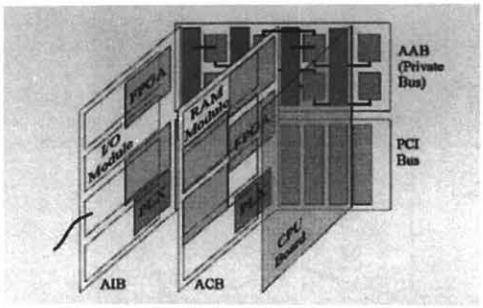

Fig. 4 ATLANTIS system overview

Finally, modularity is used on the sub-board level by allowing different memory types or different I/O interfaces per board type. The ATLANTIS system thus consists of four main components:

- The computing board – ACB
- The I/O board – AIB
- The private bus on the active backplane – AAB
- The host CPU with PCI backplane.

A schematic view of the whole system is shown in Figure 4.

The ATLANTIS computing board along with the host CPU are used for volume rendering. We therefore describe in the following section the ATLANTIS computing board (ACB) in more detail.

B. *ATLANTIS Computing Board (ACB)*

The core of the main processing unit of the ATLANTIS system consists of a 2x2 FPGA matrix (see Fig. 5). Assuming a usable gate count of approximately 180k per chip for the Lucent ORCA 3T125 the gate count per board sums up to 720k FPGA gates. Each FPGA has 4 different ports:

- 2 ports @ 72 lines to a neighboring FPGA each in vertical and horizontal direction
- 1 logical I/O port @ 72 lines and
- 1 module port @ 206 lines.

[2] In order to emphasise the close coupling between host-computer and FPGA system we use the term "hybrid" system.

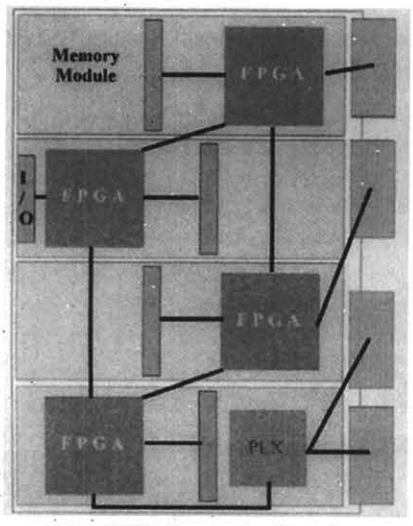

Fig. 5 ATLANTIS Computing Board

These 4 ports use a total amount of 422 I/O signals per FPGA. The 72 lines of FPGA interconnect provide high bandwidth as well as multi-channel communication between chips.

The module port is built from 2 high-density mezzanine connectors @ 124 pins. Depending on the application, memory modules with very different architecture or modules with additional logic resources like DSPs can be used to optimize the system performance. Also other I/O interfaces can be implemented.

The I/O port serves different tasks on the 4 FPGAs, depending on the physical connection of the chips:

- One FPGA is connected to the PLX9080 PCI interface chip thus providing the host-I/O functionality.
- Two FPGAs are connected to a backplane bus via the upper 2 CPCI connectors (J4/P4 and J5/P5).
- One FPGA supports two 32 bit M-Link[3] interfaces (sender and receiver) with a physical VHD68 SCSI connector for high speed external I/O.

The two backplane ports support high-speed I/O of 1 GB/s @ 66MHz, 2*64bit. The host interface via PCI supports two independent DMA (on demand) channels allowing 125 MB/s max data throughput.

[3] M-Link is a FIFO like new international standard for point-to-point links. At present we use an 80 MB/s LVDS implementation similar to CERN internal S-Link interface.

IV. IMPLEMENTATION

C. System Working Flow

The single volume rendering system just uses one FPGA board (ACB) and a host CPU board connected via the CompactPCI bus of ATLANTIS.

A subboard has been additionally designed that provides the volume memory for storage of the rendered data set. It is mounted on the module ports and integrates the memory read-out stage in the rendering pipeline.

Our design makes heavy use of the reconfiguration properties of FPGAs, i.e., by appropriate reconfiguration we define special purpose processors for different operations for which they are optimally suited. There are 4 main configuration modes of the ACB for volume rendering:

- Initialization
- LUT and reflectance map adjusting
- Distance calculation
- Rendering configuration

Each of them is set up within 35 ms up to 140 ms depending on the number of FPGAs which have to be reconfigured.

The first two configuration modes are only used for loading data into registers and memory whereas the other two configurations are used for setting internal parameters for the rendering process concerning the shading process and the assignment of voxel gray value to opacities.

A.1.C.1 Distance calculation

After startup the extension of the transparent regions within the data set is determined. This is done by distance transforms. For this purpose one FPGA has to be reconfigured with a two pass non-linear digital filter.

A.1.C.2 Rendering configuration

The rendering configuration executes 3 tasks in parallel:

- Downloading of ray parameters
- Implementation of the rendering algorithm
- Upload of rendering results

Downloading of ray parameters

For multithreading we use a 8×8 bunch of rays (64). For them several parameters have to be downloaded: The starting point (3×24 bit), the minimal distance to the next sampling point (3×16 bit) and the maximal ray length (10 bit) have to be stored for each ray on the FPGA in a parameter RAM.

To be able to overlap the time of downloading parameters and rendering, an additional local storage is selected. For this purpose the FPGA is used.

Implementation of the rendering algorithm

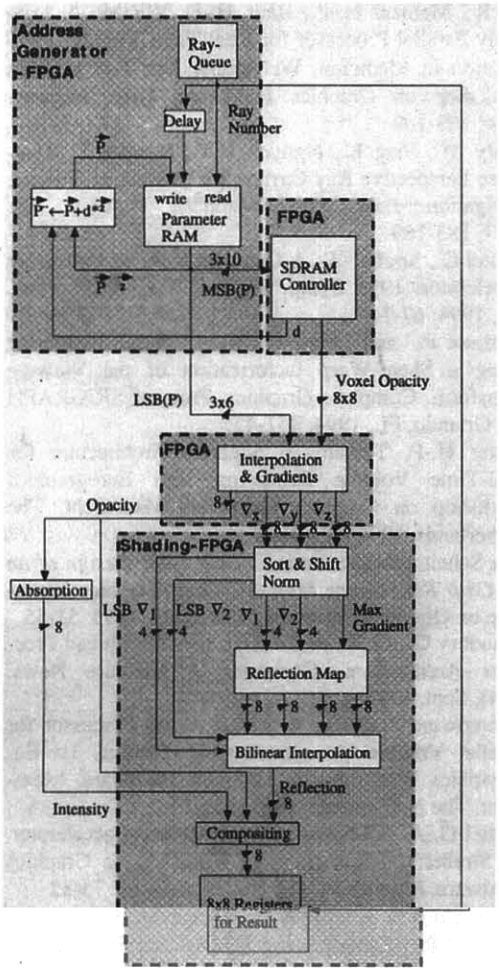

Fig. 6 Detailed view of the pipeline implementation. The three gray regions are the building blocks that are concentrated on one FPGA respectively.

The most interesting part for optimization is the ray queue. It implements the context switch for multithreading (see Fig. 6). This queue stores ray numbers that are used for addressing the parameter RAM.

Together with the address calculation (24 bit fixed point) and the ray parameters, one ORCA3T125 is completely filled. The three offset values (X,Y,Z) with 16 bit resolution have to be multiplied by the 8 bit distance value and added to the last sample point position of the ray, which is implemented as three times 24 bit.

One FPGA is needed for the SDRAM controllers of the volume memory since this volume memory is subdivided into 8 separate banks to increase the data throughput.

The 8×8 bit opacity values are together with 3×6 bit LSBs (following the MSBs) of the sample point position the input of the tri-linear interpolation stage. It fills one complete FPGA as well. Output of the interpolation stage are one 8 bit value opacity and three 8 bit gradient values estimated from the neighboring voxels of the sample point [VSC 95].

Shading is done according to Smit [PFI 95] where the complex calculation is essentially performed by a LUT.

In parallel to shading the absorption of the ray with the 8 bit opacity (A) and intensity (I) values is calculated by I=I(1-A).

The compositing step uses the 8 bit result of the bilinear interpolation of the reflection map and the intensity of the rays. It handles all 64 rays by addressing the result RAM and intensities with the ray number.

V. Performance Results

The volume rendering system is currently under construction and should be up and running summer 1999. Before design of the system began it has been simulated in C and VHDL. These simulation results were performed assuming a 100 MHz system.

In the simulation a fully pipelined hardware is assumed. The memory read-out operation consists of three stages and the remaining steps in the volume rendering pipeline are requiring 9 pipeline stages.

For the simulation we used a CT data set from a human jaw with 256×256×128 voxels. This data set is viewed from three different viewing directions and three different levels of opacity for soft tissue is applied (see Fig. 7).

On average one achieves efficiencies of between 90% and 97%. The number of sample points varies between 10-15% of all voxels if the data set consists mainly of empty space and opaque objects and 25-40% for semi transparent opacity levels.

The above results correspond to rendering rates from 20 Hz on semi-transparent data sets to 138 Hz. The results are achieved from images of size 256×128. Perspective views reduce the rendering speed by a factor of 2.

These frame rates have to be compared with those achievable on standard PCs. Rendering rates on high-end PCs are in the range of one second to about 10 seconds for the same sort of data. A 100 MHz ASIC achieves a speedup of a factor > 100 while FPGA systems are currently limited to a factor >50.

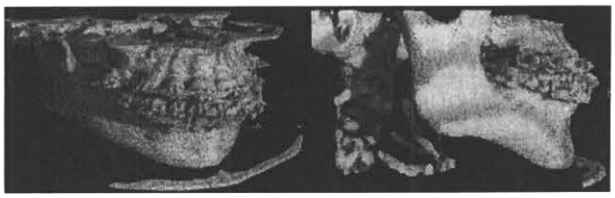

Fig. 7 The images show different views on the jaw data set if the surface is rendered opaque.

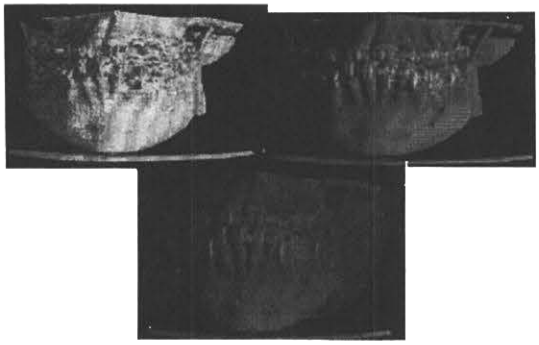

Fig. 8 The images show the frontal view with different levels of opacities of the boundary between bone and the surrounding area. The last image uses semi-transparent soft-tissue.

The main reason why FPGA systems perform such well compared to ASICs is that the performance bottleneck is the memory speed which is limited to 100 MHz to most devices. Recent SDRAMs achieve 150MHz operation frequency, RDRAMs even allow data read out rates of up to 500 MHz so that higher performance of ASIC solutions could be assumed.

The performance of this system is difficult to compare to other approaches in literature. The main reason is that no other hardware uses both techniques of algorithmic optimizations; and (besides the VIRIM system in 1995 [GUE 94]) only one hardware system has actually been built. This system is the VolumePro board that persuades rendering rates of 30 Hz for 256^3 data sets (www.3dvolumegraphics.com).

VI. Conclusions and Outlook

The ATLANTIS computing board turned out to be a versatile hardware for the volume rendering application. It allows to integrate the full board complexity on a few FPGAs only. Free pins on the board allow integrating a special purpose subboard that contains the volume memory. Finally, the reconfigurability gives us the possibility to minimize the necessary amount of hardware by reloading the configuration for different computing tasks that occur during rendering.

VII. References

[LEV 88] Levoy M. Display of Surfaces from Volume Data. IEEE Computer Graphics & Appl., 8(5), 1988: 29-37

[GUE 94] Günther T., Poliwoda C., Reinhart C., Hesser J., Männer R., Meinzer H.-P., Baur H.-J.. VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine. W. Straßer, 9th Eurographics Workshop on Graphics Hardware, Oslo, Norway, 1994: 103-108

[BRA 97] Brady M., Jung K., Nguyen H.T., Nguyen T. Two-Phase Perspective Ray Casting for Interactive Volume Navigation. Proc. Visualization'97, Phoenix, AZ, 1997: 183-189

[KNI 94] Knittel G., Straßer W. A Compact Volume Rendering Accelerator. 1994 Symp. on Vol. Vis., ACP press., NY, 1994: 67-74

[LAC 94] Lacroute P. and Levoy M. Fast Volume Rendering Using a Shear-Warp factorization of the Viewing Transform. Computer Graphics, Proc. of SIGGRAPH '94, Orlando, FL, 1994: 451-457

[PFI 95] Pfister H.-P. Towards a Scalable Architecture for Real-Time Volume Rendering 10th Eurographics Workshop on Graphics Hardware, Maastricht, The Netherlands, 1995: 123-130

[VSC 95] Van Scheltinga J.T., Smit J., Bosma M. *Design of an On-Chip Reflectance Map.* 10th EuroGraphics Workshop on Graphics Hardware, Maastricht, 1995: 51-55

[CHA 94] Chaudhry G., Li X.. A Case for the Multithread Processor Architecture. Computer Architecture News, 22(4), Sept. 1994

[LIC 95] Lichtermann J. Design of a Fast Voxel Processor for Parallel Volume Visualization. W. Straßer, 1st Eurographics Workshop on Graphics Hardware, Maastricht, The Netherlands, 1995: 83-92

[KNI 95] Knittel G. A PCI-based volume rendering accelerator. W. Straßer,10th Eurographics Workshop on Graphics Hardware, Maastricht, The Netherlands, pp. 73-82

[ZUI 92] Zuiderveld K.J., Koning A.H., Viergever M.A. Acceleration of Ray-Casting using 3D Distance Transforms. Proc. Vis. in Biomed. Comp., Chapel Hill, 1992: 324-335

[FOL 90] Foley, van Dam, Feiner, Hughes. Computer Graphics: Principles and Practice. Addison Wesley, Reading, MA, 2d. ed., 1990

[KUG 99] A. Kugel, H, Singpiel, J Hesser., R. Männer. ATLANTIS: A hybrid approach combining the power of FPGA and RISC processor based on CompactPCI in Proc. FPGA `99, Monterey, CA, 1999.