VIZARD - Visualization Accelerator for Realtime Display

Günter Knittel, Wolfgang Straßer
WSI/GRIS, University of Tübingen, Germany[†]

ABSTRACT

Volume rendering has traditionally been an application for supercomputers, workstation networks or expensive special-purpose hardware. In contrast, this report shows how far we have reached using the other extreme: the low-end PC platform. We have alleviated the mismatch between this demanding application and the limited computational resources of a PC in three ways:

- several stages in the visualization pipeline are placed into a preprocessing step,
- the volume rendering algorithm was optimized using a special data compression scheme and
- the algorithm has been implemented in hardware as a PCI-compatible coprocessor (*VIZARD*).

These methods give us a frame rate of up to 10Hz for 256^3 data sets at an acceptable image quality, although the accelerator prototype was built using relatively slow FPGA-technology.

In a low-cost environment a coprocessor must not be more expensive than the host itself, and so VIZARD was designed to be manufacturable for a few hundred dollars. The special data compression scheme allows the data set to be placed into the main memory of the PC and eliminates the need for an expensive, separate volume memory.

The entire visualization system consists of a portable PC with two built-in accelerator boards. Despite its small size, the system provides perspective raycasting for realtime walk-throughs. Additional features include stereoscopic viewing using shutter glasses and volume animation.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture - Graphics Processors; I.3.3 [Computer Graphics]: Picture/Image Generation.

Additional Keywords and Phrases: Volume Rendering Accelerator, PCI-Coprocessor

†Universität Tübingen
Wilhelm-Schickard-Institut für Informatik -
Graphisch-Interaktive Systeme (WSI / GRIS)
Auf der Morgenstelle 10, C9
D-72076 Tübingen, Germany
Phone: ..49 7071 29 76356, FAX: ..49 7071 29 5466
email: [knittel,strasser]@gris.uni-tuebingen.de
web: http://www.gris.uni-tuebingen.de

1 MOTIVATION

Currently we can observe a number of research projects dealing with the design of volume rendering accelerators: VIRIM [7] is already operational since some years, as well as the massively parallel system described in [10]. DIVVA [11] is currently being assembled and tested.

Common to all approaches is that they represent large and expensive *coprocessing systems* with a separate, highly interleaved volume memory and complex arithmetic units. The controlling workstation is merely considered as an intelligent display, which passes user inputs to the deskside accelerator. An exception is Cube-4 [12], since it is being designed as a PCI-card. However, it still has an own volume memory and will fall outside the typical PC price range.

Considering the current trend in the area of surface-oriented graphics accelerators, we follow a radically different approach. Volume data sets are extremely large, and thus the user acceptance for a dedicated memory of this size is limited. Moreover, workstations and PCs are shipped with more and more memory, up into the gigabyte range, and so it is no longer understandable why these enormous resources should not be used for volume rendering also.

The volume rendering accelerator presented in this work is a step towards the ultimate goal of a single-chip visualization coprocessor. The central problem to be solved is the limited main memory bandwidth, which we try to solve using data compression and specialized caches. The achieved performance and the image quality are encouraging, although this project is still in its prototype stage.

The paper is organized as follows: in section 2, the underlying algorithm is explained in short terms. The hardware architecture is explained bottom-up from the coprocessor architecture to our *BlackMagic* visualization system in section 3 through section 7. Operational principles of the system are explained in section 8 and section 9. Performance figures are given in section 10. Image quality is illustrated at the end of the paper by some examples from medical imaging.

2 ALGORITHM

The underlying visualization paradigm is *perspective raycasting*, although this is one of the most expensive algorithms, in order to allow for proper walk-throughs and stereoscopic viewing. Transparent display of selected materials also adds to the complexity of the algorithm, but can in no way be omitted.

The overall processing from the raw data set to the image is shown in Figure 1. Segmentation, shading and our special way of data set compression are done in software once per data set. The transformed data set can then be visualized in realtime with hardware acceleration.

At the first glance it may be considered as a disadvantage to move the segmentation and shading steps out of the interactive loop. However, a reliable classification in standard applications (e.g., separating a tumor from healthy brain tissue) is far too complicated and expensive to be done in realtime during the visualization. Therefore, in most relevant cases, segmentation will be done separately anyway.

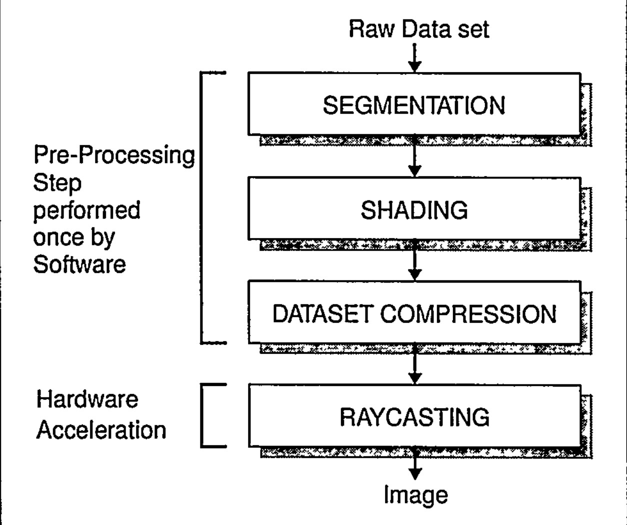


Figure 1: The Rendering Pipeline

The consequences of pre-shading are that the light sources travel with the data set as it is moved, and cannot be brought to other positions. Thus, the data set is viewed like a work of art at an exhibition. An advantage is, that the number of light sources and the complexity of the illumination model are not limited by hardware considerations.

The special data set compression scheme, which we call ***RBC*** (Redundant Block Compression), reduces the memory bandwidth requirements and therefore allows the data set to be placed in the main memory of the host. It is a 3D-extension of the well-known ***BTC*** (Block Truncation Coding) scheme, which was invented in 1979 for grayscale 2D-images [5]. RBC has also been described comprehensively in [8]. Here we give only a short summary. VIZARD currently supports only monochrome data sets, and so we restrict our discussion to the processing of grayvalues.

The grayvalues of a 12-voxel-block are quantized into 2 new grayvalues C_l and C_u such that losses are minimized. Each voxel position is assigned one decision bit to select one of the grayvalues. Given 8 bits for both of the new grayvalues, a 12-voxel-block can be compressed into a 32-bit word, as shown in Figure 2. The

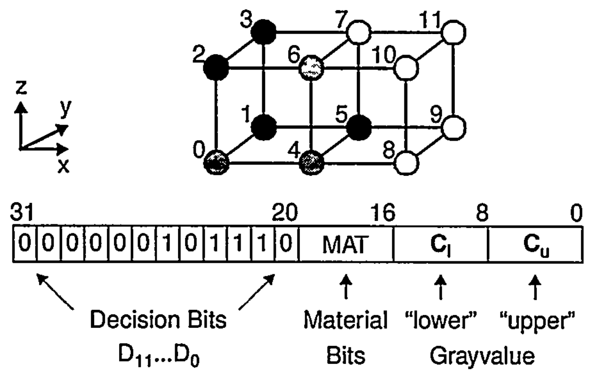


Figure 2: Data Element representing one Voxel Block

remaining 4 bits can be used to identify the different materials inside the block. This compression is done redundantly, as depicted in Figure 3 (not shown but also done in y-direction). Consequently, all eight samples needed for the tri-linear interpolation can always be fetched from memory in a single access.

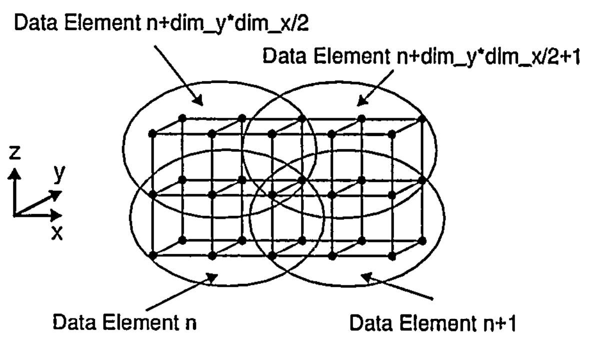


Figure 3: Redundant Block Compression

Besides the memory bandwidth requirements, computational expenses are also significantly reduced. The tri-linear interpolation of the raypoint value C at offsets α, β and γ from eight voxels $C_{0..7}$, given by

$$C = C_0(1-\alpha)\,(1-\beta)\,(1-\gamma) + C_1(1-\alpha)\,\beta\,(1-\gamma) + C_2(1-\alpha)\,(1-\beta)\gamma + \ldots + C_7\cdot\alpha\cdot\beta\cdot\gamma \qquad (1)$$

can be factorized as shown below, since there are only two different grayvalues C_u and C_l in any given volume cell:

$$C = C_u\cdot(\omega_a + \omega_b + \ldots + \omega_c) + C_l\cdot(\omega_d + \omega_e + \ldots + \omega_f) \qquad (2)$$

The weightfactors ω_n sum up to 1. If ω_l is the compound weight for C_l, then

$$C = C_u\cdot(1-\omega_l) + C_l\cdot\omega_l = C_u - \omega_l\cdot(C_u - C_l)\ . \qquad (3)$$

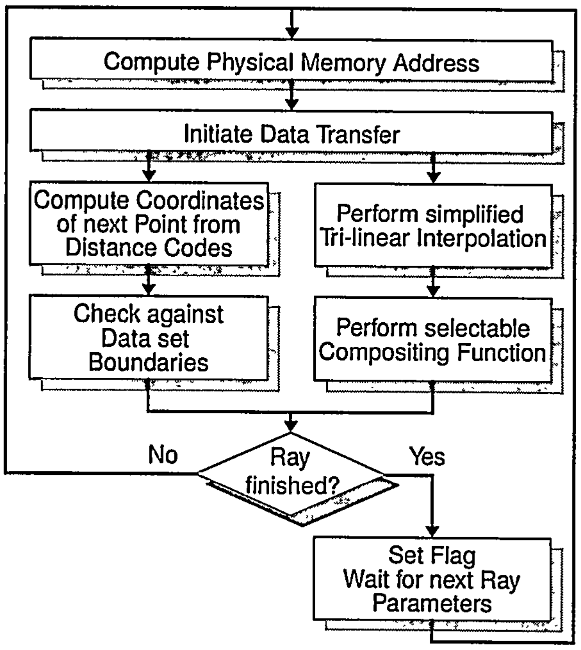
The compound weight ω_l depends on α, β and γ (which we limit to 4 bit precision each), and on 8 decision bits, giving a total of $1M = 2^{20}$ different configurations. Thus we can easily precompute the weightfactors for each possible configuration and store them in a table. Furthermore, as implied by (3), we do not store C_u and C_l in the data elements, but instead C_u and (C_u-C_l). Then, a complete tri-linear interpolation is performed by

- ❑ assembling the weightfactor address from the decision vector and the offsets,
- ❑ one table look-up and
- ❑ one multiplication and one subtraction.

A further, significant speedup is achieved by integrating a special *distance coding* into the data set. If a voxel block is empty, i.e., $C_u=0$, the remaining bits of the data element are redefined and contain information about the neighborhood of the block. There is a certain probability that parts of this neighborhood are empty as well, which can then be skipped during raycasting [8].

3 COPROCESSOR FUNCTIONALITY

The coprocessor autonomously traces a given ray through the data set until volume exit or encounter with an opaque surface. After being set up with the coordinates of the first raypoint and the vector to the next, it starts processing as shown in Figure 4. Accordingly, the coprocessor has one address-pipeline and one data-pipeline. For the simplified tri-linear interpolation as explained above, the coprocessor is accompanied by a PROM holding the weightfactors in 8-bit precision. The weightfactor address is assembled as
$D_{7..0}\gamma_{-1..-4}\beta_{-1..-4}\alpha_{-1..-4}$ if $X_0=0$, or $D_{11..4}\gamma_{-1..-4}\beta_{-1..-4}\alpha_{-1..-4}$ if $X_0=1$.



Compute Physical Memory Address
Initiate Data Transfer
Compute Coordinates of next Point from Distance Codes
Perform simplified Tri-linear Interpolation
Check against Data set Boundaries
Perform selectable Compositing Function
No
Ray finished?
Yes
Set Flag Wait for next Ray Parameters

Figure 4: Coprocessor Operation

The only large arithmetic unit is an 8×8 into 8 bits multiplier, so that the entire functionality fits into a single FPGA (XC4013 from XILINX [1]).

Due to the distance coding, there is a certain idle time between the receipt of a data element and the initiation of the next transfer. For this reason, the accelerator board has two coprocessors following two different rays in parallel.

4 MEMORY HIERARCHY

For further speedup, we use on-board and on-chip caches, which in combination with the main memory and the harddisks form a four-level memory hierarchy.

The atomic transfer unit is a 2×2×2 voxel block, defined by four data elements as shown in Figure 5. Whenever a memory access

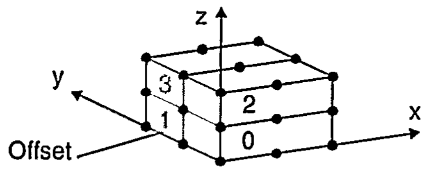


Figure 5: Atomic Transfer Unit

occurs, four data items are read out and stored in the on-board and on-chip cache. Accordingly, the on-chip cache has four 32-bit entries and is addressed by Z_0Y_0 of the raypoint coordinates.

The hit ratio of the on-chip cache can be considerable, if the distance from one raypoint to the next is very small. This in turn is desirable in favor of a high image quality.

The task of the on-board cache is to hold all blocks a ray has traversed, as shown in Figure 6. If the next ray passes through the neighboring pixel, it will hit a certain percentage of blocks then already present in the cache (ray-by-ray coherence). However, a

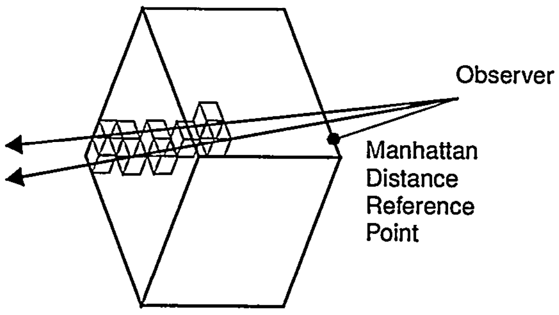


Figure 6: Blocks to be cached

standard direct-mapped or n-way-associative cache organization would lead to an unacceptable cache size. For this reason, a specialized cache architecture has been developed. The cache is addressed by the Manhattan Distance of the voxel block relative to the Manhattan Distance Reference Point, which is the closest point of the volume to the observer. Given 2×2×2 cache blocks and a data set resolution of 256^3, the maximum Manhattan Distance is 384, and thus the cache has 384 entries. Each entry consists of the physical memory offset of the cache block (the cache tag), which is 21 bits wide, and the 128-bit voxel block. This would result in a total cache capacity of only about 7KByte.

However, it is desirable to have a larger cache capacity for an increased hit rate. The most economic way to do this is to increase the cache line size to 4×4×4 voxel blocks, or 32 data elements. Then, we have only 192 cache entries, and an 18-bit wide tag. However, the cache is still filled in units of 4 data elements, so that we have to provide additional 8 valid bits. This gives a capacity of 4992 bits for the tag RAM, and 24KByte for the cache.

This tiny system is able to cache a data set of 32MByte and still deliver a very high hit ratio [9].

The operational principle of the cache tag architecture is shown in the block diagram in Figure 7.

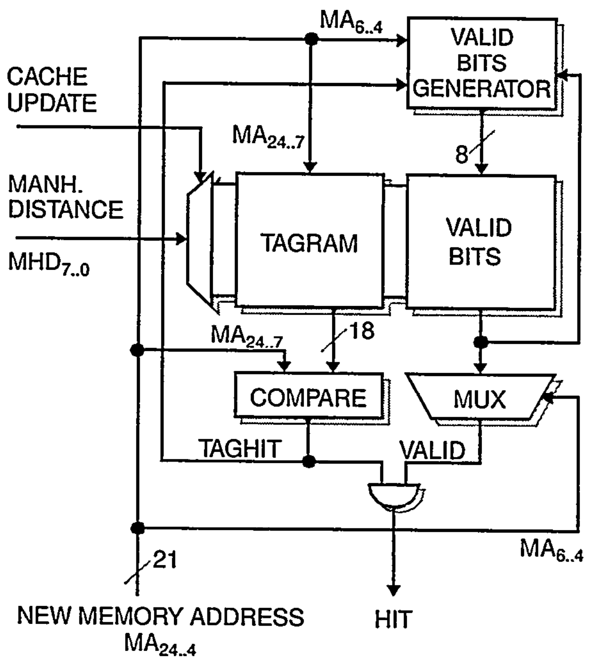


Figure 7: On-Board Cache Tag Circuitry

Both coprocessors compute the Manhattan Distance along with the logical coordinates of the raypoints, and have a private cache tag system as shown in Figure 7. The tag systems are implemented in a third XC4013 device, making use of the Distributed Memory Feature of the XILINX 4K-family [1]. The 4013-device offers up to 18.432 bits storage capacity.

If a hit occurs, an external 32-bit high-speed SRAM is addressed by $MHD_{7..0}\, MA_{6..4}$ to deliver the appropriate voxel block during four on-board transfers. For maximum performance, the data element containing the raypoint is transferred first.

For a high hit rate, a strong ray-by-ray coherence is needed. Therefore, rays are not generated in scanline order, but instead in the order of screen squares.

5 ACCELERATOR ARCHITECTURE

The missing part of a complete accelerator system is the data transfer controller, which is implemented using an XC3195A FPGA. It incorporates a dual-channel PCI master/target interface for burst transfers and controls all on-board activities. An atomic transfer unit can be read from main memory without wait states, since the transfer controller has a 4×32-bit register pipeline.

The block diagram of the accelerator is given in Figure 8. A photography of the board is shown in Figure 9. The different units discussed so far can easily be identified:

- the two completely identical coprocessor units, each consisting of one 4013 device (unit A or unit B), accompanied by two Flash Memory chips,
- the third 4013 FPGA (unit C), which incorporates the cache tag systems and controls the high-speed SRAM, and
- the PCI interface and system controller (unit P).

For future use there is a high-speed multiply-and-accumulate (MAC) unit, which can perform a 16×16 into 32 bit multiplication with subsequent accumulation within 25ns.

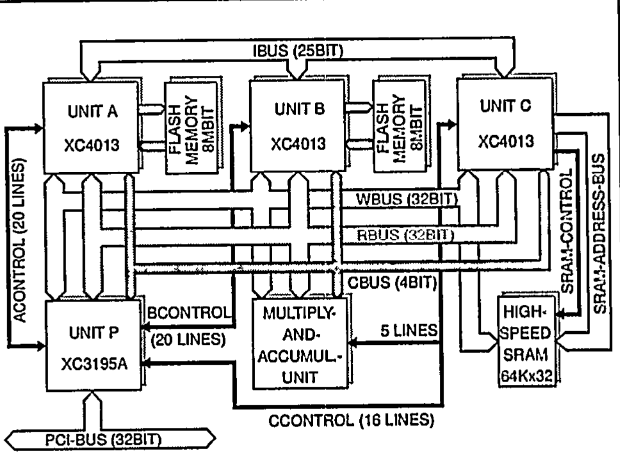


Figure 8: Accelerator Block Diagram

6 SYSTEM ARCHITECTURE

The host system can be any standard PC having a PCI-bus. Since the data set must fit entirely into the main memory, the PC must be equipped with 32MByte for 256^3 data sets. A Pentium-CPU running at 200MHz is sufficient for the software part of the visualization process. In the current implementation, MMX would not increase the frame rate significantly.

The system architecture of a typical PCI-based PC is shown in Figure 10. In our system, control functions are carried out by the TXC-unit (cache and memory control, host-to-PCI-bridge) and the PIIX3-unit (ISA-bridge, disk interface), both manufactured by Intel. For detailed information about the PCI-bus and the system control units, please refer to [2], [3] and [4].

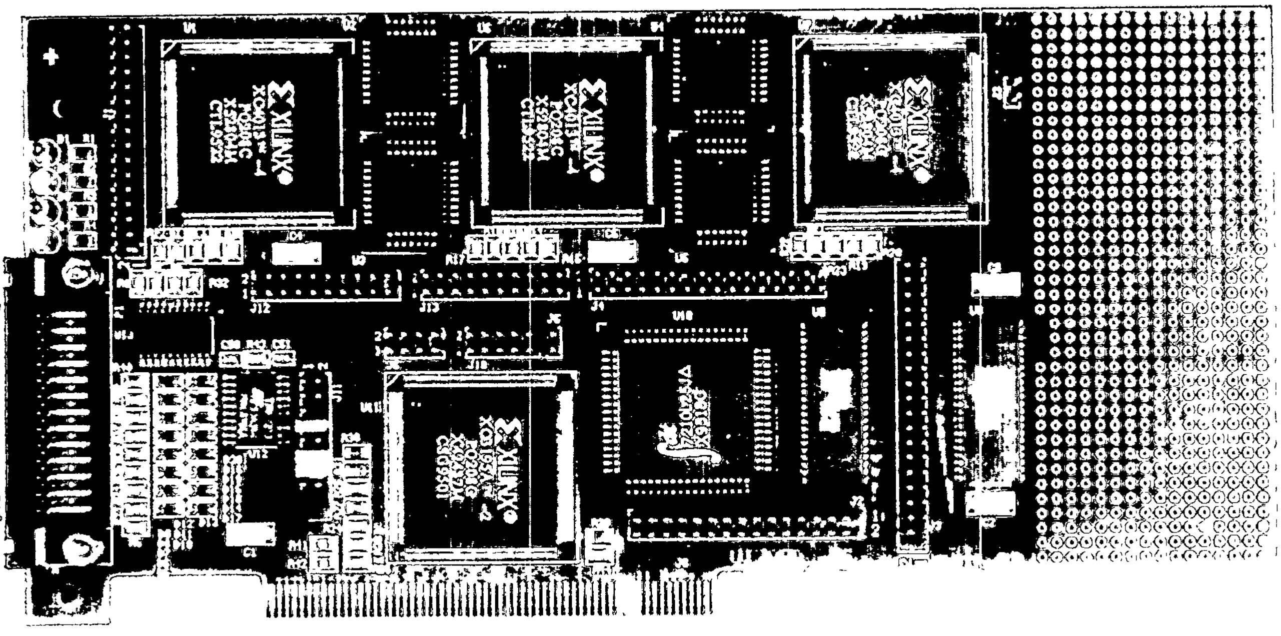


Figure 9: VIZARD

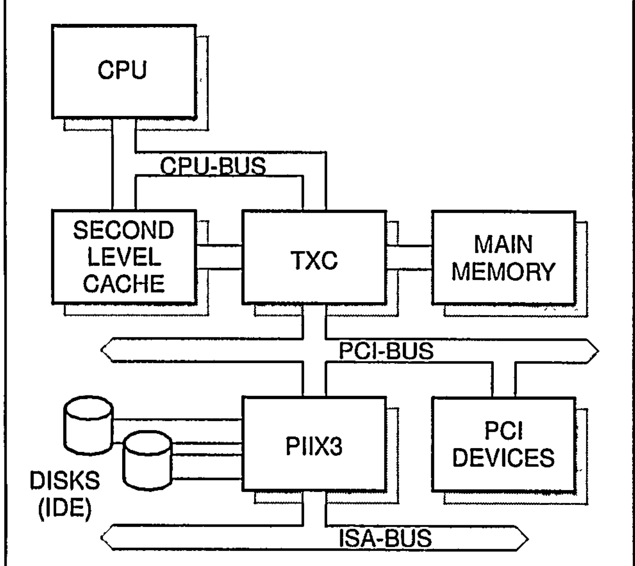


Figure 10: PC System Architecture

7 BLACKMAGIC VISUALIZATION SYSTEM

Compactness has been driven to the extreme for our BlackMagic demonstration system. It is a portable PC with a standard PCI mainboard, a 200MHz Pentium-CPU, 512KByte second level cache, 128MByte main memory, two 2.5GByte harddisks and two VIZARD accelerator boards. A VGA-adapter, which can drive both the built-in LCD and an external monitor simultaneously, completes the PC. The graphics adapter provides double-buffering and even fourfold-buffering (for stereoscopic viewing) for screen resolutions of up to 640×400×8bit.
A photography of the BlackMagic system together with a collection of 3D input devices is shown in Figure 11.

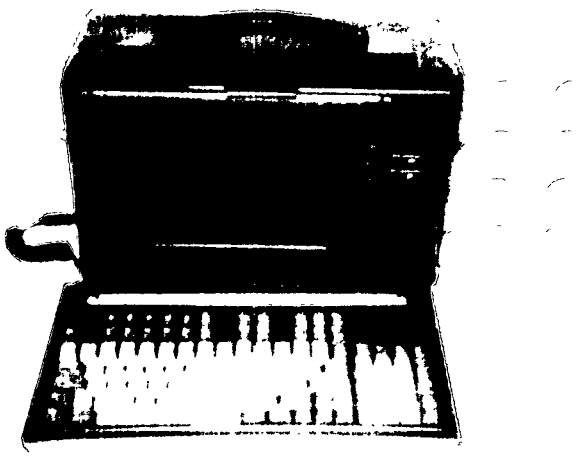


Figure 11: The BlackMagic Visualization System

8 PARALLEL OPERATION

At program start the data set is loaded into memory and the coprocessors are set up with the starting address and bounding volume of the data set. The two accelerator boards work in parallel, providing a total of four raycasting engines. An even workload is established by assigning screen tiles of 32×32 pixels to the different coprocessors.
For maximum performance, all system components must work in parallel. The system architecture implies that the software part should entirely fit into the CPU's second level cache. This has been achieved by optimizing the critical parts in assembly language. Since the CPU never accesses the data set itself, perfect parallelism can be established.
The software performs ray generation and intersection tests in floating-point format. If a ray hits the volume, the coordinates of the intersection point and the components of the vector to the next raypoint are converted into integer format and stored along with other visualization parameters in main memory. The appropriate data transfer controller (unit P) is triggered, which transfers the parameter block to the targeted visualization coprocessor. This unit in turn starts raycasting as explained in section 3. Meanwhile the CPU generates the next ray. Upon completion, the CPU checks if the last ray has been terminated. If so, it reads the pixel value, which again triggers the appropriate data transfer controller to transfer the next parameter block. If not, the CPU generates a ray for the next coprocessor. After a very short while, all four coprocessors and the CPU are working in parallel.
After image completion, the CPU optionally performs 2D operations (e.g., bi-linear interpolation in the case of subsampling), transfers the image to the frame buffer on the VGA card in a block transfer, and switches the display buffers.

9 VOLUME ANIMATION

By volume animation we mean the display of a sequence of data sets. However, due to the lack of a high-speed I/O-interface in the PC architecture, we assume that the data sets are already stored on the local disks.
As long as the sequence fits entirely in the main memory, changing from one data set to the next just means changing a pointer. Since the main memory usually is much larger and can more easily be upgraded than a specialized volume memory, this approach has a clear advantage for this kind of applications. For example, a main memory capacity of 1GByte will be common in the near future, giving room for more than 30 data sets of 256^3 16-bit voxels.
However, the PC architecture does not yet support such a large main memory, and so the question was examined at which rate the data sets could be swapped in from harddisks during visualization. As can be seen in Figure 10, the PIIX3 can control two IDE channels in parallel. IDE defines a practically no-cost harddisk interface and is the low-end counterpart to SCSI. The disks attached are two Seagate ST52520A drives with an individual peak transfer rate of 8.5MByte/s from the outermost tracks.
The PIIX3 provides an independent DMA channel for each IDE disk, and one 32-Byte buffer. This allows burst-mode write accesses to main memory via the PCI-bus to proceed at peak transfer rates.
The largest block of data the disk can deliver without CPU intervention is 16KBytes. Then an interrupt is generated, and the disk must be set up for the next block read.
The smallest unit of allocation on a disk is called a *cluster*, in our case containing 32KByte. All clusters are numbered. Cluster numbers are 16 bits wide. The clusters of a file are organized as a linked list.
At program start, a cluster list for the entire data set sequence is set up in main memory for both drives. This eliminates excessive head movements during the animation. As an example, the cluster list would occupy 256KByte for a sequence of 4GByte.
Two memory regions are defined, one for the data set being ren-

dered, and one for the data set being loaded. After having loaded the first data set, DMA and interrupt structure are set up such that all subsequent data sets are loaded concurrently to the visualization with little CPU overhead.
Whenever the system has finished a frame, it checks if there is a new data set present. If so, it exchanges the data set pointers, and renders the new data set using any new user inputs. In the opposite case, user inputs are used to display the old data set.
Clearly, the system cannot load large data sets at a realtime animation rate. For evaluation purposes, we used 100 timesteps from an astrophysical simulation. A gas eruption on a rotating sun, which leads to accretion disks, has been simulated using *Smoothed Particle Hydrodynamics* [6]. The results of each timestep were sampled on a 256×256×32 grid, giving 4MByte for each data set. The first half of each data set has been written on the one disk, and the second on the other, starting on the outermost track of each disk.
The entire sequence of 100 timesteps is loaded from disk and visualized within 48 seconds. This gives a sustained animation rate of 2.1 data sets per second. Thus, the disk system can provide a sustained data stream of about 8.3MByte/s into the main memory, which at the same time is frequently accessed by the visualization coprocessors.

10 PERFORMANCE AND IMAGE QUALITY

The performance figures given below have been measured in the running system using a logic analyzer, or have been derived from our design. The PCI-bus clock runs at 33MHz, which is also the clock for the accelerators.
On average, a 200MHz Pentium-CPU can generate the parameters for one ray within 4μs, resulting in a generation rate of about 4Hz for 256×256 rays.
Transferring 512×400 pixels from memory to the frame buffer takes 5ms.
The PCI-bus can transfer more than 85MBytes per second. However, due to the relatively short bursts of four transfers per memory access (atomic transfer unit), memory latency reduces the data rate significantly. A four-word-burst takes 12 cycles of 30ns, giving a peak data rate of 44MByte/s.
A coprocessor can accept data elements every 150ns, giving a maximum performance of 6.67M raypoints per second if the data is available in the on-chip caches. This gives a system peak performance of 26.7M raypoints per second.
A transfer from the on-board cache into the on-chip cache takes 15 cycles, resulting in a rate of 2.2M raypoints per second per accelerator.
Finally, if misses occur in both caches, two consecutive memory accesses are separated by 31 clocks, giving a worst-case rate of about 1M raypoints per second per system.
The application performance was evaluated using a CT-stack of a human head containing 256×256×222 voxels. The achievable image quality is illustrated by several images in Figure 12.
For the images on the right side, 256×200 rays have been shot through the data set. The images on the left side have been created using fourfold subsampling, i.e., by sending 128×100 rays through the volume. In any case, the images are bi-linearly interpolated (in software) to a final screen resolution of 512×400 pixels. The system uses subsampling during motion, and switches to the normal resolution as soon as motion has stopped.
For images a) and b) the skin was set to opaque. Therefore, we can achieve high frame rates by means of the distance coding and early-ray-termination.
The skin surface was set to translucent, and the bones were set to opaque in images c) and d). All interior tissue was discarded. The frame rate drops because rays can only be terminated after encounter with the bone surface, and the distance coding does not apply to material which is discarded during rendering.

Finally, the bones were set to translucent, and the intensities of all raypoints within bone have been accumulated to give the X-ray-like images e) and f). In terms of frame rates this represents the worst-case, since each and every ray has to go through the entire data set.

The frame rates, which have been measured by rotating the data set around the z-axis, are given in Table 1.

Image	No. of Rays	Frame Rate [Hz]
a)	128×100	9.4
b)	256×200	3
c)	128×100	3.3
d)	256×200	1.1
e)	128×100	1.5
f)	256×200	0.5

Table 1: Frame Rates

The preprocessing step of the visualization pipeline is currently implemented in C for UNIX workstations. However, it has never been optimized and is still in its debug version. On an SGI-Indy running at 100MHz the preprocessing takes between 15 and 30 minutes. Future workpackages therefore include the implementation of all preprocessing steps in hardware, making use of the in-system-programmability of the FPGA devices. In this way, the preprocessing time could potentially be brought into the range of seconds.

11 PERFORMANCE IMPROVEMENTS

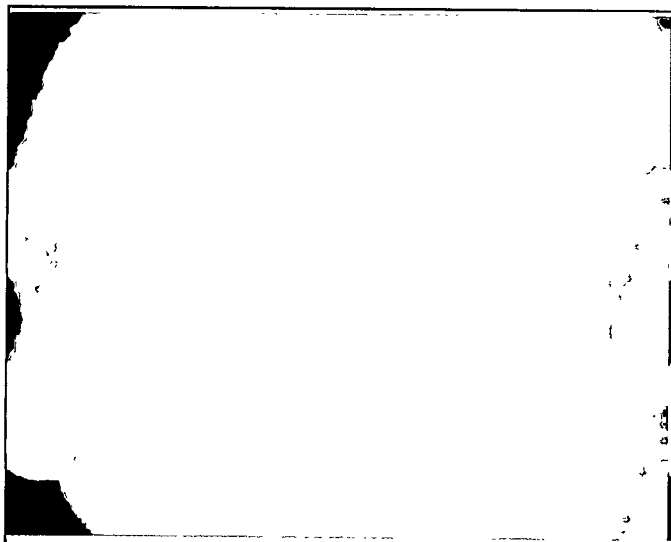
The limiting factors are the FPGA-technology, the PCI-bus and the CPU (in that order). Architectural improvements, however, should also be made.

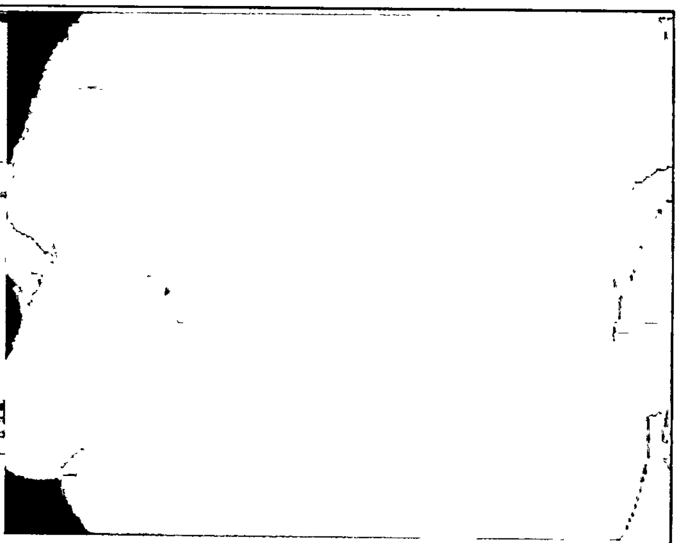
If we could use ASIC-technology, we could implement larger on-chip caches and complete one raypoint every clock, possibly at 66MHz or more. Also, the burst length of the PCI-transfers could be extended to 32 data elements by means of larger FIFOs. On the architectural side, it is a clear performance bottleneck that two coprocessors share one on-board (second level) cache. A redesign should provide private second level caches for each ray casting engine (possibly also on-chip).

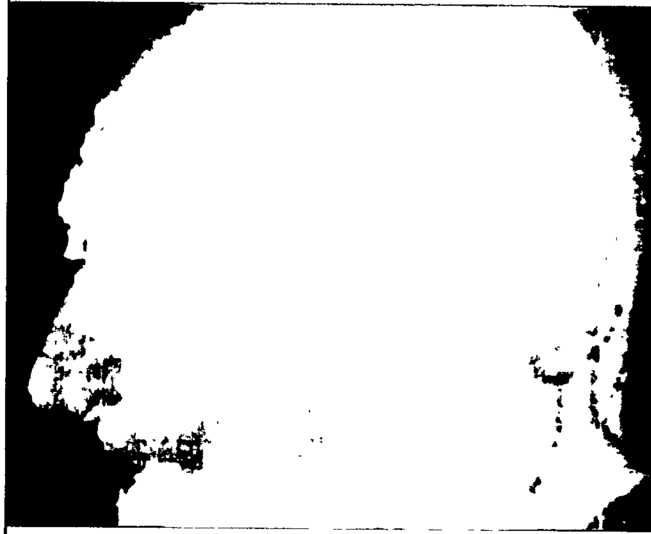
A higher transfer bandwidth towards the main memory will be provided by the upcoming *Accelerated Graphics Port (AGP)*, which will have a peak transfer rate of 528MByte/s.

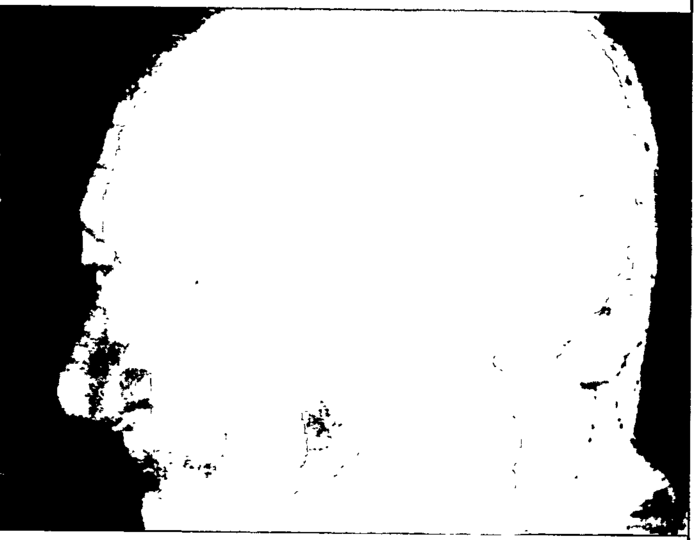
Using a Pentium-II-class CPU should also yield a significant performance increase, since the geometric computations involve a large number of floating-point calculations.

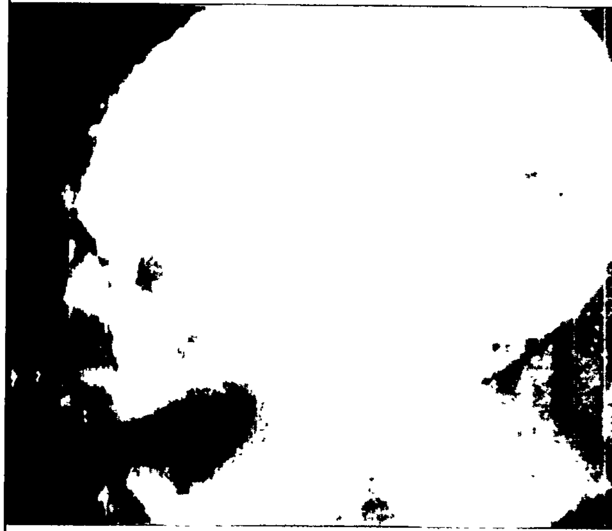
A performance increase by a factor of two for an ASIC-solution of the coprocessor, a factor of two for the AGP over the PCI-bus and a factor of 1.5 for a Pentium-II over its predecessor can be considered a pessimistic estimation. In this case, the system would run at twice the speed, giving frame rates of 18Hz and 6Hz for images a) and b), respectively. However, one should keep in mind that using VLSI-technology, this performance could be provided by a single-chip coprocessor.


a)


b)


c)


d)


e)

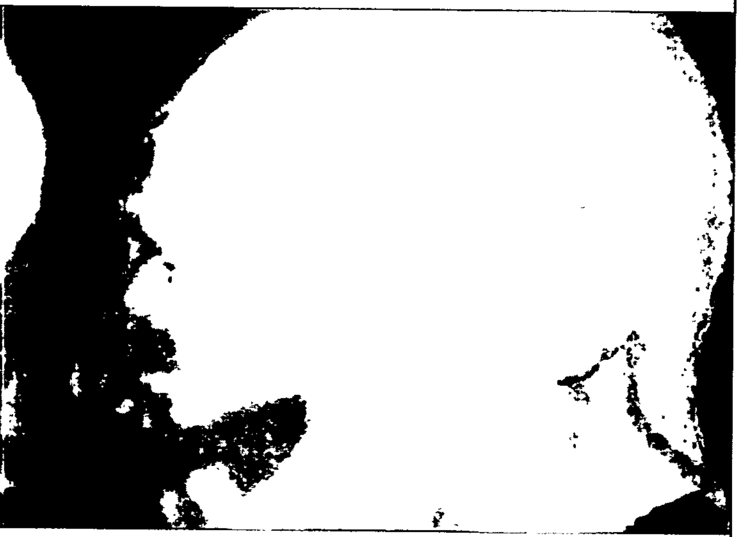

f)

Figure 12: Examples from Medical Imaging. Left Column: 128×100 Rays. Right Column: 256×200 Rays

12 CONCLUSIONS AND OUTLOOK

We have presented probably the world's most compact volume rendering accelerator existing today. Nevertheless, performance and image quality are highly competitive, although the underlying algorithm uses lossy data compression. We lay special emphasis on the fact that this architecture does not need a specialized volume memory, as opposed to all other existing or academic designs. The evolution of surface-oriented graphics accelerators tells us that this will be the prerequisite for a broad market acceptance. Using ASIC-technology, we could not only increase rendering speed, but also implement this architecture into a single chip.

However, using lossy data compression is not acceptable in some applications. Thus, our future research activities will be directed towards lossless compression schemes, which still allow the decompression units to be fast and compact.

13 ACKNOWLEDGMENTS

This work was done for the research project SFB 382, funded by the German Science Foundation DFG. The SPH-simulations have been done at the Theoretical Astrophysics Institute of the University of Tübingen. Thanks to R. Speith for his cooperation and to my colleague René Rau for preparing the SPH-data sets. Special thanks to Andreas Schilling for his never ending source of inspiration.

14 REFERENCES

[1] **Anonymous,** *"The Programmable Logic Data Book"*, XILINX Inc., San Jose, CA, 1994

[2] **Anonymous,** *"PCI Local Bus Specification, Rev. 2.1"*, PCI Special Interest Group, PO Box 14070, Portland, OR 97214, April 1993

[3] **Anonymous,** *"Intel 430HX PCISET 82439HX System Controller (TXC)"*, Intel Corporation, P.O. Box 58119, Santa Clara, CA, 1996

[4] **Anonymous,** *"82371FB (PIIX) and 82371SB (PIIX3) PCI ISA IDE Xcelerator"*, Intel Corporation, P.O. Box 58119, Santa Clara, CA, 1996

[5] **E. J. Delp and O. R. Mitchell,** *"Image Compression Using Block Truncation Coding"*, IEEE Transactions on Communications, Vol. COM-27, No. 9, Sept. 1979, pages 1335-1342

[6] **O. Flebbe, S. Münzel, H. Herold, H. Riffert and H. Ruder,** *"Smoothed Particle Hydrodynamics: Physical Viscosity and the Simulation of Accretion Disks"*, Astrophys. Journal 431, 1994, pages 754-760

[7] **T. Günther, C. Poliwoda, C. Reinhart, J. Hesser, R. Männer, H.-P. Meinzer, H.-J. Baur,** *"VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine"*, Proceedings of the 9. Eurographics Hardware Workshop, Oslo, September 12-13, 1994, pages 103-108

[8] **G. Knittel,** *"High-Speed Volume Rendering Using Redundant Block Compression"*, Proceedings of the 1995 IEEE Visualization Conference, Atlanta, GA, Oct. 30 - Nov. 3, 1995, pages 176-183

[9] **G. Knittel,** *"A PCI-based Volume Rendering Accelerator"*, Proceedings of the 10th Eurographics Workshop on Graphics Hardware '95, Maastricht, NL, August 28-29, 1995, pages 73-82

[10] **A. Krikelis,** *"A Modular Massively Parallel Processor for Volumetric Visualisation Processing"*, Proceedings of the Workshop on High Performance Computing for Computer Graphics and Visualisation, Swansea, UK, July 3-4, 1995, Springer, pages 101-124

[11] **J. Lichtermann,** *"Design of a Fast Voxel Processor for Parallel Volume Visualization"*, Proceedings of the 10. Eurographics Hardware Workshop, Maastricht, NL, August 28 - 29, 1995, pages 83-92

[12] **H. Pfister and A. Kaufman,** *"Cube-4 - A Scalable Architecture for Real-Time Volume Rendering"*, Proceedings of the 1996 Symposium on Volume Visualization, San Francisco, CA, October 28 - 29, 1996, pages 47-54