A Compact Volume Rendering Accelerator

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

Abstract

We describe the architecture of a hardware accelerator for volume rendering. The system basically consists of four VLSI chips and the volume memory and represents a single-board solution to the computational challenges of volume visualization. It generates arbitrary perspective projections, so that walk-through examinations are possible. The classification of structures of interest is an integral part of the rendering pipeline. Image quality is enhanced by providing Phong shading, depth-cueing and support for translucency. Despite its compactness and algorithmic complexity, the system is able to render 256^3 data sets at a sustained frame generation rate of about 2.5Hz.

CR Categories and Subject Descriptors: I.3.1 [**Computer Graphics**]: Hardware Architecture - *graphics processors*; I.3.3 [**Computer Graphics**]: Picture/Image Generation - *display algorithms*

Additional Keywords and Phrases: graphics hardware, VLSI designs, volume rendering, ray casting

1 Introduction

Volume rendering, i.e. the visualization of scalar functions of three spatial dimensions, can be so computation intensive that even on supercomputers or large workstation networks no satisfactory rendering speed is achievable. Therefore, scientists have begun to develop special purpose hardware systems already in the early 80s [3],[4]. Recent work in this area is mainly directed towards maximum rendering speed with little or no concern of the costs. For the CUBE architecture in its latest version [12], the authors claim to achieve a frame rate of 25Hz for 512^3 data sets. However, the system will fill a complete rack while still requiring tomorrow's technology. Currently under construction is the VIRIM system [2], which uses specialized hardware units for rotation and a DSP farm for shading to provide a frame rate of 10Hz for $256^2 \times 128$ data sets. Although the authors claim to have a scalable architecture, the minimum configuration consists of 4 large boards holding a total of 64 DSPs. Considering the evolution of surface-oriented graphics hardware, it is clear to see that those approaches will not survive. Leaving the experimental stage, any volume rendering architecture will have to face the tough requirements of the market place and the "pizza-box" sized desktop workstation in the user's mind. The increasing economical and social relevance of volume visualization (e.g., in medical diagnosis and non-destructive evaluation) will open up a large number of applications where ultimate speed is not at premium, but instead the availability of a voxel graphics system at each user's workplace.

Exploiting the broad scientific success in algorithmic research and the advances in VLSI-technology, which enable us to construct large memories and fast arithmetic units, we therefore focus our work on the design of an inexpensive, compact and powerful hardware accelerator which can be attached to or plugged into any standard workstation. In our opinion, the requirements for a voxel graphics system can be formulated as follows:

- ❑ interactive to real-time rendering speed,
- ❑ low costs and small size,
- ❑ hardware support for interactive classification,
- ❑ semi-transparent display of structures of interest,
- ❑ arbitrary perspective projections allowing walk-through examinations,
- ❑ freely moveable light source with non-parallel light and
- ❑ meaningful images by sophisticated illumination models, e.g., Phong shading and depth-cueing.

In the following sections we will discuss the underlying algorithms and show how we meet these contradictory requirements. Rendering speed and image quality are illustrated with some examples at the end of the paper.

2 Algorithm

This architectural approach follows the work of Levoy [10] and Drebin et al. [1]. The algorithm proposed by Levoy performs the shading and classification operations on the acquired or prepared samples of the data set. For each grid point, the local gradient is approximated using the samples in the 6-neighborhood. The samples are Phong shaded using the gradient as surface normal. Classification assigns the samples a certain opacity, which is taken from an opacity map using the function value and the gradient magnitude as pointers. The results are two new data sets: one holds the color of the shaded samples, the other their opacity. For the image generation, the ray casting algorithm is performed on

[†]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 5461
FAX: ..49 7071 29 5466
email: [knittel,strasser]@gris.informatik.uni-tuebingen.de

both of the data sets. The reconstruction via tri-linear interpolation at the resample locations along a ray accordingly operates on colors and opacities. The compositing finally sums up the color of all points on a ray in back-to-front order according to their opacity to give the pixel color.

However, the algorithm is not well suited for interactive exploration. Any change in classification or shading parameters requires the data set holding the color or opacity to be reconstructed. For the visualization, however, we have to go through the data set a second time.

A direct hardware realization would show unnecessary high storage costs. Since in-place computation (replacing the operands by the results) is not possible, we would have to provide memory for the original sample values, their color and opacity. Given a data width of 16, 24 and 8 bits, respectively, the memory would have three times the size of the original data set.

Thus, the algorithm is reorganized to allow an easy hardware implementation. The ray-casting algorithm is performed on the original data set. For each resample location, a specific set of neighboring samples is read out, from which the function value, the local gradient and the gradient magnitude are computed. Function value and gradient magnitude are then used as pointers into several look-up tables, which hold the classification transfer function (opacity Ω), the color assignment (*RGB*) and material properties (such as the specular reflection coefficient k_s) for shading. Phong shading is then applied to the resample location, and the intensities of all points on a ray are then composited in front-to-back order according to their opacity.

Thus, the volume memory must hold only one data set. All processing is done on the fly by specialized VLSI units, which can ideally be placed into a pipeline.

Two modes of operation, differing in the way the gradient is approximated, can be used: High-Speed Rendering and High-Quality Rendering.

2.1 High-Speed Rendering

In this mode, the gradient is approximated from the 8 samples needed by the tri-linear interpolation. Figure 1 shows a volume element which is defined by the eight samples $S_0..S_7$ at the corners. The offset of the resample location within the

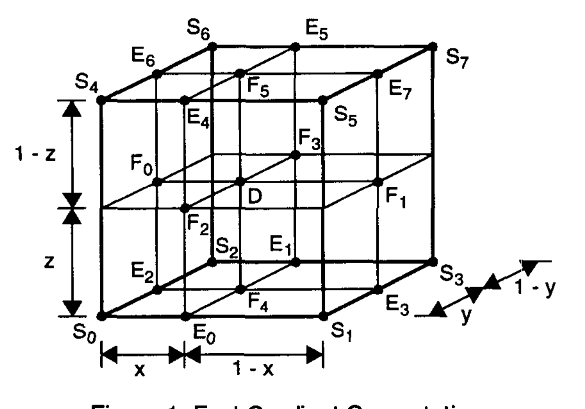

Figure 1: Fast Gradient Computation

volume element is given by *(x,y,z)*. The tri-linear interpolation is decomposed into a sequence of linear interpolations, i.e., first the quantities $E_0..E_7$ are linearly interpolated, from which the quantities $F_0..F_5$ are again computed by linear interpolations. The desired value D and the gradient components G_x, G_y and G_z at that resample location are then approximated by:

$$D = F_4(1-z) + F_5 z \text{ and} \quad (1)$$

$$G_x = F_1 - F_0; \quad G_y = F_3 - F_2; \quad G_z = F_5 - F_4. \quad (2)$$

The memory system (see section 3.1) facilitates the parallel access to the 8 corner samples (called an *S-set*) of any volume element, so that all needed quantities for the processing of a given resample location are extracted from the data set in one single memory access.

2.2 High-Quality Rendering

For a more accurate gradient approximation, the samples in an extended neighborhood are taken into account. Figure 2 shows a volume element with a collection of adjacent samples. The computation of the function value and the gradient

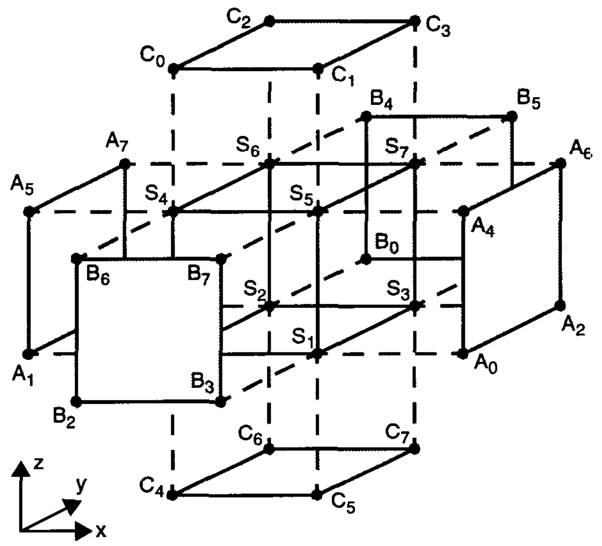

Figure 2: Extended Neighborhood

components now take place in four steps. First the samples labeled *S*, which bound the volume element holding the resample location, are fetched from memory and passed to the tri-linear interpolation. It is obvious that a memory system capable of delivering an *S*-set in parallel can also provide all samples labeled *A* (an *A-set*) in a single access. Then, together with the previously fetched *S-set*, all data is available to compute the x-components of the gradients at the original sample points, i.e.:

$$G_x|_0 = (S_1 - A_1)/2, \quad \ldots \quad G_x|_7 = (A_6 - S_6)/2. \quad (3)$$

The x-components of the gradients are then tri-linearly interpolated at the resample location. The remaining two steps refer to the samples labeled *B* and *C*, and produce the interpolated y- and z-components of the local gradient, respectively.

All together, one resample location requires 4 memory accesses, so that High-Quality Rendering runs at approximately one fourth the speed of the High-Speed mode.

3 Hardware Architecture

The system architecture is a one-to-one mapping of the described algorithmic steps to appropriate hardware units. The accelerator is organized as a pipeline, which, at peak performance, completes the processing of one resample location each clock cycle. An overview of the voxel graphics system, which grew out of our previous work in this area [5],[6], is given in Figure 3.

The Address SeQuencer (ASQ), a VLSI unit, performs the ray-casting algorithm. After having obtained all ray parameters, it subsequently generates all points on that ray, clips against clip planes and volume boundaries, computes the associated memory addresses and schedules the different access types in High-Speed and High-Quality Rendering mode.

The volume memory (VoluMem) is designed to deliver the eight samples of any set in parallel. It uses address interleaving and address pipelining and exploits the Page Mode of standard DRAMs to reach a sustained bandwidth of about 750 MByte/s.

The Reconstructor and EXtractor (REX), a VLSI chip, performs the reconstruction of the scalar function at the resample location via tri-linear interpolation, the gradient approximation in both High-Speed and High-Quality Rendering mode and the gradient magnitude calculation.

Function value D and gradient magnitude G are then passed to several look-up tables: the color (RGB) of a point on a ray is derived from its function value, the visibility (opacity Ω) and the appearance (specular reflection coefficient k_s) of the surface the point lies on are taken from two-dimensional look-up tables addressed by D and G. Both the Ω- and k_s-functions are stored at a lower resolution and are bi-linearly interpolated at the exact position; this allows the use of fast but small memory devices.

The most complex VLSI unit is the Phong Shader. For each resample location, the unrestricted Phong illumination model (non-parallel light, perspective projection) is solved. Depth-cueing is provided according to the travelling distance of the light from the source to the eye. The key idea to reach the single-chip target is to transform the operands into the logarithm at various places in the computing pipeline, thus replacing divisions and multiplications by simple subtractions and additions, and to backtransform the results into the number domain. Provided the logarithm converters are compact and fast, significant gains in chip space can be achieved. Of particular usefulness is the fact that the exponentiation of the specular term can then be done by a multiplication instead of a table-look-up. The basic macrocell, a fast logarithm converter, has been developed and is described in [9].

The Compositing Unit finally sums up the intensity contributions of all points on a ray in front-to-back order, and passes the pixel color to the framebuffer. Basically this VLSI chip is just an arrangement of multiply-and-accumulate pipelines. Additionally, the unit will support the virtual integration of surface-oriented objects into the volume data set.

Here we will focus on the front half of the volume graphics system: ASQ, VoluMem and REX, which, taken together, already represent a very useful voxel graphics subsystem. The "back end" of the accelerator, the Phong Shader and the Compositing Unit, are still under development and will be described in a later document.

3.1 VoluMem

We start the detailed description of the hardware units with the volume memory, since it is the part which defines the performance and the costs of the entire system.

A data set can have up to 512^3 samples. The coordinates of a resample location have a 9 bit integer part and an 8 bit fraction. The coordinates (X,Y,Z) of the original samples

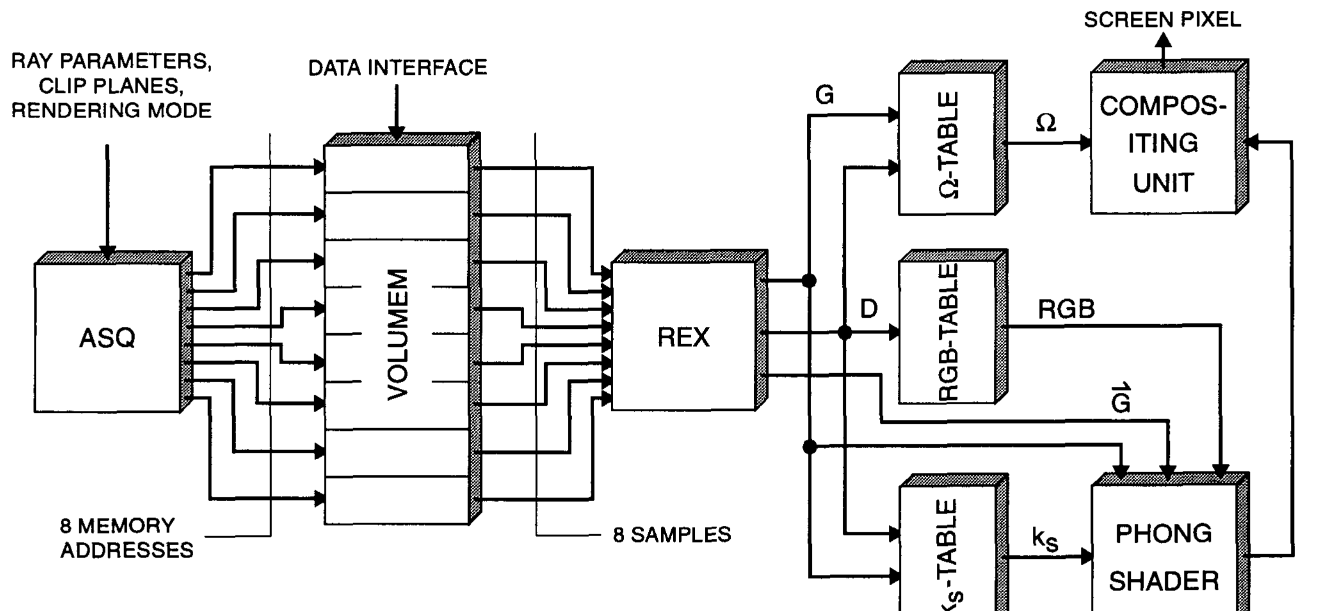

Figure 3: Architectural Overview

$S(\vec{X})$ are 9 bit unsigned integers.

We define the ***Reference Point*** for a given resample location $\vec{X}_R = (X_R, Y_R, Z_R)$ by

$$\vec{P} = (P, Q, R) = (\lfloor X_R \rfloor, \lfloor Y_R \rfloor, \lfloor Z_R \rfloor) \,. \quad (4)$$

For performance reasons, any set of samples (*S*-, *A*-, *B*-, and *C*-set) relative to a given Reference Point must be obtainable by a single memory access. This requires 8 independent memory banks and a conflict-free distribution function. Grouping the samples according to the least significant bits (X_0, Y_0, Z_0) of their coordinates will avoid any access conflicts. The bank number of a given sample is then given by

$$\beta = \beta_2\beta_1\beta_0 = Z_0Y_0X_0 \,, \quad (5)$$

the location of a sample within its bank is given by

$$\vec{U} = (U, V, W) = (\lfloor X/2 \rfloor, \lfloor Y/2 \rfloor, \lfloor Z/2 \rfloor) \,. \quad (6)$$

For further performance increase in High-Speed Rendering mode, we use ***address pipelining*** and apply ***address interleaving*** to each memory bank. Any sample contributes to a set of 8 volume elements, what we call its catchment area. For the samples of a given memory bank, the catchment areas are all non-overlapping. Each memory bank is subdivided again into 8 memory units, so that adjacent catchment areas in each direction have their samples in different memory units. In terms of address arithmetic, the samples are distributed according to $(U_0, V_0, W_0) = (X_1, Y_1, Z_1)$ among the eight memory units.

Although the spacing can be arbitrary per ray, we optimize the system for the following assumption:

- ❑ for any point, the next resample location falls into the same volume element or one of its 26-neighborhood.

Then we make the following observations:

- ❑ Any memory unit is either accessed subsequently an arbitrary number of times or
- ❑ at least two accesses refer to other units before the next access to this unit can occur.

In the latter case a memory cycle can take three clock cycles, i.e. 50ns at a target clock frequency of 60MHz. This is sufficient for the so-called *Page Mode* of standard DRAMs.

The basic architecture of memory devices is an array of storage cells [13]. Any random access to a DRAM device takes place in two steps: first, the addressed row (or page) must be loaded completely into an internal output register (row access), from where the desired data item can be accessed in a second step (column access). If the following memory cycle refers to the same row, the row access can be skipped since the data still exists in the output register (Page Mode access). The Page Mode cycle time is about 40ns. Thus, the task is to arrange the samples within the memory devices so that the Page Mode can be used the most often.

Data sets of 512^3 samples, 16 bits each, require a memory capacity of 256MByte. With 16MBit DRAM technology we need 128 devices. We use 2M×8 (2048 rows × 1024 columns × 8 bits) organized DRAMs, so that each pair of devices forms one of 64 memory units. We provide each memory unit with address registers and control logic so that it can operate independently. Eight such memory units are integrated into one Intelligent Memory Module (IMM), of which again eight are needed to build the complete volume memory. One row across all devices has 1MBit, that is 64K samples. Since there are no principal ray directions, we place the samples of a 32×64×32 subvolume, called "P-block", into one page.

If subsequent accesses to the same memory unit occur, however, ***always the same sample*** is addressed. Placing a fast register (a one entry cache) behind each unit avoids any performance penalty in this case.

In this way one set of addresses can be issued and one *S*-set is delivered each clock cycle as long as we move around in one P-block.

High-Quality Rendering also benefits from this memory architecture. Due to the high interleaving factor and the one-entry caches, all the samples of the *S*-, *A*-, *B*- and *C*-sets can be fetched within 4 clock cycles as long as no set is distributed across more than one P-block.

The block diagram of an IMM is shown in Figure 4. Each

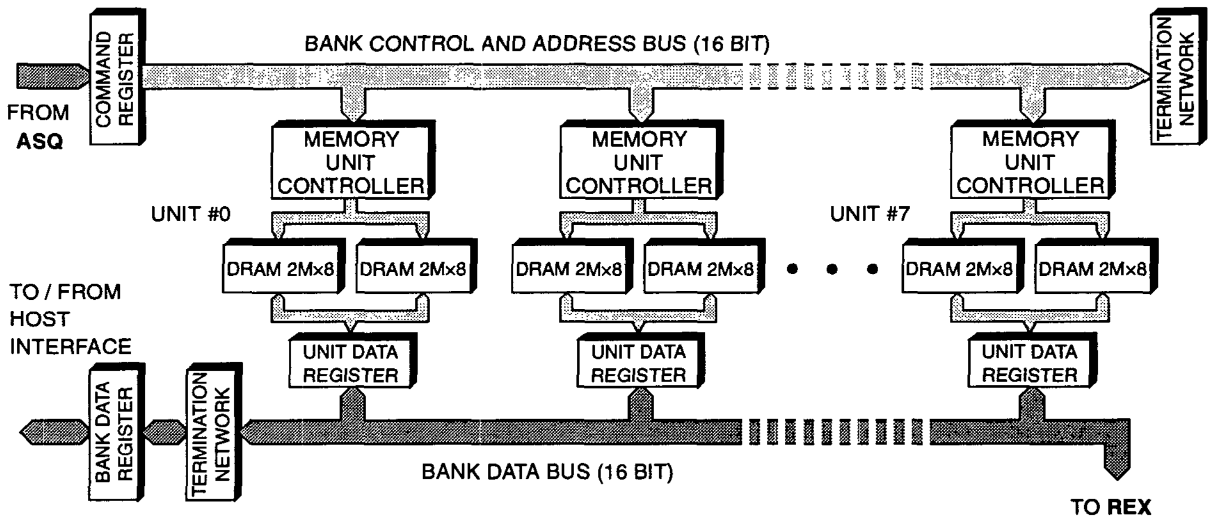

Figure 4: Intelligent Memory Module

clock a new command (row access, read, write, no operation, etc.) is written into the bank command register and broadcast to the eight units. A Memory Unit Controller (MUC), if selected, executes a command immediately after receipt. For write cycles, the host is required to place the eight samples of an S-set into the bank data registers before one write operation can be completed. This can be done sequentially or all samples at a time. Assumed there is a large Silicon Disk holding a sequence of data sets, one S-set can be loaded each clock, giving a load time of about 35ms for a 256^3 data set. In the case of a read cycle, the sample is clocked into the unit data register (the one-entry cache) and placed onto the (tri-state) bank data bus the appropriate number of times. Due to the strict behavior of the memory units, the MUCs can easily be integrated into small EPLDs (Erasable Programmable Logic Devices).

3.2 Address SeQuencer (ASQ)

A block diagram of the Address SeQuencer is shown in Figure 5. ASQ generates the memory addresses for both read and write operations. For write cycles, additional synchronization mechanisms are needed, which are not described in this paper.

Here we will focus on read cycles, since they are more challenging and more relevant for the usefulness of the machine. ASQ was designed as a pipelined unit, which processes a complete ray autonomously and issues accesses at the maximum rate defined by the memory system.

The host interface is a collection of registers, accepting

- ❑ rendering mode, clip planes and volume boundaries, set up once per frame or session;
- ❑ the starting point $\vec{X}_{R,S}$ on the ray, the vector $\Delta\vec{X}_R$ to the next resample location, the initial distance from the view point A_S and its increase ΔA, programmed once per ray.

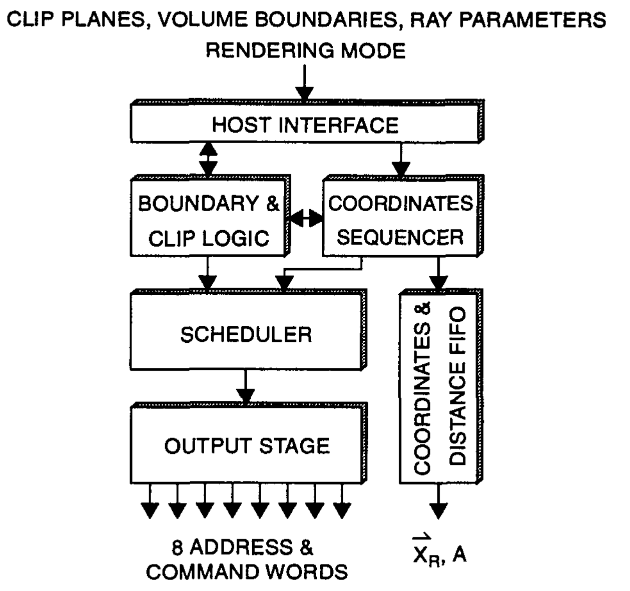

Figure 5: ASQ

The coordinates sequencer is just a set of adders, which can generate any sequence of evenly spaced points on a straight line. In the High-Quality Rendering mode, the coordinates of any resample location are issued four times and tagged as an S-, A-, B-, or C-type access, respectively.

The coordinates are passed to the boundary & clip logic, which invalidates the location if the region of interest has not been reached yet or terminates the ray if it was left. In the latter case the processing of a new ray will be started automatically if its parameters are already present in the host interface. A FIFO assures that all parameters of a given resample location arrive synchronously at the inputs of REX.

The ***access function*** α for an S-set defined by a Reference Point (P,Q,R) returns the coordinates of the samples in dependency of the bank number β:

$$\alpha:\vec{P} \to \vec{U}(\beta) = \begin{pmatrix} U = \lfloor P/2 \rfloor + P_0 & \text{for} & \beta_0 = 0; & U = \lfloor P/2 \rfloor & \text{for} & \beta_0 = 1 \\ V = \lfloor Q/2 \rfloor + Q_0 & \text{for} & \beta_1 = 0; & V = \lfloor Q/2 \rfloor & \text{for} & \beta_1 = 1 \\ W = \lfloor R/2 \rfloor + R_0 & \text{for} & \beta_2 = 0; & W = \lfloor R/2 \rfloor & \text{for} & \beta_2 = 1 \end{pmatrix}. \tag{7}$$

For an A-set, replace the first line by:

$$U = \lfloor P/2 \rfloor + \bar{P}_0 \quad \text{for} \quad \beta_0 = 0; \quad U = (-1)^{\bar{P}_0} + \lfloor P/2 \rfloor \quad \text{for} \quad \beta_0 = 1\ . \tag{8}$$

For a B-set, replace the second line by:

$$V = \lfloor Q/2 \rfloor + \bar{Q}_0 \quad \text{for} \quad \beta_1 = 0; \quad V = (-1)^{\bar{Q}_0} + \lfloor Q/2 \rfloor \quad \text{for} \quad \beta_1 = 1\ \cdot \tag{9}$$

For a C-set, finally, replace the third line by:

$$W = \lfloor R/2 \rfloor + \bar{R}_0 \quad \text{for} \quad \beta_2 = 0; \quad W = (-1)^{\bar{R}_0} + \lfloor R/2 \rfloor \quad \text{for} \quad \beta_2 = 1\ . \tag{10}$$

The ***address function*** σ transforms the coordinates into 8 linear memory addresses:

$$\sigma:\vec{U}(\beta) \to N(\beta) = W_{3..1}(\beta) \times 2^7 + V_{4..1}(\beta) \times 2^3 + U_{3..1}(\beta) \text{ for Page Mode (column) accesses and} \tag{11}$$

$$\sigma:\vec{U}(\beta) \to N(\beta) = W_{7..4}(\beta) \times 2^7 + V_{7..5}(\beta) \times 2^4 + U_{7..4}(\beta) \text{ for page (row) accesses.} \tag{12}$$

Figure 6: Functional Description of the Output Stage

The scheduler issues and delays accesses according to the momentary state of the memory system. As explained in the previous section, one access can be initiated each clock as long as all samples of the addressed set reside in the same P-block, once the appropriate page is loaded. However, if any set is spread over multiple P-blocks, a certain subset of memory banks is required to load a new page into their output registers. Then the scheduler halts the internal pipeline of ASQ, inserts a number of wait-states to finish any pending memory cycle, sends a row access command to the memory banks in question and issues the next read command after two additional wait-states.
Besides that, the scheduler detects any resample location stepping outside the 26-environment of its predecessor. In this case the read operations are all random accesses.
The scheduler also maintains synchronization between the different memory banks. There are neither synchronization elements between the memory banks nor between the memory units; any unit completes a read or write access within a fixed period after receipt.
The output stage finally transforms the coordinates of the Reference Point into a set of memory addresses as described in Figure 6, using just 6 adders/subtractors (8-bits wide) and eight multiplexers.

3.3 Reconstructor and EXtractor (REX)

Although parts of this unit have already been presented [7],[8], we give a short description so that the reader has a clear understanding of how we reach the single chip target.
REX falls into five parts:

- the data entry stage, which generates the quantities to be tri-linearly interpolated,
- the tri-linear interpolator, which computes the function value $D(\vec{X}_R)$ and the gradient $\vec{G}(\vec{X}_R)$,
- the gradient magnitude unit and
- two FIFO memories to maintain synchronization.

In High-Speed Rendering mode, the chip accepts one set of input parameters $\{S_0..S_7, \vec{X}_R\}$ and produces one set of output parameters $\{D, \vec{G}, G\}$ each clock cycle. In High-Quality Rendering mode, four clock cycles are needed to enter the input vectors $\{S_0..S_7, \vec{X}_R\}$, $\{A_0..A_7, \vec{X}_R\}$, $\{B_0..B_7, \vec{X}_R\}$ and $\{C_0..C_7, \vec{X}_R\}$.
A block diagram of the chip is given in Figure 7. The data entry stage passes the samples of an *S*-type access unmodified to the tri-linear interpolator, but stores them in a set of input registers. Together with the samples of subsequent *A*-, *B*- and *C*-type accesses (if any), it computes the x-, y- and z-components of the gradient at the original sample locations and feeds them into the tri-linear interpolator. Basically, the data entry stage consists of a number of registers, 8 subtractors and a set of multiplexers.
The tri-linear interpolator is a three-layered, straight-forward arrangement of 15 linear interpolators (LI), which in the first layer compute all values on the edges (labeled E_n in Figure 1). These quantities are then passed to the second layer to produce the quantities named F_n. The third layer consists of one LI which computes the reconstructed func-

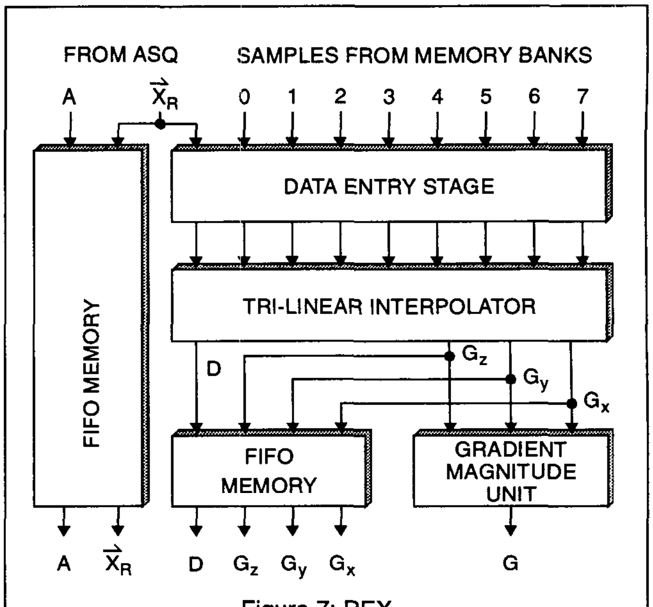

Figure 7: REX

tion value or gradient component at the resample location. Besides that, there are three subtractors to compute the gradient components in High-Speed Rendering mode.

The gradient magnitude unit consists of three square units, a triple input adder and a square root unit. All units are pipelined to reach the clock frequency target. We will explain the square root unit in greater detail, which is shown in Figure 8.
The squared gradient length Γ arrives as a 34 bit unsigned number $\Gamma_{33}\Gamma_{32}..\Gamma_0$ at the inputs. For every two digits of the square, the integer part of the square root $G = G_{16}G_{15}..G_0$ has one digit. The most significant bit G_{16} of the root can be calculated from $\Gamma_{33...32}$ independently of the other bits.
Thus:

$$G_{16} = \Gamma_{33} \vee \Gamma_{32}, \tag{13}$$

a possible remainder $R^{16}_{33..32}$ is given by

$$R^{16}_{33} = \Gamma_{33} \wedge \Gamma_{32} \text{ and } R^{16}_{32} = \Gamma_{33} \wedge \bar{\Gamma}_{32}. \tag{14}$$

The square root of the binary number $\Gamma_{33}\Gamma_{32}\Gamma_{31}\Gamma_{30}$ is still approximated by $2{\times}G_{16}$, the remainder $R^*_{33..30}$ is represented by $R^{16}_{33}R^{16}_{32}\Gamma_{31}\Gamma_{30}$. If we add G_{15}, the new root is $2 \times G_{16} + G_{15}$, and therefore its square is increased by $4 \times G_{16} \times G_{15} + G^2_{15}$.
So the following relation must be satisfied:

$$R^*_{33..30} \geq 4 \times G_{16} \times G_{15} + G^2_{15}. \tag{15}$$

Assuming $G_{15}=1$ requires that

$$R^*_{33..30} \geq 4 \times G_{16} + 1. \tag{16}$$

Thus, G_{15} is just the result flag of the above compare operation. For $G_{15} = 1$ the new remainder $R^{15}_{32...30}$ is given by

$$R^{15}_{32..30} = R^*_{33..30} - 4 \times G_{16} - 1, \tag{17}$$

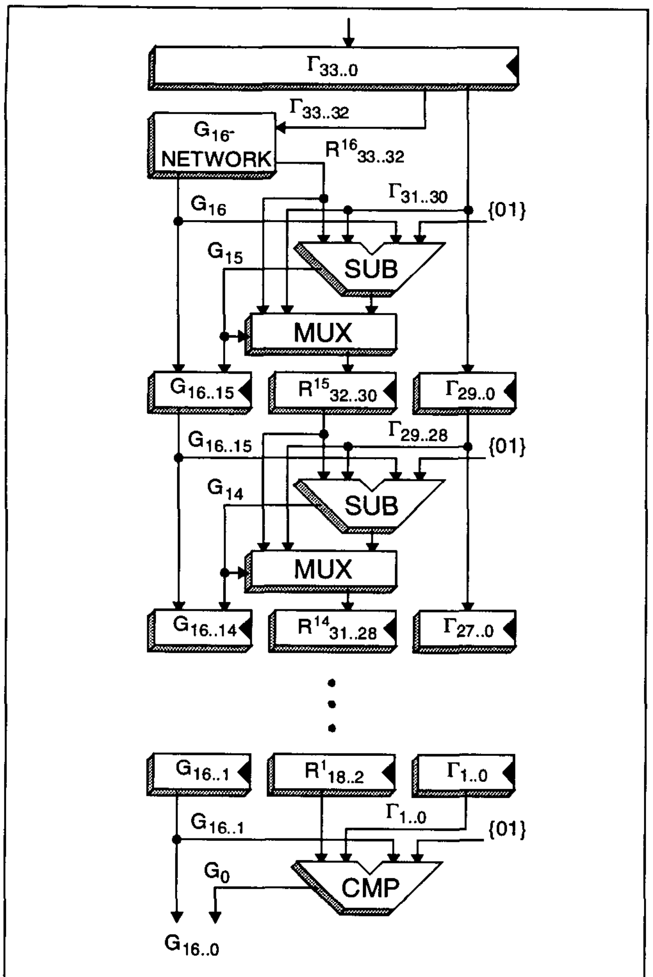

Figure 8: Square Root Pipeline

for $G_{15} = 0$ the remainder is left unchanged. This operation is repeated for every result bit. Accordingly, the pipeline has 16 stages.

In this way, one square root operation is completed every clock. The density D and the gradient components travel through a FIFO memory and arrive synchronously with the gradient magnitude at the outputs. For further processing, the parameters of the resample location are then passed to the classification, shading and compositing units. REX has app. 350 I/O-pins and uses about 175.000 gates.

4 Image Quality

A software simulation of the algorithm was written to show the differences between the High-Quality (Figure 9) and the High-Speed Rendering mode (Figure 10) under worst-case conditions. A computer-generated, unfiltered data set of 256^3 samples, containing a discretized sphere of 240 units in diameter, was rendered. We used simple thresholding, and the rays were terminated after the first encounter of the sphere surface. The surface was illuminated using the Phong illumination model with 10% ambient, 30% diffuse and 60% specular reflected light and the specular exponent set to 25.

5 Performance

Since the required rendering time for a sequence of perspective views is hard to predict, a simulator was written which performs the functions of the coordinates sequencer and the scheduler within ASQ. Two round-trips around a 256^3 data set were simulated: for round-trip #1 the observer was placed at (128, 512∗cosα+128, 512∗sinα+128), for trip #2 at (362∗cosα+128, 362∗cosα+128, 512∗sinα+128). A 100×100 view plane was located 256 units apart from the observer and rendered at a 256^2 resolution using a stepsize

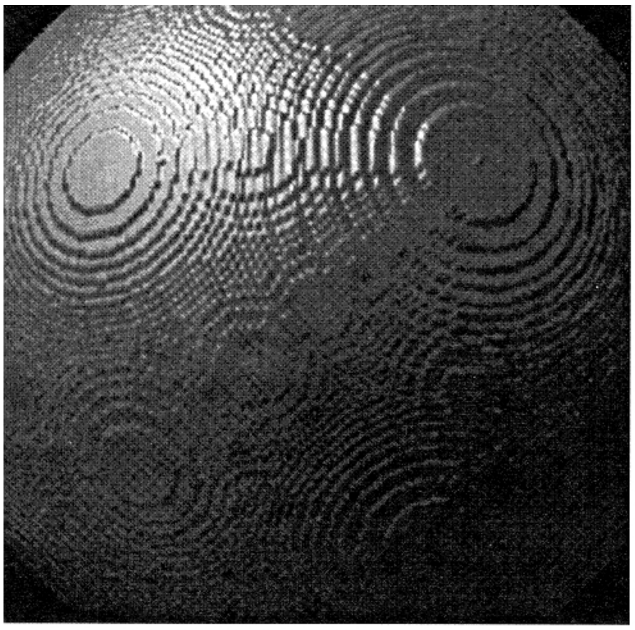
Figure 9: High-Quality Rendering

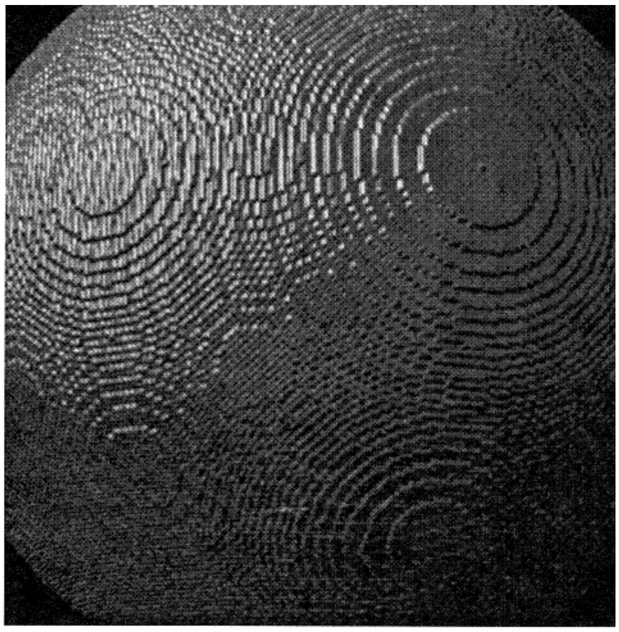
Figure 10: High-Speed Rendering

of 0.95 along each ray. Each round-trip produced 360 frames, and was simulated in High-Speed and High-Quality mode. Provided the clock frequency target of 60MHz can be reached, which is the upper limit given by the DRAM devices, we'll obtain the following results:

Round-Trip	Rendering Mode	Total Time	Average Time per Frame	Frame Rate
1	HS	139.5s	0.388s	2.58Hz
2	HS	148.8s	0.413s	2.42Hz
1	HQ	601.2s	1.67s	0.6Hz
2	HQ	647.7s	1.8s	0.56Hz

Thus, for example, rendering 10 frames in High-Speed mode to find a new viewpoint and one frame in High-Quality mode afterwards will take roughly 5.75s.

6 Conclusion and Future Work

A voxel subsystem, formerly associated with a rack-based system, has now taken shape of a PC slotboard. Adaptive refinement [11] or subsampling during motion can bring the frame generation rate for previewing purposes up into the real-time range, turning this small system into a very useful machine.

Our short-term research activities are devoted to the design of the remaining circuitries, the Phong Shader and the Compositing Unit. In principle, all problems have been solved, work must only be done to find the optimal solution and to define implementation details.

The next major effort is dedicated to the definition of parallelism on system level. Significant speed-up can only be achieved by operating multiple units in parallel. However, simply duplicating the entire data set (and the memory costs, by the way) is definitely not acceptable. In general, the solution will be to distribute certain subcubes of the data set among the different units, which process any given ray as long as it traverses through their own subcube. On exit, each unit assembles a communication packet defining the ray properties up to this point and sends it to the unit holding the subcube the ray is about to enter. Finding the optimal *granularity* is one problem: if the subcubes are too small, the communication overhead is the performance bottleneck, if the subcubes are too large, an uneven workload may paralyze the system. However, our architecture has a "natural" granularity: the P-blocks defined by the page size of the DRAMs. Crossing a P-block boundary causes a certain overhead anyway, and there are chances that assembling a communication packet and scheduling a new ray will not increase this overhead. Then, a linear speed-up over the number of units is achievable. The other problem is that samples in the neighborhood of the "last" resample location normally reside in other units. Thus, duplicating a certain number of layers of the data set might be unavoidable, what in turn might complicate the address arithmetic considerably.

7 Acknowledgments

Thanks to Reinhard Klein and Andreas Schilling for their numerous valuable suggestions.

8 References

1. **R. A. Drebin, L. Carpenter, P. Hanrahan,** "Volume Rendering", Computer Graphics, Vol. 22, No. 4, August 1988, pages 65-74
2. **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", presented at the 9. Eurographics Hardware Workshop, Oslo, September 12-13, 1994
3. **D. Jackèl and W. Straßer,** "Reconstructing Solids from Tomographic Scans - The PARCUM II System", in Advances in Computer Graphics Hardware II, edited by A.A.M. Kuijk and W. Straßer, Springer-Verlag, Berlin, 1988, pages 209-227
4. **A. Kaufman, R. Bakalash, D. Cohen and R. Yagel,** "A Survey of Architectures for Volume Rendering", IEEE Engineering in Medicine and Biology, Vol. 9, No. 4, December 1990, pages 18-23
5. **G. Knittel and W. Straßer,** "An Accelerator for Voxel Graphics", Proceedings of the 2. ITG/GI Workshop on Workstations, Hagen, May 24-27, 1993, VDE-Verlag, Berlin, pages 69-78
6. **G. Knittel,** "VERVE - Voxel Engine for Real-time Visualization and Examination", Computer Graphics Forum, Vol. 12, No. 3, September 1993, pages 37-48
7. **G. Knittel,** "A VLSI-Design for fast Vector Normalization", Proceedings of the 8. Eurographics Hardware Workshop, Barcelona, September 6-7, 1993, pages 1-14
8. **G. Knittel,** "PROVEN - Prompt Vector Normalizer", Proceedings of the 6. IEEE International ASIC Conference, Rochester, NY, September 27 - October 1, 1993, pages 112-115
9. **G. Knittel,** "A Fast Logarithm Converter", Proceedings of the 7. IEEE International ASIC Conference, Rochester, NY, September 19-23, 1994
10. **M. Levoy,** "Display of Surfaces from Volume Data", IEEE Computer Graphics & Applications, Vol. 8, No. 5, May 1988, pages 29-37
11. **M. Levoy,** "Volume Rendering by Adaptive Refinement", The Visual Computer, Vol. 6, No. 1, February 1990, pages 2-7
12. **H. Pfister and A. Kaufman,** "Real-Time Architecture for High-Resolution Volume Visualization", Proceedings of the 8. Eurographics Hardware Workshop, Barcelona, September 6-7, 1993, pages 72-80
13. **B. Prince,** "Semiconductor Memories", Wiley & Sons, Chichester, 1991, pages 250-255