A Scalable Architecture for Volume Rendering

Günter Knittel
WSI / GRIS †
University of Tübingen, Germany

Abstract

We describe the operational principles of a scalable hardware accelerator for volume rendering. The basic philosophy is to provide an atomic unit which already provides sophisticated volume graphics at interactive rendering speed. Realtime speed can then be achieved by operating multiple units in parallel. The basic unit consists of just four VLSI chips and the volume memory and thus meets the requirements of a small size and low costs. Nevertheless it provides arbitrary perspective projections (e.g., for walk-throughs), Phong shading, a freely moveable light source, depth-cueing and interactive, non-binary classification (semi-transparent display) at a frame rate of about 2.5Hz for 256^3 data sets.

Keywords: graphics hardware, VLSI designs, volume rendering, ray casting

1 Introduction

In recent years, volume rendering has made its way out of the research labs into the real world. It has gained a large number of application fields, such as medical diagnosis and surgery planning, non-destructive evaluation, weather forecast or exploration of mineral resources to name a few. Besides the fact that large scalar arrays must be classified and visualized, the different applications do not have much in common. In a clinical environment, reliability and performance is at premium with little concern of the costs. In atmospheric research, the ultimate rendering speed may not be the most wanted feature, but instead the ability to handle extremely large data sets. For the visualization of simulated dynamic processes any machine which does not provide realtime *volume animation* is useless, whereas in the workstation market the price is the most crucial quantity.

† 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@gris.informatik.uni-tuebingen.de

Thus, the task is to develop a generic *voxel graphics architecture*, which defines a consistent view from the algorithmic and software side, but nevertheless allows different *implementations* exactly tailored to the requirements of the application.

We take the following visualization paradigms as the common architectural basis:

- direct volume rendering using ray casting,
- integrated, interactive classification as opposed to pre-segmentation,
- semi-transparent display of structures of interest,
- arbitrary perspective projections allowing walk-through examinations and
- meaningful images by sophisticated illumination models, i.e., non-parallel light, Phong shading and depth-cueing.

2 Algorithm

This architectural approach follows the work of Levoy [8] 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. The classification step 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 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 operation 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 off-

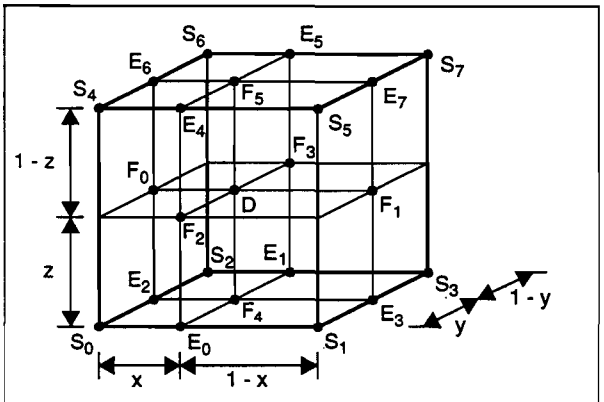

Figure 1: Fast Gradient Computation

set of the resample location within the 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 compu-

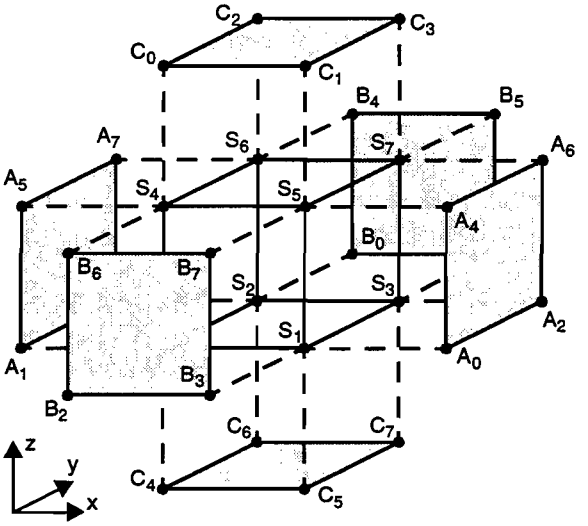

Figure 2: Extended Neighborhood

tation of the function value and the gradient 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 Basic Ray Casting Engine

The 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 engine, which grew out of our previous work in this area [4], [7], 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 750MByte/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 largest VLSI unit is COLOSSUS (**CO**mplex **LO**gic for **S**hading at **SU**per **S**peed). For each resample location, the unrestricted Phong illumination model (non-parallel light, perspective projection) is evaluated. 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 back-transform 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 [6].

COMET (**COM**positing Uni**T**) 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

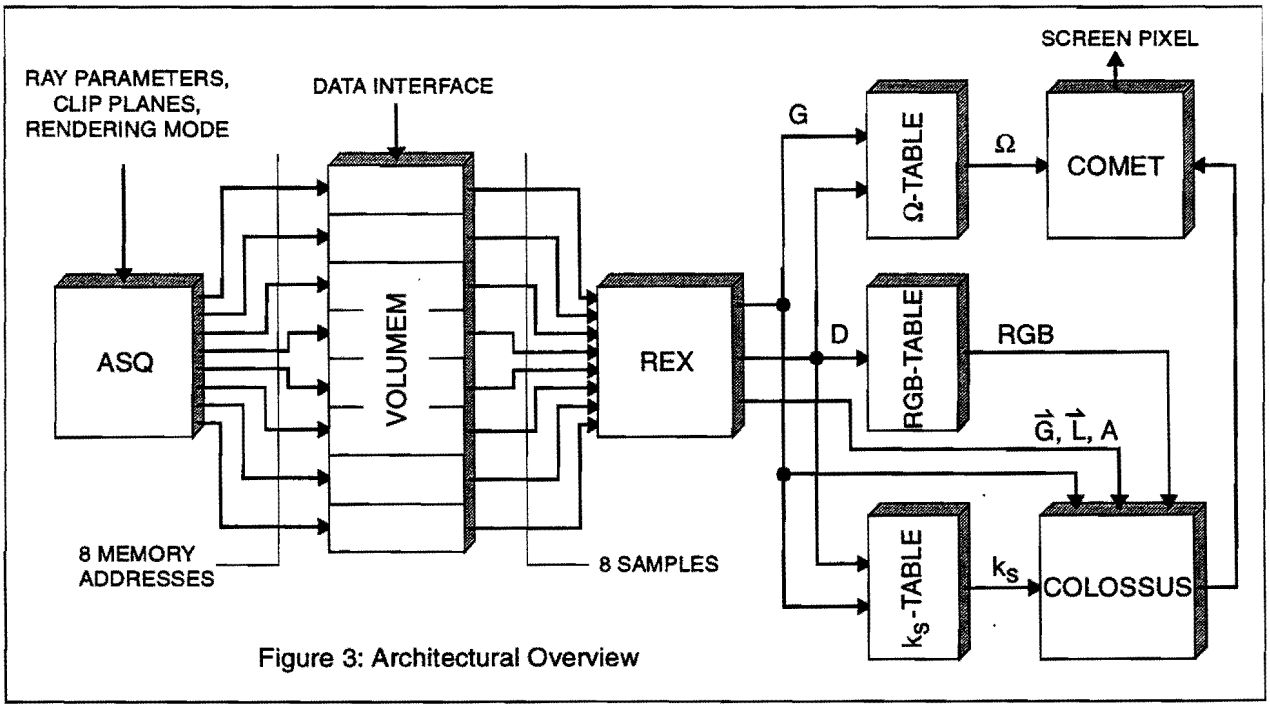

Figure 3: Architectural Overview

voxel graphics subsystem. The "back end" of the accelerator, COLOSSUS and COMET, 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 $S(\vec{X})$ are 9 bit unsigned integers.

We define the ***Reference Point*** for a given resample location $\vec{L} = (L, M, N)$ by

$$\vec{P} = (P, Q, R) = (\lfloor L \rfloor, \lfloor M \rfloor, \lfloor N \rfloor), \quad (4)$$

where P denotes the component in x-, Q the component in y- and R the component in z-direction.

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 [10]. 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

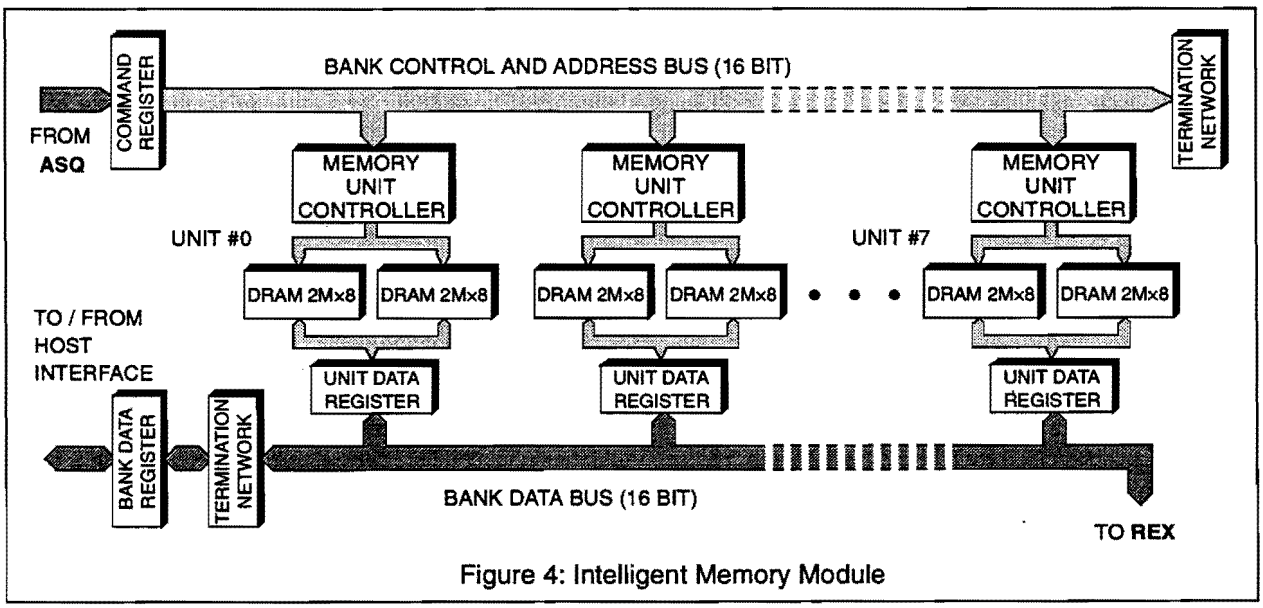

Figure 4: Intelligent Memory Module

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 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 command word has 16 bits and carries the row or column address (11 or 10 bits, a unit address (3 bits) and a command word (2 or 3 bits)). That is, the bus from ASQ to VoluMem is 128 bits wide. 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 mecha-

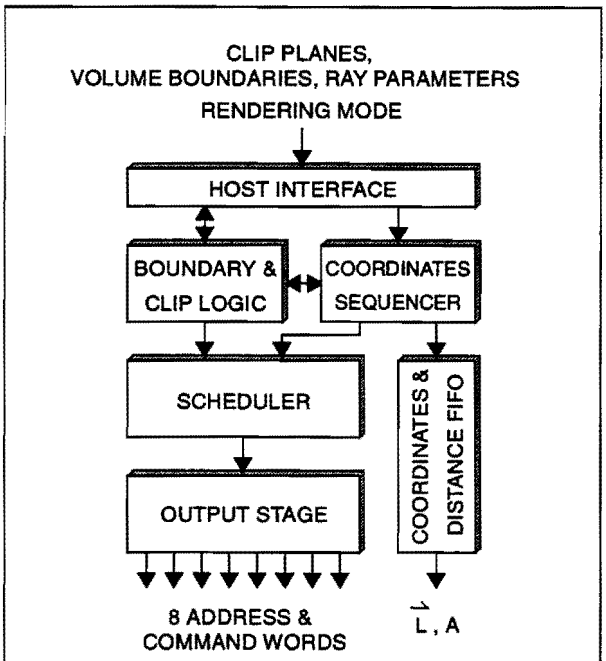

Figure 5: ASQ

nisms 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 $\vec{L}_S$ on the ray, the vector $\Delta\vec{L}$ to the next resample location, the initial distance from the view point A_S and its increase ΔA, programmed once per ray.

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 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 [4],[5], 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 $D(\vec{L})$ and the gradient $\vec{G}(\vec{L})$,
- the gradient magnitude unit and
- two FIFO memories to maintain synchronization.

In High-Speed Rendering mode, the chip accepts one set of input parameters $\{S_0..S_7, \vec{L}\}$ 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 parameter sets $\{S_0..S_7, \vec{L}\}$, $\{A_0..A_7, \vec{L}\}$, $\{B_0..B_7, \vec{L}\}$ and $\{C_0..C_7, \vec{L}\}$.

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 function value or gradient component at the resample location. Besides that, there are three

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:

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

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

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

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

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

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

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

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

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

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

Figure 6: Functional Description of the Output Stage

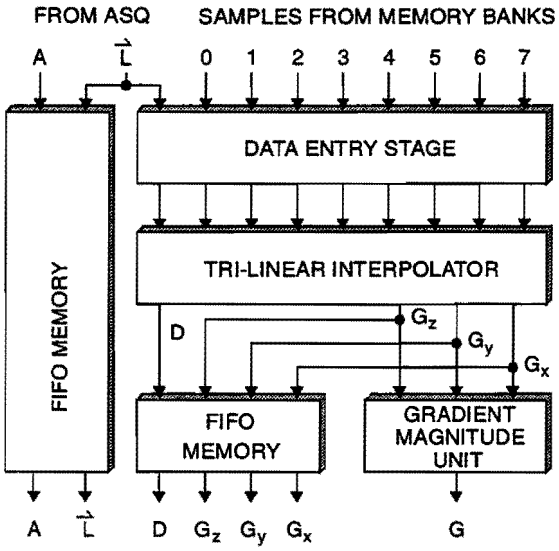

Figure 7: REX

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

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

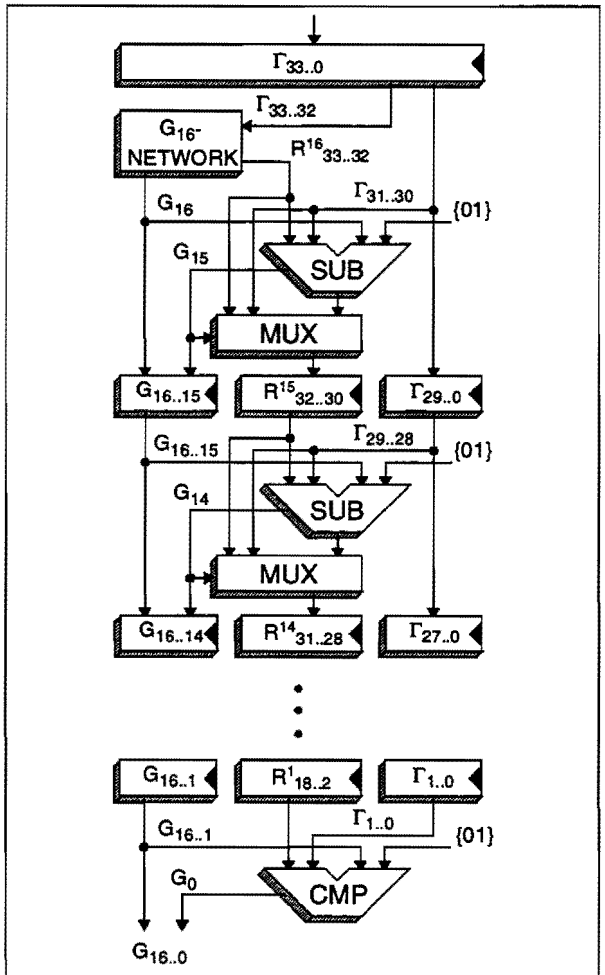

Figure 8: Square Root Pipeline

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 100x100 view plane was located 256 units apart from the observer and rendered at a 256^2 resolution using a stepsize 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 Parallelizing Ray Casting Engines

Although advances in technology will allow us to increase the clock frequency up to a certain point, significant speed-up can only be achieved by operating multiple engines in parallel. Due to the high costs, however, providing each engine with memory for the entire data set is definitely not acceptable.

The general idea is to divide the data set into certain subvolumes and to distribute these subvolumes over different ray casting engines. Each engine processes any given ray as long as it traverses through its own region. On exit, each engine assembles a communication packet defining the ray properties up to this point and sends it to the engine holding the subvolume the ray is about to enter. Thus, at peak performance, the number of rays which can be traced in parallel through the data set equals the number of engines.

The first problem to be solved is to find the optimal *granularity*: 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, especially in the case of walk-throughs. 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 assembling a communication packet and scheduling a new ray will not increase this overhead significantly. Although one can always find cases in which any parallel machine operates at low efficiency (in this case, for example, rendering a frame from a single P-Block), an almost linear speed-up over the number of units is achievable in

Figure 9: High-Quality Rendering

Figure 10: High-Speed Rendering

most practical cases.
Another problem is presented by resample locations which fall in the space between two P-blocks residing on different engines. Collecting the appropriate neighborhood for this sample point will result in an excessive communication overhead. Thus, a special replicating and partitioning scheme, which assures that each engine always has all required samples available, is needed. However, any modification of the data set requires to adapt the ray casting algorithm. Data set partitioning and distributed ray casting are discussed in the following two sections.

6.1 Data Set Preparation and Partitioning

For explanation purposes, let's assume a P-Block size of 32×32×32 and a maximum data set size of 512^3 samples, which should be distributed over 8 ray casting engines. For the moment, we will only consider High-Speed Rendering.

Thus, all boundary layers between subvolumes must be duplicated. This introduces a certain redundancy into the data set and causes a small loss in maximum data size. The data set preparation can be explained by the following lines of pseudo-code:

```
FOR X = 511 TO 32 DO
  FOR Y = 0 TO 511 DO
    FOR Z = 0 TO 511 DO
      sample(X,Y,Z) = sample(X - ⌊X/32⌋,Y,Z);

FOR Y = 511 TO 32 DO
  FOR X = 0 TO 511 DO
    FOR Z = 0 TO 511 DO
      sample(X,Y,Z) = sample(X,Y - ⌊Y/32⌋,Z);

FOR Z = 511 TO 32 DO
  FOR X = 0 TO 511 DO
    FOR Y = 0 TO 511 DO
      sample(X,Y,Z) = sample(X,Y,Z - ⌊Z/32⌋);
```

Or, explained differently but more comprehensively: the layer with $X=31$ was copied into the layer with $X=32$ after having shifted right all samples with $X \geq 32$ by one grid location. Then, using this modified data set, the layer with $X=63$ was copied into the layer with $X=64$ after having shifted right all samples with $X \geq 64$. We proceed up to the volume boundary, discarding the layers with $497 \leq X \leq 511$. The same procedure is then performed in y- and z-direction. Thus, the maximum data set size was reduced to 497×497×497. The loss can be centered by first shifting the data set 8 locations in negative x-, y- and z-direction.
The prepared data set is then distributed over the 8 engines according to $\{Z_5 Y_5 X_5\}$, thus establishing P-Block interleaving in each direction.

6.2 Distributed Ray Casting

Rays can be cast into this distributed data set as usual, however, one has to observe the following rule:

- a resample location $\vec{L} = (L, M, N)$ in the logical space must be transformed by

$$\vec{L}' = \left(L + \left\lfloor \frac{L}{31} \right\rfloor, M + \left\lfloor \frac{M}{31} \right\rfloor, N + \left\lfloor \frac{N}{31} \right\rfloor\right) \quad (18)$$

  into machine coordinates.

This must be done once per ray by the host to compute the starting point, and by each engine every time a ray leaves a P-Block (except, of course, on volume exit). For simplicity, we assume that the 26-neighborhood rule is always satisfied. Then, the exit-condition for a P-Block is given by $X_B \vee Y_B \vee Z_B$, where

$$X_B = P'_4 P'_3 P'_2 P'_1 P'_0, \quad (19)$$

$$Y_B = Q'_4 Q'_3 Q'_2 Q'_1 Q'_0 \text{ and} \quad (20)$$

$$Z_B = R'_4 R'_3 R'_2 R'_1 R'_0. \quad (21)$$

The coordinate transformation which must be done by each engine on P-Block exit is simple:

$$L'_{exit} = (-1)^{S\Delta L} \times X_B + L', \quad (22)$$

$$M'_{exit} = (-1)^{S\Delta M} \times Y_B + M', \quad (23)$$

$$R'_{exit} = (-1)^{S\Delta N} \times Z_B + R', \quad (24)$$

where $S\Delta L$, $S\Delta M$ and $S\Delta N$ are the sign flags of the increments in x-, y- and z-direction, respectively.
That is, increase every coordinate of the resample location which cause a P-Block exit if the increment in this direction is positive, else decrement the coordinate. This function is performed by ASQ. The address of the target engine is derived from the transformed coordinates.
An example of this kind of data partitioning and ray casting is given in Figure 11. For clarity, a two-dimensional view, where only 4 engines are involved, is shown.

6.3 Ray Definition Packet (RDP)

The ray properties are given by the following data items (*n.m* denote *n* integer and *m* fraction bits):

Property	Name	Format	# of Bits
(sub-) pixel address	U,V	10.2	24
accum. color and opacity	RGBΩ	8.10	72
next resample location	$\vec{L}$	9.10	57
increments	$\Delta\vec{L}$	1.10	33
distance to observer	A	10	10
increase in distance	ΔA	2.10	12
normalized viewing vector	$\vec{V}$	1.15	48

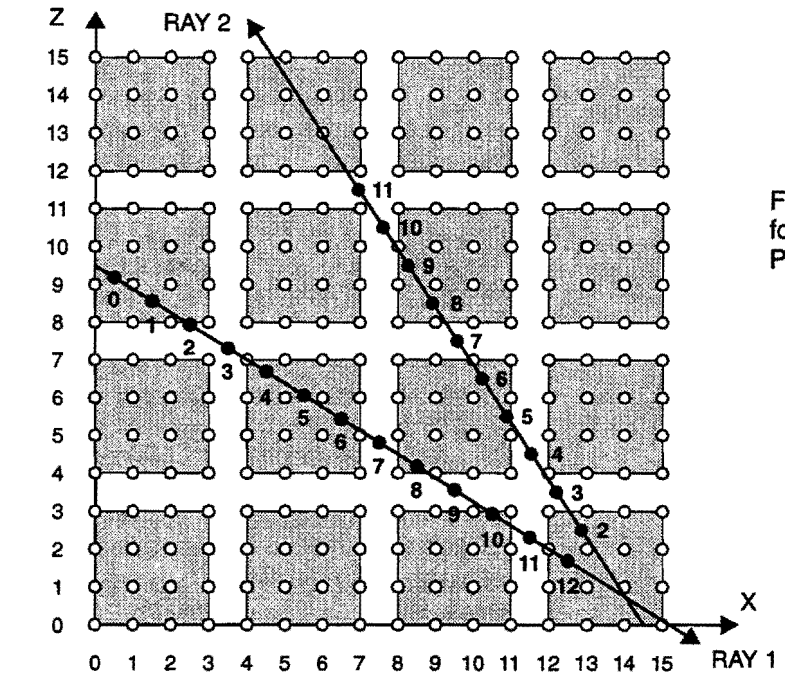

Figure 11a: Ray Casting performed on the logical data set. P-Block size is 4×n×4.

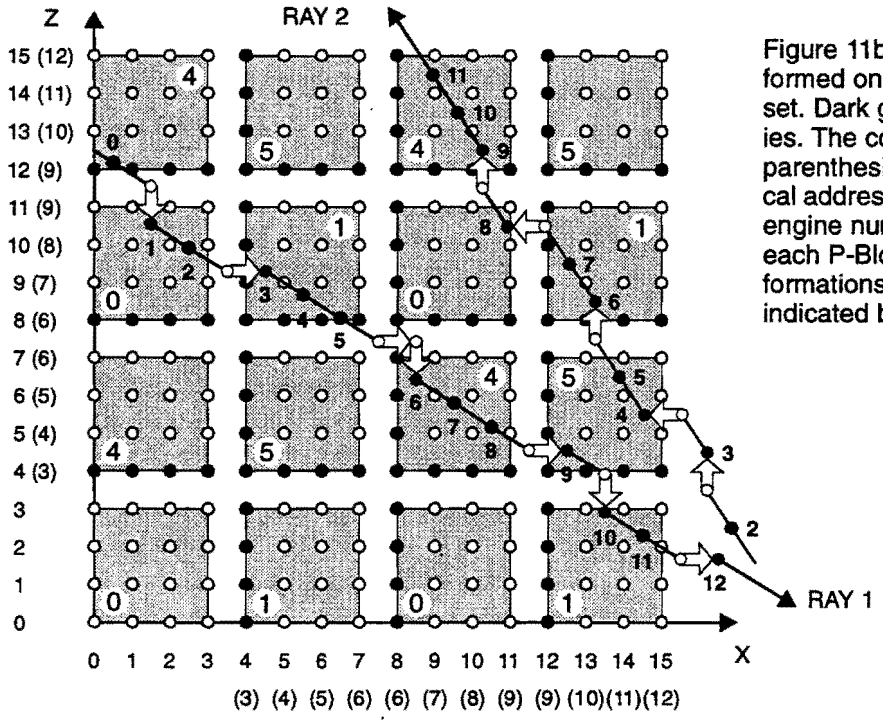

Figure 11b: Ray Casting performed on the distributed data set. Dark grey samples are copies. The coordinates given in parenthesis represent the logical address of the samples. The engine number is shown for each P-Block. Coordinate transformations on P-Block exits are indicated by small arrows.

Thus, an RDP contains 256 bits. The precision requirements of the operands stem from the following consideration: the accumulation error should not reach the integer part. Thus, for example, assuming a maximum of 1024 resample locations along a ray, the increments can be truncated after 2^{-10}.

All quantities are readily available in internal registers of ASQ, COLOSSUS and COMET. On volume exit, the color components are sent to the host as 8 bit integers, together with the ray-ID *(U,V)*.

6.4 Parallel Hardware Architecture

We will now transform the demonstration example into the real world. The maximum data set size of 512^3 samples, 16 bits each, distributed over 8 ray casting engines gives 256^3 = 16M samples or 32MBytes per engine. The volume memories are constructed from 256K×16 DRAM devices, organized as 512×512×16. Then, each memory unit consists of one DRAM device, a memory bank contains 8 devices and the memory system on one

engine has 64 DRAM chips. One page across all devices of a single engine has 512KBits, that is 32K samples. In this way, P-Blocks spanning 32x32x32 samples are constructed.

A simplified block diagram of one ray casting engine is given in Figure 12. RDPs are received from the packet bus via the input FIFOs, processed and sent through the output FIFOs to the packet bus again or to the host via the pixel bus.

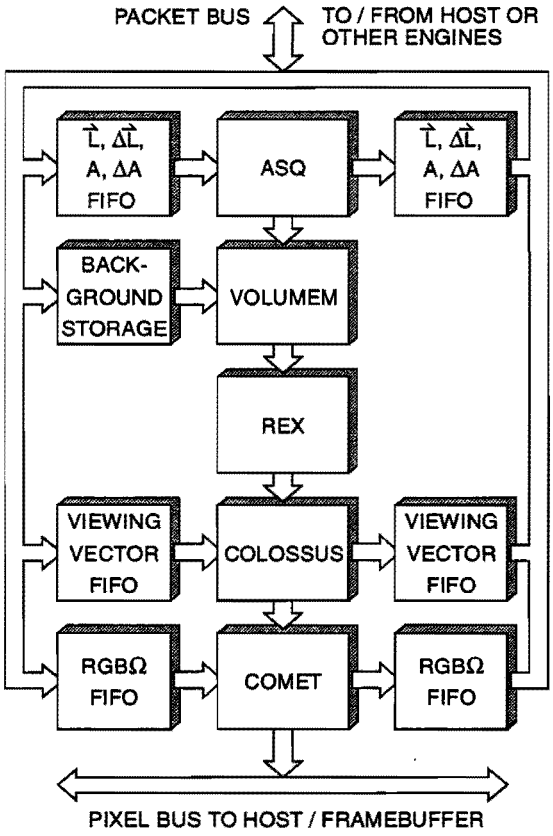

Figure 12: Ray Casting Engine for Parallel Operation and Volume Animation

The bus topology is of special importance in any parallel architecture. In the simplest configuration, as shown in Figure 13a, the packet bus is a multi-master bus which provides time-multiplexed access. In the worst case, a ray travels diagonally through the data set and steps into each and every P-Block boundary. Then it requests a data transfer approximately every 20 clocks. In the best case (parallel projection along one of the main axes), bus requests occur once every 32 clocks. Assuming a typical average bus request frequency of once every 24 clocks per ray, a bus cycle may take 3 clocks. Assuming further that one bus phase is consumed by overhead (arbitration and bus turnaround time), an RDP must be transferred within two clocks. This requires to have a 128 bit bus running at 60MHz, what represents the current state of technology.

However, any larger number of parallel ray casting engines does not permit the use of a bus. In the ideal case, each engine should be connected to every other engine in its 26-neighborhood. Of course, the hardware expenses would be too large. An often used compromise is the ring-connected cube network, where each engine is connected to its 6 neighbors in x-, y- and z-direction. The interconnection scheme is shown in Figure 13b for a two-dimensional arrangement. The maximum distance a RDP must travel is 3 channels, and each engine needs appropriate routing capabilities. In the case of 64 engines, we need 192 bidirectional links.

6.5 High-Quality Rendering Mode

In general, the same principle can be used to distribute the data set and to perform distributed ray

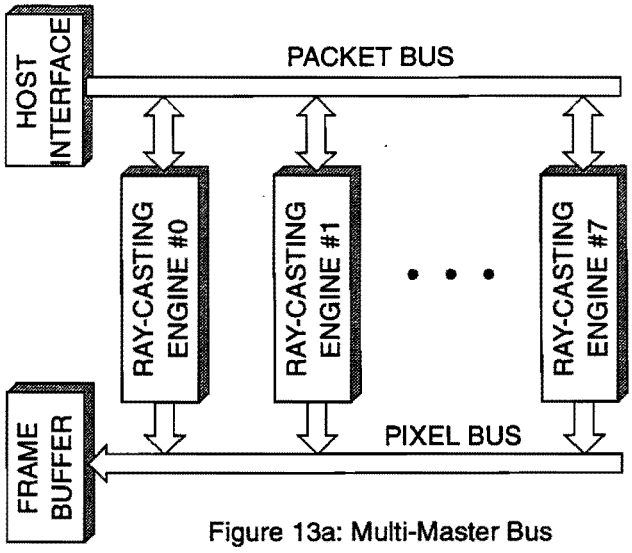

Figure 13a: Multi-Master Bus

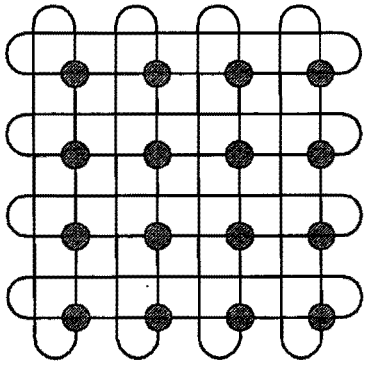

Figure 13b: Torus Network

casting in High-Quality Rendering mode. However, due to the enlarged neighborhood used to compute the gradient, the loss in maximum data size becomes substantial. Another drawback is that the data set must be partitioned differently for High-Speed and High-Quality Rendering mode. Thus we propose to always prepare and partition the data set for High-Speed mode, and to use High-Quality Rendering only within the P-Blocks. The slightly decreased gradient accuracy at the P-Block boundaries will most probably cause no visible artifacts.

7 Volume Animation

Using a large background storage system (see Figure 12) with a sufficiently short access time, complete sequences of (properly prepared and partitioned) data sets can be visualized. As explained in section 3.1, one *S-set* can be loaded each clock per engine. Thus, using 8 engines in parallel, the load time for a 256^3 data set is about 4.4ms, and a 512^3 data set is loaded in 35ms (using a clock frequency of 60MHz). For any given configuration or data set size, the load time can be neglected versus the rendering time.

8 Conclusion and Future Work

We presented a scalable volume rendering architecture which covers a wide range of application requirements. The basic ray casting engine already provides sophisticated visualization techniques and interactive rendering speed, but nevertheless fits on a single PC slotboard. Adaptive refinement [9] or subsampling during motion can bring the frame generation rate for previewing purposes up into the real-time range.

The parallelization scheme presented in this paper supports realtime volume rendering and animation. An easy-to-build parallel system of 8 engines achieves a peak generation rate of 20Hz for 256^3 data sets; a highly-parallel system of 64 engines provides a peak performance of 20 frames per second for 512^3 data sets, while still presenting moderate design complexity. In this configuration, the performance compares to or even exceeds that of other realtime architectures such as CUBE [3] or VIRIM [2].

Our short-term research activities are devoted to the design of the remaining circuitries, the Phong Shader (COLOSSUS) and the Compositing Unit (COMET). Next, a distributed simulation framework will be implemented on a workstation network to verify the design and to evaluate performance limits.

9 Acknowledgments

This work was supervised by Prof. Straßer and is part of the advanced graphics accelerator project at WSI/GRIS, University of Tuebingen, supported partially by the CEC's ESPRIT programme. Thanks to Reinhard Klein and Andreas Schilling for their numerous valuable suggestions.

10 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", Proceedings of the 9. Eurographics Hardware Workshop, Oslo, September 12-13, 1994

3. **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

4. **G. Knittel,** "VERVE - Voxel Engine for Realtime Visualization and Examination", Computer Graphics Forum, Vol. 12, No. 3, September 1993, pages 37-48

5. **G. Knittel,** "A VLSI-Design for fast Vector Normalization", Proceedings of the 8. Eurographics Hardware Workshop, Barcelona, September 6-7, 1993, pages 1-14

6. **G. Knittel,** "A Fast Logarithm Converter", Proceedings of the 7. IEEE International ASIC Conference, Rochester, NY, September 19-23, 1994

7. **G. Knittel and W. Straßer,** "A Compact Volume Rendering Accelerator", Proceedings of the ACM/IEEE Symposium on Volume Visualization, Washington, DC, October 17-18, 1994

8. **M. Levoy,** "Display of Surfaces from Volume Data", IEEE Computer Graphics & Applications, Vol. 8, No. 5, May 1988, pages 29-37

9. **M. Levoy,** "Volume Rendering by Adaptive Refinement", The Visual Computer, Vol. 6, No. 1, February 1990, pages 2-7

10. **B. Prince,** "Semiconductor Memories", Wiley & Sons, Chichester, 1991, pages 250-255