A Low-Cost Memory Architecture For PCI-Based Interactive Ray Casting

Michael Doggett, Michael Meißner*, Urs Kanus†

WSI/GRIS‡ University of Tübingen, Germany

DD&T GmbH§ Reutlingen, Germany

Abstract

In this paper we present a low-cost memory architecture running at 100 MHz which is suited for any PCI-based volume rendering accelerator using the ray-casting approach.

Current SDRAM technology, parallel access to all voxels required for trilinear interpolation, a cubic addressing scheme, and a buffering mechanism accommodating memory latency are applied to achieve high frame-rates. A total of four off-the-shelf standard DIMM modules are required enabling up to 9 Hz (averaged over a representative set of views) for datasets of 256^3 voxels, using early ray termination as the only algorithmic optimization.

The presented memory architecture is a good balance of cost versus feasibility on a standard PCI card - accepting data replication - and will be used for the VIZARD II ray casting accelerator.

CR Categories: B.3.2 [Memory Structures]: Design Style, Associative and Cache Memories; I.3.1 [Computer Graphics]: Hardware Architecture, Graphics Processors; I.3.3 [Computer Graphics]: Picture/Image Generation, Display Algorithms

Keywords: Graphics hardware, volume visualization, volume rendering accelerator, raycasting, memory architecture.

1 Introduction

Numerous architectures for hardware accelerated volume rendering have been proposed over the last decade [9, 10, 3, 13, 1, 12]. Despite the large number of proposals, only a few implementations are available [4, 14, 11], including [21] which is expected in June 1999. The reasons for the rarity of implemented systems are varied.

Volume Rendering is a computationally intensive process and puts high demands on the memory interface. Furthermore, features like over-sampling, cut-planes, multiple classification spaces, segmentation, etc. require trade-offs depending on the individual architecture. The computational complexity of volume rendering, and hence the required real estate, is no longer the major issue[1]. However, as in polygonal graphics, the requirements for the memory interface are very high (even higher) and currently the ultimately limiting factor of any volume rendering accelerator. Nevertheless, while different memory architectures for ray-casting based volume rendering have been presented only a few have been described in detail or analyzed with respect to their feasibility and correct timing behavior.

1.1 Related Work

Kaufman et al. [8] presented an early approach for hardware accelerated volume rendering. Using parallel operating pipelines – each containing one memory unit – the memory requirements are reduced by exchanging data locally between pipelines. Furthermore, a skewing scheme is used to guarantee a conflict-free access to any axis-parallel beam of voxels. While this scheme aggravates the integration of features like multiple classification spaces or oversampling of the viewplane, a high memory bandwidth can be accomplished enabling real-time volume rendering [22, 7]. However, the Cube architectures are more of an object space approach simulating ray-casting by limiting the process to parallel projections, than a true ray-casting accelerator providing perspective projections and requiring random access. Therefore, the memory requirements and the memory architecture are highly specialized and outside the range of what is presented in this paper. For example, the EM-Cube architecture [20] works on the assumption that all voxels are read from memory to generate one image so that burst mode reading of sub-cubes or blocks can be used.

In Verve, Knittel presented a memory interface that uses eight different memories, such that all eight voxels necessary for trilinear interpolation can be fetched in parallel [9]. A total of 16 SIMMs were estimated for a 512^3 dataset. Similar to Verve, VIRIM uses eight memory units to allow parallel access for each sample. Additionally, each unit contains two banks allowing for ping-pong readout. Although the authors refer to "data cubes", it is not clear whether a cubic addressing scheme, like the one presented here, is used or not.

The DIV^2A system, as proposed by Lichtermann, uses multiple image processors, each having its own private voxel memory [14]. To meet the timing of the processor's clock frequency, static memory is used and hence, no special memory architecture is necessary. However, this prevents the system from being considered as a low-cost volume rendering accelerator, feasible on a PCI card.

In 1995, Knittel presented the first PCI-based volume rendering accelerator – later named VIZARD – which can be plugged into any PC [10, 11]. Instead of using – back then – expensive dedicated volume memory, the main memory of the PC is used via DMA. Memory bandwidth was reduced using a Redundant Block Compression scheme. Additionally, a SRAM-based Cache which enabled the exploitation of ray-to-ray coherence was included, resulting in up to seven frames per second on a 256^3 dataset. How-

*e-mail : {miked,meissner}@gris.uni-tuebingen.de

†e-mail: urs@dd-t.com

‡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

§"Digital Design & Technology", Krämerstraße 13, D-72764 Reutlingen Germany, phone: +49 7121 43308 11, fax : +49 7121 43308 19

[1]When higher order interpolation methods are applied this might be different.

ever, ray-to-ray coherence only improved the frame-rate for close-ups and fly-throughs. The successor of VIZARD – VIZARD II [17] currently in development – uses dedicated memory to store the volume data on the PCI-card. This step has been driven by the dramatic price-reduction of standard memory modules.

In 1996 de Boer et al. presented a volume memory architecture based on RAMBUS DRAM [2]. Similar to the VIRIM system, volume data is distributed among eight memory units. Additionally, two or more SRAM caches are used for each memory unit (non-blocking caches). Depending on the size of the sub-cubes, the total cache size can range from 32 MByte to 100 MByte of SRAM. Obviously, this is not feasible within a single PCI card, nor low-cost; RDRAM requires a special interface chip (ASIC) to accomodate the 400MHz or higher clock speed and non-standard signal voltages of RDRAM. Furthermore, for a true ray-casting accelerator, short transfers are frequent and this is handled very inefficiently by the RAMBUS protocol [16].

In [6] a cubic memory addressing scheme for access to 64 binary voxel values in parallel was presented. While the cubic addressing presented here is similar much larger voxel sizes such as 32bit and the use of modern memory devices such as SDRAM are now required to enable high quality images and interactivity.

Very recently and independently, a memory architecture similar to the one in section 3.2 has been presented [23]. The architecture uses standard SDRAMs making use of the four internal banks. However, an accurate simulation and timing analysis – as given in this paper – has not been presented.

Within the remainder of this paper we will present several SDRAM based volume memory architectures suited for ray-casting based volume rendering accelerators. Section 2.1 introduces the memory terminology we use and goes through the performance enhancing mechanisms we will use in our architecture. Section 3.1 shows an "optimal" interface providing SRAM access time. A more realistic but feasible memory architecture is presented in Section 3.2. Finally, we present a feasible low-cost memory interface based on standard SDRAMs using four DIMMs in Section 3.3. In Section 4 timing results are presented and the architectures discussed. Finally, we conclude and present future work.

2 Memory Access

To describe and measure the performance of a memory architecture the addressing and behaviour of modern memory devices must be considered first. In this Section, we will first go through the use of SDRAMs for ray casting and propose a model that can be used for measuring an SDRAM's performance when used with an application that requires constant random access. We will then describe the cubic addressing scheme which uses a subcube as the basic unit of addressing, unlike traditional addressing schemes. The cubic addressing scheme will be extended to show the calculation of addresses when using eight parallel memory modules as shown in [9].

2.1 SDRAM terminology

Interactive ray casting places very specialised demands on the use of SDRAMs and a clear model of SDRAM behaviour is necessary to correctly estimate performance. All SDRAM providers readily supply datasheets that reveal a varying range of information concerning their product. This information ranges from electrical specifications right through to command descriptions. Using the device datasheets and the requirements of our application we must determine the suitability and performance of the device.

The first restraint we have is that a new read command with a random address is sent to the SDRAM every clock cycle. This means that the SDRAM must be used with a burst length of one and therefore we cannot use one of the main features of SDRAMs which is the ability to issue precharge and bank select commands while long burst reads are in progress. We want to minimise these precharge and bank select commands since they represent an increase in the memory latency. To model the operation of a memory architecture and determine its performance given these constraints and limitations we have designed a simplified SDRAM state diagram shown in Figure 1.

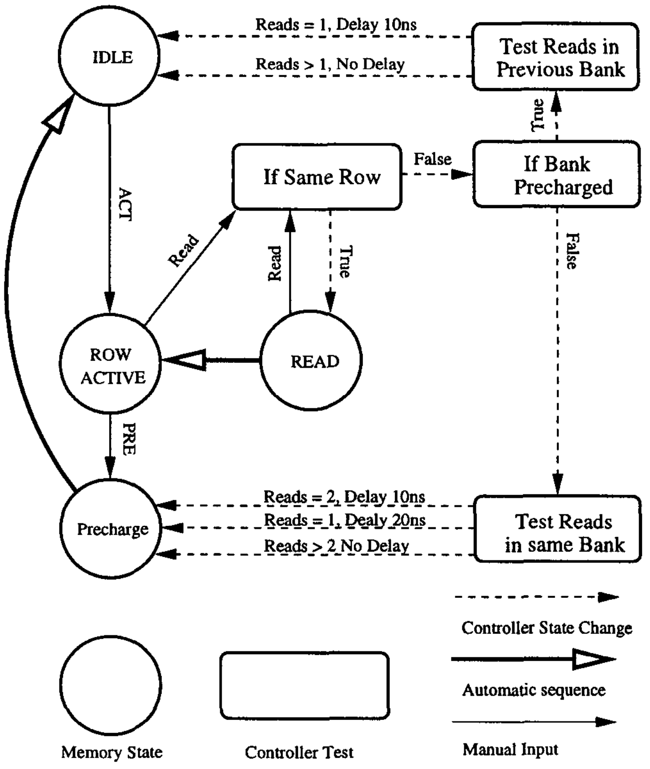

Figure 1: Simplified SDRAM State Diagram for continuous reads with burst length 1 and absolute values based on NEC SDRAM.

The SDRAM state diagram focuses on read operations only and the delays experienced in consecutive reads. At power on, the SDRAM starts in the precharge state and moves to the idle state. To save precharge time, precharge is issued for all four banks and four registers are set to indicate all banks are precharged. When a bank activate is required a check is made on the bank's precharge register to see if the bank is precharged and and if so then an activate command can be issued without precharge. Once in the idle state a row activate command is issued to select a bank and row address followed by a read command which sends the column address to the SDRAM.

Before the next read, the memory controller checks its address and compares it to the previous read to check if a bank active or precharge is required. Firstly, the memory controller checks if the next read is within the currently active row and in this case another read is issued with the next address. In this paper we will use the terms caches and pages to refer to the currently active row in one bank of an SDRAM. If it is not in the same row, then the appropriate bank will need to be activated. Before bank activation the memory controller checks the bank has been precharged, if not, a precharge will need to be issued otherwise the memory controller issues an activate command for that bank.

There are several other timing constraints that affect the bank switching time of an SDRAM. Firstly, an interleaved bank activate command cannot occur within a time period of t_{RRD}. The test for the number of reads in the previous bank ensures that this condition is met by adding a small delay if not enough time has passed while reading in the current bank. Two further conditions are the minimum time interval between successive bank activate commands to the same bank, t_{RC}, and the minimum time between an activate bank command and a precharge command t_{RAS}. These conditions can be met by adding delay time after the number of reads in the current bank is tested.

The perfomance and requirements of SDRAMs from different manufacturers can vary affecting the precharge times, bank switching times and other timing constraints. Figure 1 and the simulation results are based on the timing information for NEC 256Mbit 4 bank SDRAMs [18]. These were chosen because they have the fastest performance when continuous single reads are required. The timings are shown in Table 1. There are SDRAMs that run at speeds greater than 100MHz, but the SDRAMs performance will be limited by the board and its components onto which they are finally incorporated. Since we are aiming for a 100 MHz VIZARD II board, we have chosen the 100MHz chips.

Parameter	Time (ns)
Clock cycle time	10
Precharge time, t_{RP}	20
Row Activate, t_{RCD}	20
Read data (CAS) latency, 2	20
Minimum Times	
Activate one to activate another, t_{RRD}	20
Activate one to activate same, t_{RC}	70
Activate to precharge, t_{RAS}	50

Table 1: Characteristics for NEC SDRAM

2.2 Cubic Addressing Scheme

The main difference between traditional addressing and the cubic addressing scheme is that the basic unit for address calculation is a s^3 sub-cube instead of a single voxel. Sub-cube addressing has been used previously in the field of parallel volume rendering [15, 19, 5]. However, in this paper voxels are grouped into sub-cubes which fit into a row of an SDRAM. A cubic address first divides the x, y and z coordinates by s to find the s^3 sub-cube that the voxel is in and then finds the modulus by s of the x, y and z coordinates to determine the voxel position within the s^3 sub-cube. The address, A, is calculated using the following set of equations :

$$A = s^3 A_C + A_V$$

$$A_C = \lfloor \frac{V_x}{s} \rfloor + \frac{D_x}{s} \lfloor \frac{V_y}{s} \rfloor + \frac{D_x D_y}{s^2} \lfloor \frac{V_z}{s} \rfloor$$

$$A_V = V_x \bmod s + s(V_y \bmod s) + s^2(V_z \bmod s)$$

where,

A	is the cubic address,
A_C	is the address of the voxel's sub-cube,
A_V	is the voxel address inside the sub-cube,
s	is the size of the sub-cube,
V_x, V_y, and V_z	are the voxel coordinates,
D_x, D_y, and D_z	are the dimensions of the dataset.

If $s = 8$ and a cache size of 512 voxels is used, one sub-cube of 8^3 voxels is stored in the cache. Assuming the traditional method of calculating the address, A, is $x + ny + n^2 z$ is used and the cache is filled linearly then a ray travelling parallel to the x axis would only require a cache refresh at the beginning of the ray and have a 100% cache hit to miss ratio. But when the ray traces parallel to the z axis and a cache refresh is required for every trilinear neighbourhood read the cache is missed every time. When comparing cubic addressing to traditional memory addressing the cache results for ray casting along the x,y and z axes all have a 87.5% cache hit to miss ratio since every eighth voxel requires a cache refresh.

2.3 Parallel Memory Access

Figure 2 shows the arrangement of data in eight parallel memory modules. In Figure 2, MO to $M7$ represent the eight parallel memories and for each sample point each one of the eight parallel memories is required to deliver one voxel. If (V_x, V_y, V_z) is divisable by two then the coordinates used to calculate the address for each memory are the original coordinates divided by two. The calculation of the addresses becomes more complex when the sample point is between two neighbourhoods of the previously mentioned case. This is discussed in [9] and the base address of a neighbourhood is modified using an address modification unit (incrementer) before each memory bank. The combination here of both Cubic Addressing and Parallel Memory Access results in address modification affecting the calculation of both the sub-cube address and voxel address within the sub-cube. Therefore the recalculation of only the voxel coordinates for each parallel memory is presented.

For example, consider the set of new memory addresses for sample $S2$ compared to the addresses for sample $S1$, where samples are depicted as a cross marked on the ray in Figure 2. If $M0$ for sample $S1$ is the origin then all eight memory addresses are identical. But, the address calculations for sample $S2$ will result in new addresses for memories $M0$, $M1$, $M2$ and $M3$, while the addresses for memories $M4$, $M5$, $M6$ and $M7$ will remain the same.

When calculating the eight memory addresses for the current sample point, the sample address must first be divided by two before memory addresses are calculated (implemented as a shift operation), because each memory stores only every second voxel. For example, given that memory $M0$ is at the origin, for each memory address calculation the sample's x coordinate must be divided by two and the modulus by two of the x coordinate added to it. For memory $M1$ the sample's x coordinate only needs to be divided by two. The newly calculated coordinates to be used when calculating the address for each of the eight memories are as follows :

$$M_x = \frac{V_x}{2} + V_x \bmod 2$$

$$M_y = \frac{V_y}{2} + V_y \bmod 2$$

$$M_z = \frac{V_z}{2} + V_z \bmod 2$$

$$U_x = \frac{V_x}{2} \qquad U_y = \frac{V_y}{2} \qquad U_z = \frac{V_z}{2}$$

$$\begin{array}{ll} M0 : (M_x, M_y, M_z) & M1 : (U_x, M_y, M_z) \\ M2 : (M_x, U_y, M_z) & M3 : (U_x, U_y, M_z) \\ M4 : (M_x, M_y, U_z) & M5 : (U_x, M_y, U_z) \\ M6 : (M_x, U_y, U_z) & M7 : (U_x, U_y, U_z) \end{array}$$

2.4 Memory Access Buffering

Using the cubic addressing scheme described above, consider the situation when the ray in Figure 2 crosses the y-z plane of the next s^3 sub-cube of voxels and causes memories $M0$, $M2$, $M4$ and $M6$ to change their coordinates and refresh their cache with the values of the next s^3 sub-cube. At this point the memories $M1$, $M3$, $M5$

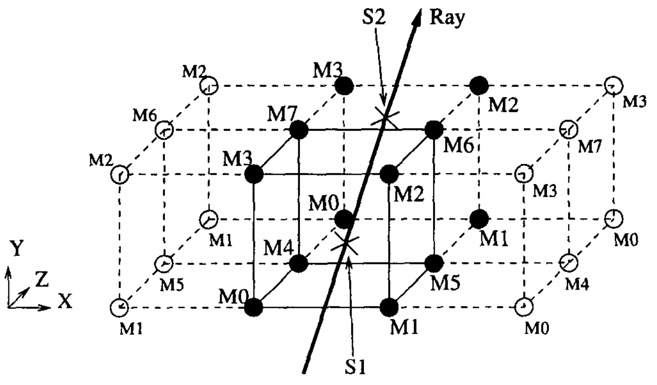

Figure 2: The positioning of voxels in eight parallel memory banks and two sample points along a ray.

and $M7$ will not yet have left their current sub-cube of voxels and so have not had a cache refresh. If the individual memory cache refreshes described above happen in separate cycles of the pipeline, then the pipeline will have to stall twice. Once for the first four memories to refill their caches and also at a later time when the second four memories refill their caches. This is a simplified example and depending on the direction in which the ray crosses between sub-cubes a particular memory can have up to 2 cache misses consecutively. To minimise the effect of cache misses and subsequent pipeline stalling a FIFO buffer is introduced into a ray casting pipeline as shown in Figure 3.

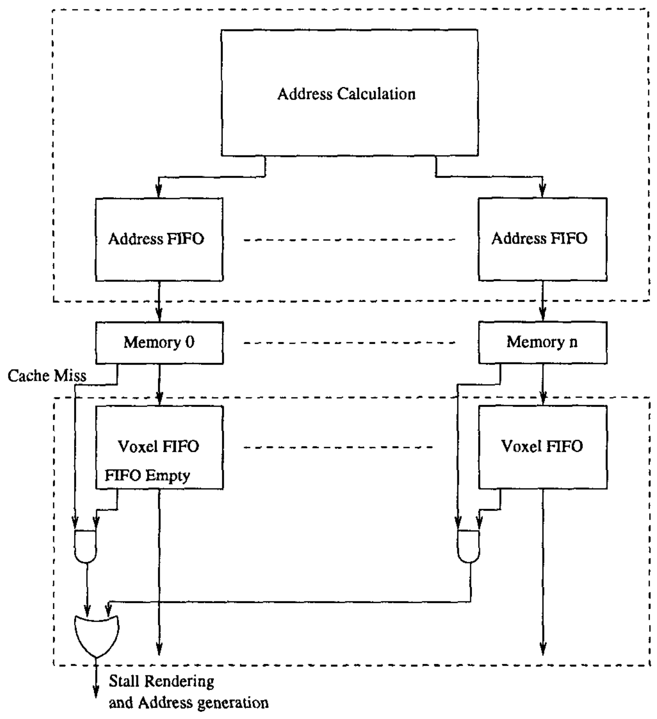

Figure 3: The FIFO buffers for addresses and voxel values.

With this buffering the only time the pipeline must stall is when the voxel FIFO of a particular memory is empty at the same time that this memory requires a cache refill. The frame time is now dependent on only the performance of the slowest SDRAM, whereas with unbuffered memory access the frame time is dependent on the sum of the slowest SDRAM access at every read of a trilinear neighbourhood.

3 Memory Architectures

The memory architecture chosen for implementation is dependent on a number of factors which are constantly changing given currently available technology. This section presents three memory architectures and shows the progression from solutions with optimal performance and poor feasibility to increased practicalility using four DIMMs and maintaining high performance.

3.1 Sixty Four Memories

An ideal solution for an arbitrary, but conflict free memory access without any danger of pipeline stalls is the use of 64 independantly accessible SDRAMs. For simplicity, the principle idea is depicted in 2D with 16 SDRAMs in Figure 4. The four currently accessed SDRAMs are highlighted (0,1,2,3). Ensuring there is no access penalty while crossing a page boundary, the 2×2 Neigborhoods across the page boundary have to be in four of the other 12 SDRAMs not currently accessed. SDRAMs 4-15 can be precharged and activated while accessing SDRAMs 1-4, so the new voxels are available within the next clock cycle when crossing the page boundary. This configuration works for two principal axes directions, with 64 SDRAMs, it can be extended for accesses along

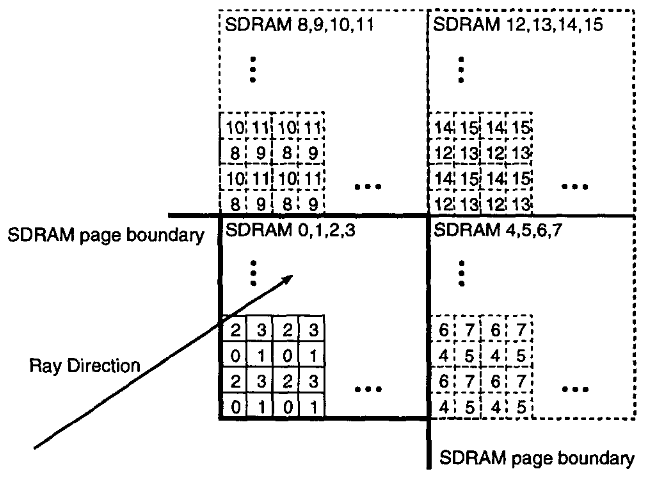

Figure 4: Memory access across page boundaries with 64 SDRAMs. Interpolation neigborhoods consist of 2×2 squares, the numbers denote SDRAM delivering the voxel.

all three principal axes directions, no matter if we use cubic addressing or not. While giving us the fastest possible access time, this solution has a severe limitation. It requires the calculation and distribution of 64 independant addresses which is expensive to build and exceeds the board space budget for a PCI-Card by far.

One alternative is to simply use 8 SDRAMs, which is the minimum required to fetch a trilinear interpolation subcube with one memory access. While this is more likely to fit on a PCI board, the solution suffers from unbalanced access times and a smaller solution is possible if we employ the banks in SDRAMs.

3.2 Eight Logical Memories

By taking advantage of the shorter switching times between banks in modern SDRAMs, a near optimal and more practical solution using eight logical memory banks can be used. If we take the row size of one bank of a SDRAM and place the neighbouring subcubes in the x,y and z directions into neighbouring SDRAM banks we can place three into the same SDRAM and use a second SDRAM to provide four more banks for the remaining neighbouring subcubes, assuming 4 banks per SDRAM. This results in another level of cubic hierarchy in the memory architecture. If we want to have 32 bits per voxel as proposed in the original VIZARD II design then two 16bit wide devices will be required per logical memory. The resulting layout of the memory architecture and the positioning of SDRAM banks in the dataset's coordinates is shown in Figure 5. When a ray passes into the neighbourhood of the eight banks a precharge is issued across all eight banks so that a switch between banks will only take as much time as a bank activate instead of a precharge and activate command. This design means that the same results can be expected in each of the three principal axes directions. Using the distance between voxels in the original dataset the effective size of each individual bank, A - H in Figure 5, is 16^3 and the size of the cube between precharge commands is now 32^3.

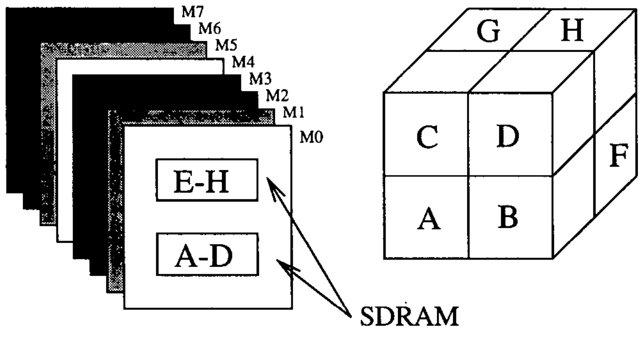

Figure 5: Memory architecture using 8 logical memories.

Vetterman et. al. [23] plan to build exactly this memory structure for their new interactive volume rendering architecture. However, an important consideration when building any hardware accelerator is the maximum speed of the system and effectively the speed that the memory can operate at. We would prefer to make use of low cost off-the-shelf memory boards and maintain a system clock of 100MHz by having a simpler overall design.

3.3 Four DIMMs

When considering the previously presented memory architectures, it is apparent that favourable access times can be achieved by exploiting the SDRAM caches to store large voxel neighborhoods. However, for a low-cost single PCI board solution, these architectures are not well suited.

Targeting low-cost, makes the use of standard off-the-shelf components mandatory. Dual In-line Memory Modules (DIMMs) are readily available, extremely cheap, and in daily use. Another advantage is that DIMMs can easily be exchanged allowing for "upgradability" of the memory. However, aiming for a single PCI board solution constrains the possible implementation. In discussions with industrial partners it has become clear that more than four DIMMs is not realistic for a single PCI board solution. A memory architecture using four DIMMs is shown in Figure 6.

Using only four DIMM modules limits the number of individual memory addresses. Hence, fetching the eight voxels – required for

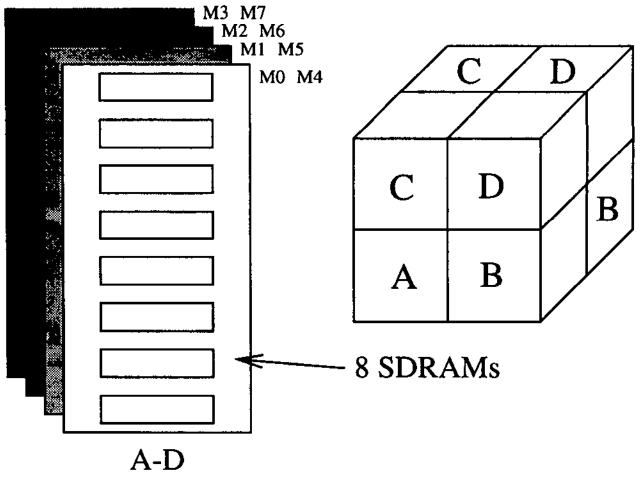

Figure 6: Memory architecture using 4 DIMMs.

trilinear interpolation – in parallel, is only feasible using either two cycles (as presented in [17]) or by replicating data. The memory addresses used to fetch a trilinear neighbourhood using four DIMMs in shown in Figure 7 Although data replication is not a desirable solution, the feasibility means that it is an acceptable trade off. Furthermore, DIMM modules are relatively inexpensive, for example, a dataset of 512^3 voxels with eight bit data stored using data replication would be in the range of a few hundred US$.

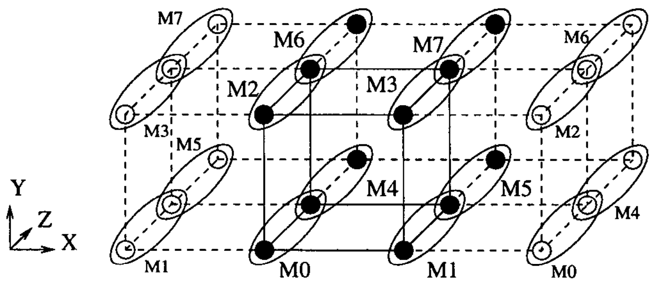

Figure 7: The addressing values for voxels in the four DIMMs memory architectre. Address value Mn is sent to DIMM (n mod 4).

As a result of the data replication, we need to store two consecutive voxels instead of one. This means an addressing scheme where only four addresses are calculated is required. These four addresses are the same as those presented in Section 2.3 and will be addresses $M0 - M3$ when the z coordinate is divisable by two and $M4 - M7$ when it is not. Additionally, only four internal banks per SDRAM on the DIMM are available (A-D), and hence, the voxel neighborhood contained over all caches of all memory devices is reduced by 50% compared to the previous architecture. This increases the number of bank activates and precharge commands therefore reducing the average access-time. However, as is shown in Section 4, this reduction has hardly any impact on the frame-rate, due to the relatively large page caches.

4 Results

To show the benefits of using buffered memory accessing and determine the performance difference between the presented memory architectures (sixty four memories, eight logical memories, and four DIMMs), a software simulation was used to determine timings using the SDRAM model as described in Section 2.1 was applied.

To gain a good perspective on the performance that a system would give, the results have been averaged over a set of twenty representative views. The views are rendered with their view direction determined by the center point of each triangle in an icosahedron centered around the dataset. Furthermore, all measurements have been performed on a total of three different datasets, which are:

Foot: This is a CT scan of the front part of a human foot with 256^3 voxels. Images of this dataset are given in Figure 8(a) and (b). Two different materials can be classified; Bone and flesh.

Arteries: CT angiography has been used as the acquisition device to generate this dataset of 256^3 voxels. A single material – the blood vessels of one half of the human brain – can be classified, see Images in Figure 8(c) and (d).

Statue: The foot of a bronze statue has been scanned, resulting in a dataset of $341^2 \times 91$ voxels with a non uniform spacing of $1, 1, 0.25$. Up to four materials can be classified; Bronze, an iron cylinder which is behind the foot, resin, and plaster. Images of this dataset are shown in Figure 8(e) and (f).

For each presented memory architecture, measurements have been performed casting 256^2 rays onto each of the datasets using perspective projection. As a result, an average frame rate, average memory access time, and cache hit ratio have been calculated using the twenty views, as mentioned above. The results of these measurements are shown in Table 2.

On average, the number of frames per second achievable if in each voxel neighbourhood a sample is generated – assuming a 256^3 dataset – is 5 for buffered memory and 4 for unbuffered memory. However, due to early ray termination and perspective projection much higher frame rates can be achieved, except the worst case where all voxels values are classified semi-transparent, thus preventing any early ray termination at all. Fortunately, for most application fields, this happens rarely. Overall, this frame rate gives an upper bound to the number of samples possibly being generated per second. The last row of the table using 256^3 samples shows frames per second of 7 for buffered and 6 for unbuffered where the speed up is due only to the reduction in the number of samples due to perspective projection.

The buffered accessing scheme as presented in Section 2.4 performs very well, as shown in Table 2. For all datasets, the average access time is reduced and hence more samples can be generated which adds between two and four more frames per second. The Four DIMMs architecture doesn't need buffering in z direction, therefore buffering does not improve frame-rates as much as for the eight logical memories. In the above described case, where in each sample neighbourhood a sample is generated, the access buffering increases the frame rate by 20%.

An advantage of the memory architectures are the caches once applying oversampling. Figure 8(f) shows the impact of oversampling along the viewing direction and Figure 8(d) for oversampling in all dimensions. In general, higher sampling rates along the cast rays decrease the average frame rate according to the oversampling rate. At the same time, the average access time to the SDRAMs is decreased because more samples occur within the same cache. This increases the overall cache hit ratio and hence, the frame rate is not reduced by 50 % but only by 40 %, as it can be seen in Table 2 (Statue 1 and Statue 2). The average cache hit ratio increases from approximately 90 % to 95 % while at the same time reducing the impact of the access buffering scheme.

As expected, the non feasible memory architecture using sixty four memories always reaches the optimum of 10 ns and a 100 % cache hit ratio. However, the difference between the results for the four DIMMs and eight logical memories is always less than 1 frame per second. As it can be seen, the four DIMMs architecture can be used with virtually no loss in performance but large gains in practicality, implementability and upgradeability, thus enabling a single PCI board solution running at full 100MHz.

5 Conclusion

We presented a low-cost memory architecture based on off-the-shelf DIMM modules running at 100 MHz. Using parallel accesses, a cubic addressing scheme, FIFOs accomodating memory latency, and data replication, we achieve high frame-rates. Averaged over a set of representative views, up to 9Hz can be achieved. Using such an onboard memory interface makes ray-to-ray coherence caches – as presented in [11, 17] – redundant. The presented memory architecture will be part of the VIZARD II system (first prototype estimated for fall 1999).

So far, early ray termination has been applied as the only algorithmic optimization. However space-leaping can greatly increase the frame-rate by skipping homogeneous areas. Due to the latency of current SDRAMs (addressing to data out), the integration of a threading scheme as presented by [23] might be worth-while and its integration is currently being investigated.

Acknowledgements

This work is funded by the Commission of the European Communities (CEC) and by the German Research Council (DFG) grant SFB 382. Thanks to Dr. Bernhard Illerhaus from the Bundesanstalt für Materialforschung und Materialprüfung, Berlin for the Statue dataset and Roland Proksa from Philips Research, Hamburg for the Arteries and Foot datasets.

References

[1] I. Bitter and A. Kaufman. A ray-slice-sweep volume rendering engine. In *Proceedings of the 1997 EUROGRAPHICS/SIGGRAPH Hardware Workshop*, Los Angeles, CA, 1997.

[2] M de Boer, A Gröpl, J Hesser, and R Männer. Latency and hazard-free volume memory architecture for direct volume rendering. In *Eurographics Workshop on Graphics Hardware*, pages 109–119, August 1996.

[3] Michael Doggett. An array based design for real-time volume rendering. In *Eurographics Workshop on Graphics Hardware*, pages 93–101. EuroGraphics, August 1995.

[4] T. Günther, C. Poliwoda, C. Reinhart, J. Hesser, R. Männer, H.-P. Meinzer, and H.-J. Baur. VIRIM: A massively parallel processor for real-time volume visualization in medicine. In *Eurographics workshop on Graphics Hardware*, pages 103–108, September 1994.

[5] W. M. Hsu. Segmented ray-casting for data parallel volume rendering. In *Proceedings of the 1993 Parallel Rendering Symposium*, pages 7–14, San Jose, CA, 1993.

Dataset	Memory Type	Av. FPS		Av. Time (ns)		Cache hits	Image
		Early Ray Termination					
		Buffered	Unbuffered	Buffered	Unbuffered		
CT foot	Sixty Four Memories	12.3	12.3	10	10	100	8(a)(b)
	Eight Logical Memories	11.0	8.3	11.2	14.9	95.3	
	Four DIMMs	9.7	8.4	12.7	14.7	90.7	
Arteries	Sixty Four Memories	10.6	10.6	10	10	100	8(c)
	Eight Logical Memories	9.5	7.2	11.2	14.8	95.3	
	Four DIMMs	8.4	7.2	12.7	14.7	90.8	
Statue 1	Sixty Four Memories	7.4	7.4	10	10	100	8(e)
	Eight Logical Memories	6.7	5.1	11.2	14.7	95.4	
	Four DIMMs	5.9	5.1	12.6	14.6	91.1	
Statue 2	Sixty Four Memories	5.0	5.0	10	10	100	8(f)
	Eight Logical Memories	4.8	4.1	10.6	12.4	97.6	
	Four DIMMs	4.5	4.1	11.3	12.4	95.4	
256^3	Sixty Four Memories	9.8	9.8	10	10	100	
	Eight Logical Memories	8.7	6.6	11.2	14.8	95.4	
	Four DIMMs	7.7	6.6	12.7	14.7	90.9	

Table 2: Memory timing for three different datasets.

[6] D. Jackel. The graphics parcum system: A 3d memory based computer architecture for processing and display of solid models. *Computer Graphics Forum*, 4(1):21–32, 1985.

[7] U. Kanus, M. Meißner, W. Straßer, H. Pfister, A. Kaufman, R. Amerson, R. J. Carter, B. Culbertson, P. Kuekes, and G. Snider. Implementations of Cube-4 on the teramac custom computing machine. *Computers & Graphics*, 21(2):199–208, 1997.

[8] Arie Kaufman and Reuven Bakalash. Memory and processing architecture for 3D voxel-based imagery. *IEEE Computer Graphics and Applications*, 8(11):10–23, November 1988.

[9] Günter Knittel. VERVE : Voxel Engine for Real-time Visualization and Examination. *Computer Graphics Forum*, 12(3):37–48, 1993.

[10] Günter Knittel. A PCI-based volume rendering accelerator. In *Eurographics Workshop on Graphics Hardware*, pages 73–82, August 1995.

[11] Günter Knittel and Wolfgang Straßer. VIZARD: Visualization accelerator for realtime display. In *1997 Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 139–147, August 1997.

[12] K. Kreeger and A. Kaufman. PAVLOV: A Programmable Architecture for Volume Processing. In *Proc. of Eurographics/SIGGRAPH workshop on graphics hardware 1998*, pages 77–86, Lisboa, Portugal, 1998.

[13] B. Lichtenbelt. Design of a high performance volume visualization system. In *Proceedings of the 1997 Eurographics/SIGGRAPH Hardware Workshop*, Los Angeles, CA, 1997.

[14] J. Lichtermann. Design of a fast voxel processor for parallel volume visualization. In *Proceedings of the 10th Eurographics Hardware Workshop*, pages 83–92, Maastricht, The Netherlands, 1995.

[15] K. Ma, J. Painter, C. Hansen, and M. Krogh. A Data Distributed Parallel Algorithm for Ray-Traced Volume Rendering. In *Proceedings of IEEE Symposium on Parallel Rendering*, pages 15–22. ACM Press, October 1993.

[16] Joel McCormack, Robert McNamara, Christopher Gianos, Larry Seiler, Norman P. Jouppi, and Ken Correll. Neon: A single-chip 3d workstation grapihcs accelerator. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 123–132, August 1998.

[17] Michael Meißner, Urs Kanus, and Wolfgang Straßer. VIZARD II, A PCI-Card for Real-Time Volume Rendering. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 61–67, August 1998.

[18] NEC, http://www.necel.com. *μPD45256163 256M-bit Synchronous DRAM*, 1998.

[19] U. Neumann. Parallel volume-rendering algorithm performance on mesh-connected multicomputers. In *1993 Parallel Rendering Symposium Proceedings*, pages 97–104, San Jose, CA, October 1993.

[20] Randy Osborne, Hanspeter Pfister, Hugh Lauer, Neil McKenzie, Sarah Gibson, Wally Hiatt, and TakaHide Ohkami. EM-Cube: an architecture for low-cost real-time volume rendering. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 131–138, August 1997.

[21] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. In *Computer Graphics, Proc. of SIGGRAPH 99*. ACM, 1999.

[22] Hanspeter Pfister and Arie E. Kaufman. Cube-4 - A scalable architecture for real-time volume rendering. In *1996 Volume Visualization Symposium*, pages 47–54. IEEE, October 1996.

[23] B. Vettermann, J. Hesser, and R. Männer. Solving the hazard problem for algorithmically optimized real-time volume rendering. In *International Workshop on Volume Graphics*, March 1999.

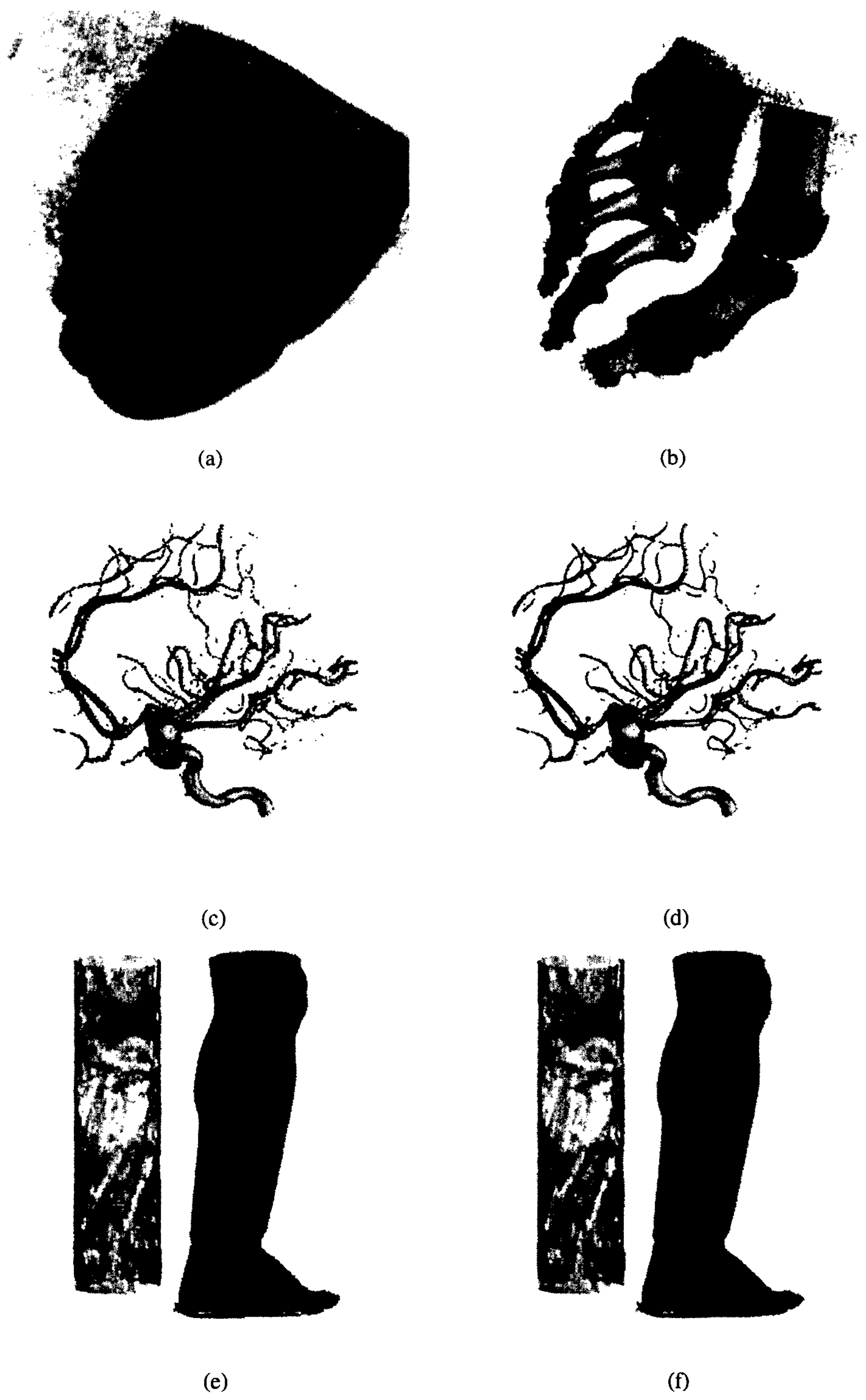

Figure 8: The dataset of (a) and (b) is a CT scan of the upper part of a human food. The dataset is of size 256 × 256 × 256, casting 256 × 256 rays. (a) Bone is displayed in opaque white and meat in pink using high transparency. (b) Bone only. (c) and (d), a CT angiography scan of human brain vessels is shown. The dataset is of size 256 × 256 × 256 casting 256 × 256 rays. (c) unifrom sampling in all three dimensions. (d) oversampling in all dimension by two. Finally, (e) and (f) show a dataset of a foot of a statue casting 371 × 371 rays. The dataset is of size 341 × 341 × 91 with spacing 1, 1, 0.25. (e) Uniform sampling in all dimensions. (f) Oversampling in viewing direction by a factor of two.

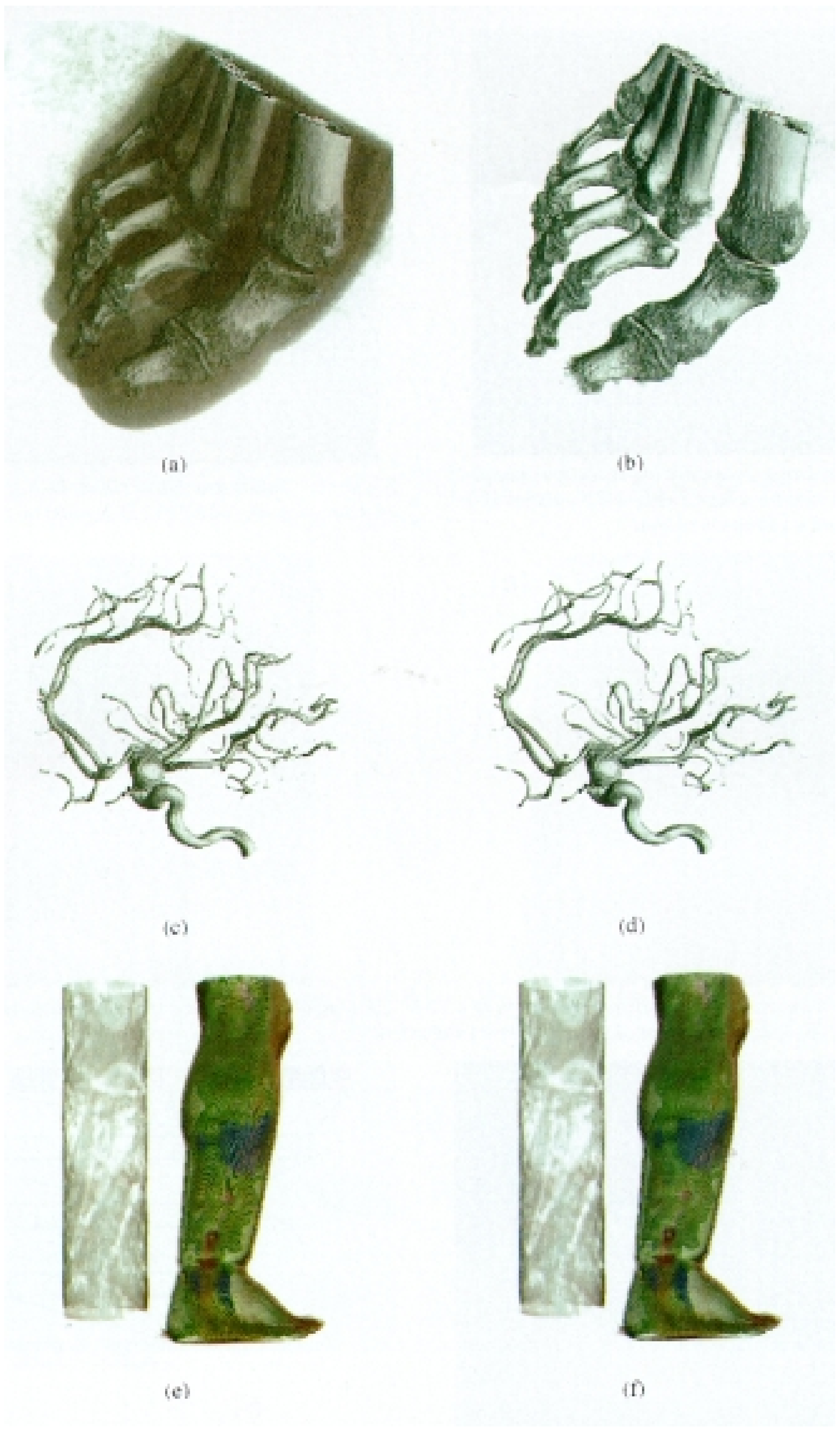

Figure 8: The dataset of (a) and (b) is a CT scan of the upper part of a human food. The dataset is of size $256 \times 256 \times 256$, casting 256×256 rays. (a) Bone is displayed in opaque white and meat in pink using high transparency. (b) Bone only. (c) and (d), a CT angiography scan of human brain vessels is shown. The dataset is of size $256 \times 256 \times 256$ casting 256×256 rays. (c) unifrom sampling in all three dimensions. (d) oversampling in all dimension by two. Finally, (e) and (f) show a dataset of a foot of a statue casting 371×371 rays. The dataset is of size $341 \times 341 \times 91$ with spacing 1, 1, 0.25. (e) Uniform sampling in all dimensions. (f) Oversampling in viewing direction by a factor of two.