Efficient Space Leaping for Ray casting Architectures

M. Meißner[*] M. Doggett[*] J. Hirche[*] U. Kanus[†] W. Straßer[*]

Abstract

One of the most severe problems for ray casting architectures is the waste of computation cycles and I/O bandwidth, due to redundant sampling of empty space. While several techniques exist for software implementations to skip these empty regions, few are suitable for hardware implementation. The few which have been presented either require a tremendous amount of logic or are not feasible for high frequency designs (i.e. running at 100 MHz) where latency is the one of the biggest issues.

In this paper, we present an efficient space leaping mechanism which requires only a small amount of SRAM (4 Kbit for a 256^3 volume) and can be easily integrated into ray casting architectures. For each sub-cube of the volume, a bit is stored in an occupancy map, which can be generated in real-time, using the VIZARD II architecture. Hence, space leaping can be classification dependent achieving yet another significant speed-up over skipping only the empty space (voxel $= 0$). Using a set of real-world datasets, we show that frame-rates well above 15 frames per second can be accomplished for the VIZARD II architecture.

Keywords: Graphics hardware, volume rendering accelerator, ray casting, space leaping.

1 INTRODUCTION

Ray casting is one of the most popular volume visualization techniques. Generally, samples along cast rays are interpolated, classified, shaded, and composited. The two main problems involved with ray casting are the computational complexity and the memory I/O.

The shear-warp algorithm [9] reduces memory I/O to an absolute minimum by visiting each voxel, at a maximum, only once per frame. Furthermore, by using three run length encoded datasets and processing data only on the given slices, almost interactive frame-rates are achieved. However, the generation of the run-length encoded volume is classification dependent and requires significant pre-processing time. Furthermore, for datasets where most voxels contribute, the frame-time is dominated by the computations needed for each voxel and is in the order of many seconds for a 256^3 volume. Hence, for a large number of contributing samples, hardware acceleration is mandatory to provide interactivity for real-time frame-rates.

The VolumePro board [11] uses a similar approach to the shear-warp algorithm. Projecting data onto a base-plane, memory I/O is reduced to a minimum by visiting each voxel exactly once. The system is fully pipelined and therefore, delivers classification independent real-time frame-rates. However, no algorithmic optimizations are integrated and hence, for many datasets most of the cycles are wasted sampling empty space which could potentially be used to accomplish even higher frame-rates or potentially real-time frame-rates for larger volumes. Furthermore, only parallel projections are supported.

Knittel [8] presented a true ray-casting architecture capable of providing almost interactive frame-rates for 256^3 datasets. The PCI based accelerator uses the main memory to store the volume data. Before rendering, the data is pre-shaded and compressed in software using a lossy compression scheme. Each ray is then processed by adding the ray increment and fetching the corresponding voxel package. An octant based distance coding scheme, stored in the empty voxel blocks, is used to skip empty octants by storing a factor to multiply the increment vector with [7]. Generally, due to the ray casting process, voxels are accessed multiple times but the access latency of the PCI bus is reduced by using an on-board SRAM cache.

The successor of this architecture, VIZARD II [10], has dedicated volume memory providing a much higher memory bandwidth than [8]. This architecture uses a prefetching scheme presented in [3] which overlays the prefetch time of the different memory modules and reduces memory access time to a minimum. Using this memory scheme, a higher frame-rate can be delivered than in [8]. However, no space leaping has been integrated and voxel values are again fetched multiple times.

Vettermann et al. [14] proposed a ray casting architecture which can integrate algorithmic optimizations such as early ray termination and space leaping, hiding most of the latency. In this architecture, an additional distance volume is used to store the Euclidean distance to the closest contributing voxel. Since the distance volume is stored in SDRAM, latency is high. This is circumvented by interleaving the processing of multiple rays. However, this significantly aggravates the task of coordinating the memory accesses and requires a significant amount of logic for implementation. Furthermore, once a certain threshold of remaining rays is reached, stall cycles are introduced. Nevertheless, the presented space leaping mechanism works effectively to solve the associated latency problem.

The proposed RACE architecture [12] exploits early ray termination and estimates real-time frame-rates for parallel and perspective projection. Space leaping capabilities have not yet been integrated but the authors state that succeeding architectures will include this and an acceleration of a factor of two is expected [12].

Overall, the architectures mentioned so far either do not provide space leaping functionality [11, 10, 3, 12] or require an entire distance volume to be pre-computed [8] and stored [14]. Generally, the generation of distance volumes has been widely investigated for CSG [1], volume rendering [13, 16], haptics in volume rendering [4], iso-surface extraction [6] and others. However, the computational costs are not neglectable and for 8 bit distance and voxel values, the memory requirements are doubled which is impractical for larger volumes.

There have also been numerous publications on approaches which do not require distance volumes but make use of volume animation [5, 15]. Here, in addition to storing the color image, a depth image is generated. This depth image is warped and used to determine valid starting points within the volume. Yagel et al. [15] state that their technique does not suffer from image degradation. However, ray-casting as used in the above architectures is purely sampling based and can miss fine detail – unless infinite sampling

[*]WSI/GRIS (Wilhelm Schickard Institut für Informatik, Graphisch Interaktive Systeme, University of Tübingen, Auf der Morgenstelle 10/C9, D-72076 Tübingen, Germany, E-mail:{meissner,miked,jhirche,strasser}@gris.uni-tuebingen.de

[†]DD&T GmbH ("Digital Design & Technology"), Krämerstraße 13, D-72764 Reutlingen, Germany, E-mail: urs@dd-t.com

frequency is used – which results in an invalid depth image. Therefore, for ray-casting driven architectures, such an approach cannot be used without potential image degradation.

In contrast, our new space leaping mechanism does not require an entire distance volume nor does it suffer from image degradation. It is extremely well suited for hardware implementation, because it only uses a small occupancy map. Since the occupancy map can be computed in real-time, it can be classification dependent, yielding much better results than coding empty voxels only (voxel = 0). Compared to [14], our approach takes more cycles to jump over empty space extending over several sub-cubes, it introduces much less latency – even for a 100 MHz design – since the occupancy map is very small (4 Kbit for a 256^3 volume) and can be stored in an on-chip SRAM separately from the main voxel memory. This facilitates any ray queuing scheme since only a few rays need to be processed simultaneously and only a small amount of logic is needed to coordinate the memory access of these few rays within the SDRAM pages.

In the following section, we will describe our space leaping mechanism using an occupancy map. In Section 3, the integration of this method into the VIZARD II architecture is demonstrated. However, occupancy maps can easily be integrated into other ray casting architectures, i.e. [14] and possibly [11]. Finally, we present an analysis of the achievable frame-rates using a set of real-world datasets and conclude our paper in Section 5.

2 SPACE LEAPING ALGORITHM

Ray casting using space leaping is performed by advancing the ray to the next sample point of interest instead of incrementing the ray's current sample position by a small increment. We propose to only skip sub-cubes of empty space, thus reducing the complexity of precomputation significantly. Our algorithm also calculates an estimated conservative distance to the next sample point within the next sub-cube, if the current sub-cube is empty.

2.1 Basic Algorithm

Initially, we map the sub-cubes of interest within the dataset by calculating a small occupancy map containing a single bit per sub-cube of the volume dataset. The bit indicates whether a sub-cube is empty or contains data requiring sampling. This leads to a very space efficient discretized representation of the volume, i.e. for a 256^3 volume and sub-cubes of 16^3 voxels, a total of 4 Kbits of memory is required. In Figure 1, a simple example of an occupancy map is given for the two dimensional case.

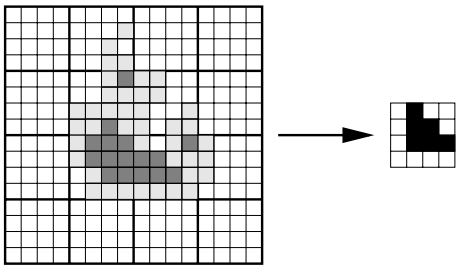

Figure 1: The Space-Leap-Volume for the 2D case.

For each sample along the ray, the corresponding bit in the occupancy map is checked using the upper address bits of the sample position. If the entry in the occupancy map indicates a non-empty sub-cube, sampling along the ray is simply continued. Otherwise, this sub-cube will be skipped by a distance, conservatively approximated to the first sample within the subsequent sub-cube by adding the increment vector multiplied by the amount of samples to be skipped. The number n_s of samples the ray may skip can be

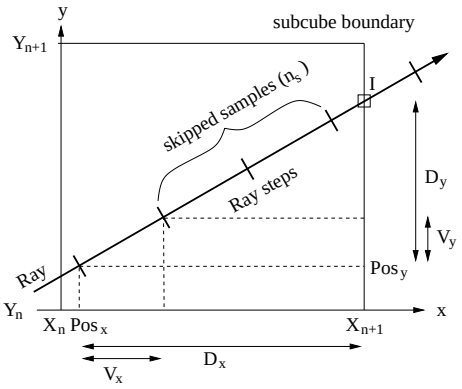

Figure 2: Calculation of the skipping

calculated by dividing the distance from the actual ray position to the intersection point (I_x, I_y, I_z) of the ray with the next sub-cube boundary by the length of one ray increment:

$$n_s = \left\lceil \frac{\sqrt{(I_x - P_x)^2 + (I_y - P_y)^2 + (I_z - P_z)^2}}{\sqrt{V_x^2 + V_y^2 + V_z^2}} \right\rceil$$

with (P_x, P_y, P_z) being the actual ray position and V_x, V_y and V_z the ray increment in the corresponding direction, as shown in Figure 2 for the x coordinate. Since this involves the calculation of two Euclidean distances, which would be extremely expensive, a simpler approach was chosen for implementation.

The same result can be obtained if the relative distance to the boundaries in each direction n_s^i ($i \in \{x, y, z\}$) is calculated. The minimum value determines the amount of steps that have to be taken to reach into the next sub-cube.

$$n_s^i = \begin{cases} 1 & \text{, if } V_i = 0 \\ \left[\frac{D_i}{V_i}\right] & \text{, else} \end{cases}$$

$$n_s = min(1, n_s^i)$$

In order to increase the space leaping efficiency even further, a second level of hierarchy can be added. The entries of eight neighboring sub-cubes ($2 \times 2 \times 2$) can be grouped into an eight bit entry in the occupancy map. When all eight entries are zero (the entire byte is zero), the whole group of sub-cubes may be skipped, achieving a higher space leaping efficiency since more samples can be skipped in less cycles.

Generally, using the above described space leaping algorithm does not require any voxel to be accessed as long as samples along the ray are "taken" in empty sub-cubes. Thus, frequent swapping of memory pages due to rays stepping once in a sub-cube and skipping to the next sub-cube, is prevented increasing the overall memory utilization significantly.

3 HARDWARE IMPLICATIONS

The additional costs in terms of computational hardware to implement our space leaping algorithm are quite modest. In this section, we will present the hardware units required to calculate the increment when skipping a sub-cube and introduce ray queuing into the ray casting pipeline to handle the latency of the increment calculations.

3.1 Space Leaping Hardware

To avoid calculating the exact distance in 3D space, we only calculate the relative distance along each axis as described in the previous section. The pipeline for computing the new increment used if a sub-cube is skipped is shown in Figure 3. The first operation

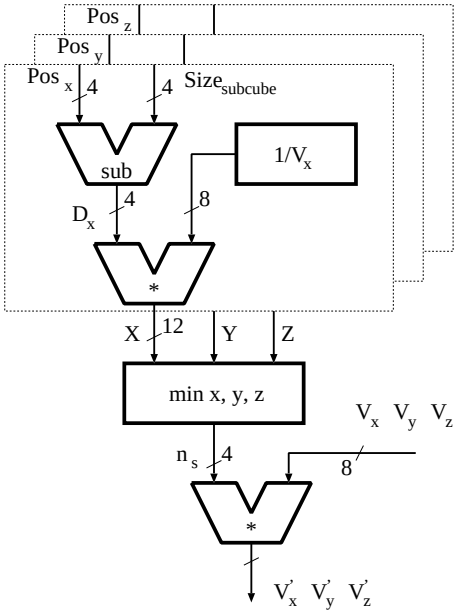

Figure 3: Pipeline for the skip calculator.

in the pipeline involves finding the integer distance to the next sub-cube. If we use 16^3 sub-cubes, a four bit subtraction is necessary to calculate each of D_x, D_y and D_z.

The second operation is to divide D_x, D_y and D_z by the ray increments V_x, V_y and V_z. To reduce the complexity of this division, we can pre-calculate the inverse of the V_x, V_y and V_z values once per ray and store the values, or store a larger adaptive division Table covering the entire range of possible ray increment values. The division is then calculated by multiplying D_x, D_y and D_z by the pre-calculated $1/V_x, 1/V_y$ and $1/V_z$ value. This multiplier does not require a very high precision since we only compare its result with the results from the other relative directional increments to determine the minimum number of samples the ray may skip n_s. As long as the value is rounded down to the nearest integer, we might step a slightly shorter number of steps, slowing the overall ray casting marginally but reducing the size and latency of this multiplier.

The results from each component are then compared to determine the minimum number of steps n_s, which is then multiplied by the actual increment V along the ray. This second multiplier must have at least the same precision as the increment value, or rounding errors and possible image artifacts will result. We have found that with increment values with only 6-bit precision, image artifacts are present, so at least an 8-bit precision for the increment value is required. The newly calculated increment value is then added to the current address.

At the same time as the new increment value is calculated, the decision whether to skip the data is looked up from a small SRAM containing the occupancy map. If the sub-cube is to be skipped, then a mux selects the newly calculated increment value or the original increment value and sends it to the Ray Queue.

The occupancy map can be easily recalculated by the VIZARD II ray casting architecture by scanning over the entire volume. The VIZARD II architecture provides an extremely high memory bandwidth capable of scanning a 256^3 dataset in 20 ms[1], allowing interactive control of transfers functions to work with the space leaping hardware. Scanning for voxels $= 0$ can be performed during down-load of the volume, as the data will pass through the Xilinx FPGA, which can compute the occupancy mask as the down-load proceeds. A possible extension is to store multiple occupancy maps, one based on skipping voxels $= 0$ and others based on different classifications.

The logic to implement the skip calculator described above requires 124 CLB slices of a Xilinx Virtex XCV1000 FPGA, utilizing 1 % of the FPGA logic, not counting the required memory[2]. The clock frequency of the resulting pipeline is well above 100 Mhz and adds 8 cycles of latency to the address computation. Therefore, the processing of eight or more rays will be sufficient to accommodate latency.

3.2 Multiple Rays

The latency introduced by calculating the new skip increment means that if only one ray is processed the pipeline will stall, waiting for the calculation of the new skip distance to complete. To avoid this latency and maintain a full ray casting pipeline, we trace the path of more than one ray at a time. An example of four rays and their location relative to the memory banks used in the VIZARD II architecture is shown in Figure 4. A sample point taken along one

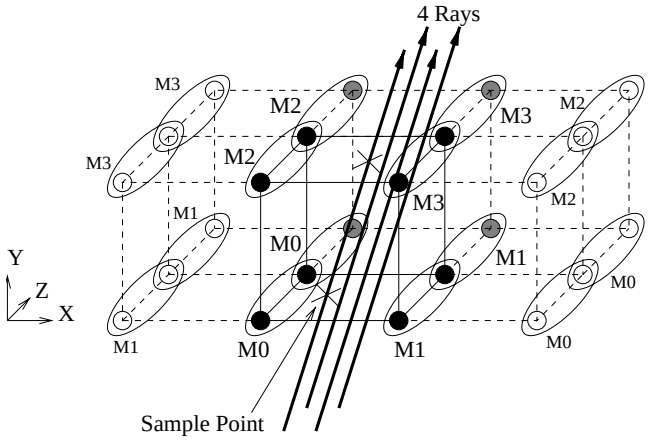

Figure 4: Four rays and associated voxels showing relevant memories.

ray is marked by an $\times$ and the closest eight voxels making up its voxel neighborhood are emphasized as black filled circles. This voxel neighborhood must be read from memory and trilinearly interpolated to calculate the new value, called the sample, at that sample point. The complete memory system of the VIZARD II board is described in [3] and is only treated briefly in this paper. In Figure 4, the memory modules that the voxels will be read from are indicated by Mn, where n is the memory module number. The volume is stored in four memories, each providing two susequent voxel values in Z. When tracing multiple rays it is important to keep the current sample along each ray within close proximity of the current samples on the other rays to ensure memory addresses are similar.

[1]The memory of the VIZARD II architecture is eight times interleaved. Hence, a volume dataset of 16 Msamples read from 100MHz SDRAM can be scanned 50 times per second.

[2]The occupancy map can be implemented using some of the 131 Kbits of on-chip Block SRAM of the Xilinx Virtex XCV1000 FPGA.

If one ray advances beyond the other rays then every time voxels are read from memory for the advanced ray a page fault will occur and the ray casting will stall during memory row activation and precharge time.

The main pipeline stages required for ray casting are shown in Figure 5(a). To this pipeline, we add a *Ray Queue* and the *Skip Calculator*, as described above. In Figure 5, we also add stages for Address Calculation and SDRAM Control, as required to interface to the memory. Also an additional Address FIFO and Voxel FIFO are added to improve memory performance by allowing different memories to change pages at different times as described in [3].

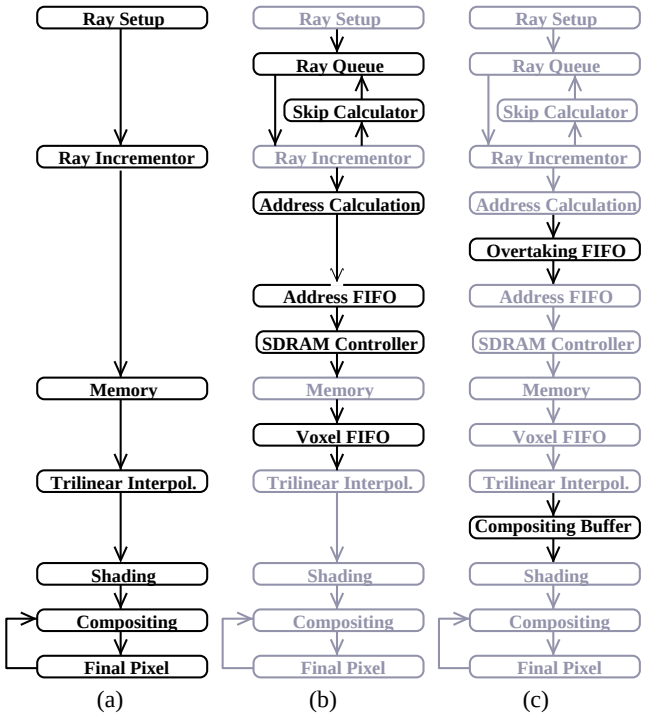

Figure 5: The stages in a ray casting pipeline. (a) Standard pipeline stages, (b) Extra units required for multiple rays and improved memory performance, (c) Further improving memory performance by sorting rays.

To minimize the memory stalling effects when using several rays, an *Overtaking FIFO* is introduced as shown in Figure 5(c) and presented earlier in [2]. The Overtaking FIFO reorders the memory addresses in order to minimize page changes in the memory. A *Compositing Buffer* is also introduced to correctly sort the samples for use in the Shading and Compositing stages.

4 RESULTS

We have implemented a software simulation (C++) of the above described algorithm to measure the quantitative performance gains by counting the number of cycles needed per frame. Rather than using the software simulation of the VIZARDII architecture, we could have recorded measurements using our VHDL simulation of our ray casting pipeline. However, our hardware simulation takes 36 hours to render a 256^3 dataset, making the decision for how to calculate results straight forward.

For the evaluation of the described space leaping approach, we carefully selected a set of five different real-world datasets which are described in Table 1. The first two datasets are both simulation

Dataset	Size	Source	Occupied voxels
fuel	64^3	simulation	5.24 %
neghip	64^3	simulation	46.38 %
foot	256^3	CT-anio	28.94 %
skull	256^3	CT	88.42 %
vessel	256^3	CT-anio	1.01 %

Table 1: Set of datasets which have been used to evaluate the space leaping approach. Occupied voxels are voxels with value > 0.

data, but the number of occupied voxels varies significantly. The other three are scanned datasets. The skull is a very compact block of occupied voxels and due to noise, only a few unoccupied voxels – further-on referred to as empty voxels – exist. In contrast, the vessel dataset contains narrow structures which are present across the entire dataset but a large number of voxels is empty since there is almost no noise present. Finally, the foot dataset is a relatively compact block of occupied voxels with a moderate amount of noise. Figure 8 presents images of these datasets; for each dataset, there is one image showing all data present and another image where a classification revealing the important content has been applied.

4.1 Dataset characteristics

The percentage of empty voxels is given in Table 1, but does not reveal any estimate of the potential gain that can be expected from the presented space leaping mechanism. The potential benefit depends on the distribution of the empty voxels within sub-cubes. We therefore measured the amount of skipable sub-cubes, using different sub-cube sizes ranging from 2^3 to 64^3 for each dataset. The results of these measurements are shown in Figure 6. Obviously,

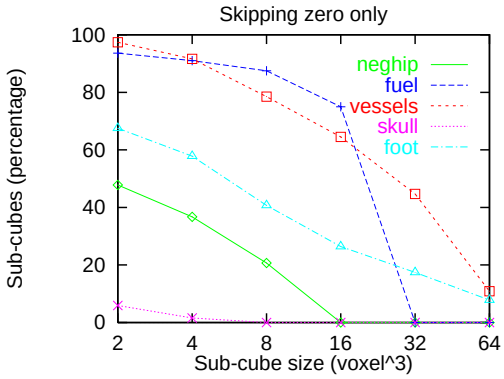

Figure 6: Percentage of skipable sub-cubes. Only empty voxels have been exploited.

the smaller the sub-cubes, the higher the percentage of skipable sub-cubes. For the fuel datasets, no gain is achieved for sub-cubes of 32^3 and larger, since the fuel stream is concentric and symmetric such that all 32^3 sub-cubes always contain occupied voxels. For the neghip dataset, this already applies for sub-cubes of 16^3 and larger since the data is much more distributed over the entire grid. Not surprisingly, for the skull dataset the percentage of skipable sub-cubes is close to zero. This is due to the present noise which prevents any classification independent space leaping mechanism. On the other hand, the foot dataset is noisy as well, but the noise is mostly around the tissue of the foot itself, which leaves many sub-cubes empty. Finally, the vessel dataset has a large amount

of skipable sub-cubes, despite the fact that the arteries are present across the entire volume. The larger the sub-cubes, the lower the percentage of non-contributing sub-cubes. However, for each sub-cube at least one sample needs to be processed. Therefore, space leaping performance is not necessarily better for smaller sub-cube sizes.

Generally, classification is used to remove noise – if possible – and other structures which are not of interest. We therefore used transfer functions visualizing the important content in each dataset. Images resulting from these transfer functions are given in Figure 8. Furthermore, Figure 7 illustrates the percentage of skipable voxels making use of these transfer functions. For the fuel and the neghip

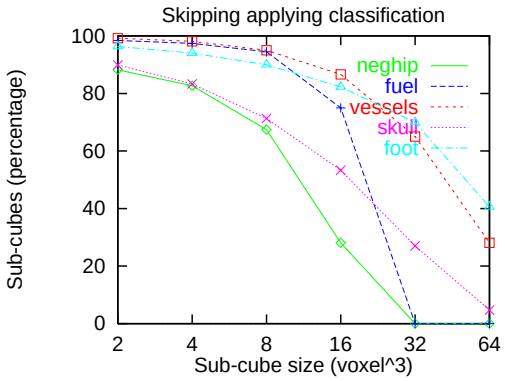

Figure 7: Percentage of skipable sub-cubes exploiting the given classification.

dataset, the percentage of non-skipable sub-cubes can be further reduced by 50%. Additionally, for the neghip 25% of sub-cubes of size 16^3 can now be skipped. The foot and the skull are the two datasets gaining the most; the percentage of non-skipable sub-cubes drops by a factor of 2 to 5. Overall, while for sub-cubes of size 2^3 more than 85% of all sub-cubes are empty, this does not translate into the best space leaping efficiency, which will be elaborated in the next section.

As mentioned earlier, the occupancy map is stored in an on-chip storage (SRAM), to keep latency during the space leaping process as small as possible. Unfortunately, the capacity of this type of memory is limited, which puts an upper bound on the size of the feasible occupancy map. Table 2 illustrates the amount of memory needed for different dataset and sub-cube sizes. For the selected datasets, 4 Kbits offer a good percentage of skipable sub-cubes compared to the memory size. However, this may change for larger datasets (512^3), where one might achieve better results using 32 Kbit, which is still feasible.

Volume size/ Sub-cube size	64^3 [256 KBytes]	128^3 [2 MBytes]	256^3 [16 MBytes]
2^3	32 Kbits	256 Kbits	2 Mbits
4^3	**4 Kbits**	32 Kbits	256 Kbits
8^3	512 bits	**4 Kbits**	32 Kbits
16^3	64 bits	512 bits	**4 Kbits**
32^3	8 bits	64 bits	512 bits

Table 2: Storage needed for the space leap map.

4.2 Experiments and Discussion

The percentage of skipable sub-cubes does not exhibit the actual performance gains. What determines the optimal sub-cube size is given by the compactness of the structure(s) contained in the dataset and how much skipable space is around. Performance gains are generally limited to those parts of rays, which pass through empty sub-cubes. For a thorough analysis, we generated animations of 72 frames for all five datasets rotating around the center of the dataset starting with the views given in Figure 8. For each frame, we accurately measured the number of cycles needed for image generation. These measurements include cycles for:

1. Taking all samples along all rays.
2. Applying early ray termination (ERT)
3. Applying space leaping additional to ERT, using sub-cube sizes ranging from 4^3 to 32^3.

The results are shown in Figure 9 where the graphs illustrate the view and classification dependent performance. To better comprehend the information of Figure 9, we translated it into frame-rates using the average memory access time of 12.5 nsecs per voxel, as described in [3]. The results are shown in Table 3. Generally, early

Acceleration		none	ERT	4^3	8^3	16^3	32^3
fuel	'0'	16.1	16.4	27.3	41.9	45.8	16.4
	class	16.1	16.3	28.4	47.3	45.2	16.3
neghip	'0'	18.6	23.7	28.1	27.9	23.7	23.7
	class	18.6	23.3	36.2	46.2	31.5	23.3
foot	'0'	4.4	5.3	7.3	7.7	7.1	6.5
	class	4.4	5.3	9.0	14.4	19.1	16.8
skull	'0'	4.3	8.2	8.2	8.2	8.2	8.2
	class	4.3	7.8	12.9	17.3	16.0	10.8
vessel	'0'	4.3	4.7	7.8	10.4	10.7	8.3
	class	4.3	4.6	8.0	13.3	17.3	12.0

Table 3: Frame-rates for the five datasets skipping empty voxels only ('0') and exploiting the given classification (class). The frame-rates are averaged over 72 frames (see Figure 9). Acceleration *none* stands for processing all samples along all rays and *ERT* stands for early ray termination.

ray termination is not a very efficient acceleration technique, unless the viewpoint is close to an highly opaque object covering large areas of the screen-space. For the presented views, performance gains vary from almost zero for the fuel dataset to 25% for the neghip dataset. The only exception is the skull dataset, where a 90% performance is gained due to the opaque skull.

Space leaping based on skipping empty sub-cubes, only gives poor speed-ups for datasets with a high percentage of occupied voxels. This is illustrated with the skull dataset where 88% of all voxels are occupied (see Table 1). A similar observation can be made for the foot dataset. Only for the fuel dataset performance gains of 280% can be observed which are due to the large areas of non-occupied voxels surrounding the compact union of occupied voxels.

Generally, much higher frame-rates can be achieved exploiting the given classification, resulting in performance gains ranging from 200% for the neghip dataset to 375% for the vessel dataset (additional to early ray termination). The performance gain for the neghip is only 200%, since a large number of samples still contribute to the final image. Overall, for the presented datasets of 256^3 voxels, we achieve frame-rates well above 15 frames per second.

While achieving good speed-ups additional to early ray termination, the selection of the appropriate sub-cube size is dataset and

classification dependent. As a rule of thumb, a sub-cube size of 8^3 is suited for the smaller datasets (64^3) and sub-cubes of 16^3 for the larger datasets (256^3), even though for a few cases slightly higher frame-rates can be achieved for the next smaller or larger sub-cube size. Finding heuristics for the best suited sub-cube size is still subject to further research.

The reported frame-rates will be even higher by skipping larger sub-cubes exploiting the given two-level hierarchy, as described in Section 2. In case all eight at even boundary addresses aligned neighboring sub-cubes are empty, a much larger distance can be skipped within a single cycle. Furthermore, for samples taken in empty sub-cubes, no memory request needs to be sent down the pipeline and hence, the overall memory efficiency will increase due to less page reloads. This is not yet incorporated in the numbers presented in Figure 9 and Table 3, since this requires a full simulation of the entire pipeline.

5 CONCLUSIONS

We presented a new space leaping mechanism using an occupancy mask instead of an entire distance volume. With only $4Kbit$ of SRAM, we were able to significantly reduce the latency caused by reading distance volumes from the voxel memory. The latency due to the calculation of the new skipping value can be accommodated by processing eight rays, even in an FPGA design running at 100 MHz. The amount of extra logic required for the presented space leaping mechanism is less than 1% (130 CLBs) of an Xilinx Virtex XCV1000 FPGA.

Furthermore, we have shown that for the VIZARD II architecture, the occupancy map can be classification dependent, since VIZARD II provides a sufficient memory bandwidth to update the occupancy map 50 times per second. Using a set of real-world datasets, we demonstrated the efficiency of the occupancy map in the VIZARD II architecture. We achieved frame-rates well above 15 frames per second for datasets of 256^3 voxels, providing parallel and perspective projections, as well as arbitrary sampling rates in all three dimensions.

Our future work will include exploiting the natural two-level hierarchy given by an occupancy map stored in eight bit entries further increasing the space leaping efficiency. Furthermore, we will investigate heuristics for determining the “ideal” sub-cube size for given datasets.

Acknowledgements

This work has been funded by the SFB grant 382 of the German Research Council (DFG). The vessel dataset is courtesy of Philips Research, Hamburg, Germany, the skull dataset is courtesy of Siemens Medical Systems, Forchheim, Germany, and the fuel injection dataset is courtesy of SFB 382 of the German Research Council (DFG).

%small

References

[1] D. E. Breen, S. Mauch, and R. T. Whitaker. 3D Scan Conversion of CSG Models into Distance Volumes. In *Symposium on Volume Visualization*, pages 7–14, Research Triangle Park, NC, October 1998.

[2] Michael Doggett. A ray queueing and sorting design for real time ray casting. In *International Symposium on Circuits and Systems*. IEEE, May 2000.

[3] Michael Doggett, Michael Meißner, and Urs Kanus. A low-cost memory architecture for pci-based interactive ray casting. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 7–14, August 1999.

[4] S. Gibson. Using Distance Maps for Accurate Surface Representation in Sampled Volumes. In *Symposium on Volume Visualization*, pages 23–30, Research Triangle Park, NC, October 1998.

[5] B. Gudmundsson and M. Randen. Incremental generation of projections of CT-volumes. In *Proc. of the First Conference on Visualization in Biomedical Computing*, pages 27–34, Atlanta, GA, May 1990. IEEE Computer Society Press, Los Alamitos, California.

[6] R. Klein, A. Schilling, and W. Straßer. Reconstruction and simplification of surfaces from contours. In Bob Werner, editor, *Proc. of the Seventh Pacific Conference on Computer Graphics and Applications*, pages 198–207, Seoul, Korea, October 1999.

[7] G. Knittel. A pci-based volume rendering accelerator. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 73–82, Maastricht, The Netherlands, August 1995.

[8] G. Knittel and W. Straßer. Vizard - visualization accelerator for realtime display. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 139–146, Los Angeles, USA, August 1997.

[9] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp factorization of the Viewing Transform. In *Computer Graphics*, Proc. of ACM SIGGRAPH, pages 451–457, July 1994.

[10] M. Meißner, U. Kanus, and W. Straßer. VIZARD II, A PCI-Card for Real-Time Volume Rendering. In *Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 61–68, Lisboa, Portugal, August 1998.

[11] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The volumepro real-time ray-casting system. In *Computer Graphics*, Proc. of ACM SIGGRAPH, pages 251–260, August 1999.

[12] H. Ray and D. Silver. A Memory Efficient Architecture for Real-Time Parallel and Perspective Direct Volume Rendering. Technical Report CAIP-TR-237, Department of Computer Aids for Industrial Productivity, Rutgers University, 1999.

[13] M. Sramek. Fast Surface Rendering from Raster Data by Voxel Traversal Using Chessboard Distance. In *Proc. of IEEE Visualization*, pages 188–195, Washington, D.C., October 1994.

[14] B. Vettermann, J. Hesser, and R. Männer. Solving the Hazard Problem for Algorithmically Optimized Real-Time Volume Rendering. Proc. of 1st Workshop on Volume Graphics, March 1999.

[15] R. Yagel and Z. Shi. Accelerating volume animation by space-leaping. In *Proc. of IEEE Visualization*, pages 62–69, San José, CA, October 1993.

[16] K. Z. Zuiderveld, A. H. J. Koning, and M. A. Viergever. Acceleration of ray catsing using 3D distance transform. In R. A. Robb, editor, *Proc. of Visualization in Biomedical Computing*, pages 324–335, Chapel Hill, NC, October 1992. SPIE, Vol. 1808.

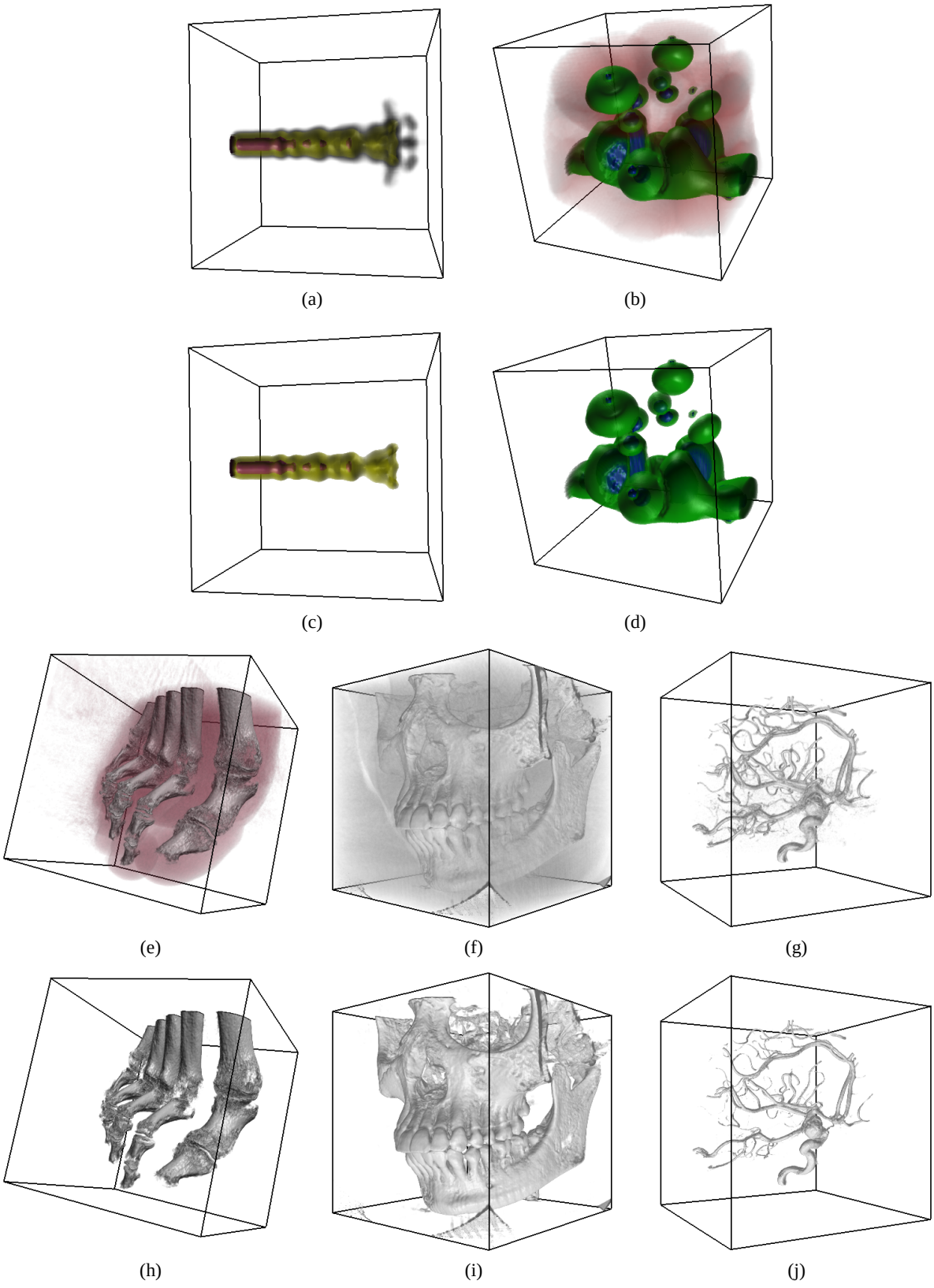

Figure 8: Test datasets: (a,b) and (e-g) have been rendered visualizing all occupied voxels. The other images were rendered applying the transfer functions used throughout the paper.

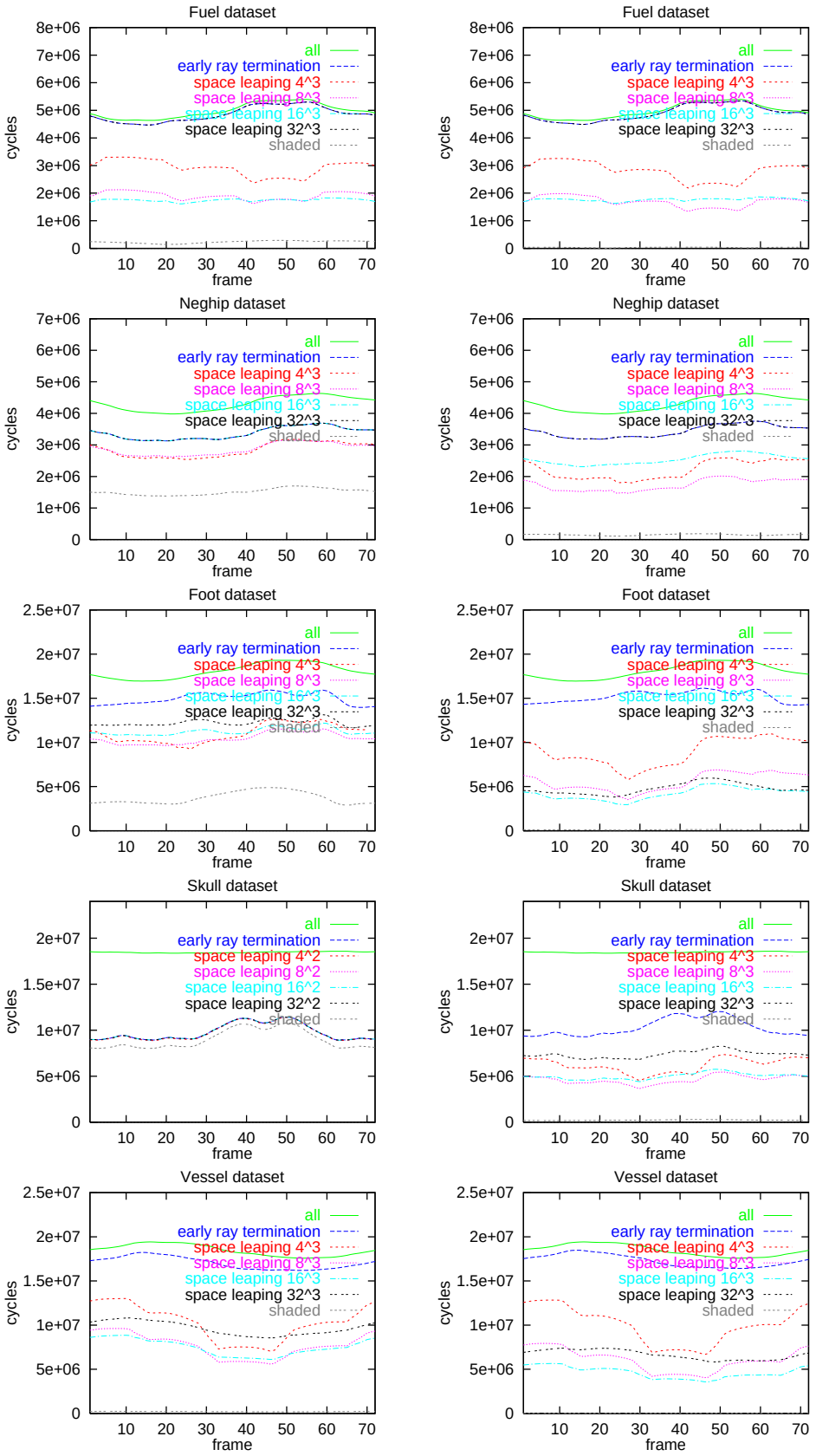

Figure 9: Cycles per frame, with (left column) and without applying classification (right column).