Accelerating Transfer Function Update for Distance Map based Volume Rendering

Michael Rauter*
Competence Center for Preclinical
Imaging and Biomedical Engineering,
Faculty of Health, University of Applied
Sciences Wiener Neustadt, Austria

Lukas Zimmermann
Department of Radiation Oncology,
Medical University of Vienna, Austria

Markus Zeilinger
Competence Center for Preclinical
Imaging and Biomedical Engineering,
Faculty of Health, University of Applied
Sciences Wiener Neustadt, Austria

Figure 1: Volume renderings for kingsnake [2], stag beetle [9], and manix [18] datasets applying different transfer functions

ABSTRACT

Direct volume rendering using ray-casting is widely used in practice. By using GPUs and applying acceleration techniques as empty space skipping, high frame rates are possible on modern hardware. This enables performance-critical use-cases such as virtual reality volume rendering. The currently fastest known technique uses volumetric distance maps to skip empty sections of the volume during ray-casting but requires the distance map to be updated per transfer function change. In this paper, we demonstrate a technique for subdividing the volume intensity range into partitions and deriving what we call partitioned distance maps. These can be used to accelerate the distance map computation for a newly changed transfer function by a factor up to 30. This allows the currently fastest known empty space skipping approach to be used while maintaining high frame rates even when the transfer function is changed frequently.

Index Terms: Computing methodologies—Computer graphics—Rendering, Theory of computation—Design and analysis of algorithms—Data structures design and analysis.

1 INTRODUCTION

Direct volume rendering is an important technique to get insight into a volume dataset. Direct volume rendering has the advantage of not relying on volume segmentations while producing high quality renders. The main technique in direct volume rendering is volume ray casting where a ray is cast from the camera eye point to every pixel in the target render image [15, 13]. Discrete steps are taken along the ray covering the volume's bounding geometry. At each position the corresponding volume's voxel values are sampled

---

*e-mail: michael.rauter@fhwn.ac.at

and interpolated. Those sampled voxel values are then mapped using a transfer function (TF) to rendered color and opacity values as described by Kniss et al. [12]. This can be done by either using a 1D mapping from intensities, or a 2D mapping from intensities and gradient magnitude. Voxels along a single ray's path are blended together for the final projected pixel color in the rendered image.

The high computational demands of volume rendering pose a challenge but with the growing computing performance of graphics processing units (GPUs) in recent years and optimized acceleration techniques volume rendering with high interactive frame-rates and high rendering resolutions became a reality. Even especially demanding use cases as volume rendering in virtual reality are possible [7, 3, 11, 6, 19, 22, 17, 5, 8].

Two important acceleration techniques are early ray termination and empty space skipping. Early ray termination stops the ray stepping iteration for a given pixel as soon as the aggregated opacity exceeds a certain threshold. Empty space skipping allows to step over voxel positions along the ray that do not contribute to the output pixel's color. This is done by using an acceleration data structure providing information about how much space can be skipped[4].

Our paper demonstrates an acceleration technique for a certain part of distance map based empty space skipping as described by Deakin and Knackstedt [1]. We show how the acceleration data structure (the distance map) can be recomputed faster which is required in case the TF is altered. This is achieved by subdividing the whole volume's intensity range into partitions. We derive individual distance maps from the respective intensity partitions. Given a newly changed TF, these partitioned distance maps can be efficiently combined into an updated distance map.

2 RELATED WORK

Empty space skipping (ESS) methods can be classified into object-order and image-order ESS methods [14]. Here, we focus on the latter class.

Krüger and Westermann use an occupancy map encoding disjoint blocks of 8^3 voxels storing the blocks' minimum and

maximum intensity values with non-zero transparency contributions [13]. This basically amounts encoding a single level of an min/max octree, in contrast to standard min/max octree implementations [14, 21]. Alternatively, the acceleration structure can be an octree of binary values encoding the occupancy of its nodes as employed by Levoy [16].

SparseLeap is a hybrid object- and image-order ESS method proposed by Hadwiger et al. [10]. An occupancy histogram tree is used to extract the dataset's bounding geometry. Per-pixel linked lists of skippable ray segments are created to efficiently skip empty space when rendering the volume. A drawback is the requirement of recomputing the histogram tree and the bounding geometry during TF changes.

A different approach uses distance maps storing the smallest distance to the nearest non-transparent block of voxels or rectilinear grids as proposed by Sramek and Kaufman [20].

Deakin and Knackstedt [1] generate a reduced occupancy map and the corresponding Chebyshev distance map encoding the distance to the nearest non-transparent block of voxels utilizing GLSL compute shaders on the GPU. In terms of rendering time, their volume renderer outperforms other approaches, e.g., depending on the dataset used, they report runtimes on average twice as fast as octree-based methods and SparseLeap. Recomputing the distance map on a TF change is a limiting performance factor in their approach.

Faludi et al. [5] use an octree of nodes storing a bitfield, where each bit inside the bitfield encodes the occupancy of an intensity subrange. Similarly, a bitfield represenation of the currently selected TF is derived. Testing if an octant only contains transparent voxels according to the TF and thus may be skipped, can be determined with a binary operation between its bitfield and the bitfield of the TF. Changing the TF just requires recomputing the TF bitfield and does not affect performance noticeably. They report ray marching times 35% slower than Deakin's approach on average.

3 METHOD

We introduce an optimization technique for updating the distance map in volume rendering using distance map based ESS. Our approach allows computing the distance map in less time than before by utilizing precomputed individual distance maps encoding voxel occupancies inside specific intensity subranges. The update step is an aggregation step combining several of those distance maps to produce the final distance map used for ESS. The TF dictates which distance maps require merging.

3.1 Intensity-Partitioned Occupancy/Distance Maps

Our approach is inspired by the idea of subdividing the intensity range similar to how it was done by Faludi et al. [5]. We also split the intensity range into multiple subranges but instead of storing the occupancy information in an octree of bitfields we store it in individual distance maps for the different intensity partitions. The total range of intensity values used in the volume dataset is subdivided into n intensity partitions. The partitions can be uniformly distributed over the whole intensity range, but this is not a necessary requirement. For example, it can be beneficial to use a separate partition for the minimum intensity value ρ_{min} encountered inside the volume and distribute the other intensity values to the remaining partitions. A single partition p is defined by the lower and upper border of its intensity range denoted as integer indices ρ_{min}^p and ρ_{max}^p.

We compute n block-reduced occupancy maps for the respective subranges which we call partitioned occupany maps (POMs) from now on. A block of voxels is considered occupied if it contains at least one voxel with an intensity value from within the intensity subrange $\left[\rho_{min}^p, \rho_{max}^p\right]$. Block-reduced means that the original volume dataset is subdivided into non-overlapping blocks of voxels of size b^3. We set b to 4 which was also used as block size by Deakin

and Knackstedt [1]. It is important to point out that the POMs do only depend on the volume's voxel intensity values and not on the TF used for rendering.

Next, we compute n partitioned distance maps (PDMs) from the n individual POMs. The PDMs encode the Chebyshev distance to non-empty voxels in the respective intensity subrange.

Both the POMs as well as the PDMs are in block-reduced size and their computation can be done in a preprocessing step (this resembles a one-time initialization step which only needs to be done once when a volume is initially loaded).

Whenever the TF is loaded or updated, merging the relevant PDMs into a final single distance map D' is done (the TF dictates the map selection required for merging). D' is the only entity directly used for ESS in the renderer. D' is assembled in two steps:
Step 1: Determine all intensity partitions containing at least one value getting mapped by the TF to a non-zero alpha value.
Step 2: Compute the element-wise minimum of the corresponding PDMs. This resembles the combined distance map D' in Equation 1.

$$D'(x,y,z) = \overset{p \in S}{min}(PDM_p(x,y,z)) \tag{1}$$

S is the set of indices p of partitions determined in step 1, x, y, z are the integer coordinates of a block.

See Figure 2 for a simplified 2D example of our technique. In general, D' may differ from the standard non-partitioned distance map (D). This happens if different intensity values from the same intensity partition get mapped to both zero and non-zero alpha values by the TF. The values of D' are always an element-wise lower limit of the values in D (e.g. $D' \leq D$). As a consequence ESS run-time performance may be worse because less voxels will be skipped. It can never happen that more voxels than allowed are skipped making D' a reasonable approximation and thus replacement of D. Again, refer to Figure 2 and compare D' to D when applying TF$_A$ ($D' = D$) and when applying TF$_B$ ($D' \leq D$). D' is - as already stated - the only component that needs to be recomputed when a TF update occurs. Taking the element-wise minimum of the distance values from the PDMs is computationally less expensive than computing the actual distance transform algorithm which makes our approach more efficient.
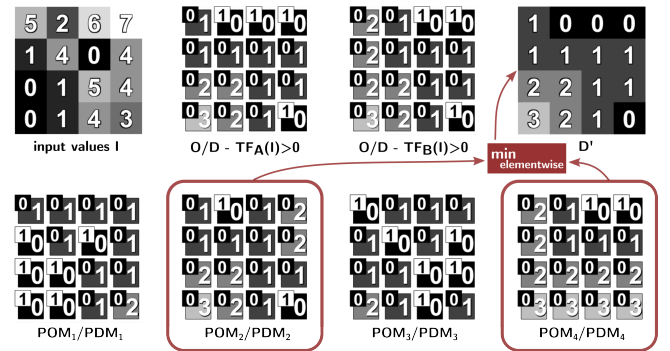

Figure 2: Our method demonstrated for a simplified 2D case (without block-reducing): 1st row from left to right: input intensity map I, occupancy and corresponding distance map (small squares contain occupancy values) after applying TF$_A$ resp. TF$_B$, D' derived from the PDMs ($D' = D'_{TF_A} = D'_{TF_B}$ in this example), 2nd row: the POMs and PDMs using partitions with intensity ranges $[0,1]$, $[2,3]$, $[4,5]$, and $[6,7]$; TFs' intensities to alpha mappings ($\emptyset$... non-zero alpha): TF$_A(0,...,7) = \{0,0,\emptyset,\emptyset,0,0,\emptyset,\emptyset\}$, TF$_B(0,...,7) = \{0,0,0,\emptyset,0,0,\emptyset,\emptyset\}$ and thus, $S = \{2,4\}$ for both.

Table 1: Comparison of time required to compute the acceleration structure used for ESS of the original approach [1] compared to our approach with different numbers of intensity partitions. For each dataset, we report timings for the different TFs used (TFs are those from Figure 3).

dataset	size	computation type	transfer function	distance map [1]	PDMs (16 partitions)	PDMs (32 partitions)	PDMs (64 partitions)	PDMs (128 partitions)	PDMs (256 partitions)
manix	$512 \times 512 \times 460$	one-time initialization [ms]	TF1-TF6	-	58.1	115.3	227.9	454.4	905.2
		update time on TF change [ms]	TF1	3.50	0.20	0.27	0.42	0.69	1.30
			TF2	2.55	0.20	0.26	0.44	0.70	1.27
			TF3	4.31	0.18	0.19	0.28	0.44	0.69
			TF4	4.03	0.17	0.22	0.28	0.41	0.71
			TF5	3.85	0.20	0.26	0.38	0.58	0.98
			TF6	3.66	0.18	0.20	0.26	0.44	0.71
stag beetle	$832 \times 832 \times 494$	one-time initialization [ms]	TF1-TF6	-	146.4	299.9	608.7	1233.0	2491.8
		update time on TF change [ms]	TF1	10.10	0.41	0.55	0.87	1.40	2.58
			TF2	0.40	0.41	0.58	0.85	1.38	2.59
			TF3	12.23	0.31	0.39	0.58	0.81	1.40
			TF4	11.30	0.33	0.40	0.58	0.80	1.42
			TF5	11.13	0.45	0.51	0.74	1.14	2.03
			TF6	11.12	0.33	0.40	0.57	0.85	1.42
kingsnake	$1024 \times 1024 \times 795$	one-time initialization [ms]	TF1-TF6	-	228.2	458.2	913.7	1825.9	3659.2
		update time on TF change [ms]	TF1	9.63	0.80	1.25	1.97	3.58	6.71
			TF2	7.28	0.80	1.25	1.99	3.56	6.70
			TF3	21.40	0.63	0.77	1.23	1.94	3.58
			TF4	14.59	0.63	0.76	1.23	1.94	3.59
			TF5	15.28	0.79	1.08	1.68	2.86	5.39
			TF6	9.76	0.66	0.79	1.26	2.09	3.60

3.2 Implementation

We implemented our method in a custom direct ray-casting volume rendering shader using OpenGL/GLSL with C++. Rendered images from our renderer can be seen in Figure 1. The computation of the POMs and PDMs as well as computing D' was implemented with GLSL Compute Shaders. POMs and PDMs use an *uint8* data type for performance and memory consumption reasons, distances higher than the maximum unsigned 8-bit integer value are clamped. Combining the PDMs in the compute shader is done in several passes, aggregating distances by binding only distance maps as input textures that have a contribution to D' according to the TF. In every pass up to a maximum of 6 PDMs are bound and combined (2 additional textures being used for flip-flopping input (result from previous passes) and output texture). The number of necessary passes depend on the number of PDMs that need to be combined derived from the non-zero alpha mappings in the TF.

4 RESULTS

To demonstrate the benefits of our approach we conducted a series of experiments to show the performance gain for the distance map update on a TF change and the implications to the frame rendering performance. A video showing volume rendering results with frame rendering performance is provided as supplementary material.

4.1 Benchmark Configuration

For our performance tests, we use a *Windows11* workstation equipped with an *Intel Core i7-13700KF* CPU running at 3.4 GHz, 32 GB RAM and a *NVIDIA RTX 4090* GPU. The timing measurements for the computation times of the distance maps generation and update depend on the size of the volume dataset, the TF used and the intensity partition configuration chosen (we benchmarked 16, 32, 64, 128, and 256 partitions). The timing measurements for frame rendering depend on the viewport size, the viewport coverage by the volume, the TF used and the chosen rendering options/parameters (e.g., if early ray termination or ESS is used).
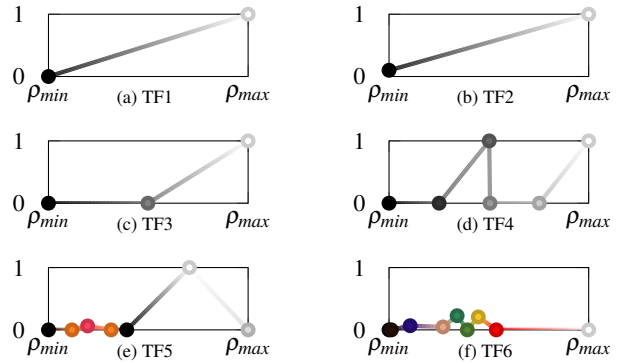

Figure 3: The different transfer functions used: TF1: all intensities but ρ_{min} are mapped to non-zero alpha values. TF2: all intensity values are mapped to an alpha value greater than zero. TF3: intensities in the right half are mapped to non-zero alpha values. TF4: non-zero alpha values in 2nd and 4th quarter of intensity range. TF5: used for rendering of kingsnake dataset in Figure 1. TF6: used for rendering of stag beetle dataset in Figure 1.

Viewport size was chosen as 2160^2, we made the rendering of the volume cover the vertical viewport extent, early ray termination (if used) was set to terminate the ray at 0.98 of the maximum possible alpha value. For the frame rendering benchmark, the camera was rotated around the vertical axis going through the center of the volume datasets completing two full rotations over the time period of 10 seconds. With this setup, we tried to reproduce the setup chosen by Faludi et al. [5].

4.2 Performance Evaluation

We use a few representative TFs for timing measurements in our experiments (see Figure 3). The choice typically affects both the runtime of computing D' as well as the actual frame rendering times. Computation time of D' is affected because the number of maps to

be combined depends on the TF used. Frame rendering times are affected because depending on the number and configuration of the partitions, both intensity values that influence the rendering as well as those that do not can fall into the same partition. If this happens this may degrade runtime performance since voxels of those intensities will be handled as non-skippable. One occurrence of this issue can be seen in Figure 5 for the setup with 32 uniform partitions. The required number of partitions to prevent measurable performance degradations depends on the volume and TF used.

In Table 1 we show the computation times for generating and updating the PDMs for a set of partition numbers. We compare the timings with the original distance map approach [1] which has to recompute the occupancy and the distance map on a TF change (we report the combined time). In our partitioned distance map approach we can precompute the POMs and PDMs. The more partitions are used, the longer it takes to compute D'. On the other hand, using fewer partitions may affect frame rendering performance negatively (see thoughts in the beginning of this section and regarding ESS run-time performance in subsection 3.1). Comparing the computation time of the original distance map, we see an improvement of up to factor 10 for 256 partitions and up to factor 30 for 16 partitions. As one can see the TF used affects the actual timings. On the stag beetle dataset using TF2 (which maps all intensity values to non-transparent output values) our approach performs worse than the original implementation by Deakin and Knackstedt [1]. This is an edge case where the original occupancy map implementation is able to immediately return when computing a block's occupancy state (since all blocks are occupied). In contrast to this, individual POMs may likely have unoccupied blocks depending on the distribution of intensity values in the volume dataset (if all voxels of a block contain intensities from outside the partition's intensity range). Computing the distance map is especially fast for this edge case since the occupancy map is fully occupied and thus, the distance computation can be exited early.

In Figure 4 timings for different volume datasets rendered using different TFs are shown. Rendering times in our PDM approach are similar compared to the original distance map approach by Deakin and Knackstedt [1]. For comparison, we also show the timings measured for their block ESS method. In Figure 5 timings for rendering the stag beetle dataset using TF6 are compared for different ESS methods. For our method we show timings when using different numbers of intensity partitions, both with uniform partition of the intensity space as well as using a special partition for ρ_{min}. For TF6, a different number of intensity partitions does not change rendering times significantly with the exception of 32 partitions where the uniform partitioning breaks down. The reason is that TF6 maps different intensities from the 1st partition to zero (e.g. ρ_{min}) and non-zero alpha values resulting in non-skippable intensity values. The stag beetle dataset contains a high number of voxels with intensity ρ_{min} which degrades ESS performance. With a sufficient number of partitions, all intensities of the first partition are transformed to the same class of mapped alpha values and no performance breakdown occurs. A special partition for ρ_{min} in the 32 partitions case can also prevent the performance degradation.

5 DISCUSSION

Our method allows for TF updates with low performance impact, nevertheless update times are higher than in the method by Faludi et al. [5] which does not require recomputing the acceleration structure. On the other hand, our method is faster in terms of rendering performance as it extends the fastest currently known volume rendering method (distance-map based ESS) allowing frequent TF updates while maintaining higher frame rates and reducing latency.

Our approach requires additional GPU memory for storing the PDMs. The total extra amount of memory required depends on the block size b of the reduced volume and the number of partitions
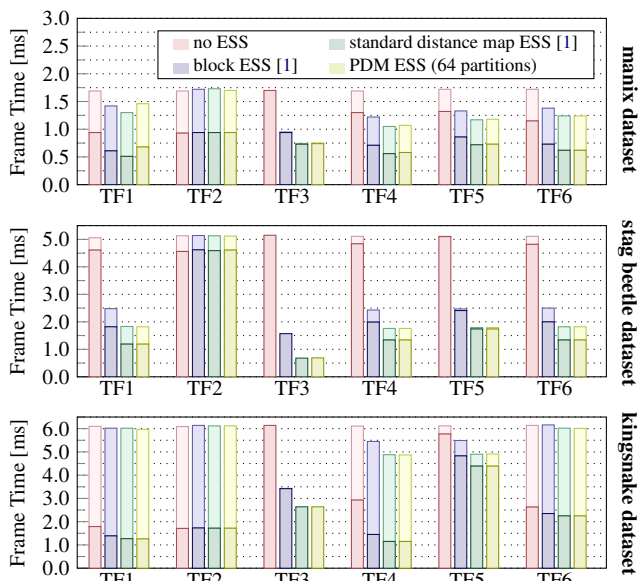

Figure 4: Frame rendering times for different datasets (viewport size 2160^2) with different TFs and ESS variants. Stacked bars indicate rendering time w/o early ray termination (darker/lighter). PDM ESS setup: 64 uniform intensity partitions for TF2-TF6, for TF1: special partition for intensity value ρ_{min} and the other intensity values uniformly partitioned into the remaining 63 partitions.
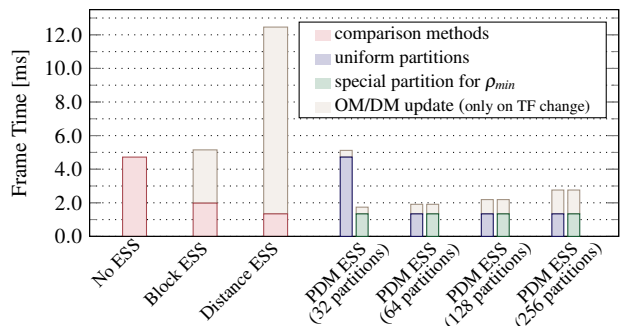

Figure 5: Rendering performance for ray marching on the stag beetle dataset with TF6 from Figure 3f with viewport size 2160^2 with early ray termination for different ESS variants. Additionally, occupancy/distance map update times are shown (these only apply if the TF is updated).

n. Since the distance maps are textures of unsigned 8-bit integer data type, the required memory amount in bytes is $n \times \frac{volumesize}{b^3}$ with *volumesize* being the number of voxels in the original volume dataset. The POMs, which are needed to compute the PDMs, are just an intermediate result. Each partition is computed independently from each other, therefore only one additional texture for computing the individual POMs is needed. For this purpose, one of the flip-flopping textures required for computing D' can be reused.

6 CONCLUSION

We improved distance map based ESS in volume rendering to better support frequent TF updates. Our method is valuable when volume rendering performance is the highest priority but also frequent TF updates need to be done. Nevertheless changing the TF will impact performance more than in the method by Faludi et al. [5]. Drawbacks of our approach are additional initialization time for creating the PDMs as well as extra GPU memory needed which does affect the maximum possible volume size for in-core volume rendering.

REFERENCES

[1] L. Deakin and M. Knackstedt. Accelerated volume rendering with Chebyshev distance maps. In *SIGGRAPH Asia 2019 Technical Briefs*, SA '19, p. 25–28. Association for Computing Machinery, New York, USA, 2019. doi: 10.1145/3355088.3365164 1, 2, 3, 4

[2] DigiMorph.org, The University of Texas High-Resolution X-ray CT Facility (UTCT), and NSF grant IIS-9874781. Lampropeltis getula, Common Kingsnake, 2005. http://digimorph.org/specimens/Lampropeltis_getula/egg/whole/ [Accessed: 2024-jul-29]. 1

[3] M. El Beheiry, S. Doutreligne, C. Caporal, C. Ostertag, M. Dahan, and J.-B. Masson. Virtual reality: Beyond visualization. *Journal of Molecular Biology*, 431(7):1315–1321, mar 2019. doi: 10.1016/j.jmb.2019.01.033 1

[4] K. Engel, M. Hadwiger, J. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-Time Volume Graphics*. A. K. Peters/CRC Press, New York, 2006. doi: 10.1201/b10629 1

[5] B. Faludi, N. Zentai, M. Zelechowski, A. Zam, G. Rauter, M. Griessen, and P. C. Cattin. Transfer-function-independent acceleration structure for volume rendering in virtual reality. In *High-Performance Graphics - Symposium Papers*. The Eurographics Association, 2021. doi: 10.2312/hpg.20211279 1, 2, 3, 4

[6] B. Faludi, E. I. Zoller, N. Gerig, A. Zam, G. Rauter, and P. C. Cattin. Direct visual and haptic volume rendering of medical data sets for an immersive exploration in virtual reality. In D. Shen, T. Liu, T. M. Peters, L. H. Staib, C. Essert, S. Zhou, P.-T. Yap, and A. Khan, eds., *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*, pp. 29–37. Springer International Publishing, Cham, 2019. doi: 10.1007/978-3-030-32254-0_4 1

[7] J. Fons, E. Monclús, P.-P. Vázquez, and I. Navazo. Rendering and Interacting With Volume Models in Immersive Environments. In *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2018. doi: 10.2312/ceig.20181153 1

[8] Y. Gao, C. Chang, X. Yu, P. Pang, N. Xiong, and C. Huang. A VR-based volumetric medical image segmentation and visualization system with natural human interaction. *Virtual Reality*, 26(2):415–424, jun 2022. doi: 10.1007/s10055-021-00577-4 1

[9] M. E. Gröller, G. Glaeser, and J. Kastner. open-scivis-datasets, 2005. https://klacansky.com/open-scivis-datasets/ [Accessed: 2024-jul-29]. 1

[10] M. Hadwiger, A. K. Al-Awami, J. Beyer, M. Agus, and H. Pfister. Sparseleap: Efficient empty space skipping for large-scale volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):974–983, jan 2018. doi: 10.1109/TVCG.2017.2744238 2

[11] K. Kalarat and P. Koomhin. Real-time volume rendering interaction in virtual reality. *International Journal of Technology*, 10:1307, nov 2019. doi: 10.14716/ijtech.v10i7.3259 1

[12] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, nov 2002. doi: 10.1109/TVCG.2002.1021579 1

[13] J. Krüger and R. Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization, 2003. VIS 2003.*, pp. 287–292, 2003. doi: 10.1109/VISUAL.2003.1250384 1, 2

[14] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, p. 451–458. Association for Computing Machinery, New York, USA, 1994. doi: 10.1145/192161.192283 1, 2

[15] M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, may 1988. doi: 10.1109/38.511 1

[16] M. Levoy. Efficient ray tracing of volume data. *ACM Trans. Graph.*, 9(3):245–261, jul 1990. doi: 10.1145/78964.78965 2

[17] F. Nysjö, F. Malmberg, and I. Nyström. Raycaching: Amortized isosurface rendering for virtual reality. *Computer Graphics Forum*, 39(1):220–230, feb 2020. doi: 10.1111/cgf.13762 1

[18] OSIRIX. DICOM image library, manix dataset, 2005. https://www.osirix-viewer.com/resources/dicom-image-library/ [Accessed: 2024-jul-29]. 1

[19] I. Scholl, A. Bartella, C. Moluluo, B. Ertural, F. Laing, and S. Suder. MedicVR. In H. Handels, T. M. Deserno, A. Maier, K. H. Maier-Hein, C. Palm, and T. Tolxdorff, eds., *Bildverarbeitung für die Medizin 2019*, pp. 152–157. Springer Fachmedien Wiesbaden, Wiesbaden, 2019. doi: 10.1007/978-3-658-25326-4_32 1

[20] M. Sramek and A. Kaufman. Fast ray-tracing of rectilinear volume data using distance transforms. *IEEE Transactions on Visualization and Computer Graphics*, 6:236 – 252, jul – sep 2000. doi: 10.1109/2945.879785 2

[21] I. Wald, H. Friedrich, A. Knoll, and C. D. Hansen. Interactive isosurface ray tracing of time-varying tetrahedral volumes. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1727–1734, nov 2007. doi: 10.1109/TVCG.2007.70566 2

[22] A. Waschk and J. Krüger. FAVR - accelerating direct volume rendering for virtual realitysystems. In *IEEE Visualization Conference (VIS)*, pp. 106–110, 2020. doi: 10.1109/VIS47514.2020.00028 1