Accelerated Volume Rendering with Chebyshev Distance Maps

Lachlan Deakin
Applied Mathematics
Australian National University
Canberra, ACT, Australia
lachlan.deakin@anu.edu.au

Mark Knackstedt
Applied Mathematics
Australian National University
Canberra, ACT, Australia
mark.knackstedt@anu.edu.au

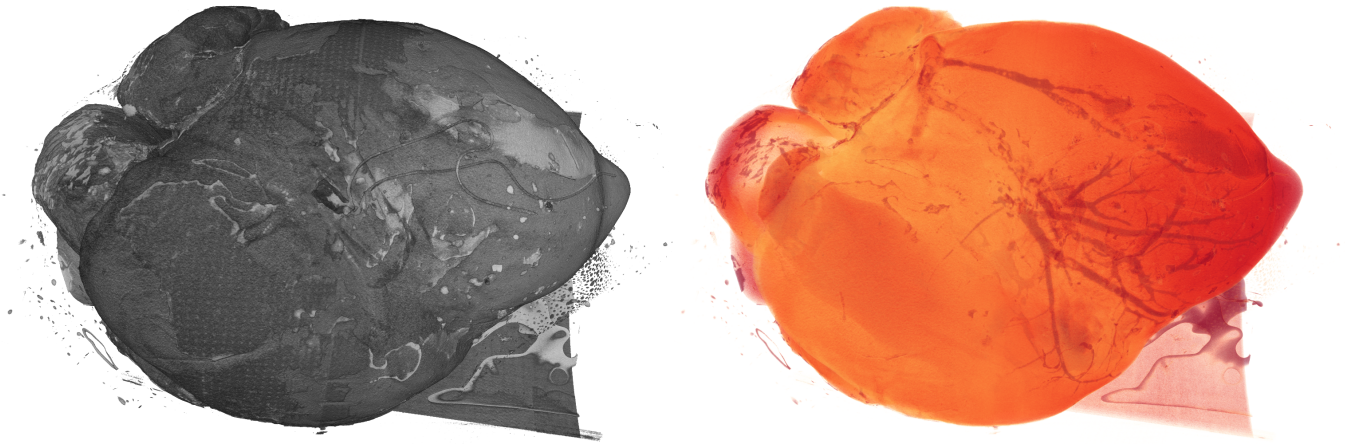

Figure 1: Volume renderings of the pig heart volumetric image (2048x2048x2612 voxels) available from Klacansky [2019]. Renders were at 1920x1200 and were accelerated with early ray termination and our Chebyshev distance empty space skipping. An NVIDIA GeForce GTX 1080 GPU was used. The transfer function is opaque (left) and translucent (right).

ABSTRACT

Volume rendering has useful applications with emerging technologies such as virtual and augmented reality. The high frame rate targets of these technologies poses a problem for volume rendering because of its very high computational complexity compared with conventional surface rendering. We developed an efficient empty space skipping algorithm for accelerating volume rendering. A distance map is generated which indicates the Chebyshev distance to the nearest occupied region (with non-transparent voxels) within a volume. The distance map is used to efficiently skip empty regions while volume ray casting. We show improved performance over state-of-the-art empty space skipping techniques.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**.

KEYWORDS

volume rendering, ray casting, empty space skipping, distance map

SA '19 Technical Briefs, November 17–20, 2019, Brisbane, QLD, Australia

ACM ISBN 978-1-4503-6945-9/19/11…$15.00
https://doi.org/10.1145/3355088.3365164

ACM Reference Format:
Lachlan Deakin and Mark Knackstedt. 2019. Accelerated Volume Rendering with Chebyshev Distance Maps. In *SIGGRAPH Asia 2019 Technical Briefs (SA '19 Technical Briefs), November 17–20, 2019, Brisbane, QLD, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3355088.3365164

1 INTRODUCTION

Volume rendering is a method of visualising a volumetric image which is typically a three-dimensional regular grid of scalar values (voxels). Volume ray casting [Kruger and Westermann 2003] maps each voxel to a colour and opacity which is then projected and composited onto a framebuffer.

Visualising volumetric images on virtual and augmented reality devices has many potential applications. The high computational complexity of volume rendering is a problem for these devices as they require high frame rates for user comfort. Volume ray casting can be accelerated by skipping transparent or occluded voxels using techniques such as Early Ray Termination (ERT) and Empty Space Skipping (ESS) which are described in Sec. 2. Ray casting acceleration techniques are also applicable to isosurface rendering.

A novel ESS technique for acceleration of GPU-based ray casting is introduced in Sec. 3 which is compared with state-of-the-art approaches in Sec. 4. The method requires only a small memory overhead for a distance map which is sampled to determine the Chebyshev distance to the nearest occupied region of a volume. The distance map is used to efficiently skip empty regions and reduce unnecessary texture samples.

2 BACKGROUND

2.1 Ray Casting

Volume ray casting [Kruger and Westermann 2003] projects a ray from each pixel of a GPU framebuffer which steps through the volumetric image (volume) and composites the colour and transparency of voxels it passes through. The scalar value of voxels (and additional image data such as local gradient) can be mapped to colours and transparencies using a lookup table known as a *transfer function* [Kniss et al. 2002]. The transfer function manipulates the appearance and visibility of each material in a volume. Image effects such as shadowing and specularity can also be applied.

Kruger and Westermann [2003] rendered the front and back faces of a bounding box of the volume in two rendering subpasses to indicate where the rays projecting from each pixel enter and exit the underlying 3D texture of the volume. Alternatively, the ray exit point can be computed from the entry point (or vice versa) directly requiring only a single rendering subpass [Stegmaier et al. 2005]. Early ray termination is a standard acceleration technique which stops rays early if an opacity limit is reached rather than where the ray exits the volume [Kruger and Westermann 2003].

2.2 Empty Space Skipping

Empty space skipping is another standard acceleration technique which leaps rays over empty regions of a volume. Kruger and Westermann [2003] created a reduced resolution *occupancy map* which encodes which 8^3 regions of the volume are empty. The ray skips over these regions until an occupied region is found and then begins ray casting on the volume image. This type of approach is considered an image-order ESS algorithm.

Object-order ESS algorithms commence ray casting at the surface of the bounding geometry of a volume rather than at the front of the bounding box. The bounding geometry may be generated through an isosurface extraction [Hadwiger et al. 2005] or other techniques. These types of approaches reduce the number of texture samples but introduce overhead to rasterise the bounding geometry.

Some early CPU-based ray casters used precomputed distance maps encoding the distance from each voxel to the nearest occupied voxel [Sramek and Kaufman 2000]. Large empty regions of a volume could be skipped given a single distance map sample. These approaches were not favoured in the transition to GPU-accelerated ray casting perhaps due to GPU memory constraints.

Large-scale volume rendering algorithms only store occupied bricks (regions) on the GPU and are capable of rendering sparse volumes which would otherwise not fit entirely into GPU memory [Beyer et al. 2015; Gobbetti et al. 2008]. An effective ESS technique is to store brick references in hierarchical structures (such as an octree) and skip empty bricks as the structure is traversed.

A recent hybrid object and image-order ESS algorithm is SparseLeap [Hadwiger et al. 2018] which rasterises the geometry into per-pixel linked lists which indicate where ray segments intersect with occupied hierarchically referenced bricks. Only the occupied segments of the ray go through the ray casting pipeline.

In this paper, we describe a GPU-accelerated ray casting volume renderer which uses distance maps to efficiently skip through empty regions. Our method innovates the early work of Sramek and Kaufman [2000] and Kruger and Westermann [2003].

3 OUR METHOD

3.1 Block Empty Space Skipping

A volumetric image is stored in a 3D texture with dimensions $\mathbf{d} = [width, height, depth]$ and can be sampled using either normalised texel coordinates, $\mathbf{t}$, which map the texture to $[0.0, 1.0]$ on each dimension or unnormalised texel coordinates, $\mathbf{u}$, which are in the range $[0.0, width/height/depth)$. The mapping between these coordinate systems is

$$\mathbf{u} = \mathbf{d} \cdot \mathbf{t}. \tag{1}$$

Voxel colours and transparencies are sampled at n equidistant points along a ray which intersects the volume at $\mathbf{t}_{\text{front}}$ and $\mathbf{t}_{\text{back}}$ respectively. We compute the number of steps along the ray as

$$n = \left\lceil \overrightarrow{\max}(\mathbf{d}) \cdot \|\mathbf{t}_{\text{back}} - \mathbf{t}_{\text{front}}\| \cdot f \right\rceil, \tag{2}$$

where $\overrightarrow{\max}$ is the operator for the maximum element of a vector and f is a sampling factor greater than or equal to 1 used for quality adjustment. The change in normalised texture coordinates between each sample point is

$$\Delta\mathbf{t} = \frac{\mathbf{t}_{\text{back}} - \mathbf{t}_{\text{front}}}{n-1}, \tag{3}$$

and the normalised texel coordinates at the i^{th} point along the ray are given by

$$\mathbf{t}_i = \mathbf{t}_{\text{front}} + i \cdot \Delta\mathbf{t}. \tag{4}$$

We create an *occupancy map*, M, which describes which 4^3 (or larger) blocks of the volume are empty for the given transfer function. The occupancy map has dimensions $\mathbf{d}_M = \lfloor \mathbf{d}/B \rfloor$ where B is the block size. The occupancy map requires only $1/64th$ of the memory of the actual volumetric image if B is 4.

Normalised texel coordinates of the occupancy map are computed as

$$\mathbf{t}_M = \frac{\mathbf{d}}{B \cdot \mathbf{d}_M}\mathbf{t} \tag{5}$$

as $\mathbf{d}$ may not be evenly divisible by B which causes an incorrect alignment if $\mathbf{t}$ were to be used directly for sampling from M. The occupancy map voxel at $\mathbf{t}_M$ can be sampled to determine if the current block is empty. If the block is empty, then the number of ray steps which can be skipped needs to be computed.

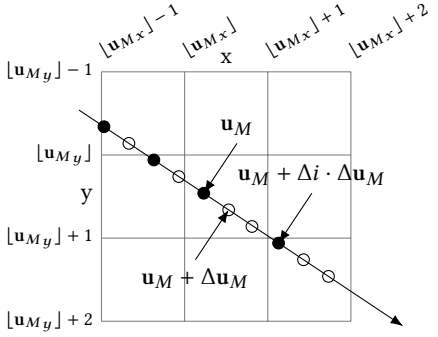

Figure 2: Schematic (2D) of ray casting through occupancy map. Filled points are sampled and unfilled points are skipped if the underlying block is empty.

The change in unnormalised texel coordinates of the occupancy map per step is

$$\Delta \mathbf{u}_M = \mathbf{d}_M \cdot \mathbf{t}_M = \frac{\mathbf{d}}{B} \Delta t, \tag{6}$$

which follows from Eqn. 1 and 5. The number of ray steps required to be in the next voxel of the occupancy map along dimension $j \in \{x, y, z\}$ is derived geometrically from Fig. 2 as

$$\Delta i_j = \begin{cases} \left(1 + \lfloor \mathbf{u}_{Mj} \rfloor - \mathbf{u}_{Mj}\right) \Delta \mathbf{u}_{Mj}^{-1} & \text{if } \Delta \mathbf{u}_{Mj} > 0 \\ \left(\lfloor \mathbf{u}_{Mj} \rfloor - \mathbf{u}_{Mj}\right) \Delta \mathbf{u}_{Mj}^{-1} & \text{otherwise.} \end{cases} \tag{7}$$

The minimum Δi_j across the three dimensions is identified and rounded up to an integer to find the first sampling point which is in another voxel of the occupancy map. This process simplifies to

$$\Delta i = \max\left(\overrightarrow{\min}\left\lceil (step\,(\Delta \mathbf{u}_M) + \mathbf{r}_M)\,\Delta \mathbf{u}_M^{-1} \right\rceil, 1\right), \tag{8}$$

where *step* is the Heaviside step function and $\mathbf{r}_M = \lfloor \mathbf{u}_M \rfloor - \mathbf{u}_M$.

The ray caster repeatedly samples the occupancy map and skips empty blocks until an occupied block is found and then standard volume ray casting begins. The volume is sampled using trilinear interpolation and voxels from multiple neighbouring blocks may be sampled if a sample point is near a block edge. For this reason the step number is decremented by 1 when ray casting begins to ensure exact image consistency as if ESS were disabled.

Empty space within and behind objects is skipped by resuming ESS when a consecutive segment of transparent voxels of sufficient length is found. A length of one and a half times the block size was found to be a good default.

This approach (which we will refer to as *block ESS*) is a formalisation and simplification of the ray casting and empty space skipping method of Kruger and Westermann [2003] which runs entirely in a single rendering subpass on modern fully programmable GPUs. The performance limiting factor of this method is that only one block of the volume is skipped per sample of the occupancy map.

3.2 Chebyshev Distance Empty Space Skipping

This section introduces an ESS algorithm which we refer to as *Chebyshev distance ESS* which can skip multiple blocks of the volume with a single texture sample. This addresses the key performance limitation of the simple ESS approach described in Sec. 3.1.

Chebyshev distance is the greatest difference between two points along any dimension and is defined as

$$D_{\text{Chebyshev}}(\mathbf{p}_1, \mathbf{p}_2) = \overrightarrow{\max}(|\mathbf{p}_1 - \mathbf{p}_2|), \tag{9}$$

where $\mathbf{p}_1$ and $\mathbf{p}_2$ are the two points of interest.

If the Chebyshev distance from a block to the nearest occupied block is known then a ray can skip at least that many blocks along any dimension before it may reach an occupied block. We compute and store these distances in a *distance map* derived from the *occupancy map* using the approach of Saito and Toriwaki [1994] adapted for integer Chebyshev distance (rather than Euclidean distance) and GPU-accelerated. Occupied blocks are assigned a distance of 0.

The number of steps required to move to a block D blocks away on dimension j is determined geometrically from Fig. 2 as

$$\Delta i_j = \begin{cases} \left(D + \lfloor \mathbf{u}_{Mj} \rfloor - \mathbf{u}_{Mj}\right) \Delta \mathbf{u}_{Mj}^{-1} & \text{if } \Delta \mathbf{u}_{Mj} > 0 \\ \left(1 - D + \lfloor \mathbf{u}_{Mj} \rfloor - \mathbf{u}_{Mj}\right) \Delta \mathbf{u}_{Mj}^{-1} & \text{otherwise} \end{cases} \tag{10}$$

and then the actual number of steps is

$$\Delta i = \max\left(\overrightarrow{\min}\left\lceil (step\,(-\Delta \mathbf{u}_M) + sign\,(\Delta \mathbf{u}_M) \cdot D + \mathbf{r}_M)\,\Delta \mathbf{u}_M^{-1} \right\rceil, 1\right) \tag{11}$$

following the same approach as for Eqn. 8. Using Eqn. 11 in place of Eqn. 8 is all that is required for the ray caster described in Sec. 3.1 to skip multiple blocks with only a single sample of the distance map. Note that Eqn. 8 and 11 are equivalent if D is 1.

Chebyshev distances were previously exploited for CPU-based ray casting by Sramek and Kaufman [2000]. Our approach is fully GPU-accelerated, and distance map traversal is more efficient and can be resumed in empty space within and behind objects. The distance map is at a lower resolution than the volume which gives a performance improvement and a lower memory overhead.

3.3 Implementation

A ray casting volume renderer was implemented using the Vulkan graphics and compute API. The renderer supports gradient-based transfer functions, ERT, and the ESS approaches described in Sec. 3.

We exploit the sparse partially-resident image functionality in Vulkan to only store occupied bricks on the GPU which enables rendering of large-scale sparse images which do not fit directly in GPU memory. The brick size (device dependent) is much larger than the block size used when generating occupancy and distance maps. The reduction in frame rate from storing images in this way is less than 1% when evaluated on the images shown in Fig. 3.

(a) Present 492x492x442 **(b) Stag beetle 832x832x494** **(c) King snake 1024x1024x795**

Figure 3: Volumetric images (via Klacansky [2019]) rendered with our volume renderer. Image dimensions are shown.

4 RESULTS

Table 1 compares the performance of several volume renderers which use different ESS approaches. Frame rates were measured on the volumetric images shown in Fig. 3, however, they were axis-aligned and orthographically projected to maximise viewport coverage and different transfer functions were used. Renders were to a 1200x1200 viewport on a system with an NVIDIA GeForce GTX 1070 GPU. Note that Fig. 1 was rendered on a different system.

VTK [2019] is a popular open-source visualisation library with an object-based ESS accelerated volume renderer. Rays start at the object surface which is precomputed as a triangulated isosurface. We evaluated VTK (version 8.2.0) on the same system with matched rendering parameters. VTK was modified to disable ERT and sample consistently with our volume renderer in order to produce effectively identical images and allow for a fair frame rate comparison.

Table 1: Unaccelerated frame rates (baseline) of several volume renderers and the relative speedup from various ESS methods. Several volumetric images (see Fig. 3) were evaluated with one or two transfer functions. ERT is disabled for all measurements.

Image	Occupied voxels	Baseline (frames/s)			Relative frame rate speedup from baseline with ESS				
		Ours[1]	VTK[2]	SparseLeap[3]	Block[1]	Chebyshev distance[1]	Isosurface[2]	Octree[3]	SparseLeap[3]
Present	7.23%	79.2	69.4	17	1.9x	2.2x	1.2x	0.94x	1.1x
	0.483%	79.6	77.9	17	3.7x	8.0x	2.3x	2.5x	4.2x
Stag beetle	3.95%	40.1	36.8	12.5	4.7x	7.8x	1.9x	2.3x	3.6x
King snake	43.90%	40.0	24.6	5.5	1.5x	1.5x	0.95x	0.55x	1.0x
	0.670%	40.2	33.8	5.5	4.1x	7.1x	1.2x	1.5x	2.7x
				Average	3.2x	5.3x	1.5x	1.6x	2.5x

[1]Measured with our volume renderer (see Sec. 3). [2]Measured with VTK [2019]. [3]As reported by Hadwiger et al. [2018].

The same volumetric images were previously evaluated by Hadwiger et al. [2018] with their octree-based ESS implementation and novel SparseLeap ESS. The transfer functions used in VTK and our volume renderer were designed to match the image occupancies they evaluated. We only consider the relative frame rate speedup of their ESS approaches for comparison (rather than actual frame rates) as not all rendering parameters could be matched exactly.

Our Chebyshev distance ESS outperformed all other approaches on every test image and more than doubled the performance of SparseLeap on average. Our Block ESS was also surprisingly efficient and it outperformed the ESS approaches of VTK and Hadwiger et al. [2018] in all but one case. SparseLeap was faster than block ESS for the "present" image with very low voxel occupancy. Our methods improved the frame rate by 1.5x on the "king snake" image with 43.90% voxel occupancy whereas the others did not improve it. This indicates our methods have low computational overhead.

The distance map for the king snake at low occupancy took the longest to generate at 43.8 milliseconds (the distance map resolution was 256x256x198 voxels). This seems fast enough for most purposes as transfer functions are typically only changed occasionally. Distance map updates that occur when the transfer function is changed could be deferred to another GPU thread and swapped in when ready to reduce stuttering on the main rendering thread.

The "pig heart" image shown in Fig. 1 was rendered to evaluate the sparse partially-resident image functionality of our volume renderer. The original image was roughly $11GB$ which reduced to around $7.5GB$ when stored on the GPU with empty space discarded. The opaque rendering in Fig. 1 (left) renders at 157 frames/s, while the highly translucent rendering (right) runs at only 7 frames/s due to the large number of texture samples of the occupied space.

ESS and ERT have limited effectiveness when images with high occupancy are rendered with low opacity transfer functions. A future direction for volume rendering acceleration is to reduce the number of texture samples within occupied regions. The real challenge is to do this in a way which maintains high image fidelity.

5 CONCLUSIONS

A ray casting method was developed which efficiently traverses a distance map to skip over empty regions of a volumetric image. This enables volumetric images with empty space to be visualised with techniques such as volume rendering and isosurface rendering at higher frame rates due to reduced computational complexity.

The frame rate (relative to an unaccelerated baseline) was evaluated for multiple existing empty space skipping approaches on several images. Our Chebyshev distance-based approach more than doubled the performance of the state-of-the-art approach on average. The method requires only a small memory overhead for the distance map and a short initial computation time to generate it.

ACKNOWLEDGMENTS

This work is funded by the Australian Government Research Training Program (AGRTP) with additional support from the Australian National Laboratory for X-ray Micro Computed Tomography.

REFERENCES

Johanna Beyer, Markus Hadwiger, and Hanspeter Pfister. 2015. State-of-the-Art in GPU-Based Large-Scale Volume Visualization. *Comput. Graph. Forum* 34, 8 (Dec. 2015), 13–37. https://doi.org/10.1111/cgf.12605

Enrico Gobbetti, Fabio Marton, and José Antonio Iglesias Guitián. 2008. A Single-Pass GPU Ray Casting Framework for Interactive out-of-Core Rendering of Massive Volumetric Datasets. *The Visual Computer* 24, 7 (July 2008), 797–806. https://doi.org/10.1007/s00371-008-0261-9

Markus Hadwiger, Ali K. Al-Awami, Johanna Beyer, Marco Agus, and Hanspeter Pfister. 2018. SparseLeap: Efficient Empty Space Skipping for Large-Scale Volume Rendering. *IEEE transactions on visualization and computer graphics* 24, 1 (Jan. 2018), 974–983. https://doi.org/10.1109/TVCG.2017.2744238

Markus Hadwiger, Christian Sigg, Henning Scharsach, Khatja Bühler, and Markus Gross. 2005. Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. *Computer Graphics Forum* 24, 3 (2005), 303–312. https://doi.org/10.1111/j.1467-8659.2005.00855.x

Pavol Klacansky. 2019. Open Scientific Visualization Datasets. https://klacansky.com/open-scivis-datasets/

J. Kniss, G. Kindlmann, and C. Hansen. 2002. Multidimensional Transfer Functions for Interactive Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics* 8, 3 (July 2002), 270–285. https://doi.org/10.1109/TVCG.2002.1021579

J. Kruger and R. Westermann. 2003. Acceleration Techniques for GPU-Based Volume Rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. IEEE Computer Society, Washington, DC, USA, 38–. https://doi.org/10.1109/VIS.2003.10001

Toyofumi Saito and Jun-Ichiro Toriwaki. 1994. New Algorithms for Euclidean Distance Transformation of an N-Dimensional Digitized Picture with Applications. *Pattern Recognition* 27, 11 (Nov. 1994), 1551–1565. https://doi.org/10.1016/0031-3203(94)90133-3

M. Sramek and A. Kaufman. 2000. Fast Ray-Tracing of Rectilinear Volume Data Using Distance Transforms. *IEEE Transactions on Visualization and Computer Graphics* 6, 3 (July 2000), 236–252. https://doi.org/10.1109/2945.879785

S. Stegmaier, M. Strengert, T. Klein, and T. Ertl. 2005. A Simple and Flexible Volume Rendering Framework for Graphics-Hardware-Based Raycasting. In *Fourth International Workshop on Volume Graphics, 2005.* Eurographics Association, New York, 187–241. https://doi.org/10.1109/VG.2005.194114

VTK. 2019. VTK - The Visualization Toolkit. https://vtk.org/