High-Performance Graphics (2021)
N. Binder and T. Ritschel (Editors)

Transfer-Function-Independent Acceleration Structure for Volume Rendering in Virtual Reality

B. Faludi[1], N. Zentai[1], M. Żelechowski[1], A. Zam[1], G. Rauter[1], M. Griessen[2] and P. C. Cattin[1]

[1]University of Basel, Switzerland
[2]Diffuse GmbH, Switzerland

Figure 1: *Screenshots from our virtual reality application that uses ray marching to visualize volumetric medical datasets and allows real-time changes to the transfer function, while rendering at the native refresh rate of a head mounted display.*

Abstract

Visualizing volumetric medical datasets in a virtual reality environment enhances the sense of scale and has a wide range of applications in diagnostics, simulation, training, and surgical planning. To avoid motion sickness, rendering at the native refresh rate of the head-mounted display is important, and frame drops have to be avoided. Despite these strict requirements and the high computational complexity of direct volume rendering, it is feasible to provide a comfortable experience using volume ray casting on modern hardware. Many implementations use precomputed gradients or illumination to achieve the targeted frame rate, and most rely on acceleration structures, such as distance maps or octrees, to speed up the ray marching shader. With many of these techniques, the opacity of voxels is baked into the precomputed data, requiring a recomputation when the opacity changes. This makes it difficult to implement features that lead to a sudden change in voxel opacity, such as real-time transfer function editing, transparency masking, or toggling the visibility of segmented tissues.

In this work, we present an empty space skipping technique using an octree that does not have to be recomputed when the transfer function is changed and performs well even when more complex transfer functions are used. We encode the content of the volume as bitfields in the octree and are able to skip empty areas, even with transfer functions that cannot efficiently be represented as a simple range of voxel values. We show that our approach allows arbitrarily editing of the transfer function in real-time while maintaining the target frame rate of 90 Hz.

CCS Concepts

• ***Computing methodologies*** → ***Virtual reality; Rendering; Ray tracing;*** • ***Human-centered computing*** → *Virtual reality;*

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

DOI: 10.2312/hpg.20211279

1. Introduction

Realistic visualizations of 3D medical data, such as Computed Tomography (CT) or Magnetic Resonance (MR) images, are important for diagnosis and surgical planning. Direct volume rendering (DVR) techniques, such as volume ray casting [Lev88] or shear warping [CU92], can be used to render an image directly from the volumetric dataset instead of segmenting the relevant tissue and generating surface mesh models for rasterization. By defining a transfer function (TF), the user is able to control the visibility, the color, and possibly other rendering parameters of different tissue types. Combined with suitable lighting and shading models, volume ray casting can achieve excellent image quality and is widely used in many medical applications [KPV14; SMBT03; FZG*19]. Real-time interactive volume rendering has been used in a clinical setting for over 25 years [JHB*96], although the image quality and rendering performance has improved considerably since then, thanks to increasing computing power. Hardware accelerated ray casting further improved the performance and user experience of these systems. A typical CT or MR image used for diagnosis consists of a single 16-bit channel and up to 1000^3 voxels. Therefore, in this work we focus on an in-core setting, where the entire dataset fits into GPU memory.

Since the recent wide-spread availability of cost-effective consumer-grade virtual reality (VR) hardware, it is now possible to visualize volumetric datasets in a virtual environment while providing a comfortable and immersive user experience. There are several benefits of such a VR application. Surgical planning in VR has been shown to benefit from an increased accuracy of the surgical plans and to improve the surgical outcome [VLA*03; SGR*02]. Through the enhanced sense of depth and scale, surgeons can get an improved spatial visualization of the pathological tissue and gain a better understanding of the geometry of complex fractures or deformities [SKR*08]. Furthermore, VR applications provide a more intuitive and direct user interface. Instead of indirectly controlling the view and move objects using a mouse, the user can naturally move through the virtual environment and interact with objects via hand tracking or motion-tracked controllers.

VR hardware manufacturers generally recommend targeting the native refresh rate of the head mounted display (HMD) for the best user experience [Ant15]. Since current HMDs do not support adaptive refresh rates, low frame rates can quickly result in nausea and a subpar user experience [KCDM19; SSB*20]. Although asynchronous timewarp [BG16] or asynchronous reprojection [Vla16] can alleviate these issues during a temporary performance drop, they cause rendering artifacts and should not be relied upon in general.

To achieve the targeted performance, a number of different optimizations can be implemented in a volume renderer. Some of these optimizations trade visual quality for better performance, for instance by allocating fewer rays to the outer periphery using variable rate shading or foveated rendering in conjunction with eye tracking [PSK*16]. Other optimization approaches involve precomputing certain aspects of the volume rendering workload, such as lighting or shading related data, thereby improving the run-time performance.

Empty space skipping (also know as space leaping) is an optimization that is employed by most ray marching renderers. Generally, it involves generating an acceleration structure that stores some information about empty regions within the volume. This information is used during ray marching to efficiently identify segments along the ray that can safely be skipped without missing any opaque voxels. Since the selected TF directly affects the opacity of every voxel, some acceleration structures have to be recomputed when the TF is changed. Depending on the chosen method and dataset, this cannot be done in a synchronous manner without affecting the rendering performance. This is especially an issue in VR, where a low performance is detrimental to user experience. For this reason, a transfer-function-dependent empty space skipping method is not an ideal choice when the frame budget is limited and real-time modifications of the TF are desired.

This work presents an empty space skipping method that uses an acceleration structure which does not depend on the currently selected TF. We encode the content of the volume using bitfields at the nodes of an octree. The bitfields are a compact representation of the voxel values found in each octant and are computed directly from the raw data. The selected TF is encoded similarly in a bitfield. Whether an octant contains any visible voxels with a given TF can be verified by a single binary operation on the corresponding bitfields. A change in the TF only requires updating a single uniform variable in the ray marching shader and does not incur a noticeable performance penalty. This allows the user to change the TF on the fly, easily visualizing different tissue types and gaining more context about the relevant anatomy.

2. Related work

The efficient rendering of volumetric data has been studied extensively, and there is a vast collection of work published in this area. Many approaches explore different techniques to group homogeneous areas of the volume, separate transparent from non-transparent voxels, and skip the transparent ones. To do this efficiently, most approaches rely on acceleration structures that are precomputed during the initial loading of a dataset.

Lacroute et al. describe a fast classification algorithm that uses a min-max octree and a multi-dimensional summed area table to efficiently classify voxels based on opacity [LL94]. An opacity change in the TF requires a relatively inexpensive update of the summed area table. Although this technique was used in conjunction with shear warp rendering and run-length encoding of transparent and non-transparent voxel runs along the scan lines, similar hierarchical acceleration structures can be used for volume ray casting as well [Lev90; DH92; KWPH06; WFKH07]. While a min-max octree does not depend on the selected TF, complex TFs with multiple disconnected non-transparent regions cannot accurately be represented by a single range, potentially resulting in unnecessary sampling of the dataset. Ljung et al. use a blockwise sample value distribution for adaptive sampling density selection and thereby as a form of empty space skipping [LLYM04]. This approach is similar to ours, but uses a segmented block histogram in a flat blocking structure and focuses on data compression and level-of-detail-based memory management of out-of-core data.

Distance maps are another popular type of acceleration structure

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

for ray marching [SK00; EI07]. These are 3D textures that store the Euclidian or Chebyshev distance to the nearest non-transparent voxel. Sampling the distance map during ray marching returns a distance that can safely be skipped without missing visible voxels. Due to the efficiency of these queries, distance maps generally outperform hierarchical structures with regards to rendering performance, but take a longer time to generate. Since distance maps have to be regenerated when voxel transparencies change, they are not ideal if the TF is changed frequently.

Deakin et al. use an occupancy map to speed up the generation of the distance map and achieve interactive TF editing with update times between 10 and 37 ms, depending on dataset complexity [DK20]. Considering the already high GPU usage of volume rendering and the typical time budget of 8 ms to 12 ms per frame in VR software, this would require an asynchronous update of the distance map to avoid frame drops, delaying the response to user feedback.

Another common technique is object-order empty space skipping, where a coarse bounding geometry of visible voxels is rasterized into depth buffers, which are used to clamp the rays to the visible areas of the volume [ASK92]. Hadwiger et al. use an occupancy histogram tree to generate the bounding geometry and rasterize it into per-pixel linked lists by merging consecutive segments of the same occupancy. This allows an efficient traversal of the volume and skipping of empty segments, however, both the histogram tree as well as the bounding geometry have to be recomputed after a TF change. Nysjo et al. combine volume ray casting with object-order rendering and exploit the spatial and temporal coherence between the eyes in a VR application [NMN19]. Primary rays are cached as brick meshes, which are rasterized in back-to-front order. A binary occupancy clipmap is used to avoid redundant bricks. This approach is able to maintain 90 FPS while editing the opacity threshold, but takes about 32 frames to converge after switching between distinct isovalues or after quick head movements.

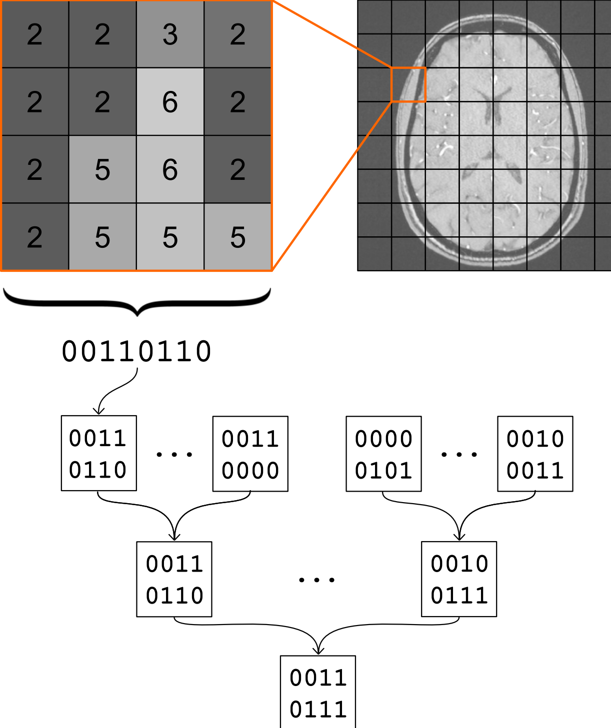


Figure 2: *An illustration of the octree generation process, simplified to 2D. In the upper right, the subdivision of the raw volume data into blocks of 4^3 is shown. To generate the leaf layer of the octree, we process each of these blocks individually, as shown in the upper left. We map each voxel to the index of a voxel value bin and set the corresponding bit of the block's bitfield to 1. Once the leaf nodes have been computed, we compute the higher octree layers consecutively. The values of the nodes in higher layers are computed by applying a bitwise OR to the values of the 8 corresponding nodes in the next lower layer.*

3. Methods

Our empty space skipping approach is based on an octree that stores a hierarchical and compact representation of the raw volume data. We use this data structure during ray marching to efficiently skip empty areas within the volume. The volume renderer was implemented as an extension of the standard rendering pipeline of the Unity Engine, and the shaders were written in DirectX 11 style HLSL. In the following sections, we describe the generation of the octree, the implementation of TF editing and the ray marching process.

3.1. Voxel data histogram

To build an octree, we require a compact description of the voxel values found within a block of the volume. First, we define a range of voxel values that we want to be able to represent and later visualize. By default, this range is automatically set to the minimum and maximum voxel values found within the entire dataset. Optionally, the visualization range can be modified manually by trimming voxel values that are irrelevant to the visualization or that we are not interested in. Changing the range requires recomputing the entire octree and therefore should be chosen such that all voxel values of interest are included.

Once the visualization range is defined, it is split into a number of equal-sized bins. Therefore, each bin corresponds to a consecutive subrange of the overall visualization range. For any given block of the volume, we can store a compact approximation of the voxel values contained within that block in a bitfield, as visualized in the upper part of Figure 2. Each bit in the bitfield corresponds to a single bin of the visualization range. A bit is set to `1` if the block contains any voxel values within the subrange of the corresponding bin, otherwise the bit is set to `0`.

For a straightforward storage of the octree data as a 3D texture on the GPU, it is desirable to limit the byte size of each bitfield to the maximum available size of a single texel. In the case of a 4-bytes-per-channel texture with 4 channels, e.g. `RGBA32UI`, this would give us a maximum of 128 bits per octree node. Alternatively, multiple 3D textures, compute buffers or a linked list data

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

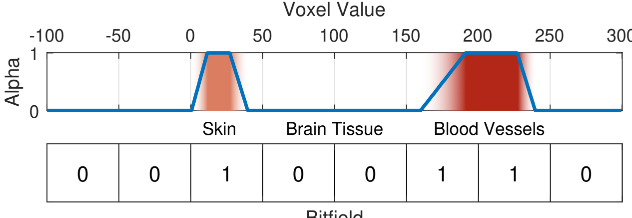


Figure 3: *Schematic illustration of a transfer function that could be applied to an MR scan to visualize the blood vessels in the brain. The horizontal axis corresponds to the voxel values within the visualization range and the overlaid color gradient shows the output of the transfer function. The vertical axis and the blue lines indicate the voxel opacity. The binary representation of the voxel opacities is shown below the color gradient for an exemplary bitfield width of* 8 bits.

structure could be used for a higher resolution representation of the block content. However, due to diminishing returns with increasing bitfield widths, it is unlikely that more than 128 bits per node would improve the performance. In Section 4.3, we go into more detail about the optimal bin size.

3.2. Transfer function

The TF maps the voxel values within the selected visualization range to color and opacity. The user can edit the TF by placing and moving a set of control points on a color gradient editor. Whenever the TF is changed, we first discretize it with a fixed resolution. This allows us to store the TF in a simple 1D lookup table or a 2D pre-integrated lookup table [EKE01], which can be used during the ray marching stage to shade visible voxels.

In addition to the compact representation of the content of blocks of the dataset, we also need a similar representation of the currently configured TF. Similarly to the method described in Section 3.1, we can also generate a bitfield of the same length that encodes a TF. Instead of processing the content of a block of voxels, we determine which bits to set by examining the transparency of the TF. In this case, each bit corresponds to a subrange of the discretized TF, as shown in Figure 3. If all color values within a subrange have an opacity of `0`, we set the corresponding bit to `0`, otherwise to `1`. The resulting bitfield is then used as a uniform shader variable during ray marching.

3.3. Octree generation

Our approach for empty space skipping is a modification of the classic octree-based approach. When loading a new dataset, we generate an octree representation of the voxel data in a one-time automatic preprocessing step. The computation of this octree structure is implemented in a compute shader and takes around 5 ms for a dataset with a resolution of 512^3 on an AMD Radeon RX 6800 XT GPU. A complete benchmark is provided in Section 4. Each node in the octree corresponds to a block of the dataset with the root node encapsulating the entire volume. For every node, we compute a bitfield that describes the voxel values found in the corresponding block of data, as described in 3.1.

We compute the octree in a depth-first bottom-up fashion. Starting at the leaf nodes of the octree, we iterate over all voxel values within the associated block of data and generate the corresponding bitfield representation. Each block is processed in parallel in a single dispatch of the compute shader. The computed bitfields are stored in a 3D texture of appropriate channel count and size to fit a bitfield into a single texel. Every texel in this texture corresponds to a node of the lowest layer of the octree.

Once the first pass of the compute shader completes and all leaf nodes of the octree have been processed, we continue with the internal nodes. An internal node has to represent all voxel values contained within any of its children. This is easily achieved by taking the bitwise OR of the bitfields of every child node, as shown in the lower part of Figure 2. The result is equivalent to computing the bitfield for the internal node from scratch. This process is repeated for the remaining layers of the octree. We store the entire data structure in a single 3D texture with the required channel count and size using a mipmap pyramid layout.

3.4. Ray marching

The ray marching part of our implementation is similar to other octree-based empty space skipping methods. It is implemented as a compute shader and is executed at the end of the rendering pipeline after other objects in the scene have already been fully rendered. We dispatch one thread for every pixel we want to render. Each thread processes a single ray starting at the camera position, going through the corresponding pixel of the image plane and possibly traversing through the volumetric data. Before the ray marching stage, each ray is transformed into the normalized texture space of the volume and clamped to its axis-aligned bounding box. Additionally, the rays are further clamped by the near plane, user-controlled clipping planes, and the scene depth information stored in the camera depth texture.

At the start of the ray marching stage, we first need to determine which block the ray is currently in because the ray origin is not necessarily on the boundary of the volume, for instance, if the camera is inside the volume or a part of the volume has been cut off by a clipping plane. We start at the highest level of the octree that we want to take into account and determine which octant contains the ray origin. If the octant is considered empty with the current TF, the entire octant is skipped. Otherwise, we have to go one level deeper in the octree until we find an empty block. If we reach the lowest level of the octree without finding an empty block, we have to march the ray through that block until the ray exits it.

During the entire ray marching stage, we keep track of the level and the block of the octree that the ray is currently passing through. Whenever we finish ray marching through a block and enter an empty block, we have the possibility of going back to higher levels of the octree and possibly skipping bigger octants at once. The whole process is repeated until the ray completely passes through the volume and exits on the other side.

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

4. Results

For the evaluation of our empty space skipping approach, we conducted a series of performance benchmarks with different datasets and a wide range of rendering parameters. In a first step, we investigated the effect of the different parameters of the bitfield octree on the rendering performance and identified a set of parameters that performed best for a variety of datasets. For a complete comparison, we also implemented other empty space skipping methods in our ray marching shader and compared their performance to our approach. Finally, we tested the performance of our approach in an interactive VR application.

4.1. Datasets

Since our primary use case is in the medical field, we mostly used CT and MR scans of the human body for our performance benchmarks. This includes the Manix dataset [Osi05] containing a human head as well as the well-known Philips Aneurysm dataset. We also tested a CT scan of a torso, which was rendered with a TF highlighting the air inside the lungs, to evaluate any performance differences in the presence of gaps in the TF. Similarly, we used an MR scan of a head to visualize the blood vessels in the brain. This requires the visibility of brain tissue to be turned off and causes an opacity gap in the TF, as shown in Figure 3. Additionally, we used a couple of well-known datasets that have been used in other volume-rendering-related works, such as a scan of a kingsnake [dig03] and of a stag beetle [GGK05].

4.2. Benchmark configuration

We ran our performance benchmarks on a standard Windows 10 workstation equipped with an Intel Core i7-9700K CPU @ 3.60 GHz, 16 GB RAM, and an AMD Radeon RX 6800 XT GPU. We identified a few render resolutions to be most frequently used in related work ranging from 512^2 to 2160^2. Although some earlier HMDs (e.g. Oculus Rift CV1) had a resolution of around 1200^2 pixels per eye, more recent models (e.g. HP Reverb G2) have been released with up to 2160^2 pixels per eye. Therefore, we decided to perform our benchmarks with a rendering resolution of 2160^2 pixels. For better comparability with non-VR related research, we report the performance when rendering just one eye. The reported frame times can be doubled for a rough estimation of the performance with a high resolution 4K HMD.

After loading the dataset, we generate the octree with the selected parameters. For every dataset, we configured the scene so that the entire volume was visible in the viewport while rotating around the vertical axis through its center. After one second of warm-up, we logged every frame delta time for a period of 10 seconds while the dataset completed 2 full rotations. We report the average frame times, as well as the slowest 1 percentile.

4.3. Octree parameters

In the first set of performance benchmarks, we focused on finding ideal parameter sets for our bitfield octree approach. The main parameters of the bitfield octree include the block size at the leaf nodes, the width of the bitfields stored for each node, and the highest level to consider when traversing through the octree. To find the optimal parameters, we performed a hyperparameter grid search where we performed a benchmark with all possible combinations of the above parameters. We considered block sizes in the range of 2^3 to 64^3 and bitfield widths of 8, 16, 32, 64 and 128. We also tested all starting levels in the octree that were possible, given the number of total octree levels with a particular configuration and dataset.

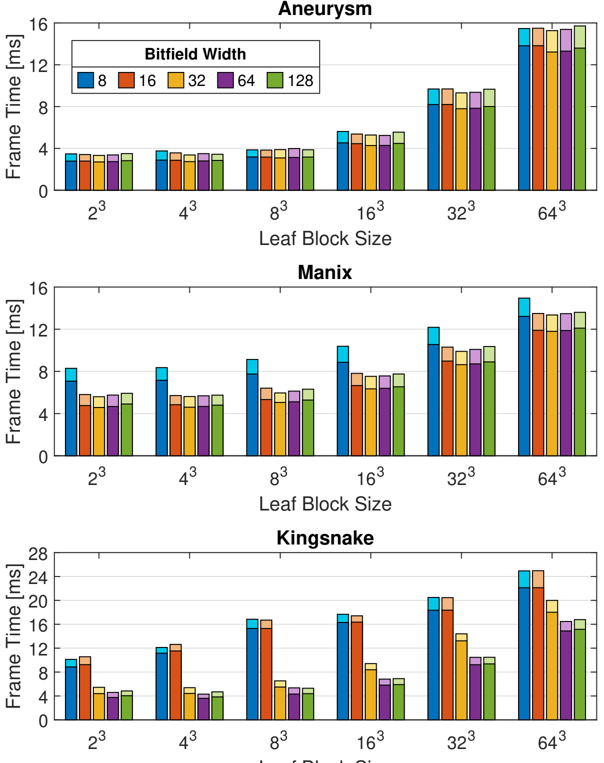
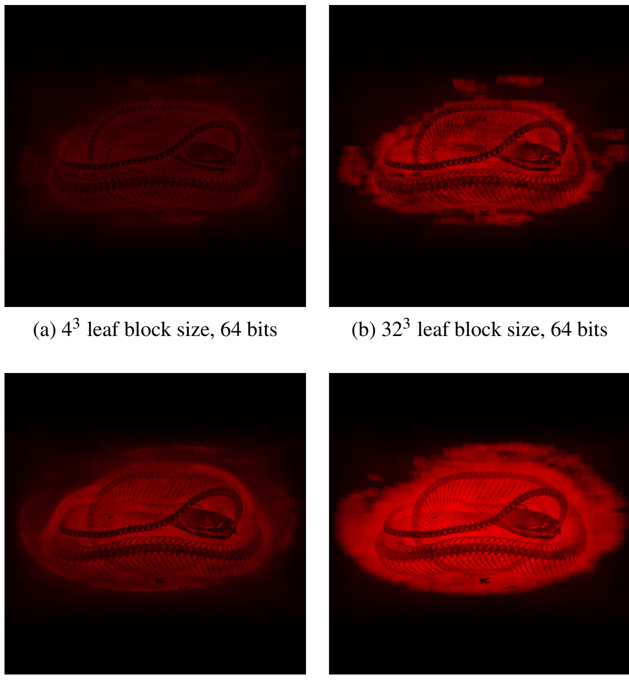


Figure 4: *The ray marching performance using different configurations of the bitfield octree. The vertical axes show the frame delta time over a time span of 10 seconds while the dataset completed two full rotations. The darker tint indicates the average frame times, while the brighter tint shows the slowest 1 percentile. The rendering resolution was set to 2160^2 pixels. Early ray termination and gradient-based shading was deactivated. The horizontal axes group the benchmark results by the block size at the lowest level of the octree. The bar colors indicate the bitfield width used to represent the content of the blocks and the selected transfer function.*

Leaf block size. Figure 4 shows the ray marching performance achieved when starting the octree traversal at the root node and using different combinations of leaf block sizes and bitfield widths. For the more complex and higher resolution datasets (e.g. king snake) the optimal leaf block size was 4^3. Larger block sizes result in a lower performance due to a coarser subdivision of the volume and a less accurate representation of higher frequency details of the dataset. This prevents skipping empty areas that are too small to

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.



(a) 4^3 leaf block size, 64 bits (b) 32^3 leaf block size, 64 bits

(c) 4^3 leaf block size, 16 bits (d) 32^3 leaf block size, 16 bits

Figure 5: *Heatmaps of the number of iterations per ray used to render the king snake dataset using different parameter sets for the bitfield octree. The heatmaps were normalized by the upper limit of allowed iterations (1024).*

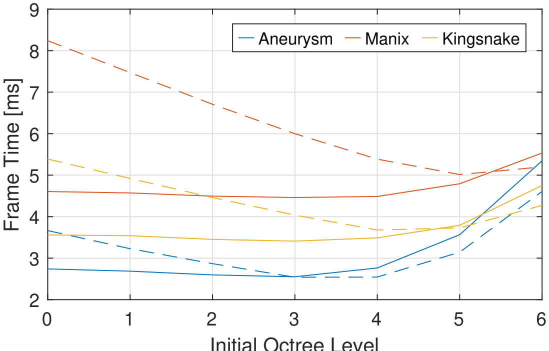


Figure 6: *The ray marching performance with different initial octree levels. Early ray termination was disabled, the leaf block size was set to 4^3 voxels, the bitfield width to 64 bits, and the rendering resolution to 2160^2 pixels. The vertical axis shows the average frame delta times over a time span of 10 seconds while the dataset completed two full rotations. The horizontal axis indicates how many levels of the octree are ignored for empty space skipping, starting at the root. Solid lines correspond to using bidirectional traversal through the octree levels and resuming at the level of the previous check. Dashed lines show the performance of only using top-down traversal and always starting at the initial level for every emptiness check.*

completely encompass an entire leaf block and leads to a higher overall ray step count. The effect of the block size on the iteration count can also be seen in Figure 5 by comparing the iteration heatmaps of the test runs with different leaf block sizes.

While a finer subdivision of the volume allows rays to skip smaller empty areas, it also exhibits diminishing returns as the overhead of computing ray-block intersections reduces the performance benefit of skipping blocks. For higher resolution datasets, this overhead outweighs the performance gain with a leaf block size of 2^3. For less demanding datasets (e.g. Aneurysm), the optimal leaf block size was 2^3.

Bitfield width. Figure 4 also shows the performance when using different bitfield widths for the octree. In the case of the Aneurysm dataset, the performance difference is negligible. When rendering the Manix dataset with a TF that only visualizes bony tissues, there is a noticeable performance decrease with a bitfield width of 8 bits. For the Kingsnake, our most challenging and highest resolution dataset, a performance decrease can also be seen when using 16 bits.

The bitfield width determines how accurately the content of a block of data can be represented in the octree. Additionally, it is used as the resolution of the TF that is used to determine the visibility of blocks as the rays march through the volume. The lowest bitfield width that we tried separates the entire visualization range into only 8 bins. A low number of bins can result in voxel values corresponding to different tissue types to be assigned to the same bin. Voxel values that are assigned to the same bin cannot be differentiated for the purpose of empty space skipping. If any of them are set to be visible by the TF, the entire bin has to be marked as containing visible data. This can result in blocks which cannot be skipped despite containing no visible voxels given the current TF. This issue can be mitigated by increasing the bitfield width and thereby increasing the resolution of the TF's binary representation and improving the ability of the octree to differentiate between tissue types with similar voxel values.

The effect of using different bitfield widths on the ray step count can also be seen in the iteration heatmaps in Figure 5. Comparing heatmap (a) to (c) and heatmap (b) to (d) demonstrates that fewer iterations are needed when the octree can store more detailed information about the content of each block.

Initial octree level. The performance can further be improved by ignoring the highest levels of the octree during ray marching. Figure 6 shows how the performance is affected by omitting an increasing number of levels, starting at the root. Assuming that all datasets that we want to visualize contain at least some visible voxels, the root node of the octree will always evaluate to not-empty and therefore can safely be ignored. Similarly, since most datasets have their main point of interest close to the center of the volume, there is a high chance that all octants of the second level will also contain something visible. Therefore, the first two octree levels can be ignored to avoid unnecessary octree sampling and emptiness checks. Depending on the dataset and the selected TF, ignoring an

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

additional level of the octree can further increase the performance. However, if too many of the higher levels are ignored and only the smaller blocks of the lower levels are used for empty space skipping, the performance can start to degrade. This is most noticeable the case of sparser datasets, such as the Aneurysm dataset, where large empty areas cannot efficiently be traversed without relying on the higher octree levels to skip larger empty blocks.

In addition to using bidirectional traversal through the octree levels, we also evaluated the performance of reducing the branching in the octree traversal by only allowing top-down traversal. With top-down traversal, we always start at the root of the octree (or the initial octree level) for every emptiness query and travel down the tree to search for an empty block. We do not resume the search on the same level of the octree for a subsequent query, because with a top-down only traversal that would prevent rays from skipping bigger blocks after passing by a visible structure in close proximity. With bidirectional traversal, we resume successive queries on the same level and allow checking higher levels if an empty block was found to potentially find a bigger empty block. When a high number of octree levels were ignored, the top-down traversal slightly outperformed the bidirectional implementation, due to a simpler shader. However, the overall best performance was reached with bidirectional traversal and by skipping 2 or 3 octree levels for most datasets.

4.4. Performance comparison

To compare our approach with other empty space skipping methods, we implemented three additional variants and conducted further performance benchmarks. First, we implemented the min-max version of the octree [LL94], where the minimum and maximum voxel values within each octant are stored. During ray marching, these values are compared with the minimum and maximum non-transparent voxel value, as defined by the current TF, and blocks are skipped if these ranges do not overlap. We also implemented a simple Boolean octree variant, which uses precomputed, transfer-function-dependent, binary occupancy flags for each octree node. This method requires a simple sampling of the octree to determine the emptiness of a block, but has to be recomputed if the visibility of voxels changes. Additionally, we compare these octree-based methods with a distance-map-based method. The Euclidean distance map is a 1-channel 3D texture where each voxel stores the length of the longest possible space leap without skipping any visible voxels, i.e., the radius of the biggest sphere centered on a given voxel that only contains invisible voxels. We applied a downsampling of $\frac{1}{4}$ to the distance map, as using a higher resolution did not improve the performance during our testing.

Table 1 shows the results of our performance comparison benchmarks. For every method and dataset, we measured the performance with different shader configurations. We separately report the ray marching performance with and without early ray termination, with only minimal shading, using a simple TF color lookup. We measured the performance impact of applying a physically-based shading model, derived from the Unity Standard Shader, which uses the Disney model [Bur12] for the diffuse component, the GGX model [WMLT07] for the specular component with a Smith Joint GGX visibility term [Hei14] and Schlick's approximation of the fresnel factor [Sch94]. Finally, we benchmarked the performance of the complete rendering pipeline, including real-time ray-marched shadows for a spotlight with a shadow map resolution of 1024^2 pixels.

For the octree-based methods, we report the frame times that were achieved with the best performing parameters, and we report these parameters as well. Additionally, we list the generation duration for each acceleration structure.

For all configurations that we tested, the distance-map-based empty space skipping resulted in the best rendering performance, with about 10–160% more frames rendered per second, compared to the octree-based methods. The generation of the distance map took about $2.5\times$ longer than the octree generation for the simpler datasets and up to $20\times$ longer for the higher resolution datasets. Comparing the octree-based methods with each other, there is a consistent, but minor performance gain with the simple Boolean octree, presumably due to the slightly less complex emptiness check. The min-max octree and the bitfield octree showed almost identical performance for the less challenging datasets and TFs. For more complex TFs, containing regions with alternating opacity, such as the example in Figure 3, the bitfield octree was up to 114% faster. This can be seen in the results for the Head MRI and especially in the Lungs dataset.

Considering that both the distance map and the Boolean octree have to be recomputed when voxel opacities change and the performance advantage of the bitfield approach over the min-max octree for more detailed TFs, our method seems to be an ideal acceleration technique when the editing of TFs in real-time is desired. Although the generation duration of the Boolean octree was less than 10 ms for the lower resolution datasets, it would still consume a considerable part of the limited time budget per frame in a VR application, which could otherwise be spent on improving the image quality with supersampling, and more accurate shading and lighting.

4.5. Real-time transfer function editing

The computational overhead of arbitrarily changing the TF is negligible. We only need to compute the bitfield representation of the TF and update a uniform shader variable to change the visibility of voxels. Additionally, we need to update the albedo color lookup texture, which is required for alpha testing and shading, irrespective of which empty space skipping method is used. If the TF is defined by a list of control points, we only need to upload a minimal amount of data to the GPU and can generate the lookup texture in a compute shader.

To demonstrate this benefit of our approach, we prepared two separate TFs for our datasets and continuously interpolated between them. Before rendering each frame, we computed the linear interpolation between the two TFs, using a time-based sine wave as the interpolant, generated the lookup table on the CPU, and uploaded it to the GPU. Figure 1 shows three exemplary states of this interpolation for the Manix dataset. We were able to render the above example at the native refresh rate (90 Hz) and default rendering resolution (2016×2240 per eye, before lens distortion correction) of an HTC Vive Pro, using an AMD Radeon RX 6800

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

Dataset	Description		None	Distance Map	Boolean Octree	Min-Max Octree	Bitfield Octree
		Marching Only [ms]	12.05	5.98	7.32	15.59	7.91
		+ Early Exit [ms]	7.83	2.76	4.50	10.16	4.75
		+ Shading [ms]	12.47	7.54	8.20	13.41	8.52
		+ Lighting [ms]	14.38	8.21	9.07	15.14	9.46
		Generation [ms]		59.2	7.7	7.1	7.0
	Lungs	Leaf Block Size			2^3	4^3	2^3, 32 bits
	$512 \times 512 \times 690$	Initial Level			5	6	5
		Marching Only [ms]	10.90	7.03	9.81	14.82	11.28
		+ Early Exit [ms]	5.41	1.51	2.33	3.11	2.56
		+ Shading [ms]	7.66	3.89	4.37	4.83	4.59
		+ Lighting [ms]	8.61	4.34	4.93	5.57	5.24
		Generation [ms]		39.6	9.4	7.9	5.3
	Head MRI	Leaf Block Size			2^3	4^3	2^3, 128 bits
	$512 \times 640 \times 184$	Initial Level			3	3	3
		Marching Only [ms]	6.41	0.94	2.46	2.47	2.46
		+ Early Exit [ms]	6.32	0.82	2.22	2.24	2.24
		+ Shading [ms]	6.78	1.18	2.33	2.34	2.35
		+ Lighting [ms]	7.36	1.33	2.54	2.56	2.57
		Generation [ms]		14.0	5.2	5.2	5.2
	Aneurysm	Leaf Block Size			2^3	2^3	2^3, 32 bits
	$256 \times 256 \times 256$	Initial Level			3	3	3
		Marching Only [ms]	9.17	2.59	4.27	4.39	4.35
		+ Early Exit [ms]	6.48	0.92	1.72	1.74	1.74
		+ Shading [ms]	7.81	2.27	2.59	2.67	2.63
		+ Lighting [ms]	9.40	2.51	2.94	3.00	2.97
		Generation [ms]		54.6	4.2	3.5	4.4
	Manix	Leaf Block Size			2^3	4^3	2^3, 32 bits
	$512 \times 446 \times 459$	Initial Level			3	3	3
		Marching Only [ms]	19.31	2.07	2.82	2.87	2.85
		+ Early Exit [ms]	17.20	0.69	1.28	1.30	1.29
		+ Shading [ms]	18.58	1.29	1.63	1.65	1.64
		+ Lighting [ms]	22.49	1.50	1.91	1.95	1.93
		Generation [ms]		118.2	12.0	12.1	9.8
	Beetle	Leaf Block Size			4^3	4^3	4^3, 32 bits
	$832 \times 832 \times 494$	Initial Level			2	2	2
		Marching Only [ms]	23.82	2.35	3.26	3.39	3.45
		+ Early Exit [ms]	21.58	1.29	2.33	2.39	2.41
		+ Shading [ms]	23.38	2.46	2.96	3.04	3.06
		+ Lighting [ms]	29.87	2.75	3.39	3.45	3.51
		Generation [ms]		350.4	18.3	23.5	23.7
	Kingsnake	Leaf Block Size			2^3	4^3	4^3, 64 bits
	$1024 \times 1024 \times 795$	Initial Level			3	3	3

Table 1: *Performance comparison for rendering a collection of datasets using different empty space skipping methods. We report the average frame delta time over a period of 10 seconds, while the dataset completed 2 full rotations. The slowest 1 percentile of frames took on average 27% longer to render and are provided as supplementary material. We run the benchmarks on an AMD Radeon RX 6800 XT GPU and set the rendering resolution to* 2160^2 *pixels. For every method and dataset, we measured the performance with various shader configurations. Marching Only: The performance achieved for ray marching only, merely using transfer function sampling for shading and disabling early ray termination. Early Exit: Ray marching performance with early ray termination enabled. Shading: The performance with gradient-based shading and a single light source. Lighting: The final performance with real-time ray-marched shadows for a single spotlight with a shadow map resolution of* 1024^2 *pixels. We report the generation time for the octrees and distance maps. Additionally, we list the best performing parameters for all octree-based methods that were used for the ray marching only results.*

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

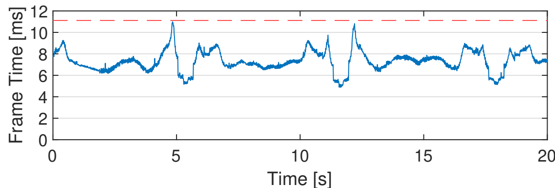


Figure 7: *Frame time plot of rendering the Manix dataset in VR at a resolution of 2016 × 2240 per eye using an AMD Radeon RX 6800 XT GPU. Head tracking was disabled and the camera was rotated around the dataset twice in 20 seconds at a distance of 75 cm from its center. The red dashed line marks the frame budget of 11.1 ms per frame.*

XT GPU. No foveated rendering was used, but early ray termination was enabled. A video recording is provided as supplementary material. Figure 7 shows the frame time plot of this example. Note that the fluctuations in performance are not due to the updating of the TF, but due to the changing rendering complexity of different points of view and different amounts of semi-transparent voxels as the TF is interpolated.

5. Discussion

When compared to other empty space skipping methods, our approach provides several benefits. Changing the TF or updating the positions of the volume, camera and light sources does not require a regeneration of any acceleration structures or caches. Many other approaches rely on distance maps, precomputed node values in an octree, or boundary meshes that have to be partially or fully recomputed when the visibility of any voxel changes. While it is possible to execute these updates in the background or to distribute the workload of recomputation over many frames, this leads to a delayed response to user input and often results in temporarily reduced rendering quality or performance.

Our method only needs to update a single uniform shader variable and a small lookup table to reflect the updated visibility of voxels when the TF is modified. Updating the bitfield representation of the TF is a negligible amount of work and the generation of the lookup texture can be implemented efficiently on the GPU. This allows the user to edit the TF while wearing an HMD and immediately see the results without a performance drop or degradation of visual quality. Being able to change the TF with no noticeable performance hit not only ensures that the user does not experience frame drops and motion sickness, but also provides a new way to explore a medical dataset by quickly changing the visible structures and obtaining a better understanding of the patient's anatomy.

The independence of our acceleration structure from the TF provides flexibility and easy extensibility for additional features. For instance, extending the volume renderer to apply different TFs on either side of a clipping plane is trivial, since we only need to add a second bitmask and lookup table for the alternate TF and can reuse the same octree.

Segmenting different areas of the volume and the ability to selectively hide any segment is a common requirement in medical applications. Automatic segmentation of up to 125 distinct bones has been shown to be feasible with the help of 3D segmentation networks [SHR*20]. Our approach could be extended to support this use case with a second octree of the same structure and up to 128 bits per node, which encoded the segmentation layers, instead of voxel value ranges. Toggling the visibility of segments would have a similarly low performance impact as changing the TF and hidden segments would be skipped efficiently.

6. Conclusions

We presented an acceleration method for volume ray marching that allows changing the TF in real-time with no noticeable performance impact in a VR application. While an empty space skipping implementation with a distance map can provide better rendering performance, it requires a full recomputation after the TF opacity is changed. Our approach outperforms the min-max octree method for TFs that contain intermittent opacity values, which can be used to visualize otherwise hidden structures, such as the air inside the lungs. Such TFs are also helpful when rendering MR scans, where tissue types can map to widely different voxel values, depending on the MRI sequence used. In future work, we want to extend our approach to support toggling the visibility of segments in a labeled dataset and efficiently skip hidden segments. We also want to explore the possibility of combining our empty space skipping method with object-order rendering methods, similarly to Sparse-Leap [HAB*18] or RayCaching [NMN19], to further improve the rendering performance without sacrificing the flexibility of TF independent acceleration.

Acknowledgements

This work was financially supported by the Werner Siemens Foundation through the MIRACLE project.

References

[Ant15] Antonov, Michael. *Asynchronous Timewarp Examined*. Mar. 2015. URL: https://developer.oculus.com/blog/asynchronous-timewarp-examined 2.

[ASK92] Avila, Ricardo S, Sobierajski, Lisa M, and Kaufman, Arie E. "Towards a comprehensive volume visualization system". *Proceedings Visualization'92*. IEEE. 1992, 13–20 3.

[BG16] Beeler, Dean and Gosalia, Anuj. *Asynchronous Timewarp on Oculus Rift*. Mar. 2016. URL: https://developer.oculus.com/blog/asynchronous-timewarp-on-oculus-rift 2.

[Bur12] Burley, Brent. "Physically-Based Shading at Disney". *ACM SIGGRAPH*. Vol. 2012. Walt Disney Animation Studios. 2012, 1–7 7.

[CU92] Cameron, George G and Undrill, Peter E. "Rendering volumetric medical image data on a SIMD-architecture computer". *Proceedings of the Third Eurographics Workshop on Rendering*. 1992, 135–145 2.

[DH92] Danskin, John and Hanrahan, Pat. "Fast algorithms for volume ray tracing". *Proceedings of the 1992 Workshop on Volume Visualization, VVS 1992*. 1992. DOI: 10.1145/147130.147155 2.

[dig03] digimorph.org. *The University of Texas High-Resolution X-ray CT Facility (UTCT), and NSF grant IIS-9874781*. 2003. URL: https://klacansky.com/open-scivis-datasets/ 5.

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.

[DK20] DEAKIN, LACHLAN J. and KNACKSTEDT, MARK A. "Efficient ray casting of volumetric images using distance maps for empty space skipping". *Computational Visual Media* 6.1 (2020). ISSN: 20960662. DOI: 10.1007/s41095-019-0155-y 3.

[EI07] ES, ALPHAN and IŞLER, VEYSI. "Accelerated regular grid traversals using extended anisotropic chessboard distance fields on a parallel stream processor". *Journal of Parallel and Distributed Computing* 67.11 (2007). ISSN: 07437315. DOI: 10.1016/j.jpdc.2007.06.011 3.

[EKE01] ENGEL, KLAUS, KRAUS, MARTIN, and ERTL, THOMAS. "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading". *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*. 2001, 9–16 4.

[FZG*19] FALUDI, BALÁZS, ZOLLER, ESTHER I, GERIG, NICOLAS, et al. "Direct visual and haptic volume rendering of medical data sets for an immersive exploration in virtual reality". *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, 29–37 2.

[GGK05] GRÖLLER, MEISTER EDUARD, GLAESER, GEORG, and KASTNER, JOHANNES. 2005. URL: https://klacansky.com/open-scivis-datasets/ 5.

[HAB*18] HADWIGER, MARKUS, AL-AWAMI, ALI K., BEYER, JOHANNA, et al. "SparseLeap: Efficient Empty Space Skipping for Large-Scale Volume Rendering". *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018). ISSN: 10772626. DOI: 10.1109/TVCG.2017.2744238 9.

[Hei14] HEITZ, ERIC. "Understanding the masking-shadowing function in microfacet-based BRDFs". *Journal of Computer Graphics Techniques* 3.2 (2014), 32–91 7.

[JHB*96] JOHNSON, PAMELA T., HEATH, DAVID G., BLISS, DONALD F., et al. "Three-dimensional CT: Real-time interactive volume rendering". *American Journal of Roentgenology* 167.3 (1996). ISSN: 0361803X. DOI: 10.2214/ajr.167.3.8751655 2.

[KCDM19] KOURTESIS, PANAGIOTIS, COLLINA, SIMONA, DOUMAS, LEONIDAS A.A., and MACPHERSON, SARAH E. *Technological Competence Is a Pre-condition for Effective Implementation of Virtual Reality Head Mounted Displays in Human Neuroscience: A Technological Review and Meta-Analysis*. 2019. DOI: 10.3389/fnhum.2019.00342 2.

[KPV14] KIKINIS, RON, PIEPER, STEVE D, and VOSBURGH, KIRBY G. "3D Slicer: a platform for subject-specific image analysis, visualization, and clinical support". *Intraoperative imaging and image-guided therapy*. Springer, 2014, 277–289 2.

[KWPH06] KNOLL, AARON, WALD, INGO, PARKER, STEVEN, and HANSEN, CHARLES. "Interactive isosurface ray tracing of large octree volumes". *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE. 2006, 115–124 2.

[Lev88] LEVOY, MARC. "Display of Surfaces from Volume Data". *IEEE Computer Graphics and Applications* 8.3 (1988), 29–37. ISSN: 02721716. DOI: 10.1109/38.511 2.

[Lev90] LEVOY, MARC. "Efficient ray tracing of volume data". *ACM Transactions on Graphics* 9.3 (July 1990), 245–261. ISSN: 07300301. DOI: 10.1145/78964.78965. URL: http://portal.acm.org/citation.cfm?doid=78964.78965 2.

[LL94] LACROUTE, PHILIPPE and LEVOY, MARC. "Fast volume rendering using a shear-warp factorization of the viewing transformation". *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994*. 1994. DOI: 10.1145/192161.192283 2, 7.

[LLYM04] LJUNG, PATRIC, LUNDSTROM, CLAES, YNNERMAN, ANDERS, and MUSETH, KEN. "Transfer function based adaptive decompression for volume rendering of large medical data sets". *2004 IEEE Symposium on Volume Visualization and Graphics*. IEEE. 2004, 25–32 2.

[NMN19] NYSJÖ, F., MALMBERG, F., and NYSTRÖM, I. "RayCaching: Amortized Isosurface Rendering for Virtual Reality". *Computer Graphics Forum* (2019). ISSN: 14678659. DOI: 10.1111/cgf.13762 3, 9.

[Osi05] OSIRIX. *DICOM Image Library, Manix dataset*. Feb. 2005. URL: https://www.osirix-viewer.com/resources/dicom-image-library/ 5.

[PSK*16] PATNEY, ANJUL, SALVI, MARCO, KIM, JOOHWAN, et al. "Towards Foveated Rendering for Gaze-Tracked Virtual Reality". *ACM Trans. Graph.* 35.6 (Nov. 2016). ISSN: 0730-0301. DOI: 10.1145/2980179.2980246. URL: https://doi.org/10.1145/2980179.2980246 2.

[Sch94] SCHLICK, CHRISTOPHE. "An inexpensive BRDF model for physically-based rendering". *Computer graphics forum*. Vol. 13. 3. Wiley Online Library. 1994, 233–246 7.

[SGR*02] SEYMOUR, NEAL E., GALLAGHER, ANTHONY G., ROMAN, SANZIANA A., et al. "Virtual reality training improves operating room performance results of a randomized, double-blinded study". *Annals of Surgery*. Vol. 236. 4. 2002. DOI: 10.1097/00000658-200210000-00008 2.

[SHR*20] SCHNIDER, EVA, HORVÁTH, ANTAL, RAUTER, GEORG, et al. "3D Segmentation Networks for Excessive Numbers of Classes: Distinct Bone Segmentation in Upper Bodies". *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 12436 LNCS. 2020. DOI: 10.1007/978-3-030-59861-7{_}5 9.

[SK00] SRAMEK, MILOS and KAUFMAN, ARIE. "Fast ray-tracing of rectilinear volume data using distance transforms". *IEEE Transactions on Visualization and Computer Graphics* 6.3 (2000). ISSN: 10772626. DOI: 10.1109/2945.879785 3.

[SKR*08] STADIE, AXEL THOMAS, KOCKRO, RALF ALFONS, REISCH, ROBERT, et al. "Virtual reality system for planning minimally invasive neurosurgery". *Journal of Neurosurgery JNS* 108.2 (1Feb. 2008), 382–394. DOI: 10.3171/JNS/2008/108/2/0382. URL: https://thejns.org/view/journals/j-neurosurg/108/2/article-p382.xml 2.

[SMBT03] SALGADO, RODRIGO, MULKENS, TOM, BELLINCK, P, and TERMOTE, J. "Volume rendering in clinical practice. A pictorial review". *JBR-BTR : organe de la Société royale belge de radiologie (SRBR) = orgaan van de Koninklijke Belgische Vereniging voor Radiologie (KBVR)* 86 (July 2003), 215–20 2.

[SSB*20] SAREDAKIS, DIMITRIOS, SZPAK, ANCRET, BIRCKHEAD, BRANDON, et al. "Factors associated with virtual reality sickness in head-mounted displays: A systematic review and meta-analysis". *Frontiers in Human Neuroscience* 14 (2020). ISSN: 16625161. DOI: 10.3389/fnhum.2020.00096 2.

[VLA*03] VICECONTI, M., LATTANZI, R., ANTONIETTI, B., et al. "CT-based surgical planning software improves the accuracy of total hip replacement preoperative planning". *Medical Engineering and Physics* 25.5 (2003). ISSN: 13504533. DOI: 10.1016/S1350-4533(03)00018-3 2.

[Vla16] VLACHOS, ALEX. "Advanced VR rendering performance". *Game Developers Conference*. Vol. 2016. 2016 2.

[WFKH07] WALD, INGO, FRIEDRICH, HEIKO, KNOLL, AARON, and HANSEN, CHARLES D. "Interactive isosurface ray tracing of time-varying tetrahedral volumes". *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), 1727–1734 2.

[WMLT07] WALTER, BRUCE, MARSCHNER, STEPHEN R, LI, HONGSONG, and TORRANCE, KENNETH E. "Microfacet Models for Refraction through Rough Surfaces." *Rendering techniques* 2007 (2007), 18th 7.

© 2021 The Author(s)
Eurographics Proceedings © 2021 The Eurographics Association.