DOI: 10.1111/cgf.13771
Eurographics Symposium on Rendering 2019
T. Boubekeur and P. Sen
(Guest Editors)

Volume 38 (2019), Number 4

Adaptive Temporal Sampling for Volumetric Path Tracing of Medical Data

J. Martschinke[1] and S. Hartnagel [1] and B. Keinert[1] and K. Engel [2] and M. Stamminger [1]

[1]University of Erlangen-Nuremberg [2]Siemens Healthineers

Figure 1: *Same-time rendering results of our method (right half) in comparison to the output generated using a single sample per pixel (left half). Frame times are equal for the left and right halves of the images, both taken during interaction (rotation of the object).*

Abstract

Monte-Carlo path tracing techniques can generate stunning visualizations of medical volumetric data. In a clinical context, such renderings turned out to be valuable for communication, education, and diagnosis. Because a large number of computationally expensive lighting samples is required to converge to a smooth result, progressive rendering is the only option for interactive settings: Low-sampled, noisy images are shown while the user explores the data, and as soon as the camera is at rest the view is progressively refined. During interaction, the visual quality is low, which strongly impedes the user's experience. Even worse, when a data set is explored in virtual reality, the camera is never at rest, leading to constantly low image quality and strong flickering. In this work we present an approach to bring volumetric Monte-Carlo path tracing to the interactive domain by reusing samples over time. To this end, we transfer the idea of temporal antialiasing from surface rendering to volume rendering. We show how to reproject volumetric ray samples even though they cannot be pinned to a particular 3D position, present an improved weighting scheme that makes longer history trails possible, and define an error accumulation method that downweights less appropriate older samples. Furthermore, we exploit reprojection information to adaptively determine the number of newly generated path tracing samples for each individual pixel. Our approach is designed for static, medical data with both volumetric and surface-like structures. It achieves good-quality volumetric Monte-Carlo renderings with only little noise, and is also usable in a VR context.

CCS Concepts

• ***Human-centered computing*** → ***Scientific visualization;*** • ***Computing methodologies*** → *Ray tracing;* • ***Applied computing*** → *Health informatics;*

1. Introduction

Path tracing of medical volumetric data can generate stunning visualizations of internal structures of the human body. The global illumination effects introduced by path tracing can improve perception significantly: The more natural look is easier to interpret, in particular for unexperienced viewers, structures become better visible, and depth relations get clearer [DHF*16]. Such renderings are of high value, for example for surgery planning, education, or for doctor-patient communication [CEGM16, GES*18]. On the downside, path tracing – in particular on volumetric data – requires a large number of samples for convergence, so the current application is mostly restricted to precomputed videos. Nevertheless, it is desirable to apply path tracing in interactive applications or even in VR, as interaction with the data (moving, zooming, placing clipping planes, setting markers) is an important part of data exploration.

A simple solution is to use *progressive rendering*, where a single sample per pixel and frame is computed during interaction and the image is continuously refined by adding new samples while the camera is at rest. This approach silently accepts severe noise and reduced user experience throughout interaction. Even worse, in VR applications the user's head is constantly moving and the camera is never at rest, so the viewer only sees very noisy images with heavy temporal flickering. This fact, together with the need for high frame rates and high resolution stereo views, makes the VR case particularly challenging.

Temporal antialiasing (TAA) is a variant of progressive rendering that is designed for the removal of aliasing effects in surface rendering. It reuses samples from previous frames, but contrary to usual progressive rendering, it allows the camera to move in between frames. To this end, pixels from previous frames are reprojected to the current view, which is only possible for surface rendering, where a 3D world position can be assigned to each sample. TAA has been successfully applied to reduce shader aliasing (e.g. at reflections along sharp edges or along shadow boundaries) and aliasing along silhouettes. However, TAA is not directly applicable in volume path tracing, since the color of a pixel stems from an integral along the eye ray, and thus it does not have a well-defined 3D position that can be reprojected. Furthermore, practical TAA implementations rely on heuristics that are not applicable to path-traced volume rendering.

Taking inspiration from TAA, in this paper we present a method to reuse volumetric path tracing samples from previous frames, even while the camera is moving. To make an approach similar to TAA applicable to the task at hand, we introduce several adaptations to overcome the problems arising due to the different nature of both the data source (volumetric vs. surface) and the targeted undersampling effect (Monte-Carlo noise vs. aliasing). Beyond that, our reprojection delivers valuable information about the history of each pixel. We exploit this by including *adaptive sampling*. For pixels with a long trail of reused samples, no or only a few new global illumination samples are generated, whereas more samples are computed for pixels with little history that probably still exhibit significant noise. As a result, we achieve high-quality volume renderings on a single GPU at VR-compatible resolution and frame rates. Our approach has been designed for medical data with both volumetric and surface-like structures. It is less efficient for purely volumetric data, such as clouds or general atmospheric effects, but it can well render volumetric and transparent anatomic structures, for example soft tissue, muscles, vessels, or skin.

2. Previous Work

Interactive Volumetric Rendering Volume rendering has been a very well-researched topic for decades. While there are some non-Monte-Carlo approaches for the rendering of participating media and volumetric data such as flux-limited diffusion [KPS*14], the vast majority of proposed algorithms are based on Monte Carlo path tracing. Especially in recent years, research in this area has seen a large amount of additional advances, e.g. new particle models [SKGM*17], [KHLN17]. In recent work, even deep learning techniques were applied to support volume rendering [KMM*17]. For a thorough overview on most recent techniques, see [NGHJ18]. For a VR application as targeted in this paper, only interactive volume rendering can be taken into account. While the majority of the aforementioned methods focuses on offline rendering, other Monte-Carlo approaches are suited for interactive rendering of volumetric data. Good overviews on the foundations of interactive volume rendering techniques are given in a book by Engel et al. [EHK*04], in a course on advanced illumination techniques for GPU-based volume ray casting by Hadwiger et al. [HLSR09], and in a survey by Jönssen et al. [JSYR12].

Caching-Based Approaches to Volumetric Path Tracing There is a large body of research on enhancing volumetric Monte-Carlo path tracing via caching. The general idea of these methods is to truncate the computationally expensive recursive part of the rendering equation by storing and reusing illumination information. This is typically done for example using virtual point lights, virtual ray lights [NNDJ12], photon mapping [HJG*13, JJD08] and irradiance/radiance caching [Jar08, KG09]. Again, VPL/VRL and photon mapping are designed for an offline setting, utilizing an amount of secondary rays which lies outside the time budget for our use case. Irradiance caching methods seem more appropriate for the problem at hand as they effectively reduce the amount of rays needed for the computation of global illumination in volumes. Khlebnikov et al. [KVS*14] even show how to build the irradiance cache on the fly and use it in interactive rendering, but with the need for highly complex GPU scheduling. As our approach is also caching-based in a certain sense, we will compare it to an irradiance cache implementation in Sec. 6.

Noise Reduction Techniques for Path Tracing Another thread of research examines the direct de-noising of path-traced images, e.g. [SKW*17]. Most of these approaches also consider the temporal domain for filtering [CSK*17], and - like in our approach - reuse samples in-between frames for noise reduction [HDMS03, NSL*07, HBGM11, MMBJ17, ZRJ*15]. However, all of the reprojection-based approaches consider surface rendering, where a reprojection is well defined, in contrast to the volumetric samples considered in this paper. There are also many other approaches to the denoising problem, e.g. [MMMG16, RJN16, BRM*16], but they all focus on offline rendering with multiple samples per frame and are therefore not applicable for our scenario. An extensive overview over adaptive sampling and reconstruction techniques for noise reduction is given in a state-of-the-art report by Zwicker et. al. [ZJL*15].

Rendering of Medical Volume Data For the purpose of the exploration of anatomical data, the goal is to visualize internal structures with a mixture of surface-like boundaries and volumetric regions. To this end, novel lighting models have been developed that combine surface shading techniques with volume rendering [Sal07, RDRS10]. These approaches generate stunning insights into anatomical structures, which has been further improved by the Exposure Renderer by Kroes et al. [KPB12], who combines full volume path tracing and lighting techniques known from high-quality rendering. This work opened the door for the visualization of volumetric medical data in a cinematic rendering style [CEGM16, GES*18]. The special nature of medical data, with its mixture of rather uniform volumetric regions and more surface-like

structures, allows us to effectively adapt temporal antialiasing for the problem at hand.

Temporal Antialiasing Aliasing of screen-space image signals due to undersampling is one of the key problems in computer graphics and image synthesis. Practically all image synthesis systems which integrate per-pixel values using a limited number of representative samples suffer from undersampling. Whenever the chosen samples are not sufficient to represent the signal, aforementioned undersampling occurs and leads to both spatial and temporal artifacts: flickering, jagged edges/string of beads and noise. In order to overcome these unpleasant artifacts, a multitude of different techniques has been developed over time. Spatial filtering techniques like morphological antialiasing (MLAA [Res09], [JME*11]) and fast approximate antialiasing (FXAA [Lot11]) try to reduce aliasing in a postprocessing step after each individual frame. Jimenez et al. [JESG12] couple MLAA with additional sampling techniques and also show how to combine it with temporal reprojection in order to overcome temporal flickering artifacts. To the present day, one of the most successful and commonly used techniques in games is the TAA variant proposed by Karis [Kar14]. Instead of relying on any of the abovementioned spatial techniques, this TAA variant is based on multisampling using temporal reprojection of a history buffer into the current frame using exponential averaging. Here, color values retrieved from the history buffer are clipped to a per-pixel color space neighborhood in the current frame in order to reduce ghosting artifacts. Marrs et al. [MSG*18] extend this TAA technique for rasterized images with adaptive ray tracing. They conservatively mask pixels for which TAA will likely fail and sparsely ray trace additional samples for these masked pixels. Their technique uses FXAA for regions for which no temporal history exists and an adaptive combination of TAA and ray tracing for the remaining regions.

3. Background

3.1. Volumetric Lighting Model

We use a lighting model which has been successfully applied in medical applications [CEGM16, GES*18], but our algorithm is not restricted to this model. The input consists of a 3D voxel grid with density values $V(x), x \in [0,1]^3$ and a transfer function that maps these values to absorption and scattering coefficients $\sigma_a(x)$ and $\sigma_s(x)$, respectively. Usually, these are RGB color values, yet to simplify notation we assume σ_a and σ_s to be scalar values.

According to [JSYR12], the radiance of an eye ray (x,ω) is:

$$L(x,\omega) = L_o(x_o,\omega)T(x,x_o) + \int_x^{x_o} (\sigma_a(x')L_e(x',\omega) + \sigma_s(x')L_s(x',\omega))T(x,x')dx, \quad (1)$$

where x_o is the point where the ray (x,ω) leaves the volume, $L_o(x_o,\omega)$ is the background radiance at this position, $L_e(x',\omega)$ is the self-emission of the volume (usually provided by a transfer function), and $L_s(x',\omega)$ is the in-scattered radiance. $T(x,x')$ is the transparency between points x and x', it is determined by accumulating the extinction $\sigma_t(x) = \sigma_a(x) + \sigma_s(x)$ along the line (x,x'):

$$T(x_0,x_1) = exp\left(-\int_{x_0}^{x_1} \sigma_t(x')dx'\right) \quad (2)$$

The above equations contain self-emission within the volume, in-scattering, and lighting from the environment. There are many variants that ignore some of these effects or make simplifying assumptions. For our application, we explicitly do not ignore in-scattering, which makes the problem a global one, and which is needed to generate global illumination effects such as shadows or indirect lighting. As background light, we apply an environment map: $L_o(x_o,\omega) = E(\omega)$.

On the other hand, we ignore self-emission, which is mainly used as a substitute if in-scattering is considered too expensive. Furthermore, we only consider isotropic scattering:

$$L_s(x',\omega) = L_s(x') = \frac{1}{4\pi}\int_\Omega L(x',\omega')d\omega', \quad (3)$$

i.e., to compute in-scattering at a point, we simply integrate over all incident directions with equal importance.

As a result, our integral can be summarized as:

$$L(x,\omega) = T(x,x_o)E(\omega) + \int_x^{x_o} \sigma(x)L_s(x')T(x,x')dx, \quad (4)$$

where we drop the subscript from σ_s.

3.2. Volumetric Path Tracing

Path tracing solves the above integral by averaging samples of the integrand. To compute one sample, the algorithm steps through the volume along the ray, and integrates opacity. At each step position x, a random variable ξ is drawn that decides whether a scattering event appears or not. If a scattering event happens, the ray is terminated, and a new ray that samples the inscattering light $L_s(x)$ is started recursively in a randomly chosen direction. The resulting color from this ray is multiplied by the color of the voxel. When a ray exits the volume, the color of the sample is determined by the environment map E. We use an upper limit to the number of scatter events allowed for one sample: When this limit is reached, the sample is considered absorbed and evaluated to black. The procedure is also illustrated in Fig. 2.

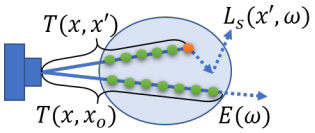


Figure 2: *Path tracing procedure. If a ray runs into a scatter event (top ray), its color is determined by a secondary ray. A ray passing the volume (bottom ray) gets its color from the environment map.*

Due to the high variance of the integrand, some dozens of samples per pixel are necessary to obtain a converged image, which is too costly for interactive rendering, even on latest GPUs. Progressive rendering either cannot be applied to VR as the camera is never at rest. Nevertheless, since camera pose changes in VR are usually relatively small, reusing samples from previous frames by reprojecting them to the current frame makes perfect sense. This concept is well known in surface rendering, where it is used for antialiasing and referred to as *temporal antialiasing* or *temporal reprojection*.

3.3. Temporal Antialiasing for Surfaces

In the most common TAA variant [Kar14], a *history buffer* is utilized which contains the previous frame, including depth. When a new frame is rendered, each new pixel sample is reprojected to the history buffer. If the corresponding pixel shows a nearby surface point, it is combined with the new color by linear interpolation using a factor α as the weight for the history value (and $(1-\alpha)$ as the weight for the new color), thus increasing the effective number of samples of this surface point by one. Otherwise, probably an occlusion or disocclusion appeared, and sampling of the newly visible surface point is restarted, which we call a *reset* in the following. The principle is illustrated in Fig. 3.

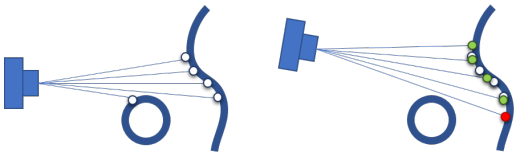


Figure 3: *History buffer for surface rendering: The samples of an image (left) are stored in a history buffer. Samples from the next frame (right) search for nearby samples in the history buffer and include these into their own color (marked green). The sample marked in red has been disoccluded and thus is restarted.*

As a result, each sample gathers previous samples at close positions, filtered with a temporal exponential filter. α can be chosen per pixel using a combination of heuristics (e.g. screen-space velocity, luma difference). A large α is preferable as it results in strong filtering and thus reduced aliasing. However, a large α bears the danger that old, inappropriate samples contribute, which results in ghosting, visible as trails behind moving objects. An effective strategy to reduce ghosting artifacts is to compute the bounding box of the color values of the direct pixel neighborhood and to clamp the history sample with respect to this box (*color clamping*) or to project it onto the box toward the current color (*color clipping*) before accumulation [Kar14].

4. Temporal Reprojection for Volumetric Path Tracing

A direct transfer of TAA to volume rendering is not possible because it is not clear which depth to store in the history buffer. If we store the depth of the first scatter event, the chance that subsequent frames offer similar positions is much lower than for surface rendering, as shown in Fig. 4.

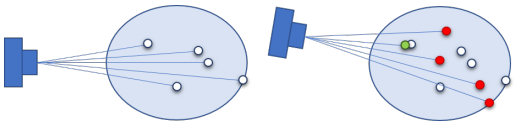


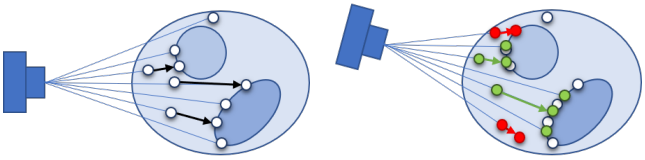
Figure 4: *For volume rendering, a history buffer with scatter event entries does not work well: Since scatter events appear at random positions, only few hits are found, even for nearby views.*

For the case of volume rendering, the colors in the history buffer represent multiple Monte-Carlo samples, accumulated along a ray. We thus have to store not only a single position in the history buffer, but information about the ray along which the samples have been accumulated. Furthermore, since a high number of equally weighted samples is required to achieve convergence, the exponential weighting should be replaced by a filter that weights samples more equally. Finally, due to the high variation of the samples, successful heuristics of TAA for ghosting prevention – color clamping and color clipping – are not applicable in the path-tracing setting.

Our approach is based on the idea of sample reprojection, but it combines three adaptions, that together lead to proper temporal reuse of path-traced volume rendering samples: (i) a heuristic to assign a depth value and information about the entire ray to history samples, (ii) a sample averaging approach that resembles progressive rendering, and (iii) an error accumulation approach that down-weights older, less appropriate samples. In the following we will detail on these adaptations.

4.1. Depth Heuristic for Volumetric Reprojection

To reproject a sample, we need a position that represents the eye ray that generated the sample, such that samples from later, similar eye rays get a similar position. To this end, we trace the primary ray (regardless of scatter events) until its accumulated opacity passes some threshold, e.g. 0.9, and use this depth for the history buffer. If the threshold is not reached before the ray exits the volume, we use the depth of the voxel with the highest opacity value. We refer to the chosen depth as "representative" depth, because it represents all events along the current ray. Later, nearby rays are likely to get a similar representative depth, so more history buffer hits are found and less resets happen. The concept is depicted in Fig. 5.



(a) previous frame **(b)** current frame

Figure 5: *Independently from where the first scatter event occurs, we pinpoint the depth to a surface-like position. This method leads to a high amount of history buffer hits in the next frame, only resetting pixels where occlusion or disocclusion occurs.*

One could think of other choices for the representative depth, for example the depth with the maximum opacity, the point where the ray enters or leaves the volume, or the midpoint. Our choice has the advantage that it gathers history samples close to solid structures, e.g. bones, which are visually the most significant parts.

While the above approach results in long trails of history buffer hits and good filtering, ghosting occurs in transparent regions: As shown in Fig. 6, we accumulate samples from similar rays in a single pixel of the history buffer. If view direction changes, the samples from the history buffer are no longer suited for accumulation, even if they have the same representative position (red ray). As a result, ghosting from the volumetric regions appears.

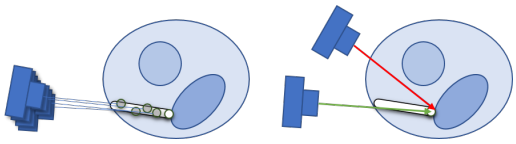


Figure 6: *The history buffer accumulates samples generated for similar view directions, depicted as a white capsule (left). For new eye rays, the accumulated samples should only be used if the directions are similar (green ray), otherwise (red ray) accumulation must restart.*

To avoid this problem, we store in the history buffer not only the representative depth, but also the average direction for which the samples have been generated. The more similar the direction of a new ray is, the more valid is the color of the history buffer. To decide whether a history buffer sample should be reused, we define as the error δ a weighted sum of depth difference and ray direction difference (1 - cosine of the angle between rays), where d denotes depth and r denotes normalized ray direction:

$$\begin{aligned} &\delta = \left|d_{history} - d_{current}\right| + \gamma \cdot \left(1 - \langle r_{history}, r_{current}\rangle\right) \\ &\delta > \varepsilon \rightarrow reset \end{aligned} \tag{5}$$

Both the weight γ for the combination of depth and direction difference and the threshold ε for rejection of samples can be chosen by the user. Besides totally rejecting samples from the history, the combined metric is also used to downweight the contribution of the history buffer, see Sec. 4.3. To be able to do so, we first have to define our sample accumulation scheme.

4.2. Sample Averaging

In original TAA, a fixed weight α is used to compose the new history h_n from a new sample c_n and the color h_{n-1} found in the history buffer, resulting in exponential averaging:

$$\begin{aligned} h_n &= \alpha h_{n-1} + (1-\alpha)c_n \\ \Rightarrow h_n &= \alpha^n c_0 + \sum_{i=1}^{n} \alpha^{n-i}(1-\alpha)c_i \end{aligned} \tag{6}$$

Small values for α lead to insufficient integration, so that noise is hardly reduced, high values result in ghosting due to integration of inappropriate samples. For classical TAA, α can be chosen (almost) arbitrarily high when color clamping or clipping is enabled, as outliers contained in the history buffer are removed using these concepts. As the underlying assumption for these corrections, the fact that values in a local pixel neighborhood are similar, does not hold true for the noisy input in our case, color clamping/clipping cannot be applied effectively due to large bounding boxes in color space.

Therefore, we opt for a different approach of weighting and alter α with respect to how many samples are already included in the history in order to implement a cumulative moving average:

$$\begin{aligned} &h_n = \alpha_n h_{n-1} + (1-\alpha_n)c_n, \quad \alpha_n = \frac{n-1}{n} \\ \Rightarrow &h_n = \left(\prod_{j=1}^{n} \alpha_j\right) c_0 + \sum_{i=1}^{n} \left(\prod_{j=i+1}^{n} \alpha_j\right)(1-\alpha_i)c_i = \frac{1}{n}\sum_{i=1}^{n} c_i \end{aligned} \tag{7}$$

We store the number of samples n included in the history so far, along with the color, depth and direction information for each pixel. When no corresponding sample is found for a certain pixel, n is reset to 1 and only the newly created sample is used. In each step, the direction information for each pixel is updated using α_n just like the color value. This way, the direction is accumulated/reset synchronously with the color and can serve as an estimate for the average direction the current history color comes from.

4.3. Error Accumulation

The choice of a surface-like point as the representative for a ray (see Sec. 4.1) lowers the threshold for a pixel in the history buffer to be considered "corresponding" to a current pixel. This makes history buffer hits more likely and thus reduces noise, but at the expense of blurriness and ghosting. Even if the error δ in depth and direction is below the threshold ε and therefore also the error in color is low for every pair of subsequent frames, it can add up over many frames and lead to wrong colors as slightly inappropriate samples are integrated. We thus keep track of the accumulated error by adding up the single error values δ over time and store this additional information with each pixel of the history buffer. Whenever a history buffer hit occurs ($\delta < \varepsilon$), the error is accumulated in this value. Once it exceeds a threshold ε_{acc}, a "soft reset" is performed by halving the assumed sample count for this pixel. As the weight for the history buffer depends on the sample count, this downgrades the influence of old samples, thus preventing the image from getting blurry over time. The accumulated error itself is also halved during a soft reset, and set to zero whenever a hard reset occurs.

5. Adaptive Sampling

Because depth is less sharply defined as for surface-based rendering, regions remain where pixels are often or even constantly reset due to varying depth values over time. Besides transparent volumetric regions this also applies to pixels along edges. In these two types of regions, history buffer hits are rare, and constant resetting leads to high-variance noise. On the other hand, in opaque regions, the history buffer already integrated many samples, and can directly be used without additional samples. We thus use the sample count information to adapt the number of new samples to be generated per pixel.

First, to reduce noise in critical regions, we keep the number of samples incorporated in each pixel above a minimum number n_{min}: Whenever the reprojection step yields only a small number m of samples stored in the history buffer, or even no samples at all ($m = 0$) due to differing depth/direction and a consequent reset, $n_{min} - m$ new samples are enforced for this pixel by tracing several paths.

Second, we also define an upper limit n_{max} on the number of integrated samples. Whenever this maximum number is found in the history buffer, the pixel is considered "converged", and no new samples are generated.

Because this scheme leads to highly varying ray numbers per pixel, load balancing is needed: In a first fragment shader step, we compute the representative depth and reproject the pixel to the history in order to determine the number of new samples needed for

each pixel. The actual path tracing part is then performed in a compute shader pass, spawning one invocation for each sample needed and synchronizing writes to the same pixel using a spinlock.

The joined minimum/maximum mechanism ensures that computation effort is used exactly where needed and combines high quality with low latency. The parameters n_{min} and n_{max} have to be chosen carefully with respect to quality and performance. For all results in Section 6, we use $n_{min} = 4$ and $n_{max} = 16$.

6. Results

Our method results in good quality renderings of volumetric data with significantly reduced noise compared to an image consisting of only one sample per frame. The accompanying video shows results from our approach during different ways of interaction with the volume.

For a thorough discussion of the outcomes, we compare our results to both the fully converged path-tracing result and the image computed with a single sample per pixel and frame as arising for progressive rendering in the VR setting, where the camera is never at rest. We evaluate our results both in terms of image quality and in terms of rendering speed, as the two criteria are closely intertwined for our approach. Since both performance and quality depend on the amount and kind of change in the viewing angle, we take into account two scenarios: only camera movement resulting from normal head motion in VR, and additional object rotation around its own axis. Constant rotation can be considered a worst-case scenario for temporal reprojection, as permanently new regions appear which cannot have an equivalent in the history yet.

Fig. 7 and Tab. 1 show quality and performance of our approach for the two scenarios described above and a variety of both data sets and transfer functions. The lighting is kept fixed for all experiments. As can be seen from Fig. 7, the rendered images can almost compete with the fully converged path-tracing results, all without the need for a separate denoising step. For both situations, our algorithm performs well, introducing only little blurriness/ghosting while effectively reducing noise. Even during interaction (rightmost column), our method converges to a result similar to the ground truth. As can be seen especially from the enlarged sections, these renderings exhibit some more noise than the ones acquired with usual head motion only (second column from the right) due to the higher amount of resets necessary, but still in an acceptable range. For both scenarios, even tiny structures like vessels (see *Head 1* and *Heart*, first and last row) or fine detail in surgical instruments (see *Torso*, second row) is reproduced quite well by our approach. Our method is most suitable for data sets exhibiting a mixture of more transparent, "volumetric" regions and more opaque, surface-like structures. This is the case for most medical data sets, where bones, vessels etc. yield good representative points. Nevertheless, we can also use the approach for data set - transfer function combinations exhibiting a high amount of volumetric regions (see *Torso* and *Head 2*, second and third row).

Frame times both for our approach and for the single sample rendering, as depicted in Tab. 1, differ depending on data set and transfer function. More interestingly, also the differences between the performance of our approach and the single sample solution vary between test cases: For head motion only, which can be considered the best-case scenario, we can mostly undercut the frame times needed for a single sample per pixel. The more regions with relatively fixed depth values occur in the views, such as bones and even tissue (for example in *Head 1* and *Head 2*), the less resetting occurs and the more our approach benefits from the maximum number imposed on the sample count due to the adaptive sampling scheme. For most data sets and transfer functions, this bonus suffices to outweigh the latency induced by both the search for the representative depth in the reprojection step and the overhead caused by the additional compute shader call. For the worst-case scenario of constant object rotation, the number of pixels for which either a soft or even a hard reset is necessary is high, leading to high numbers of samples to be generated, which is reflected in increased frame times. Nevertheless, for the chosen parameters the frame times even for this scenario do not exceed 150% of the single sample frame times for any data set.

	single sample	Ours (head motion only)	Ours (object rotation)
Head 1	44	21	44
Torso	43	43	52
Head 2	72	51	96
Heart	28	22	42

Table 1: *Frame times in milliseconds for the data set - transfer function combinations shown in Fig. 7. All measurements were conducted on a NVIDIA GeForce GTX 1080, stereo rendering to an Oculus CV1 HMD with a total resolution of* 2160×1200 *pixels.*

For a fair comparison, the parameters error threshold, accumulated error threshold, minimum/maximum sample count and directional weighting coefficient were kept fixed throughout all data sets and transfer functions ($\varepsilon = 0.038, \varepsilon_{acc} = 2 \cdot \varepsilon, n_{min} = 4, n_{max} = 16, \gamma = 0.08$). Nevertheless, these parameters can further be tweaked for each combination of data set, transfer function and application (amount of motion needed, VR/non-VR environment, focus on accuracy or on performance): Higher values for n_{min} and n_{max} as well as smaller values for ε and ε_{acc} improve accuracy at the expense of performance, in particular during strong interaction. Therefore, control over these parameters is given to the user in a simple user interface.

As already mentioned above, the performance of our approach heavily depends on the number of new samples that have to be generated. Fig. 8 visualizes the number of newly generated samples for the *Torso* data set, again for $n_{min} = 4$. Without adaptive sampling, one sample is drawn for each pixel, regardless of how many samples are included in the history buffer (see Fig. 8(a)). For the *Torso* data set, this amounts to around 228000 samples computed each frame, as many as pixels covered by the model. The number of samples needed with adaptive sampling depends on the amount of movement in the scene: For only little change, the total amount of pixels can be much smaller (about 86000 samples, see Fig. 8(b)), leading to high frame rates while preserving detail even in volumetric regions or along the borders of the object, where a minimum sample count of n_{min} is kept by generating more than

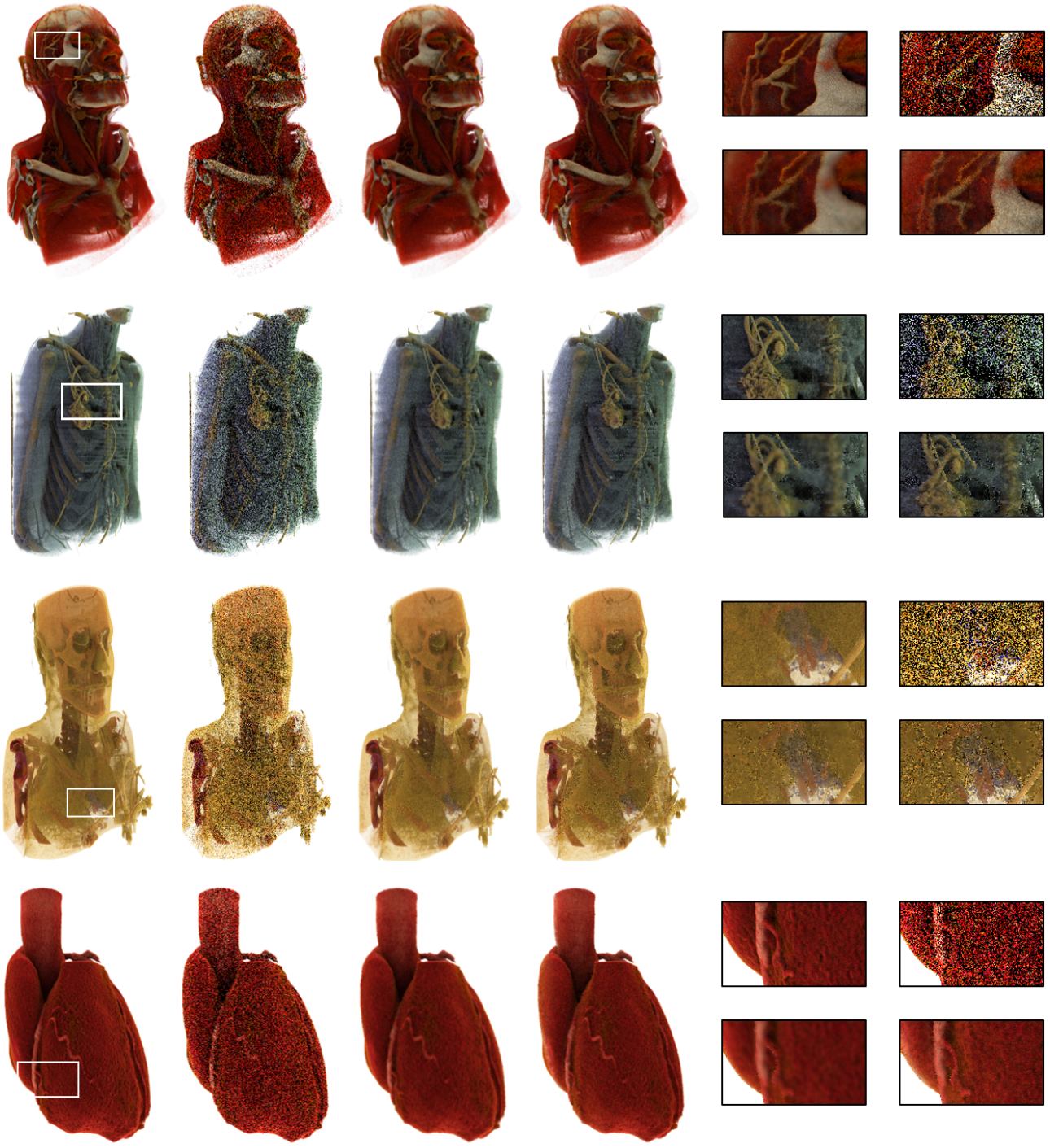
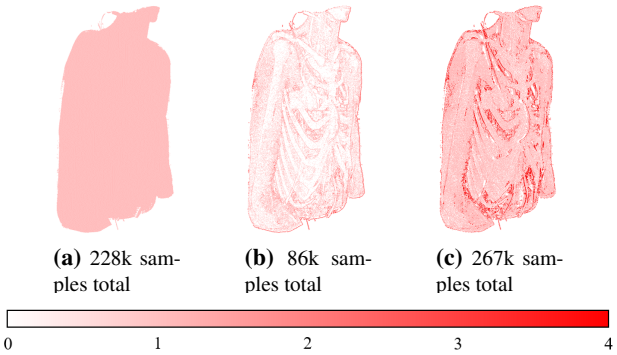


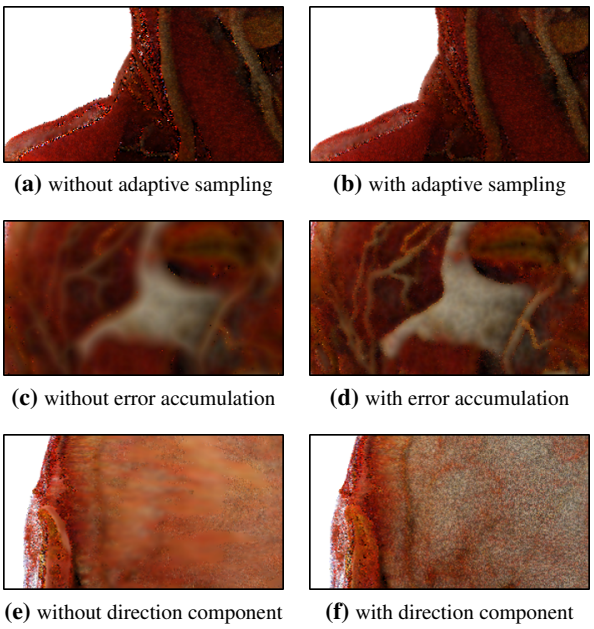
Figure 7: *Results of our method for four different data sets and four different transfer functions. From left to right: fully converged path-tracing result using 64 samples per pixel, image generated using one sample per pixel, image generated using our method during normal head movement but with object at rest, image generated using our method with combined head movement and object rotation. Respective enlarged sections are arranged top left - top right - bottom left - bottom right. The four combinations of data set and transfer functions correspond to the labels (*Head 1, Torso, Head 2, Heart*) appearing in Table 1. The* Torso *data set is rendered using an environment map which casts more blue light on one side and more green light on the other side of the volume.*



(a) 228k samples total **(b)** 86k samples total **(c)** 267k samples total

Figure 8: *Visualization of number of newly drawn samples: (a) without adaptive sampling, (b) with adaptive sampling, head movement only, (c) with adaptive sampling, additional object rotation.*

one sample per pixel. For a higher amount of motion, less samples from the history can be reused, and the average number of samples can exceed 1 (see Fig. 8(c)). On average, the same number of path-traced samples as for the single-sample solution is needed to compute a smooth result, which renders solutions based on a single sample per pixel and relying on a separate denoising post-process obsolete.



(a) without adaptive sampling **(b)** with adaptive sampling

(c) without error accumulation **(d)** with error accumulation

(e) without direction component **(f)** with direction component

Figure 9: *Close-ups of several regions of the* Head 1 *data set, exhibiting artifacts when one of the components of our approach is omitted.*

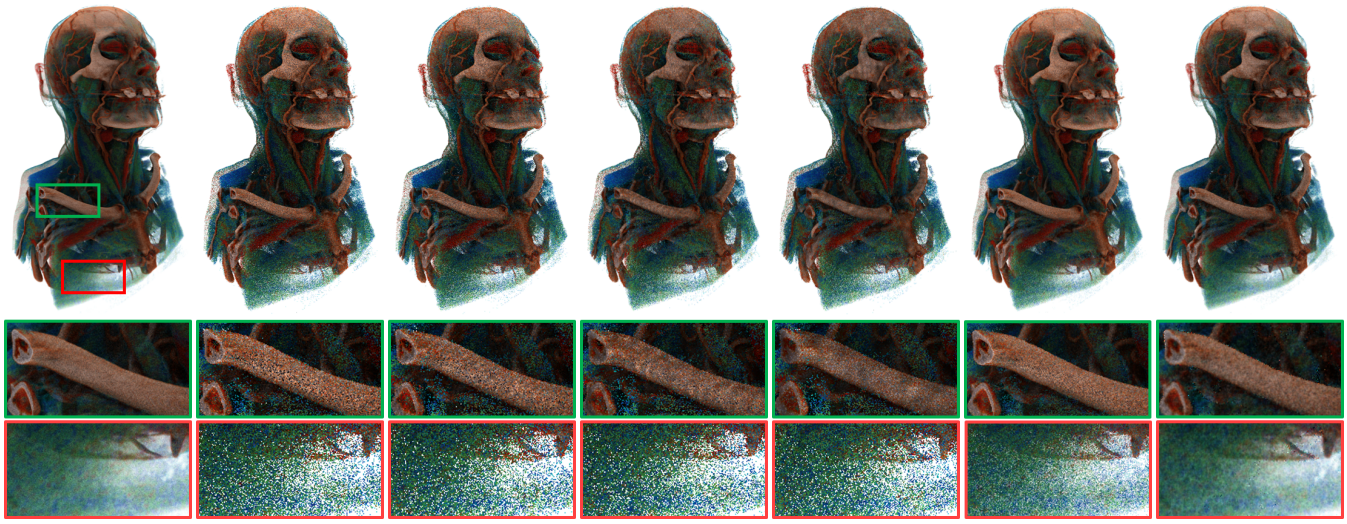
Fig. 9 illustrates the influence of several design choices made in Sections 4 and 5. Without adaptively drawing more samples for recently reset pixels, regions with high-frequency noise remain (see Fig. 9(a)), especially at borders and in highly volumetric areas. Using the adaptive sampling scheme described in Sec. 5, these regions receive a minimum of n_{min} samples, which significantly smooths the result (see Fig. 9(b)). Error accumulation as described in Sec. 4.3 greatly contributes to the sharpness of the result, as can be seen comparing Fig. 9(c) and 9(d): Without the soft reset performed once the accumulated error exceeds ε_{acc}, the results get blurry over time due to the integration of slightly inappropriate samples. When γ is set to zero, ghosting/smearing appears near to solid structures like the skull (see Fig. 9(e)) during movement of the object, as rays from different directions are integrated due to their similar representative depth. The incorporation of the ray direction solves this problem at the expense of slightly higher noise when γ is set appropriately (see Fig. 9(f)).

While most of the Monte-Carlo methods mentioned in Sec. 2 focus on offline rendering, some of them can be extended for the use in online rendering. As an approach eligible for comparison with our method, we choose irradiance caching [Jar08]. To this end, an irradiance volume is computed in a pre-pass. Each voxel stores a precomputed irradiance value obtained by drawing a number of indirect illumination samples (in our experiments: 64). At runtime, only primary rays are traced, and whenever a scatter event is triggered, the cache is queried for the irradiance value. This significantly reduces computational cost at the expense of preprocessing. Khlebnikov et al. [KVS*14] showed how the irradiance cache can be filled on demand even on the GPU, for our comparison we did not use this option and filled the cache completely in a pre-process.

	pre-pass time	frame time
IC, $\lambda = 1$ (b)	624.1*s*	11*ms*
IC, $\lambda = 2$ (c)	72.7*s*	
IC, $\lambda = 4$ (d)	10.8*s*	
IC, $\lambda = 8$ (e)	1.8*s*	
IC, $\lambda = 1$, 4 samples (f)	624.1*s*	44*ms*
ours (g)	-	27*ms*

Table 2: *Frame times in milliseconds and pre-pass times in seconds for the images shown in Fig. 10 (b) to (g). The experiments were conducted using the same setup as described in Tab. 1.*

The general idea of the approach presented in this paper closely resembles the idea of irradiance caching - storage and reuse of intermediate illumination results. A volumetric irradiance cache stores these values in a 3D data structure, whereas our method caches values in screen space. In our experiments, we came to the result that screen-space caching is more beneficial for our scenario: Tab. 2 lists both frame times and precomputation times for our approach and the irradiance caching (IC) method described above, while Fig. 10 shows the obtained results when using the two different methods (again, with fixed lighting). While the irradiance caching approach offers lower frame times than our approach, the computation of the full-resolution cache volume takes minutes, which is unacceptable for data manipulation by the user. In order to reach more reasonable cache setup times, grid resolution can be decreased by a factor λ. However, while for $\lambda = 2$ this works quite well, for $\lambda = 4$ and $\lambda = 8$ voxel boundaries become visible (see Fig. 10), so downsampling is not a useful option for the problem of high cache computation times.



(a) ground truth **(b)** IC, $\lambda = 1$ **(c)** IC, $\lambda = 2$ **(d)** IC, $\lambda = 4$ **(e)** IC, $\lambda = 8$ **(f)** IC, $\lambda = 1$, 4 s. **(g)** ours

Figure 10: *Comparison of both ground truth and our method to renderings using irradiance caching instead of our approach, utilizing different grid sizes for the storage of irradiance values. The volume consists of* $512 \times 512 \times 512$ *voxels, the resolution of the 3D cache is computed as the resolution of the volume reduced by a factor of* λ*. Image (f) was computed with four samples per pixel to approximately match the frame time of our approach. The ground truth image (a) is obtained using volumetric path tracing (as introduced in Sec. 3.2) with 64 samples per pixel.*

But even if we ignore the long pre-process times of irradiance caches, our screen space data structure has the advantage that it also accumulates along eye rays, as can be seen in the enlarged sections bordered red in Fig. 10: If we use only a single eye ray with one single scatter event, where irradiance values are read from the cache, the result is still very noisy even for fixed lighting, whereas our method accumulates along the primary ray, and thus delivers smoother results. For a fair comparison, we can grant four eye ray samples per pixel in combination with irradiance caching, which leads to a frame time comparable to our approach. Still, the amount of noise in Fig. 10(f), which manifests in highly distracting temporal flickering during data exploration, is higher than in Fig. 10(g), which was rendered using our approach. So even if we ignore the setup time for the irradiance cache, we can expect smoother results from our approach. Furthermore, as shown in the accompanying video, our approach can very quickly recover if the transfer function is changed or clipping planes are introduced, and thus the cached values become invalid.

7. Limitations

Our method has been designed for path tracing of medical volumetric data. To some degree, it is optimized for the visualization of bounded, internal structures, as they typically appear in anatomy. The approach is less suited for large homogeneous objects, such as clouds or fog. In this case, when samples are accumulated along long rays, a reprojection of history samples generates a large error, resulting in many integration resets and thus reduced image quality.

As for all temporal antialiasing methods, quality degrades when the camera moves very quickly. However, as can also be seen in the video, the approach returns very quickly to a smooth image, so that the effect is not very disturbing.

When exploring medical data, it should be possible to change transfer functions or to manipulate the data, e.g. by placing clipping planes. However, our approach assumes that the visualized data is static, so such changes are not directly supported. One option is to clear the history buffer after each change to the data, but our experience is that such manipulations are often handled well in practice, because our sampling scheme quickly blends out old, invalid samples and thus smoothly converges to the correct solution. The effect can be seen in the accompanying video.

8. Conclusion

In this paper we have shown that it is possible to apply temporal reprojection also to path-traced volume rendering. Our approach combines several adaptations to the original temporal antialiasing approach, which, in combination, allow us to integrate path tracing samples over rather long time periods. Furthermore, it uses an adaptive sampling approach to reasonably distribute rendering power between pixels. As a result, we are able to render volumetric data interactively and in good quality, with the frame rate depending on the amount and kind of change in the scene and the desired quality of the output. The achievable frame rate (on a single NVIDIA GeForce GTX 1080) in VR resolution is currently 30 fps and more for normal view points, which is acceptable for our scenario of exploring an object.

References

[BRM*16] BITTERLI B., ROUSSELLE F., MOON B., IGLESIAS-GUITIÁN J., ADLER D., MITCHELL K., JAROSZ W., NOVÁK J.: Nonlinearly weighted first-order regression for denoising Monte Carlo renderings. *CGF 35*, 4 (2016). 2

[CEGM16] COMANICIU D., ENGEL K., GEORGESCU B., MANSI T.: Shaping the future through innovations: From medical imaging to precision medicine. *CoRR abs/1605.02029* (2016). 1, 2, 3

[CSK*17] CORSO A. D., SALVI M., KOLB C., FRISVAD J. R., LEFOHN A., LUEBKE D.: Interactive Stable Ray Tracing. In *Proceedings of High Performance Graphics* (2017), ACM, pp. 1:1–1:10. 2

[DHF*16] DAPPA E., HIGASHIGAITO K., FORNARO J., LESCHKA S., WILDERMUTH S., ALKADHI H.: Cinematic rendering – an alternative to volume rendering for 3d computed tomography imaging. *Insights into Imaging 7*, 6 (2016), 849–856. 1

[EHK*04] ENGEL K., HADWIGER M., KNISS J. M., LEFOHN A. E., SALAMA C. R., WEISKOPF D.: Real-time volume graphics. In *ACM Siggraph 2004 Course Notes* (2004), ACM, p. 29. 2

[GES*18] GLEMSER P. A., ENGEL K., SIMONS D., STEFFENS J., SCHLEMMER H.-P., ORAKCIOGLU B.: A new approach for photorealistic visualization of rendered computed tomography images. *World Neurosurgery 114* (2018), e283 – e292. 1, 2, 3

[HBGM11] HENRICH N., BÄRZ J., GROSCH T., MÜLLER S.: Accelerating path tracing by eye path reprojection. 2

[HDMS03] HAVRAN V., DAMEZ C., MYSZKOWSKI K., SEIDEL H.-P.: An Efficient Spatio-temporal Architecture for Animation Rendering. In *ACM SIGGRAPH 2003 Sketches & Applications* (2003), SIGGRAPH '03, ACM, pp. 1–1. 2

[HJG*13] HACHISUKA T., JAROSZ W., GEORGIEV I., KAPLANYAN A., NOWROUZEZAHRAI D.: State of the art in photon density estimation. In *ACM SIGGRAPH Asia Courses* (2013), ACM. 2

[HLSR09] HADWIGER M., LJUNG P., SALAMA C. R., ROPINSKI T.: Advanced illumination techniques for gpu-based volume raycasting. In *ACM SIGGRAPH 2009 Courses* (2009), SIGGRAPH '09, ACM, pp. 2:1–2:166. 2

[Jar08] JAROSZ W.: *Efficient Monte Carlo Methods for Light Transport in Scattering Media*. Ph.D. Thesis, University of California, San Diego, United States – California, 2008. 2, 8

[JESG12] JIMENEZ J., ECHEVARRIA J. I., SOUSA T., GUTIERREZ D.: Smaa: Enhanced morphological antialiasing. *Computer Graphics Forum (Proc. EUROGRAPHICS 2012) 31*, 2 (2012). 3

[JJD08] JAROSZ W., JENSEN H. W., DONNER C.: Advanced global illumination using photon mapping. In *ACM SIGGRAPH Courses* (2008), ACM, p. 2:1–2:112. 2

[JME*11] JIMENEZ J., MASIA B., ECHEVARRIA J. I., NAVARRO F., GUTIERREZ D.: *GPU Pro 2*. AK Peters Ltd., 2011, ch. Practical Morphological Anti-Aliasing. 3

[JSYR12] JÖNSSON D., SUNDÉN E., YNNERMAN A., ROPINSKI T.: A survey of volumetric illumination techniques for interactive volume rendering. *Computer Graphics Forum 33*, 1 (2012), 27–51. 2, 3

[Kar14] KARIS B.: High quality temporal anti-aliasing, 2014. 3, 4

[KG09] KŘIVÁNEK J., GAUTRON P.: *Practical Global Illumination with Irradiance Caching*, vol. 4 of *Synthesis Lectures in Computer Graphics and Animation*. Morgan & Claypool, 2009. 2

[KHLN17] KUTZ P., HABEL R., LI Y. K., NOVÁK J.: Spectral and Decomposition Tracking for Rendering Heterogeneous Volumes. *ACM Trans. Graph. 36*, 4 (2017), 111:1–111:16. 2

[KMM*17] KALLWEIT S., MÜLLER T., MCWILLIAMS B., GROSS M. H., NOVÁK J.: Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *CoRR abs/1709.05418* (2017). 2

[KPB12] KROES T., POST F. H., BOTHA C. P.: Exposure render: An interactive photo-realistic volume rendering framework. *PLOS ONE 7*, 7 (07 2012), 1–10. 2

[KPS*14] KOERNER D., PORTSMOUTH J., SADLO F., ERTL T., EBERHARDT B.: Flux-limited diffusion for multiple scattering in participating media. *Computer Graphics Forum 33*, 6 (2014), 178–189. 2

[KVS*14] KHLEBNIKOV R., VOGLREITER P., STEINBERGER M., KAINZ B., SCHMALSTIEG D.: Parallel irradiance caching for interactive monte-carlo direct volume rendering. In *Proceedings of the 16th Eurographics Conference on Visualization* (2014), EuroVis '14, Eurographics Association, pp. 61–70. 2, 8

[Lot11] LOTTES T.: Fxaa, 2011. URL: https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf. 3

[MMBJ17] MARA M., MCGUIRE M., BITTERLI B., JAROSZ W.: An Efficient Denoising Algorithm for Global Illumination. In *Proceedings of High Performance Graphics* (2017), ACM, pp. 3:1–3:7. 2

[MMMG16] MOON B., MCDONAGH S., MITCHELL K., GROSS M.: Adaptive polynomial rendering. *ACM Transactions on Graphics 35*, 4 (2016), 40:1–40:10. 2

[MSG*18] MARRS A., SPJUT J., GRUEN H., SATHE R., MCGUIRE M.: Adaptive temporal antialiasing. In *ACM SIGGRAPH / Eurographics High Performance Graphics* (August 2018), p. 4. 3

[NGHJ18] NOVÁK J., GEORGIEV I., HANIKA J., JAROSZ W.: Monte carlo methods for volumetric light transport simulation. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports) 37*, 2 (2018). 2

[NNDJ12] NOVÁK J., NOWROUZEZAHRAI D., DACHSBACHER C., JAROSZ W.: Virtual Ray Lights for Rendering Scenes with Participating Media. *ACM Trans. Graph. 31*, 4 (July 2012), 60:1–60:11. 2

[NSL*07] NEHAB D., SANDER P. V., LAWRENCE J., TATARCHUK N., ISIDORO J. R.: Accelerating Real-time Shading with Reverse Reprojection Caching. In *Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (Aire-la-Ville, Switzerland, Switzerland, 2007), GH '07, Eurographics Association, pp. 25–35. 2

[RDRS10] ROPINSKI T., DÖRING C., REZK-SALAMA C.: Interactive volumetric lighting simulating scattering and shadowing. In *2010 IEEE Pacific Visualization Symposium (PacificVis)* (2010), pp. 169–176. 2

[Res09] RESHETOV A.: Morphological antialiasing. In *Proceedings of the Conference on High Performance Graphics 2009* (2009), ACM, pp. 109–116. 3

[RJN16] ROUSSELLE F., JAROSZ W., NOVÁK J.: Image-space control variates for rendering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) 35*, 6 (2016), 169:1–169:12. 2

[Sal07] SALAMA C. R.: GPU-Based Monte-Carlo Volume Raycasting. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)* (Oct. 2007), pp. 411–414. 2

[SKGM*17] SZIRMAY-KALOS L., GEORGIEV I., MAGDICS M., MOLNÁR B., LÉGRÁDY D.: Unbiased Light Transport Estimators for Inhomogeneous Participating Media. *Comput. Graph. Forum 36*, 2 (2017), 9–19. 2

[SKW*17] SCHIED C., KAPLANYAN A., WYMAN C., PATNEY A., CHAITANYA C. R. A., BURGESS J., LIU S., DACHSBACHER C., LEFOHN A., SALVI M.: Spatiotemporal Variance-guided Filtering: Real-time Reconstruction for Path-traced Global Illumination. In *Proceedings of High Performance Graphics* (2017), ACM, pp. 2:1–2:12. 2

[ZJL*15] ZWICKER M., JAROSZ W., LEHTINEN J., MOON B., RAMAMOORTHI R., ROUSSELLE F., SEN P., SOLER C., YOON S.-E.: Recent advances in adaptive sampling and reconstruction for monte carlo rendering. *Computer Graphics Forum (Proceedings of Eurographics - State of the Art Reports) 34*, 2 (2015), 667–681. 2

[ZRJ*15] ZIMMER H., ROUSSELLE F., JAKOB W., WANG O., ADLER D., JAROSZ W., SORKINE-HORNUNG O., SORKINE-HORNUNG A.: Path-space motion estimation and decomposition for robust animation filtering. *CGF 34*, 4 (2015), 131–142. 2