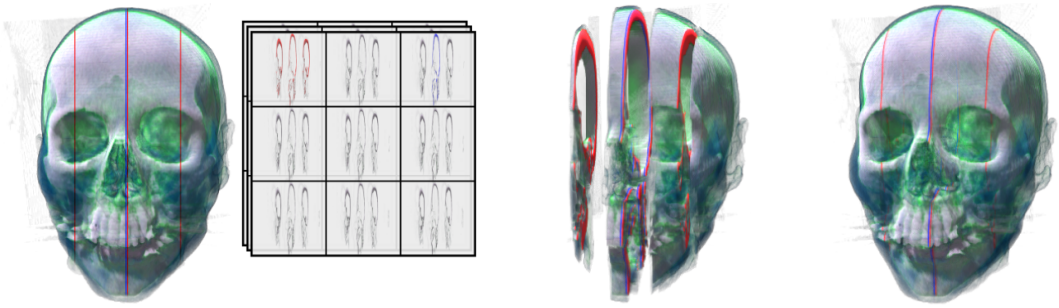
Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2017)
P. Sander and M. Zwicker (Editors)

Single-Pass Stereoscopic GPU Ray Casting Using Re-Projection Layers

Arend Buchacher[1,2] and Marius Erdt[2]

[1]University of Koblenz-Landau, Germany
[2]Fraunhofer IDM@NTU, Visual and Medical Computing, Singapore



(a) *Left View and Re-projection Texture Array* **(b)** *Image-based Compositing* **(c)** *Composed Right View*

Figure 1: *Principle of the single-pass stereoscopic GPU ray casting method. The left view is rendered via ray casting. (a) Vertical scan-lines of rays re-project ray segments to specific layers of a texture array. (b) Subsequently, the layers are blended in an image-based compositing pass. (c) This results in the right view.*

Abstract

Stereoscopic rendering of volume data for virtual reality applications is costly, as the computation complexity virtually doubles compared to common monoscopic rendering. This paper presents a single-pass stereoscopic GPU volume ray casting technique which significantly reduces the time needed to produce the second view. The approach builds upon previous work on ray segment re-projection techniques for non-parallel software ray casting that is initially inapplicable to GPU ray casting. Following the previous approach, ray casting is only executed for the left view. At the same time, ray segments are re-projected to layers of a texture array which leverages the constraints of the previous approach. In a subsequent compositing pass the layers are blended to produce the final image. Additionally, ways to determine an appropriate set of parameters are presented. Performance experiments show significant time savings on producing the second view over the naive two-pass approach achieving well over 60% speed-up in a typical virtual reality setup. The trade-off is an overhead of memory consumption that is proportional to the number of layers and image resolution and a marginal reduction in image quality. In qualitative experiments, average DSSIM values of less than 1% were recorded.

1. Introduction

In recent years, efficient stereoscopic rendering has been becoming a field of interest again. Various new consumer products of head-mounted displays (HMDs) for virtual reality (VR) applications are driving this interest. In the medical field VR is regaining interest as well where in many cases, volumetric medical data needs to be displayed [KJP*15, AHKA12]. Volumes are commonly rendered directly using GPU accelerated image-based ray casting techniques in which the volume is sampled along rays shot from the eye's center. Intuitively, to produce a stereo pair of images the volume is rendered once from the left and once from the right eye, effectively doubling the rendering time.

In software volume rendering where each pixel can be processed in any desired order, projective properties between the two views can be utilized. During the traversal of a ray for one view, each shaded sample can be re-projected to the second view's image. The technique presented by Adelson and Hansen [AH94] requires a strict vertical scan-line order of ray processing to ensure the sam-

DOI: 10.2312/sre.20171190

ples are accumulated in the correct order in the second view, i.e. front-to-back or back-to-front. The substantial increase in perfomance when using GPU accelerated ray casting techniques as presented by Roettger et al. [RGW*03] rises from the parallelization of ray processing. Unfortunately, this is in conflict with the requirement stated above which renders the fast stereo technique inapplicable. The confinement of GPU programs to access and write to multiple images and different image coordinates arbitrarily is another problem that was long insuperable. For example, in OpenGL this ability was added to the core profile only in 2011 through the *image load store* extension.

The current state-of-the-art method to visualize volumetric data in 3D at interactive frame rates is GPU ray casting [RGW*03]. One way to efficiently generate alterative views from existing data is image-based rendering. A notable technique commonly used with polygonal rendering are Layered Depth Images (LDI) that were first introduced by Shade et al. [SGHS98]. For volume rendering, in which continious transparency attenuation is a key factor to the perceived image, layered surface representations are not as expedient. An LDI adaptation for volumes are Explorable Volumetric Depth Images presented by Frey et al. [FSE13]. The view is subdivided into a sliced representation based on depth. Subsequent frames are produced by rendering frusta that approximate the volume's color and opacity between slices. The frusta must be sorted before rendering which poses the need to perform GPU-CPU synchronization. The recent work by Lochmann et al. [LRBR16] yields a similar approach in which the old view is encoded in a piece-wise analytic representation of emission and absorption coefficients. Subsequent frames are produced by sampling this representation along the new rays. In principle, the above methods may be classified as *gathering* approaches in which the resulting image is produced by gathering colors from an intermediate representation of the first view. In contrast, the fast stereo volume rendering technique may be classified as a *scattering* approach in which the resulting image is produced by scattering colors to the target image. Additionally, stereo rendering poses just one of many use-cases for the above methods. They address more generally the efficient generation of frames from arbitrary views in a common single-view setup. The work by Hübner and Pajarola [HP07] addresses the particular case of multi-view rendering for auto-stereoscopic displays. They refrain from adopting the approach by He and Kaufman [HK96] as significantly more and stronger artifacts would be expected for N views. Also, the volume is rendered using a texture-based method using viewport-aligned quadrilaterals.

The remainder of the paper is structured as follows. In Section 2 background and previous work is described, most notably the fast stereo technique that poses the foundation to this paper. In Section 3 the GPU accelerated approach is presented, describing in detail the buffering architecture, array size reduction and optimal parameter estimation. In Section 4 qualitative and quantitative experimental results using current generation hardware are presented and discussed. Section 5 concludes the paper.

2. Background and Previous Work

The technique is based on the works of Adelson and Hansen [AH94] and the extension by He and Kaufman [HK96]. Their work describes how to efficiently produce stereoscopic views of a volume.

2.1. Fast Stereo Volume Rendering

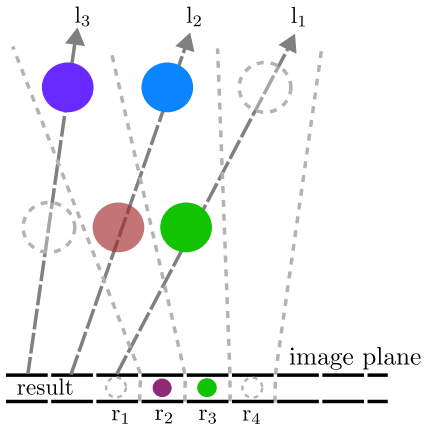


Figure 2: *Scan-line order re-projection of samples along rays (based on figure from [AH94]).* l_i*: left view's ray, colored circles: sample colors,* r_j*: right image's pixel. The projective pixel boundaries are indicated by the light gray dashed lines.*

In short, the work by Adelson and Hansen [AH94] proposes that a stereo pair of images can be produced by rendering the left view using conventional ray casting from the left eye's center. At the same time, the right view is produced by re-projecting each sample point along the left view's rays to the right view's image and accumulating the color. The right image is thus produced in a fraction of the time, since the computationally expensive sampling and shading part is only performed once. In the paper by He and Kaufman [HK96] the algorithm is further enhanced by significantly reducing the memory access operations to the right view's image pixels. By implementing interpolation steps the color-accuracy is enhanced as well. Given that the angle between two corresponding rays is generally small, multiple consecutive samples would be re-projected to the same right view's image pixel. The proposed *segment composition* scheme first accumulates a segment of a left view's ray which corresponds to the range of one right view's image pixel, before re-projecting its color.

It should be noted that both papers discuss only the case of parallel projection. The geometric properties between the views are simpler compared to perspective projection. However, the general idea can be adapted to perspective projection as done in the work by Wan et al. [WZKQ00]. Perspective projection is also more suitable for current generation HMDs and VR applications. The key observation is illustrated by Figure 2. If rays are evaluated in the correct order, the right view's rays will accumulate the values in the same order as well. If the left view's rays are processed right to left and each sampled front to back, then the right view rays will inherently accumulate the values from front to back as well [AH94]. This geometric property is also true if perspective projection is used and the image planes are chosen to be coplanar and parallel.

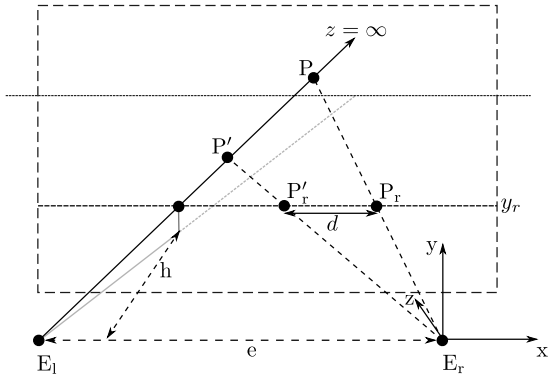


Figure 3: *3D diagram of the stereoscopic perspective projection geometry. A ray is cast from the left eye through the image plane into space at an angle. Dashed rectangle: projection plane,* E_l*: left eye,* E_r *right eye,* h*: focal length,* e*: eye distance,* P,P′*: sample points on left view's ray,* P_r,P_r'*: re-projected sample points,* y_r*: height of epipolar line,* d*: distance between re-projected points.*

2.2. Stereoscopic Perspective Projection Geometry

Figure 3 illustrates the most relevant stereoscopic perspective projection geometry properties using the pinhole camera model. A segment between two points $P(x,y,z)$ and $P'(x',y',z')$ with $h \leq z' \leq z$ on a ray that is emitted from the left eye E_l is re-projected to a segment of at most length d on the epipolar line. This length is equivalent to the distance between the re-projected start and end points $P_r(x_r,y_r,h)$ and $P_r'(x_r',y_r,h)$ with

$$d = \frac{eh}{z} - \frac{eh}{z'}. \tag{1}$$

Equation 1 is derived from the observation that for any point on a left view's ray, for its re-projected x-coordinate x_r it yields

$$\begin{aligned} x_r &= \frac{hx}{z} && (2)\\ &= \frac{h(z\tan\alpha - e)}{z} \\ &= h\tan\alpha - \frac{eh}{z} && (3) \end{aligned}$$

where α is defined by the horizontal angle of the ray to the z-axis.

The views are setup such that both images lie on the same projection plane and are only offset along the x-axis by eye distance e. If the distance e between the two eyes is small enough, the images can overlap. In any case, for every left view's ray the corresponding epipolar line runs parallel to the x-axis at height y_r on the image plane.

3. Proposed Method for GPU Ray Casting

The proposed method is split into two phases, as illustrated by Figure 1. First, the left view is rendered using regular ray casting during which accumulated colors of ray segments are re-projected and stored in layers of a texture array buffer. Second, the textures are blended in the correct per-pixel order such that the right view is composed.

3.1. Re-Projection Layers

As described in Section 2.1 the order of processing the left view's rays is crucial. To assure the segment colors are blended in a front-to-back manner, it must be in vertical scan-lines from the right-most pixels to the left. For software renderers the ray-by-ray order can easily be enforced. In GPU accelerated volume ray casting many rays are traversed in parallel [RGW*03]. However, the order is arbitrary and commonly the hardware rasterizer arranges blocks of pixels to be processed in parallel. Additionally, multiple blocks may be processed in parallel.

Attempting to control the order of fragment shader instances using the vertex shader or altering the shape of the blocks to resemble scan-lines using a GPGPU ray caster is not universally applicable. One such attempt is to reduce the rasterizer's influence on the fragment order by rendering a single vertex per pixel as a point. Vertices are arranged in scan-line order in the vertex buffer. Another idea is to use a GPGPU implementation of a ray caster and to invoke scan-lines of pixels as local work groups. In both cases, some hardware might in fact process the rays in scan-line order, whereas other processes multiple scan-lines or multiple pixels in parallel. As a result flickering artifacts occur, as the order of writes to the right image pixels is arbitrary.

The solution presented in this paper lies within decoupling the compositing from the acquisition of segment colors by introducting a texture array buffer. Instead of reading, compositing and writing the segment colors to the right image directly, each scan-line of rays is assigned a layer of a texture array instead. Following the ray casting phase, the layers are composited to obtain the final right view's image. Thus, a pixel's final color is now independent of the order in which horizontally adjacent rays are processed. Instead, it depends on the order in which the layers are blended which is fixed. The re-projection layers can be filled in parallel.

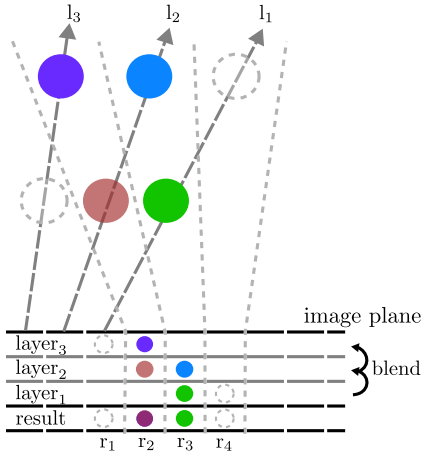


Figure 4: *Re-projection of samples along rays to layers of a texture array.* $layer_k$*: texture array layer. The blend order is front-to-back.*

The approach is illustrated by Figure 4. Based on the horizontal

image coordinate, each ray is assigned a layer of the texture array. The re-projection to right image coordinates is performed as before. The segment color is written to the corresponding position in the layer texture. When all rays have been processed, the final image is composited by blending the layers. Note that no additional depth information is stored in the layers. The blend order results from the mapping of the layers to the left view's rays. Front-to-back blending of the layers in a right-most to left-most ray order thus mimics the front-to-back ray casting of the right view.

3.2. Re-Using Layers for Multiple Rays

The naive implementation would require as many textures as the image is wide in pixels, i.e. rays. To reduce the number of layer textures it's possible to share them between multiple rays. The key observation is that each ray has a start and end point and thus only writes to a bounded horizontal segment of the texture. The space outside of this segment could be used by another ray which writes the corresponding image coordinates, but initially on a different layer. In turn, this ray also only writes to a bounded segment of the texture. Following this pattern, the same texture could be assigned to vertical scan-lines of rays in equal regular steps. The required step size can be estimated using the epipolar geometric properties between the left and right view and the rays. The layers are overlap-free if the step size is at least as big as the maximal re-projected length of any ray.

When re-using layer textures for multiple rays the compositing phase must be modified. Before, a layer's index indicated its spatial relationship to the other layers and thus the blend order. The fully transparent areas in each texture simply corresponded with the empty space that is not covered by the rays. Now, for each pixel the index of the texture must be identified in which the color of the outmost sample can be found. I.e. the layer that was assigned to the right-most left view's ray that re-projected a color to this right view's pixel position. Further details on the calculations can be found in Section 3.3.

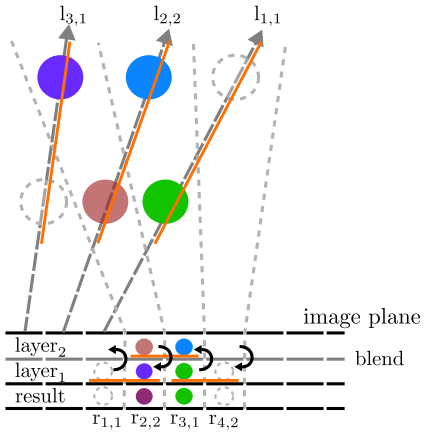


Figure 5: *Re-projection of samples along rays to reduced number of layers of a texture array. Per-pixel front-to-back blend order of layers.*

The described structure is illustrated by Figure 5. The segment each ray traverses is indicated in orange. In this example each is approximately two pixels wide. The second index of each left view's ray $l_{i,k}$ identifies the layer it writes to. The second index of each right view's pixel $r_{j,k}$ identifies the entry layer at which the front-most color is to be found. From this layer, as many layers are blended as the epipolar line segments are wide. A modulo operation using the number of layers is applied to the current index each time it's incremented.

Relevant properties that influence the pixel length d^p of a epipolar line segment are listed below.

1. Distance between cameras e, where an increase in distance increases the pixel length.
2. Distances of start and end points of ray to camera, where an increase of distance of the end point P and a decrement of distance of the start point P' increase the pixel length, respectively.
3. Angle of ray to camera view direction. For a fixed ray length, the segment is maximal when the ray is parallel to the view direction.
4. Width resolution of images w, where an increase of resolution increases the pixel length.
5. Focal length h, where a decrease of length increases the pixel length. Commonly corresponds with the distance to the near plane of the view frustum.

3.3. Estimating Texture Array Buffer Size

Depending on which parameters can be set to a constant, the suitable number of layers or rendering parameters may be calculated. For example, using one of the current consumer VR headsets, such as the HTC Vive, some of the properties can be fixed. It has a display resolution with a width of $w = 1080$ pixels per eye, a recommended field of view of $2\alpha = 110$ degrees and a default eye distance of $e = 0.065$ meters.

It is possible to estimate the minimal number of textures that still ensures that no two rays will interfere with each other. This is achieved by estimating the maximal pixel range d^p that any ray could write to. For quick reference, the following equations will be used in the subsequent descriptions. They comprise the pixel space transformation (Eq. 4, 5), finding the maximal segment lengths for infinite ray length (Eq. 6), for finite ray length (Eq. 7) and for known bounding sphere radius (Eq. 8) and finding the minimal focal length given the other parameters (Eq. 9).

$$s = \frac{w}{2h\tan\alpha} \tag{4}$$

$$d^p = ds \tag{5}$$

$$d_\infty = e \tag{6}$$

$$d_z = e - \frac{eh}{z} \tag{7}$$

$$d_r = e - \frac{eh}{h+2r} \tag{8}$$

$$h = -r + \sqrt{r^2 + \frac{edw}{2d^p\tan\alpha}} \tag{9}$$

It is sufficient to first estimate the maximal epipolar line segment length d. The subsequent transformation to pixel space is achieved

by a scalation by s using the constants, following equations 4 and 5.

The optimal, i.e minimal, value of d^{p} can be approximated through different approaches. One approach is to assume that any given ray ranges from the left camera's near plane all the way to infinity. Given that both view directions are parallel, any point at the horizon is re-projected to the same position as in the left image. A point on the near plane will be re-projected to a point which lies one eye distance e away on the image plane. Thus, the segment on the epipolar line from start to end will have length $d_\infty = e$ (cf. Equation 6).

The distance to the far plane or outmost exit point of all rays can also be taken into account. Equation 7 holds for a ray that begins at the near plane at distance h and ends at distance z, resulting in the estimate d_z. This is suitable for the case in which a volume might enclose the entire view frustum, for example a volumetric terrain.

Considering the case in which the inspected volume has a fixed or maximal size, the range can be computed by finding the longest possible ray that enters the volume at the image plane. Commonly, volumes are represented by a bounding box, thus the longest distance between two points on the bounding box is defined by its diagonal with length $2r$. The end point of the longest possible ray then lies at the distance $z = h + 2r$, which corresponds to the ray that is emitted through the center of the view. Thus, the estimate d_r is found following Equation 8. In this case the re-projected segment on the epipolar line changes with the distance to the volume. Refer to Figure 3, imagining the volume stretches between P' and P. As the entry and exit points move away from the near plane the re-projected points P'_r and P_r drift to the right while d becomes shorter. As stated in Section 3.2 the entry layer to each right view's pixel needs to be identified to ensure the correct blend order. The corresponding left view's ray lies at an offset of $\frac{eh}{z'} - e$ to P'_r. The index is determined by calculating the corresponding left view's image coordinate. Furthermore, the compositing loop can be stopped after fewer iterations as d is also shorter.

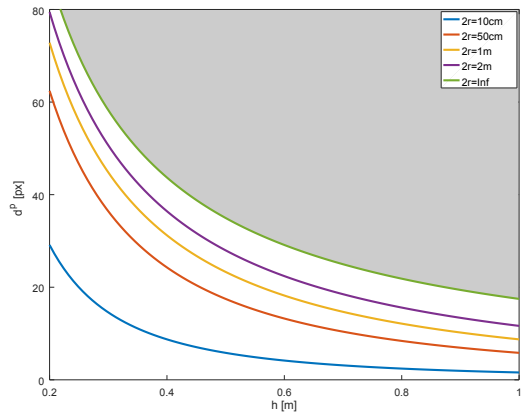


Figure 6: *Pixel range d^{p} plotted against focal length h for different object sizes $2r$ (colored lines). d^{p} calculated with $e = 0.065m$, $2\alpha = 110°$, $w = 768px$.*

Lastly, the minimal focal length h, given all the other parameters including the maximal pixel distance d^{p} can be found following Equation 9. The plot in Figure 6 exemplifies how the focal length maps to the number of layers at different object sizes. The upper bound is given by an object infinite in size, which relates to the case of a ray cast to infinity. For quick reference of practical focal lengths, Figure 7 illustrates typical VR setup configurations. Setups vary especially in distance and simulated size of the object. Deceeding a distance of 0.3m might be undesirable due to squinting. In a room-scale VR application, a volume might be largely scaled up to walk around it at a comfortable distance.

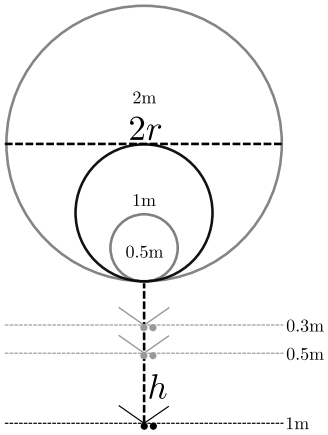


Figure 7: *Diagram of VR setups illustrating distances of HMD to object and object sizes.*

4. Experiments

The performance experiments were conducted on a Windows 10 PC equipped with a Nvidia GTX 1080 graphics card, an Intel Core i7-6700K CPU and 64 GB of RAM.

Volume	Size	Voxels
Solid Box	$64 \times 64 \times 64$	262,144
DTI	$128 \times 128 \times 58$	950,272
CT Head	$256 \times 256 \times 113$	7,405,568
Visibile Male	$128 \times 256 \times 256$	8,388,608
Engine	$256 \times 256 \times 256$	16,777,216
Sheep Heart	$352 \times 352 \times 256$	31,719,424
Piggy Bank	$512 \times 512 \times 134$	35,127,296
Bonsai	$512 \times 512 \times 154$	40,370,176

Table 1: *Overview of volume data sets used for experiments.*

A variety of CT, MRI, DTI and synthetic sample data sets of varying resolutions and density properties were used for the experiments. An overview is given in Table 1. A volume is loaded into a single-component floating point 3D texture.

The ray caster is implemented as a fragment-shader. Each ray samples the 3D texture at a step size of $\frac{1}{2}v$ between the front and back faces of the volume's bounding box where v is defined as the minimal voxel extent in texture space. The sample color is retrieved

using a 1D transfer function texture lookup. Early ray termination is implemented with an opacity-threshold of 0.99. The volume is scaled such that $r = 1$ meter. The focal length h is calculated according to Equation 9.

The structural dissimilarty measure (DSSIM) is used to assess the image quality of the result image. It's a measure derived from the structural similarity measure (SSIM) first introduced by Wang et al. [WBSS04]. Luminance, structure and contrast differences between the images influence its value. It ranges from 0 to 1 where a value of 0 indicates equality. For the experiments, standard SSIM stabilization parameters and a window size of 8×8 were used.

The definition for speed-up (time saving) is adopted from [HK96]:

$$V = 1 - \frac{T_{l+r} - T_l}{T_r}. \tag{10}$$

Here, T_{l+r} is the total time for rendering of a stereo pair in one pass, T_l is the time for rendering the left image, and T_r is the time for rendering the right image. Thus, the definition reflects that the technique does not produce a speed-up for rendering the left image, but only that time can be saved on generating the right image. The stated values for DSSIM, T_{l+r} and V are the averages over a 50-step 360° rotation of the volume around the y-axis.

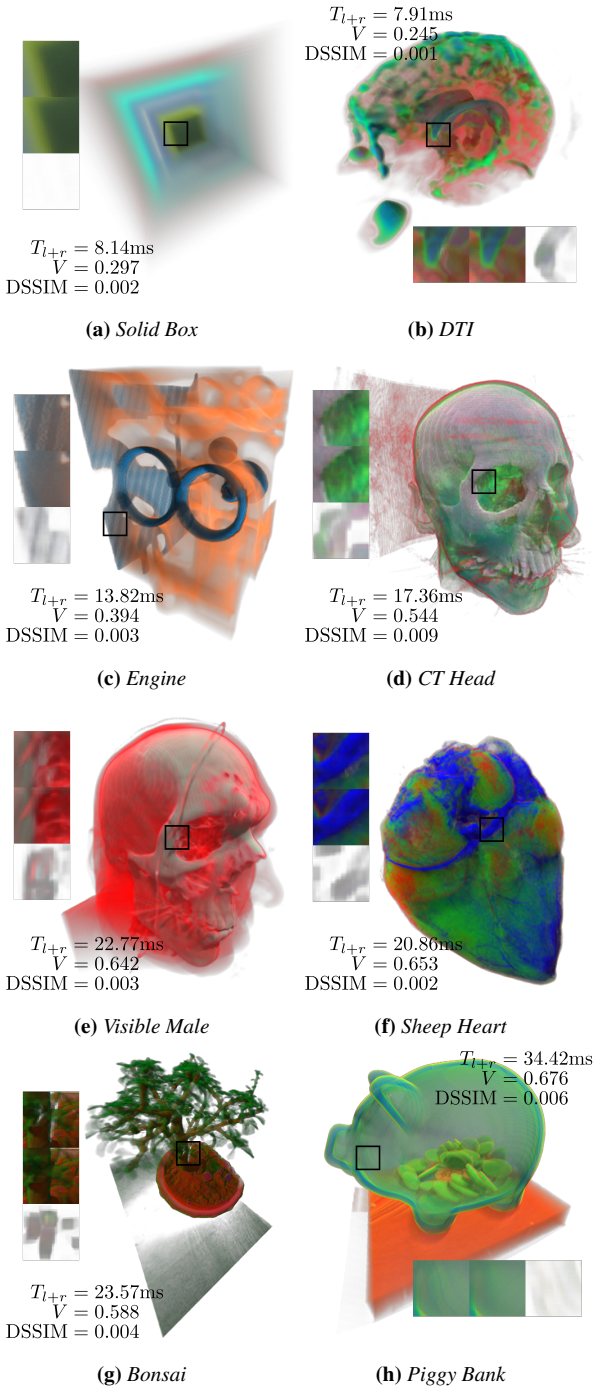
4.1. Qualitative Results

For a qualitative evaluation Figure 8 gives an overview of generated views of the sample volumes. For each ray sample a shadowing effect is applied by accumulating the opacities of 24 samples in direction of a parallel light source and multiplying the result with the sample's color. Additionally, an ambient occlusion effect is applied by averaging the opacitites of 14 samples on a sphere around the sample and multiplying the result with the sample's color.

DSSIM values are generally low, but not zero. Part of the error is due the fact that distances between samples on the left rays are slightly different from the right. Therefore the emission and absorption integral is slightly different as well. Similar to the approach in [LRBR16] the error could potentially be reduced by encoding emission-absorption-coefficients in each layer and calculating the distance between layers during the compositing phase. Another approach would be adapting the linearly-interpolated re-projection scheme proposed in [HK96]. Note that neither approach would be able to achieve full equality.

Due to early-ray termination of left view's rays, some information might be missing near high opacity structures. For example, missing samples become evident near the bonsai tree's trunk in Figure 8g. However, more often not all rays that run across the same image region are terminated early. Thus, the effect is not as drastic as in common LDI rendering where hole-filling strategies such as [SA12] become necessary.

Also note that in the experimental VR setup with a high field of view the volumes generally did not cover the entire viewport, arguably affecting the DSSIM values in a favorable way.



(a) *Solid Box* **(b)** *DTI*

(c) *Engine* **(d)** *CT Head*

(e) *Visible Male* **(f)** *Sheep Heart*

(g) *Bonsai* **(h)** *Piggy Bank*

Figure 8: *Experimental results for different volumes. Detail views from top to bottom (resp. left to right): Result, reference, per-channel DSSIM image (on white background). Rendered at resolution* 768^2, *32 layers,* $e = 0.065$m, $2\alpha = 110°$, $h = 0.435$m.

4.2. Quantitative Results

Figure 9 illustrates the arbitrary shading method used to parametrize shading complexity on a continuum. The method is

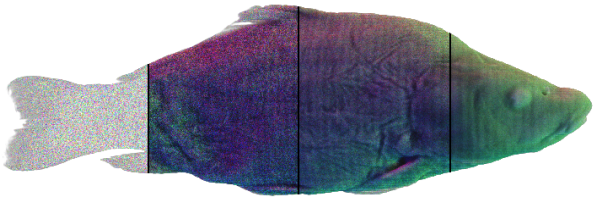


Figure 9: *Shading method using random directional vectors as directional light sources. The (x,y,z) coordinates of a vector are used as the (r,g,b) light intensities coming from that direction. Additional samples along a direction accumulate the occlusion towards that light source. Left to right: 1 direction totaling 0 samples, 1 direction totaling 4 samples, 4 directions totaling 16 samples, 16 directions totaling 64 samples.*

inspired by Monte Carlo Volume Rendering [CSK03] in that a sample's color results from additional samples taken from random directions around it. Each direction is traced for up to 4 steps to estimate the amount of light coming through from that direction. Increasing the amount of directions and number of additional samples increases image quality and reduces noise. Thus, the number of additional texture reads is used as an indicator for shading complexity in general.

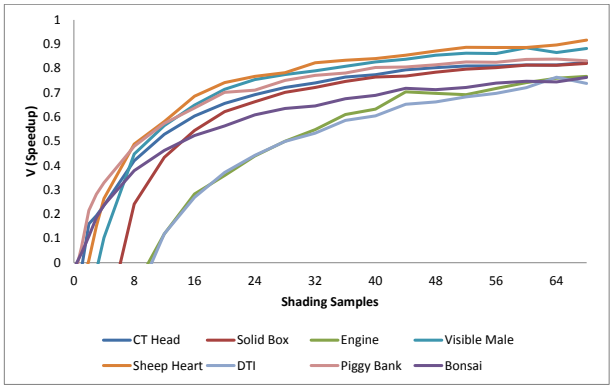


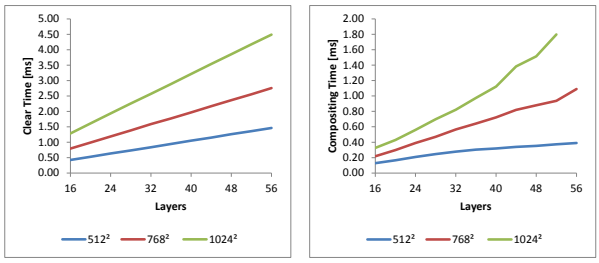
Figure 10: *Recorded speed-up V plotted against shading complexity, represented by number of additional samples required to shade a ray sample.*

Figure 10 depicts the measurements of the average speed-up versus the number of additional shading samples. All experiments were conducted at a resolution of 768^2 pixels using 32 layers. Generally, the break-even point at which the overall cost of shading outweighs the memory access overhead is reached at relatively low shading complexities. For example, with only 4 shading samples, a stereo image pair of the Visible Male volume is produced at 10.75 milliseconds, 10.2% speed-up. For experiments with lower sampling densities for reasons such as regions of fully transparent samples, lower volume resolution, frequent early ray termination, positive speed-up is reached at slightly higher shading complexity. In either case, positive speed-up is reached at real-time rendering times of 10 to 20 milliseconds per stereo image pair. At high shading complexity, speed-up of well above 75% is achieved.

Size [px]	32 layers	64 layers	96 layers
512^2	64 MB	128 MB	192 MB
768^2	144 MB	288 MB	432 MB
1080^2	284.76 MB	569.53 MB	854.29 MB
1512^2	558.14 MB	1116.28 MB	1674.42 MB

Table 2: *Memory consumption for different combinations of number of layers and image sizes. Each pixel holds four* 16 *bit floating point components (RGBA).*

The speed-up V depends on many factors. Most notably the sampling and shading complexity and the relation between the computational overhead to the total render time. The time needed each frame for clearance and compositing is nearly constant for a given size and number of layers. Figure 11 depicts measured overhead computation times. Note that while compositing is generally quick, texture clearing is more prone to become a bottleneck at short rendering times. For example, an average of 1.65 milliseconds of the 7.91 milliseconds recorded for the DTI data set (cf. Figure 8b) were spent on texture clearing. Additionally, the arbitrary image write performance is generally not as optimized as the fragment shader output pipeline. In fact, for simple volume rendering using only a transfer function look-up, but no additional shading samples, the proposed method could generally not out-perform rendering the two views seperately. The minimal focal length h dictates the near plane distance used for rendering. Fortunately for VR applications, current generation HMDs generally have a high field of view such that h is relatively small even at high resolutions.



(a) *Texture Clearing* **(b)** *Compositing*

Figure 11: *Plots of overhead computation times against number of layers for different image resolutions. Texture clearing perfomed by copying data from a pixel buffer object (PBO).*

The texture array creates a memory overhead that is not insignificant. Given an image size of w^2, layer count l, bit depth b and number of channels c the consumed memory M can simply be calculated as $M = w^2lbc$. Table 2 exemplifies the memory consumption for different image sizes and number of layers. Using higher bit depths and floating point formats reduces the effect of quantization on the values produced by ray segments.

5. Conclusion

This paper presented a single-pass stereo rendering technique for GPU ray casting. To this end, the ray casting fragment shader is extended by a re-projection phase in which ray segment colors are

written to layers of a texture array. The second view is efficiently produced in a fast compositing pass in which the buffered colors are accumulated. Multiple factors such as resolution, array size and field of view constrain the applicable near plane distance. Speed-up over rendering views seperately depends largely on the sampling and shading complexity.

Acknowledgments

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative.

References

[AH94] ADELSON S. J., HANSEN C. D.: Fast stereoscopic images with ray-traced volume rendering. In *Proceedings of the 1994 Symposium on Volume Visualization* (New York, NY, USA, 1994), VVS '94, ACM, pp. 3–9. URL: http://doi.acm.org/10.1145/197938.197945, doi:10.1145/197938.197945. 1, 2

[AHKA12] ADESHINAA A., HASHIMB R., KHALIDC N., ABIDIND S. Z.: Medical volume visualization: decades of review. *Computer Science 1* (2012), 152–157. 1

[CSK03] CSEBFALVI B., SZIRMAY-KALOS S.-K.: Monte carlo volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), VIS '03, IEEE Computer Society, pp. 59–. URL: http://dx.doi.org/10.1109/VIS.2003.10000, doi:10.1109/VIS.2003.10000. 7

[FSE13] FREY S., SADLO F., ERTL T.: Explorable volumetric depth images from raycasting. In *Proceedings of the 2013 XXVI Conference on Graphics, Patterns and Images* (Washington, DC, USA, 2013), SIBGRAPI '13, IEEE Computer Society, pp. 123–130. 2

[HK96] HE T., KAUFMAN A.: Fast stereo volume rendering. In *Proceedings of the 7th Conference on Visualization '96* (Los Alamitos, CA, USA, 1996), VIS '96, IEEE Computer Society Press, pp. 49–ff. URL: http://dl.acm.org/citation.cfm?id=244979.245000. 2, 6

[HP07] HÜBNER T., PAJAROLA R.: Single-pass multi-view volume rendering. In *Proceedings of IADIS Multi Conference on Computer Science and Information Systems* (2007), pp. 50–58. 2

[KJP*15] KING F., JAYENDER J., PIEPER S., KAPUR T., LASSO A., FICHTINGER G.: An immersive virtual reality environment for diagnostic imaging. 1

[LRBR16] LOCHMANN G., REINERT B., BUCHACHER A., RITSCHEL T.: Real-time novel-view synthesis for volume rendering using a piecewise-analytic representation. In *Vision, Modelling & Visualization* (2016). 2, 6

[RGW*03] ROETTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart hardware-accelerated volume rendering. In *Proceedings of the Symposium on Data Visualisation 2003* (Aire-la-Ville, Switzerland, Switzerland, 2003), VISSYM '03, Eurographics Association, pp. 231–238. 2, 3

[SA12] SOLH M., ALREGIB G.: Hierarchical hole-filling for depth-based view synthesis in ftv and 3d video. *IEEE Journal of Selected Topics in Signal Processing 6*, 5 (Sept 2012), 495–504. doi:10.1109/JSTSP.2012.2204723. 6

[SGHS98] SHADE J., GORTLER S., HE L.-W., SZELISKI R.: Layered depth images. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 231–242. URL: http://doi.acm.org/10.1145/280814.280882, doi:10.1145/280814.280882. 2

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (April 2004), 600–612. doi:10.1109/TIP.2003.819861. 6

[WZKQ00] WAN M., ZHANG N., KAUFMAN A., QU H.: Interactive stereoscopic rendering of voxel-based terrain. In *Proceedings IEEE Virtual Reality 2000 (Cat. No.00CB37048)* (2000), pp. 197–206. doi:10.1109/VR.2000.840499. 2