SIGRAD 2016, pp. 1–8
M. Hayashi (Editors)

Concepts of Hybrid Data Rendering

T. Gustafsson, W. Engelke, R. Englund and I. Hotz

Linköping University, Department of Science and Technology, Media and Information Technology.

Abstract

We present a concept for interactive rendering of multiple data sets of varying type, including geometry and volumetric data, in one scene with correct transparency. Typical visualization applications involve multiple data fields from various sources. A thorough understanding of such data often requires combined rendering of theses fields. The choice of the visualization concepts, and thus the rendering techniques, depends on the context and type of the individual fields. Efficiently combining different techniques in one scene, however, is not always a straightforward task. We tackle this problem by using an A-buffer based approach to gather color and transparency information from different sources, combine them and generate the final output image. Thereby we put special emphasis on efficiency and low memory consumption to allow a smooth exploration of the data. Therefore, we compare different A-buffer implementations with respect to memory consumption and memory access pattern. Additionally we introduce an early-fragment-discarding heuristic using inter-frame information to speed up the rendering.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

In this paper we discuss methods to combine different rendering concepts for volumes, meshes and texture based techniques for the purpose of interactive hybrid visualizations.

Visualization plays an important role for the analysis of data in many scientific applications. Thereby, typical real world scenarios involve multiple data sets often of different type. This requires efficient solutions for combined rendering of hybrid data in one image. Since different data types often require fundamentally different rendering techniques this can be a challenging task; e.g. the joint visualization of geometry represented as multiple transparent surfaces and volumes using direct volume rendering.

When combining multiple transparent surfaces it is important to blend them in the right order to be perceived correctly. This means the fragments need to be composited in the correct order, either front-to-back or back-to-front. In direct volume rendering, when using volume ray-casting, volume samples are composited in a similar way, in order from the entry point to the exit point. Since volume ray-casting is working on a range of samples at once it is impossible to use the default compositing methods that exists on the GPU to combine volume ray-casting with transparent surfaces.

Order Independent Transparency refers to a collection of techniques to render multiple, possibly complex, transparent surfaces independent of the order of draw calls. Most of the Order Independent Transparency techniques focus on surface representations rather than combining surfaces and volumetric data.

In this paper we will discuss concepts extending existing Order Independent Transparency techniques for interactive hybrid data rendering. In detail we consider

- Mesh rendering for geometry visualization
- Volume Ray-casting for direct volume visualization
- 3D line integral convolution (3D-LIC) for directional data.

Our approach is based on A-buffer techniques and we demonstrate the effectiveness of this on datasets from different fields including a heart dataset containing both anatomical context in the form of surface boundaries and the blood flow. As well as a protein dataset containing various surface representations and the electron charge density volume. With this we present the following contributions:

- A concept for efficient hybrid data rendering based on A-Buffer techniques.

- An improvement over existing techniques by using inter-frame coherence for a fragment discard heuristic.

The remainder of this paper is organized as follows. After related work in Section 2 we introduce some important background knowledge in Section 3. We present the methods of our approach in Section 4. In Section 5 we analyze timing and memory consumption and present our results. Finally we conclude our work with a discussion in Section 6 and name possible future developments.

2. Related Work

In the following we shortly review the most relevant literature for our work. This is at first order independent transparency, basic volume rendering and three dimensional texture rendering.

ORDER INDEPENDENT TRANSPARENCY. The correct rendering of (semi) transparent scenes is an intensively studied field in computer graphics. Early techniques were based on sorting the individual draw calls. Opaque objects are drawn first, followed by (semi) transparent objects. With increasing geometric complexity and an increasing amount of objects in typical computer graphics scenes, this technique lead to high computational costs for the CPU-based sorting of scene elements in a back to front manner. NVidia [Eve01] presented an approach, which uses consecutive peeling of the depth buffer to resolve correct transparency, independent from the order of draw calls. The individual slices of the depth buffer reveal hidden geometry in a consecutive manner. All slices are blended back-to-front in a second render path. This method was later improved by Bavoil and Myers [BM08] where a front-to-back and back-to-front peeling is performed at the same time.

Another approach to render correct transparency, independent of the order of draw calls, is sorting on fragment level. For this, all objects of a scene are rendered and their color and depth values are stored in a buffer. This *A-Buffer* approach can be implemented with layered 2D textures or per-fragment linked lists [BK11]. Maule et al. and Zhang [MCTB12, Zha14] have addressed different performance and memory related aspects of this approach. One challenge with the A-buffer based methods, is the optimization of the local memory for each pixels' fragment list. For example GPU shaders does not allow the usage of arrays with dynamic size. An early work by Sintorn et al. targeted this issue by a rapid pre-computation of the required array size [SEA08]. Later Lindholm et al. [LFS*14] introduced a different approach. Their method is called *per-pixel array optimization* and is based on using multiple shaders which uses different array sizes, and sending pixels of certain depth complexities to a shader using a similar array size. Lindholm also proposes another improvement called *per-pixel depth-peeling*, which removes the problem of having to allocate too large array sizes completely. Schollmeyer et al. presented the integration of an A-Buffer approach with a deferred rendering pipeline [SBF15].

Another method, more similar to the traditional depth-buffer, was presented by Bavoil et al. [BPL*07]. Their *k-buffer* is also capable of handling order independent transparency. It stores the front-most fragment for each pixel, but it can only store up to k fragments, sorted and blended in a single pass. One advantage of this, compared to the A-buffer is that it does not have to define a maximum scene depth.

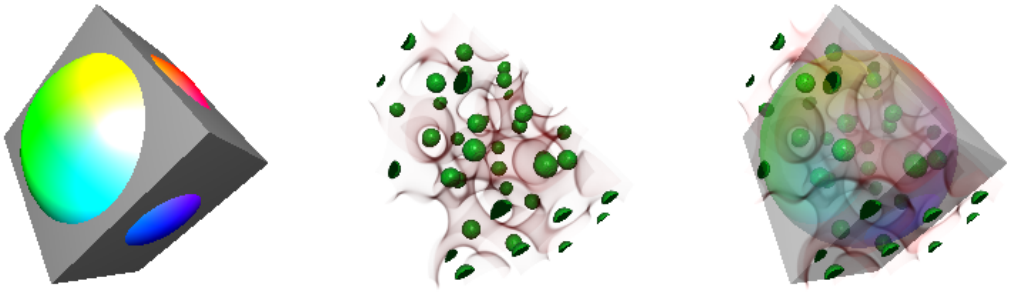
DIRECT VOLUME RENDERING. Volume raycasting is the most intuitive and popular method for direct rendering of volumetric data, with scalar quantities. Early work in this area was undertaken by Cabral et al. [CCF94] where they used 3D texture slicing. More recent approaches render proxy geometry to generate entry and exit point textures on the GPU and perform ray casting afterwards. These approaches have been improved by Krüger and Westerman [KW03] by integrating early ray termination and empty space skipping. Röttger et al. [RGW*03] targeted pre-integration technique, volume clipping and advanced lighting. A flexible framework for standard and non-standard techniques, was presented by Stegmaier et al. [SSKE05]. Their work was easy to extend and able to reproduce translucency, transparent isosurfaces, refraction and reflection. Volume rendering for general polyhedral cells was presented by Muigg et al. [MHDG11] *The Visualization Handbook* [KM05] as well as *Real-Time Volume Graphics* [EHK*06] provide an overview of direct volume rendering techniques.

LINE INTEGRAL CONVOLUTION. Dense texture based techniques are one major group in flow visualization. One representative of this category is line integral convolution, which was first presented by Cabral et al. [CL93]. Stalling and Hege [SH95] presented a fast and resolution independent approach. Later the idea was applied to 3D vector fields by Interrante [Int97] as well as Rezk-Salama et al. [RSHTE99] and surfaces by van Wijk et al. [vW03]. A comprehensive overview can be found in the work of Laramee et al. [LHD*04]. Whereas, Falk and Weiskopf [FW08] targeted 3D line integral convolution in conjunction with direct volume rendering. Additionally they incorporated adaptive noise generation. Therefore the method is independent of the input data size.

3. Background

In this section we will briefly describe the visualization methods we focused on, for the combination with our hybrid data rendering approach.

MESH RENDERING. A typical approach for mesh rendering is to transform the meshes vertices with the current modelview matrix, perform primitive assembly, clipping, perspective division, depth testing and finally put each fragment to the output buffer. OpenGL performs these stages in



(a) (b) (c)

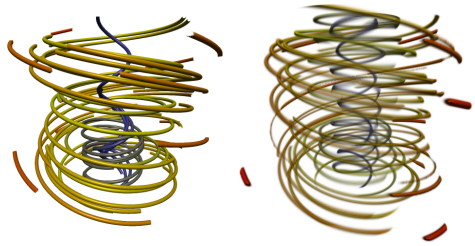
Figure 1: *Our hybrid rendering approach combines rendering of mesh and volume data into one single visualization. The shaded fragments of the mesh rendering (a) are stored with their depth information in an A-Buffer. During ray traversal the volume renderer (b) stores its results in the same A-Buffer. The result (c) is a combined visualization of both source with correct transparency.*

its configurable and customizable pipeline approach. In our current implementation we rely on triangulated meshes. We suggest [SSKLK13] for further reference.

VOLUME RAYCASTING. For direct volume rendering method, a common method is volume ray-casting. Here a three-pass approach can be used. In the first pass a proxy geometry, namely the bounding box of the volume, is rendered. During this pass, back-face culling is enabled and a color buffer is bound as render target. Its content will later serve as entry-point texture. For the second pass, front-face culling is enabled and therefore the exit-point texture is generated. Both passes will interpolate the given texture coordinates at each vertex of the bounding box in an efficient way. Afterwards the third and last pass uses the entry and exit-point textures to define the rays, used for sampling the volume. We suggest *The Visualization Handbook* [KM05] and *Real-Time Volume Graphics* [EHK*06] for further reference.

3D LINE INTEGRAL CONVOLUTION. Line Integral Convolution (LIC) is a well known texture based method for visualization of steady vector fields. The basic idea is to perform a convolution of a white noise input texture. At each location in the field a one-dimensional kernel is defined by integrating a flow tangential line a fixed distance in both, forward and backward direction. The noise texture is sampled along this line and the corresponding values are summed and normalized. Afterwards, the resulting value is placed at the actual location in the output image. Furthermore, the local one-dimensional nature of this algorithm makes it ideal for a parallel and efficient implementation. Today, LIC calculations are commonly performed on graphics hardware and can easily achieve interactive frame rates even for larger input data.

For 3D vector fields the same approach can be used. Instead of a two-dimensional noise texture, a noise volume is utilized to perform the convolution at each voxel. Afterwards, the result is visualized by direct volume rendering. Unless 2D LIC, 3D LIC is more sensitive to the type of noise used for the convolution. A sparsely populated and blurred 3D texture as input, has proven to result in less cluttered results. Figure 2 shows an analytically defined vector field visualized by streamlines as well as 3D LIC. We suggest [CL93] and [FW08] for further reference.



(a) (b)

Figure 2: *Visualizations of an analytically defined vector field using (a) streamlines and (b) 3D LIC.*

NUMERICAL INTEGRATION METHODS. For the calculation of integral lines typical methods are Euler and Runge-Kutta integration. Both methods iteratively compute the integral lines from a given start location. Thereby, Euler integration samples the field only at the beginning of the interval (e.g., current location) to compute the next point and thus suffers from low accuracy. In contrast the fourth-order Runge-Kutta scheme (RK4) samples the field at four different locations within the current interval and uses a linear combination of these samples to compute the next point. The

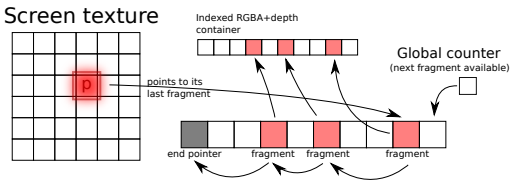


Figure 3: *A screen texture is used to store pointers to the last stored fragment, which in turn points to its previous fragment. By using these pointers, we are able to access the RGBA+depth values of each fragment.*

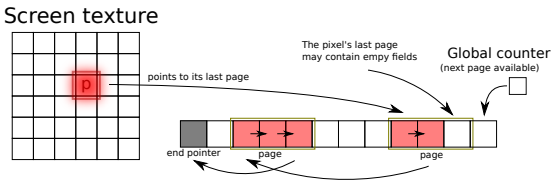


Figure 4: *Here the screen texture points to a page of fragments. In this illustration the page size is three fragments, but it can essentially be any number. By storing the fragments in pages, we get better performance on the GPU.*

RK4 integration is a good trade of between computational cost and accuracy.

4. Hybrid Data Rendering

In this paper we propose a concept to combine different rendering techniques, this is visualizing volumes using *direct volume rendering* together with (semi) transparent geometry. For a combination of mesh rendering and volume raycasting it is important to align the individual coordinate systems. Doing this ensures that the resulting depth values correspond to each other.

In this section we describe the proposed hybrid data rendering method. This includes the data types that can be visualized and the visualization options.

A-Buffer Rendering Methods. Order independent rendering of (semi) transparent scenes is commonly done in two rendering passes. The first pass renders the scene and populates the A-Buffer with fragments, the second pass sorts these fragments and performs back to front compositing to produce the final pixel colors. In order for our A-Buffer to support varying color and transparency of fragments and be able to correctly compose these, we need to be able to store the color, its transparency and the depth value. This results in five different values, in total, which have to be stored per fragment (*r, g, b, a, depth*). Therefore, we are using two buffers, one for (*r, g, b, a*) and one for *depth* values.

An A-Buffer can be implemented with different underlying technologies. The most intuitive approach is storing each rendered fragment in a layered 2D-texture. All layers, for one output fragment, are later blended in the order of their depth value. A major drawback of this approach is that it suffers from a lot of unused memory if the depth complexity varies over the scene.

Another approach uses a *per-pixel linked lists* to store the incoming fragments. This approach requires a single buffer to store all fragments, with an accompanying buffer that stores pointers between these elements. Thereby the first fragment of each coordinate points to the zero index, all other fragments point back to its previous fragment. Thus a lot of memory can be saved compared to using the texture-based method, since the buffer can be defined to be just as large as necessary. This is advantageous especially for scenes with high variation of depth complexity. Figure 3 shows the concept of using linked lists of fragments with an A-Buffer. In general this method is slower than a texture-based method, due to bad cache coherency between the fragments, since each fragment is possibly stored in different memory pages.

To overcome this disadvantage we propose to use *fragment pages*, instead of individual unrelated fragments, in the linked list. A pointer in the list will then point to a page of fragments, instead of a single fragment. This method minimizes the problem of bad cache coherency when using a linked list of fragments. Figure 4 shows how the pages are stored in the linked list.

All described methods require memory to be pre-allocated on the GPU. For the texture-based method this size is determined by a user-configurable parameter adjusting the size of the texture stack, i.e. number of depth layers per fragment. For the link-list methods the amount of memory to pre-allocate is defined globally and not per pixel, the size of this buffer is automatically adjusted during run-time. Upon loading a scene a relatively small buffer is allocated, if this buffer is fully filled up. While rendering we abort the current frame, enlarge the buffer and restart the rendering.

Performance Analysis. Out of the described methods we expect the texture method to be faster than the Linked-List method due to less cache misses. A problem with the texture-based method is that it must predefine a maximum depth complexity that it will be able to handle. The linked lists method can theoretically handle any depth-complexity until the pre-allocated memory is exhausted. The advantage with using this method though is that we will have the option to allocate less memory and still get correct results, compared to a texture-based approach, where the size of the stack must be at least as big as the highest depth complexity. According to Crassin [Cra10], the advantage in memory utilization of the method using linked lists can be huge when objects of high depth-complexity are rendered in parts of the full screen space, while other parts are either

empty or very simple. In typical scenes, exactly this is the case and is reflected by the values measured by [Cra10]. He found that a linked list based implementation consumes, depending on the screen resolution, only $6-10\%$ of the memory a texture-based approach does.

DIRECT VOLUME RENDERING WITH A-BUFFERS. Volume ray-casting is the most common approach for direct volume rendering, and by recognizing the similarity of the compositing part of our A-Buffer rendering and volume ray-casting we realize that, to combine these two methods is similar to the merging steps of merge sort and we only need to modify our composite step. After we have retrieved and sorted the fragments stored in the A-Buffer for current pixel, we query the entry and exit point and perform the ray-casting. For each step in the ray-casting loop we compare the depth of current volume sample with the foremost fragment in the sorted A-Buffer list and pick whichever is nearest. This is done until either both the ray-casting and A-Buffer fragment list is empty or we reach full opacity.

EARLY FRAGMENT DISCARDING. In many scenes, there is a lot of fragments that are stored unnecessarily. When the alpha values of the closest fragments are high enough to completely occlude the fragments behind them, we would optimally want to have a way of knowing beforehand which fragments those would be, since that would allow us to skip the occluded fragments completely in the first pass.

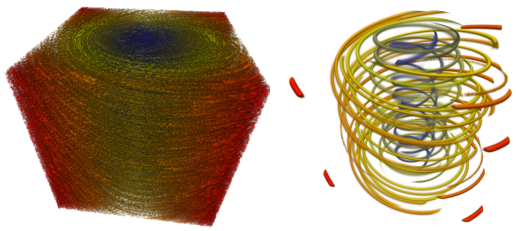
To address this issue, we present a method we call *early fragment discarding*. This technique relies on an additional buffer and a general assumption of frame coherency. The additional buffer is used to store the depth values of previous frame's fragments where full opacity was reached. With this we do a depth-test for each fragment and discard that fragment if its depth is higher than the depth value of the previous frame. This happens during the first rendering pass, while populating the A-Buffer.

By storing fewer fragments we see performance increases for multiple reasons. The first reason is that discarded fragments does not need to be written to the buffer, which is one of the bottle necks of the first pass, second reason is, having fewer fragments, sorting them in the second pass will become faster.

This approach can introduce information loss during interaction. For example, when rotating the camera the depth values close to edges might change rapidly resulting in discarding fragments that could have a strong contribution to the final pixel-color. By only discarding fragments while the user is interacting with the scene (rotation, translation etc.) and perform a full rendering once the user stops interacting, a good compromise between performance and correctness can be achieved. A way to decrease information loss can be achieved by introducing an allowed margin of depth values. Since the fragment depth is stored in the range of 0-1, we can add a percentag margin to the threshold, which will allow us to keep some of the fragments that could potentially be incorrectly discarded on interaction of the scene. A good value for this margin was found at 5%, in which most of the correct fragments were kept, with still significantly improved performance, compared to not using the method at all.

3D LINE INTEGRAL CONVOLUTION. A 3D-LIC method was implemented using Euler integration, as well as fourth-order Runge-Kutta integration with a fixed step size. For every voxel in a vector field volume, a generated noise volume is traversed with a certain amount of steps, in forward- and backward directions. For every step traversed the noise texture is sampled. After sampling all points along one integral line, their mean value is used as the final color of the current voxel.

RANDOM NOISE-VOLUME GENERATION. Unlike 2D LIC, with 3D LICs the output quality is highly dependent on the chosen strategy for random noise generation. In this case, using a 3D volume as the noise input, different types of noise were tested. These were *white noise*, where each voxel get a random value between zero and one, and a sparse noise generation based on the *Halton sequence*. As it can be observed in Figure 5, the sparse noise based on the *Halton sequence* is the most promising one, when comparing the final output images with respect to visual clutter and structure.



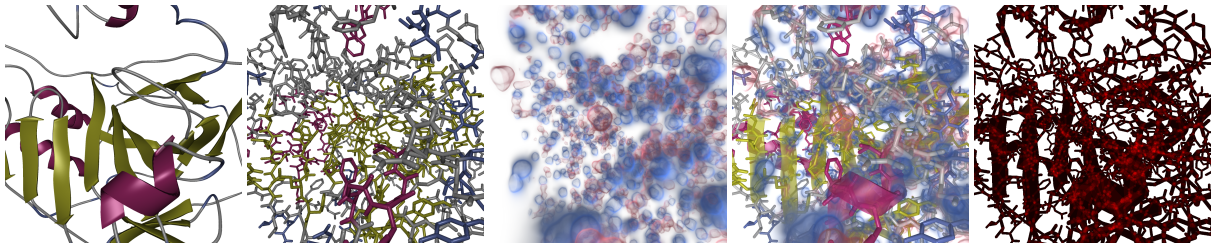
(a) (b)

Figure 5: *Two LIC visualizations of the same vector field, with different noise inputs. White noise (a) give a dense but more cluttered result. Sparse Noise based on the Halton sequence (b) results in clearer, more distinct structures.*

5. Results and Evaluation

The methods described in Section 4 have been implemented in Inviwo, an open-source framework for interactive visualization [SSK*15]. In this section we present the results, from applying the described methods on two datasets from different domains.

The first dataset is a flow dataset of a human heart, acquired using 4D Flow MRI and consists of a time-sequence of volumes. It is given over one cardiac cycle, in addition to the flow field we have a time-sequences of the anatomy as regular MRI and segmented masks for each of the various



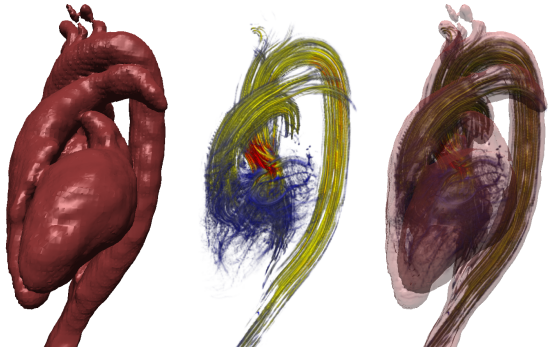
(a) (b) (c) (d) (e)

Figure 6: *Visualization of the protein human carbonic anhydrase II. (a) and (b) show two different mesh representations, in (a) ribbons are used to visualize the backbone or the protein and in (b) a licorice visualization is used to visualize the bounds between atoms. Figure (c) visualizes the electron charge density inside the protein using volume ray-casting. By using our hybrid renderer we can combine these three views into the final image, seen in (d). The depth complexity of the A-Buffer of this frame is visualized in (e).*

chambers, major arteries and veins of the heart [BPE*15]. In figure 7 we can see various representations of this dataset, created with our software. In Figure 7(a) we have extracted an iso-surface from to represent the boundary of the heart, here rendered with full opacity. To visualize the blood flow we have generated a 3D LIC volume as described in Section 4, the result of this is visible in Figure 7(b). When having only the 3D LIC rendering it can sometimes be tricky to depict the anatomical location of the flow in the heart, in such cases rendering the flow together with the surface can be useful. With our hybrid renderer we can easily combine this two representations, in Figure 7(c) we can see how the 3D LIC is rendered using volume ray-casting together with the surface mesh. Before compositing, the surface mesh is rendered into the A-Buffer with a 40% opacity.

The second dataset is the *human carbonic anhydrase II* [PDB] protein. It consists of the protein structures as meshes and volumetric data describing the electron charge density at each location. In Figure 6(b) we see a Licorice representation of the bonds between atoms in the protein, this representation is available from the data as a polygon mesh of cylinders. In Figure 6(a) a second mesh representation is available which represents the backbone of the protein using ribbons. Overlapping with the meshes we have a electron charge density volume describing positive and negative charge, this volume is rendered using regular volume ray-casting in Figure 6(c), red areas represents positive charges and blue areas represent negative charges. Using our hybrid renderer we can combine these three representations. First the Licorice mesh is rendered into the A-buffer at full opacity together with the ribbons at 60% opacity. The sorted a-buffer is then combined with the volume ray-casting resulting in a final rendering as can been seen in 6(d). In Figure 6(e) the depth complexity of the scene is displayed. This is done by coloring the pixel based on the number of fragments for that pixel where black means zero fragments and red means max number of fragments.

In scenes with very large depth-complexity, the A-buffer may sometimes run out of memory. In such situations we discard additional fragments. Since discarding fragments, during the population of the A-Buffer, leads to insufficient information during the final composite step, we show black pixels, instead of incorrect information at that location.



(a) (b) (c)

Figure 7: *Heart dataset: (a) shows the extracted surface as geometry, whereas (b) shows the result of the 3D LIC. In (c) both visualizations are combined. Note that the opacity of the geometry was reduced to reveal the inner LIC visualization.*

PERFORMANCE. The A-Buffer methods presented in this paper are based on performing two rendering passes. The first writes the fragments to a fragment container *(which may be different depending on the method used)*, then it renders them to screen in the second pass. When measuring the performance of the A-Buffer methods, it is therefore relevant to measure the performance of these passes separately.

All performance measurements were done in a setup similar to the heart rendering described above. Instead of using 3D LIC the flow is visualized using path lines rendered

as tubes and the anatomical surface rendered using volume ray-casting. All performance metrics presented in the paper have been measured on a computer using a Nvidia GTX 580 GPU, with 4 GB of graphics memory, and an Intel Xeon W3550 CPU, with four cores and clock speed of 3.07 GHz.

In order to get a good time measure of each pass, timestamps were taken before the first pass, after the first pass, and after the second pass. By calculating the differences between these we could measure the time taken for each pass. When running the tests using different options meant to speed up one of the rendering passes, it was discovered that both time differences changed drastically, in contrast to just one, which would have been expected.

In the second pass, insertion sort is used to sort all the fragments for each screen pixel, which has a time complexity of $O(N^2)$. Since most pixels do not have a depth-complexity of higher than 10 fragments for most scenes an $O(N^2)$ sorting algorithm is sufficient. Not much, if any, performance could be gained by changing the sorting method into an $O(N \log N)$ such as quick or merge sort. Disabling depth-sorting completely improves overall performance by around 20-30%, but will of course give incorrect results.

In Table 1 we present the rendering times for the two A-Buffer methods with two parameters (e.g., max depth and page size). The main difference between the methods is, how the GPU memory is used. The texture-based approach uses layered 2D-textures, during the population of the A-Buffer. In contrast the linked list approach uses pointers in GPU memory, for each fragment location (see Figure 3, 4). Section 4 describes both methods in detail. The image size for these tests was 800x800 pixels. Note that the early-fragment-discarding method was not used here.

The early-fragment-discarding method was tested with scenes of varying depth complexity, object transparency and resolutions. The results are shown in Table 2.

Here it is clear that the method is able to improve performance of the A-Buffer significantly, under the right circumstances. An unexpected result was obtained for low resolutions, where removing the margin yielded worse performance compared to using a 5% margin.

Table 1: *Performance of the two A-Buffer methods. The average depth-complexity was* ~ 12 *fragments for the scene these tests were made on.*

Method	Max Depth	Page Size	**Avg. FPS**
Texture Based	64	-	∼22
Texture Based	128	-	∼18
Linked Lists	64	8	∼18
Linked Lists	128	8	∼16
Linked Lists	128	2	∼15
Linked Lists	128	1	∼14

Table 2: *Performance of the Early Fragment Discarding method using different parameters. All tests were performed on the same dataset. The average depth-complexity of the scenes were* ~ 10 *fragments for the 400x400 px scene and* ~ 7 *for the 800x800 px scene.*

Dimensions	Alpha	EFD	Margin	**Avg. FPS**
400x400	0.2	no	-	∼30
400x400	0.2	yes	5%	∼35
400x400	1.0	yes	0%	∼44
400x400	1.0	yes	5%	∼46
400x400	1.0	yes	10%	∼43
800x800	0.2	no	-	∼24
800x800	0.2	yes	5%	∼32
800x800	1.0	yes	0%	∼41
800x800	1.0	yes	5%	∼40
800x800	1.0	yes	10%	∼39

Using 10% margin in general removed the visual artifacts. In our examples a margin of around 5% was a good compromise resulting in few artifacts and generally high image quality. Removing the margin generally increases performance, however it has a large impact on the image quality, which is considered as too big. Since the performance gain is small that using it margins is justifiable.

6. Conclusion and Future Work

CONCLUSION. With this work we presented a hybrid rendering technique, which is capable of combining different visualization methods into a single output image in an efficient way. To achieve a high-quality and interactive rendering we extended and combined state of the art methods like the A-Buffer and GPU based volume ray-casting. To improve the performance of the rendering we introduced several improvements to one of the underlying building blocks, the A-Buffer. The method has been carefully analyzed with respect to memory consumption and performance. Its efficiency has been demonstrated on two different datasets.

FUTURE WORK. As discussed in Section 5 for future work, we see room for improvement with the sorting algorithm in the second pass of the A-Buffer. Besides that, the optimal resolution of the noise input, in comparison to the output resolution could be studied in more detail, in order to improve performance while still keeping a good visual representation. Our current implementation is restricted to one volume data set. For the future we plan to support for multiple volumes, which should be a straightforward extension.

7. Acknowledgments

This work was supported in part by the Swedish e-Science Research Center (SeRC) and the Excellence Center at Linköping and Lund in Information Technology (ELLIIT). The described concepts have been realized using the Inviwo visualization framework (www.inviwo.org).

References

[BK11] BARTA P., KOVÁCS B.: Order Independent Transparency with Per-Pixel Linked Lists. *The 15th Central European Seminar on Computer Graphics* (2011). 2

[BM08] BAVOIL L., MYERS K.: Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK* (2008), 1–12. 2

[BPE*15] BUSTAMANTE M., PETERSSON S., ERIKSSON J., ALEHAGEN U., DYVERFELDT P., CARLHÄDLL C., EBBERS T.: Atlas-based analysis of 4d flow cmr: Automated vessel segmentation and flow quantification. *Journal of Cardiovascular Magnetic Resonance* (2015). 6

[BPL*07] BAVOIL L., P.CALLAHAN S., LEFOHNM A., AO L. D. COMBA J., SILVA C. T.: Multi-fragment effects on the gpu using the k-buffer. *I3D '07 Proceedings of the 2007 symposium on Interactive 3D graphics and games. Pages 97-104* (2007). 2

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. *Proceedings of the 1994 Symposium on Volume Visualization* (1994), 91–98. 2

[CL93] CABRAL B., LEEDOM L. C.: Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques - SIGGRAPH '93* (1993), ACM, ACM Press, pp. 263–270. 2, 3

[Cra10] CRASSIN C.: Opengl 4.0+ abuffer v2.0: Linked lists of fragment pages, 2010. 4, 5

[EHK*06] ENGEL K., HADWIGER M., KNISS J., REZK-SALAMA C., WEISKOPF D.: *Real-Time Volume Graphics*. A K Peters, Ltd., Wellesley, Massachusetts, 2006. 2, 3

[Eve01] EVERITT C.: Interactive order-independent transparency. *NVIDIA OpenGL Applications Engineering 2*, 6 (2001), 7. 2

[FW08] FALK M., WEISKOPF D.: Output-sensitive 3D line integral convolution. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 820–834. 2, 3

[Int97] INTERRANTE V.: Illustrating surface shape in volume data via principal direction-driven 3d line integral convolution. *SIGGRAPH '97 Proceedings of the 24th annual conference on Computer graphics and interactive techniques. Pages 109-116* (1997). 2

[KM05] KAUFMAN A., MUELLER K.: Overview of volume rendering. *Visualization Handbook d* (2005), 127–174. 2, 3

[KW03] KRUGER J., WESTERMANN R.: Acceleration techniques for GPU-based volume rendering. *Proceedings IEEE Visualization* (2003), 287–292. 2

[LFS*14] LINDHOLM S., FALK M., SUNDÉN E., BOCK A., YNNERMAN A., ROPINSKI T.: Hybrid data visualization based on depth complexity histogram analysis. *Computer Graphics Forum* (2014). 2

[LHD*04] LARAMEE R. S., HAUSER H., DOLEISCH H., VROLIJK B., POST F. H., WEISKOPF D.: The State of the Art in Flow Visualization: Dense and Texture-Based Techniques. *Computer Graphics Forum 23*, 2 (jun 2004), 203–221. 2

[MCTB12] MAULE M., COMBA J. L. D., TORCHELSEN R., BASTOS R.: Memory-Efficient Order-Independent Transparency with Dynamic Fragment Buffer. *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images* (aug 2012), 134–141. 2

[MHDG11] MUIGG P., HADWIGER M., DOLEISCH H., GRÖLLER E.: Interactive volume visualization of general polyhedral grids. *IEEE transactions on visualization and computer graphics 17*, 12 (dec 2011), 2115–24. 2

[PDB] PDB: Structure of native and apo carbonic anhydrase ii and structure of some of its anion-ligand complexes. 6

[RGW*03] ROETTGER S., GUTHE S., WEISKOPF D., ERTL T., STRASSER W.: Smart Hardware-Accelerated Volume Rendering. *Symposium on Visualization (VISSYM '03)* (2003), 231–238. 2

[RSHTE99] REZK-SALAMA C., HASTREITER P., TEITZEL C., ERTL T.: Interactive exploration of volume line integral convolution based on 3D-texture mapping. *Proceedings Visualization '99 (Cat. No.99CB37067) Li*, section 6 (1999), 233–528. 2

[SBF15] SCHOLLMEYER A., BABANIN A., FROEHLICH B.: Order-independent transparency for programmable deferred shading pipelines. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 67–76. 2

[SEA08] SINTORN E., EISEMANN E., ASSARSSON U.: Sample based visibility for soft shadows using alias-free shadow maps. *Computer Graphics Forum 27*, 4 (2008), 1285–1292. 2

[SH95] STALLING D., HEGE H.-C.: Fast and resolution independent line integral convolution. *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95* (1995), 249–256. 2

[SSK*15] SUNDÉN E., STENETEG P., KOTTRAVEL S., JÖNSSON D., ENGLUND R., FALK M., ROPINSKI T.: Inviwo - An Extensible, Multi-Purpose Visualization Framework. Poster at IEEE Vis, 2015. 5

[SSKE05] STEGMAIER S., STRENGERT M., KLEIN T., ERTL T.: A simple and flexible volume rendering framework for graphics-hardware-based raycasting. *Fourth International Workshop on Volume Graphics, 2005.* (2005), 187–241. 2

[SSKLK13] SHREINER D., SELLERS G., KESSENICH J., LICEA-KANE B.: *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. Addison-Wesley, 2013. 3

[vW03] VAN WIJK J.: Image based flow visualization for curved surfaces. *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control* (2003), 123–130. 2

[Zha14] ZHANG N.: Memory-hazard-aware k-buffer algorithm for order-independent transparency rendering. *IEEE transactions on visualization and computer graphics 20*, 2 (feb 2014), 238–48. 2