To:

Editor in Chief, Computer Graphics Forum
Associate Reviewers

Dear Editors, dear Reviewers,

This letter accompanies the major revision of our paper “Hybrid Data Visualization Based on Depth Complexity Histogram Analysis” (fast-track major revision from EG2014 submission). Below, we provide an account for how we addressed the reviewers’ comments to our initial EG submission and also list additional changes (review excerpts are set in italics).

General comments

We would like to start by clarifying the relation between our work and existing A-buffer approaches, in particular the dynamic-fragment buffers (DFB) presented by Maule et al. [MCTB12]. In short, most A-buffer literature, including DFB and per-pixel linked lists (PPLL), focus on algorithms for managing *global* GPU memory, while our algorithms improve performance by optimizing the use of *local* cache memory. Our proposed approach thus complements previous literature as it is explicitly designed to be combined with existing techniques. Instead of ours-vs-theirs comparisons we therefore provide comparisons of existing techniques with and without our components.

New material and major changes:

- More concise introduction and related work sections, revised Section 4 and 5
- Clarification that our proposed algorithms are an extension to existing A-buffer implementations
- The description of the general A-buffer approach was reduced and the figure of the A-buffer pipeline was simplified (Figure 3)
- Added figures to illustrate concepts and differences between prevalent, existing A-buffer implementations (Figure 4) and differences in local memory management between the state-of-the-art and our two proposed approaches (Figure 5)
- Added pseudo-code for both proposed approaches
- Measurements illustrating the view-dependency of depth complexity histograms (Figure 2)
- Additional performance benchmarks for fixed fragment buffer (FFB), DFB, and PPLL in combination with our algorithms (Figure 6)
- Added results for a synthetic data set demonstrating the behavior of our approaches affected by the maximum depth complexity in a worst-case scenario (Section 7.2, Figure 7)
- The number of fiber bundles in the medical scene and the number of streamlines for the flow data were increased to obtain higher depth complexity

Major Revision as detailed by primary reviewer

1. ***clearly show novelty***

   Both the abstract/introduction and results/performance sections were revised to clarify that the proposed algorithms can be combined with any existing A-buffer solution for global memory management to form the final rendering algorithm. The separation between our work and existing literature is outlined more clearly in the text and is supported by new figures illustrating global vs. local memory management (Figures 4 and 5). The presentations of the proposed algorithms were rewritten with additional pseudo-code and figures to increase clarity.

2. ***validation of techniques to show effectiveness***

   In addition to existing benchmarks with PPLL, both ppAO and ppDP were tested in combination with two additional A-buffer solutions for global memory management (FFB and DFB). The results were obtained for two different GPU architectures (NVIDIA Fermi and Kepler) and are discussed in Section 7.2 and reported in Figure 6. Additional tests were also performed to investigate the view-dependency of the depth complexity and the associated frame rate (Figure 2). A synthetic worst-case scenario is introduced to investigate algorithm behavior in less-than-ideal conditions (Figure 7).

3. ***improved comparison + demonstration of improvements (dyn. fragment buffer)***

   The performance benchmarks now include FFB, DFB, and PPLL. We compare the baseline performance (only global management) with combinations of the baseline and our algorithms (base+ppAO, base+ppDP, base+ppAO+ppDP). The overall performance gain (averaged over all scenes) is about 3-fold for ppAO and up to 8-fold for ppDP (cf. Figure 6). The effectiveness is also demonstrated for the rotating protein scene, where the scene composition is changed to contain less geometry or less volumetric parts. When applied, our approaches dramatically reduce rendering times (Figures 2 and 7).

4. ***presentation***

   All sections of the paper have been revised to improve the presentation quality of the paper. Special attention was given Sections 4 and 5 to improve the presentation of our algorithms and their relation to existing work.

   In particular, the abstract and Section 1 were revised to be more to the point. Section 3 was extended with additional figures detailing the view-dependency of depth complexity histograms. Section 4 was completely rewritten to describe the three prevalent A-buffer implementations (FFB, DFB, and PPL) and their handling of global memory. A figure has been added to illustrate the general differences in global memory management between the three implementations (Figure 4). In Section 5, the overall structure was changed and pseudo-code and figures were inserted for better illustration of our algorithms. Figure 5 depicts the two proposed algorithms for managing local memory compared to the state-of-the-art implementation while resolving the contents of the A-buffer. Section 6 was extended to include more detailed information on the rendering details for increased reproducibility and completeness.

Reviewer 1

The main weaknesses: the evaluation (in particular, a comparison to [MCTB12] needs to be included) and description that need to be improved.

As mentioned above in the general comments, the DFB approach by Maule et al. is now included and our performance comparisons detail how our algorithms increases the performance of the DFB. The general presentation of the paper was also adapted to emphasize that our algorithms are designed to complement, not replace, existing solutions.

The paper does not do a good job in explaining the important details of the novel parts, i.e., Sections 5.2 and 5.3, which take only one page. In fact, the rather dubious description significantly lowers the reproducibility of the paper. Instead, lots of space is wasted in Section 4, which basically explains an implementation of the A-buffer algorithm on the GPU.

The general description of the A-buffer algorithm was compressed and now focuses on the differences in handling global memory (Figure 4). Section 5 was extended and now contains both a graphical description (Figure 5) and the algorithms depicted in pseudo-code. (see above for a more detailed description)

Table 1: The numbers/symbols […] of the columns are not described adequately and are misleading.

Performance results are now reported in a graph. The new figure/table combination uses much fewer short notation symbols for increased readability. Performance numbers were also moved to a separate figure for increased clarity.

I could not reproduce the steps in Section 5.3. to sort the fragment list so I think it would be quite difficult to implement the approach from the provided information in the paper.

Furthermore, the description of the overflow optimization should be improved for full reproducibility.

We added a more detailed description and pseudo-code to Section 5.3 (see above).

[…] In this context, there should be a discussion of efficient GPU sorting algorithms.

The sorting of the local buffer contents relies on a simple insertion sort, utilizing only a single thread per fragment list. We provide no general discussion or references on divide-and-conquer techniques, or general GPU sorting that distributes the computations onto multiple threads as we think this would be out of scope for this paper.

Reviewer 2

The datasets tested with the presented algorithm are small compared to other previous studies (almost all less than 256^3). I strongly recommend applying the technique to larger data to demonstrate the performance benefits.

The scenes selected as showcases in our paper are visualized in a way as they would be presented in their respective fields. Due to their very nature, i.e., their high depth complexity caused by semitransparent geometrical objects like streamlines and isosurfaces, it is very challenging to render them at interactive frame rates while obtaining correct blending results.

Furthermore, the scene composition often has more impact on performance than, for example, the raw resolution of the volumetric data. The resolution of the volume data does not affect the per-pixel depth complexity since the number of entry and exit points remains the same regardless of the resolution. However, it would affect the overall rendering time because of the necessarily higher sampling rate.

Add references to earlier volume and a-buffer work (e.g., Ebert and Parent SIGGRAPH 90), other transparency volume/geometry combinations (Kniss, Hansen et. al), and alternative approaches.

The related work section has been extended with references to earlier variations of A-buffer and depth peeling implementations. We also included a reference to the book "Real-Time Volume Graphics" as an introduction to both transparency rendering and inclusion of geometry in direct volume rendering.

Minor modifications

All minor modifications suggested by Reviewer 2 were incorporated and the paper was revised accordingly.

Reviewer 3

A cuda implementation of the a-buffer seems to be the base for this work, but I don't think it is fair to refer just to the code and not to discuss the details of the CUDA implementation. It would be nice if the authors could be more precise on how data is handled, or deliver better, more explicit figures.

Our work is implemented entirely in OpenGL using GLSL shaders, allowing the graphics driver to optimize scheduling and memory assignments. This includes our two proposed algorithms (ppAO, ppDP) as well as the implementations of FFB, PPLL, and most parts of the DFB. In the DFB algorithm, CUDA is used only for one particular sorting step as explained by Maule et al. [MCTB12]. We added these details to Section 6.2.

I am not convinced however that handling and combining 128 depth layers is the right thing to do in visualization.

[…] but I am not convinced that having so many samples combined in one pixel is useful.

We agree that discussions on transparency or no transparency are interesting and that further research on the topic is necessary. Such discussions would, however, make sense mostly on a per case basis – most likely supported by user studies – which we think would be out of scope for this paper. Furthermore, a fixed maximum number of layers results in a trade-off between performance and 'approximate rendering results'. Given the potential for our algorithms to be used in a wide range of domains we prefer not to suggest a 'universally acceptable maximum depth'.

In addition, our algorithms greatly reduce the impact of this trade-off for real-world examples since a few high complexity pixels are less likely to become a bottleneck. Secondly, our algorithms improve the performance also at lower complexities and are thus not only aimed at very high complexities. Finally, the number of layers that are actually blended can be limited by, for example, early-ray-termination as discussed in the manuscript.

[…] a few important components of the implementation not discussed, and quantitative results to lack explanations and analysis.

Additional discussion is now provided in Section 6 as well as the inclusion of both a view-dependent benchmark as well as a synthetic worst-case scenario, both intended to highlight how, and in what circumstances, our algorithms are beneficial. (see Summary 4 for details)

I am not sure how the fragments produced from mesh, stream-line and volume data are blended to obtain the final colour.

Details on how to combine volume rendering with mesh data was added to Section 6.

512x512 is a little bit small, but I guess beyond this, real-time is not possible anymore.

The memory requirements of the A-buffer implementations basically limit the screen resolution. By using GPUs with more VRAM we were able to increase the viewport size. We use a rendering resolution of 1024x768 for the performance benchmark on both GeForce GTX580 (1.5GiB VRAM) and the Titan (6GiB VRAM). The results reported for the view-dependency measurements were taken for a 1024x1024 viewport on the Titan graphics card.

Minor modifications

All figures mentioned by Reviewer 3 have been replaced, including the A-buffer workflow.

Reviewer 4

The paper is well-written but the description is somewhat difficult to follow due to the many abbreviations.

The number of abbreviations has been drastically reduced. There are now five prominent abbreviations, FFB + DFB + PPLL for the prevalent A-buffer implementations and ppAO + ppDP for our proposed methods.

The description of ppDP in Section 5.3 would also benefit from pseudocode to improve the clarity. Furthermore, illustrations would have been helpful.

The presentation of ppDP and ppAO was improved and pseudo-code of the algorithms was added. We also added two figures illustrating the basic A-buffer approaches and our proposed ppDP and ppAO approaches (see general comments).

A valuable addition would be an "auto-tuning" approach to automatically select good values using a set of benchmarks.

The free parameter would be the size of the local array. In terms of performance, the bounds are clear for both ppAO and ppDP. Other configurations, or a combination between the two techniques, results in a quality-performance trade-off that would be tricky to handle by auto-tuning. Still, the concept is very interesting and is now mentioned in the manuscript.

Reviewer 5

The paper shows performance results for 4 different datasets, but did not provide any comparison against competing strategies, only a basic A-buffer reference implementation.

Please see summary 3 for a general discussion.

The fact that it relies on a single frame reduces its usage to this frame, since other views might generate different DCH with other behaviors. Although the idea is new, I do not think it was fully validate and explored.

There is a lack of analysis of the impact of the DCH in different viewpoints (how they vary) and how the algorithm can cope with these changes.

Please see summary 4.

how does the number of fragments define a color [when rendering a complexity image]? What is the color range? [0,255]?

For computing a depth complexity histogram, we use an integer framebuffer to render the complexity “image” as mentioned in the paper. This image is then analyzed to compute the depth histogram. For illustration purposes, we normalize the image according to the histogram range.

What happens when you zoom in the scene such that all pixels present high depth complexity?

This behavior was evaluated with two newly introduced benchmarks as described in summary 2.

>> Split the sequential processing of an array of size N into two parallel sequences of size N/2 suggests that the performance gain comes from the high GPU parallelism, not from the driver.

We don’t apply a divide and conquer principle here. The process of an array of size N is split into two sequential (not parallel) sequences of size N/2. The two sequences will be performed by the same thread one after the other. The appropriate sections in the paper have been revised to clarify this.

Minor modifications (additional comments and spelling)

All minor modifications suggested by Reviewer 5 were incorporated and the paper was revised accordingly.

We are grateful for the comprehensive reviews and we hope that we were able to address all concerns adequately.

Yours sincerely,

The Authors