Eurographics Workshop on Visual Computing for Biology and Medicine (2014)
I. Viola, K. Bühler, and T. Ropinski (Editors)

Visibility-Driven Processing of Streaming Volume Data

Veronika Solteszova[1,2] Åsmund Birkeland[2] Ivan Viola[2,3] Stefan Bruckner[2]

[1]Christian Michelsen Research, Bergen, Norway
[2]University of Bergen, Norway [3]Vienna University of Technology, Austria

Figure 1: *Dataset Anna, 3D ultrasound of a fetus: visualization of the raw data (left), with all voxels filtered (middle) and with only voxels contributing to the visualization filtered (right). Even though the middle and the right images are identical, the filtering operation took 0.175s for the middle image and 0.098s for the right image using a Geforce 580 GTX GPU.*

Abstract

In real-time volume data acquisition, such as 4D ultrasound, the raw data is challenging to visualize directly without additional processing. Noise removal and feature detection are common operations, but many methods are too costly to compute over the whole volume when dealing with live streamed data. In this paper, we propose a visibility-driven processing scheme for handling costly on-the-fly processing of volumetric data in real-time. In contrast to the traditional visualization pipeline, our scheme utilizes a fast computation of the potentially visible subset of voxels which significantly reduces the amount of data required to process. As filtering operations modify the data values which may affect their visibility, our method for visibility-mask generation ensures that the set of elements deemed visible does not change after processing. Our approach also exploits the visibility information for the storage of intermediate values when multiple operations are performed in sequence, and can therefore significantly reduce the memory overhead of longer filter pipelines. We provide a thorough technical evaluation of the approach and demonstrate it on several typical scenarios where on-the-fly processing is required.

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

1. Introduction

Over the past years, ultrasound imaging has experienced dramatic improvements in quality, resolution, availability, and range of indications. Ultrasonography is now a standard tool in obstetrics, cardiology, gastroenterology, and many other medical fields and the range of applications is constantly growing. The technology has progressed rapidly from initial 1D signals over standard 2D sonography to 3D volumetric ultrasound. In echocardiography, for example, ultrasound is used to diagnose heart contraction efficiency, functional-

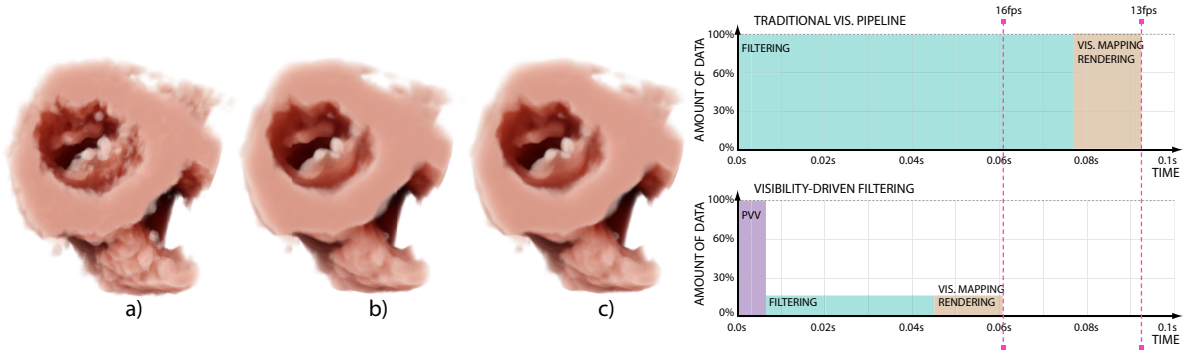

Figure 2: *Cardiac 3D ultrasound a) raw data visualization, b) filtering of all voxels, c) filtering of potential visible voxels. The graph compares the performance of the pipeline used for visualizations b) and c) on a Geforce 580 GTX GPU. Data filtering for visualization b) takes 0.076s seconds. Our approach is based on the efficient computation of the potentially visible set of voxels (PVV) so that the expensive filtering operation is executed only on the PVV. For a PVV containing 17% voxels of the entire dataset and a selected filtering method (lowest-variance filtering [vSHW*12]), our total computation time sinks to 0.045s.*

ity of the valve, and in intra-operative scenarios it is even used as a navigational aid for minimally-invasive valve replacement. Recently, the rapid increases in processing power coupled with advances in digital beamforming have enabled the real-time capture of high-resolution 3D volumes hence making ultrasound a true 4D imaging modality which paves the way for a number of new important clinical applications. 4D ultrasound offers many potential benefits in the prenatal diagnosis of neurological problems by enabling the assessment of grimacing, breathing movements, swallowing, mouthing, isolated eye-blinking, and revealing the direction of the limbs [LV11]. Similarly, 4D ultrasound provides significant advantages in assessing heart defects [Ion10]. Recent developments are even progressing towards portable 4D scanners which open up new possibilities in point-of-care medicine, e.g., for emergency medicine in crisis zones.

However, despite its advantages, the visualization of ultrasound data is plagued by several challenges. Compared to other common tomographic imaging modalities such as CT or MRI, ultrasound has a much poorer signal-to-noise ratio and suffers from various acoustic artifacts such as attenuation, focusing and interference [NPH*00]. In volume visualization these are particularly problematic as artifacts can obscure relevant structures and hence affect the diagnostic value of the resulting image. Much research has gone into the development of filtering strategies for the removal of speckle noise and other artifacts, but effective methods typically have a considerable computational cost [vSHW*12]. For static 3D data or prerecorded 4D sequences, filtering acts as a preprocessing step and is therefore not performance critical. A complex filtering operation, even if it takes seconds to process the whole volume, usually has only negligible impact since it only has to be performed once (typically during loading of the data). However, when data is streamed at many volumes per second during a live examination, such a delay becomes unacceptable. Thus, the effective visualization of live 4D ultrasound data in clinical scenarios necessitates solutions to this problem.

To address this challenge, we propose a novel approach to visibility-driven processing of streaming ultrasound data. Key to reducing the computational load of expensive filtering operations is the fact that we do not need to apply the filtering operation to the whole volume, but only to those regions that will be contributing to the visualization. Typically, only a small fraction of all voxels will be visible since they are either classified as transparent or occluded by other structures. This observation is certainly not new and in fact visibility culling and similar techniques are a mainstay of many visualization and graphics algorithms. However, in contrast to other approaches which exploit visibility information for improving processing performance, we need to account for the fact that filtering operations, since they modify the underlying data values, can themselves influence the visibility. Our work presents a solution to this problem by quickly identifying voxels which are potentially visible after a subsequent filtering operation has been applied. The proposed scheme guarantees that the resulting rendered image is equivalent to applying the filtering operation to the whole volume. The main contributions of this paper can be summarized as follows:

- We introduce a new approach for visibility-driven filtering of streaming volume data which avoids processing of invisible (both due to transparency and due to occlusion) regions of the volume but still guarantees the same visualization result as processing the whole volume. This is demonstrated in Figure 1.
- Based on this concept, we also present a novel architecture which exploits visibility information to reduce

intermediate storage requirements for complex filtering pipelines with multiple stages.
- We present a thorough analysis of the performance of our system and demonstrate that the proposed approach enables high-quality visualization of streaming 4D ultrasound data on current graphics hardware.

2. Related work

The topic of *visibility* has been extensively studied in computer graphics and amounts to the classification of all objects in a scene as either being at least partially visible or totally invisible. The task of rapidly identifying entirely invisible objects is of importance in any rendering system, but it is especially crucial when interactive frame update rates are desired. Ideally, it would be desirable to exactly identify those objects that do not contribute to the generated image in any way at zero cost and to draw only the visible parts of those objects. The surveys by Bittner and Wonka [BW03] and Cohen-Or et al. [COCSD03] provide a good introduction into basic techniques. Approaches include image-space methods such as hierarchical z-buffering and hierarchical occlusion maps, spatial subdivision (e.g., octrees, BSP trees, K-D trees), portals, and potentially visible sets.

Deferred shading is a further related concept which aims to move expensive operations (such as the evaluation of complex shaders) to later processing stages where they are only applied to those image regions that actually need to be processed. First introduced by Deering et al. [DWS*88], a common realization uses two rendering passes where the first pass simply stores the image-space depth and normals of visible objects. The second pass then uses these values compute the final image. The deferred shading pipeline allows for fast computation of differential surface properties, e.g., curvature magnitudes and directions. Hadwiger et al. [HHSG05] use deferred shading on isosurfaces to realize advanced shading effects such as ridge and valley lines [HKG00, KWTM03] and curvature-based flow advection.

In the context of volume visualization, many techniques for exploiting visibility information to accelerate the rendering process have been proposed. As occluded parts of the volume can be easily skipped in ray casting (early-ray termination), most methods have focused on avoiding processing of empty space. Early work by Levoy [Lev90] proposed a hierarchical subdivison of the volume to skip empty regions. A very common approach, originally introduced by Lacroute et al. [LL94] for the shear-warp factorization volume rendering algorithm, is the use of a min-max octree in combination with a summed area table of the transfer function to quickly identify empty parts of the volume. Boada et al. [BNS01] proposed a management policy which uses a hierarchical approach in homogeneous regions and regions of low interest instead of a one-texel-per-voxel logic on the entire dataset. Kraus and Ertl [KE02] introduced several variants of adaptive texture maps and applied them in volume rendering. Mora et al. [MJC02] proposed the use of hierarchical occlusion maps for hidden volume removal in an object-order volume rendering algorithm. The elimination of occluded but non-transparent parts of the volume is particularly important in the visualization of large data where costly disk-to-main-memory or main-memory-to-GPU-memory transfers need to be avoided. Most approaches employ a form of bricking where the volume is subdivided into smaller blocks [BHMF08, LLY06]. Gobetti and Marton [GM05] introduced a voxel-based multi-resolution framework for interactive rendering of large and complex models. They coupled visibility culling, out-of-core construction, and view-dependent rendering to achieve interactive frame rates. Kniss et al. [KLF05] use the term deferred filtering to refer to a two-pass volume rendering approach for compressed formats. In the first pass, a local subset of the data is reconstructed and the second pass can then exploit the GPUs native interpolation capabilities. Crassin et al. [CNLE09] suggested the use of information extracted from the previous frame to guide data streaming from slower memory units. For the visualization of petascale volume data, Hadwiger et al. [HBJP12] presented a visibility-driven rendering method which employs a virtual memory architecture. They use visibility information to only fetch and reconstruct needed bricks from a multi-resolution hierarchy. Fogal et al. [FSK13] presented a detailed analysis of ray-guided approaches for large volume data visualization including an evaluation of the effects of common tunable parameters such as brick size. A common limitation of all these approaches, with respect to our application, is that the volume is considered static, i.e., its values do not change from one frame to the next. Jeong et al. [JBH*09], for instance, perform on-demand filtering of electron microscopy data only for visible blocks. Pre-computed histograms are efficiently updated during the filtering pass only for affected blocks. However, this approach is not feasible for streaming volume data, where the entire volume is replaced continuously. In this paper, we present a solution that specifically addresses this scenario. Another very important difference to our approach is that they do not consider the situation when the invisible blocks become visible due to filtering.

Only few works have addressed the visualization of 4D streaming ultrasound data. The poster by Bruder et al. [BJE*11] shows how the Voreen volume rendering engine can be used to visualize real-time 4D ultrasound data and Elnokrashy et al. [EEH*09] present the basic pipeline setup for GPU-based ultrasound rendering, but neither work discusses the important issue of filtering. Most related to our approach is later work by Elnokrashy et al. [EEH*09] where they restrict themselves to single isosurfaces and apply a smoothing filter to the z-buffer. In contrast, our method enables true filtering of the scalar field in an integrated volume rendering pipeline for live streamed 4D ultrasound.

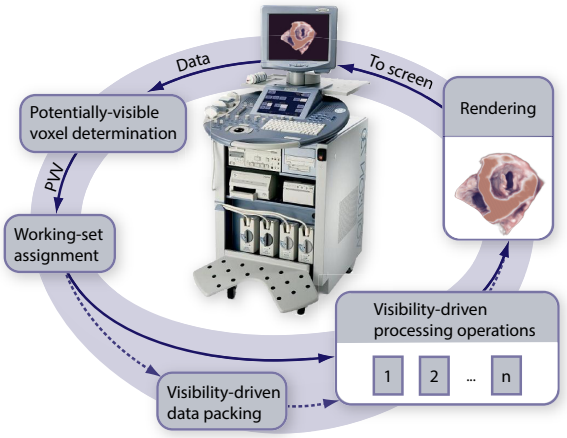

Figure 3: *The pipeline for visibility-driven data processing: 4D data is streamed directly from the ultrasound scanner. In the first stage the visible set of voxels is calculated based on the opacity transfer function. The neighborhood information is then evaluated and the set of potentially visible voxels is passed to the next stage. If the data does not fit into the GPU memory, we can optimize memory consumption by performing a visibility-driven data packing before processing the data. Finally, the processed data is rendered.*

3. Visibility-Driven Processing Pipeline

In the standard visualization pipeline, data enhancement (e.g., filtering) directly follows the acquisition stage. The visual mapping and rendering steps then operate on the enhanced data as illustrated in Figure 2. The problem arises when in-situ visualization of streamed data is required. In this case, for every rendered frame the data needs to undergo an expensive processing operation which may be significantly more costly than the mapping/rendering stages themselves. Examples of such costly common data enhancement algorithms are iterative anisotropic diffusion filtering [PM87], iterative bilateral filtering [TM99], lowest-variance filtering [vSHW*12] and vessel-enhancing filtering [FNVV98]. To solve this problem, we observe that in most volume visualization applications, only a fraction of the data is displayed at a time. Transfer functions limit the range of displayed data values and large portions of the volume may be occluded by other structures. Additionally, particularly for ultrasound data, clipping planes or more advanced clipping geometries are commonly used to remove unwanted parts of the data [BBBV12]. Furthermore, when zooming-in to investigate small details, large parts of the volume may simply lie outside of the viewing frustum. For such scenarios, it is worthwhile to execute the data enhancement only on a subset of the data that is potentially visible, i.e., potentially-visible voxels (PVV).

In this section, we describe an approach for visualizing streaming volume data which exploits visibility information to leverage computational resources only to contributing parts of the data to enable high-performance on-the-fly data enhancement operations. An overview of our approach is depicted in Figure 3. Our pipeline receives one volume of the ultrasound stream at a time. This volume can then undergo multiple sequential processing operations and the result of these operations is then displayed by a volume rendering algorithm. Our strategy to enable on-the-fly processing of live streamed volume data is that processing operations only have to be applied to those regions which affect the final displayed image. This means that completely transparent voxels do not have to be processed, but it also means that occluded regions (regions where viewing rays have already accumulated full opacity before reaching them) can be skipped.

To make this possible, we need to make certain assumptions about the nature of the permissible processing operations and how they affect the visibility in the volume. Our input volume is a scalar-valued volumetric function $f : \mathbb{R}^3 \rightarrow \mathbb{R}$. In general, a processing operation $g(p)$ replaces the original value at a voxel position p with a new value. For any such operation, we require that the filtered function value $g(p)$

1. is only dependent on values of f in a finite neighborhood Ω_p around a position p, i.e, that it has compact support, and
2. that $g(p) \in [\inf \Omega_p, \sup \Omega_p]$, i.e., the new value falls within the minimum and maximum values in that neighborhood.

Both requirements are trivially true for any convolution with normalized weights (mean, Gaussian, etc.), but also hold for a wide range of other smoothing or denoising operations including nonlinear filters such as the median or bilateral filters. An edge-detector, containing negative values in the operator mask, naturally does not conform with this requirement. Without the above requirements a processing operation could, in principle, be a random number generator and there would be no way to predict the resulting visibility. In order to exactly determine the final visibility at every position, we would have to perform the full processing and volume rendering steps. Given these two assumptions, however, we can now develop methods to conservatively predict the visibility of a voxel position p, *prior* to the application of g. We introduce a visibility prediction function $v(p) \in \{0,1\}$ which, for every position p, is 0 when the application of the processing operation at that position can be safely (i.e., without changing the final image) skipped and has a value of 1 otherwise. Ideally, this prediction function is much cheaper to compute than g itself and we can therefore not only amortize its costs but also reduce the overall processing time by only applying g to those positions that are potentially visible (i.e, where $v(p) = 1$). We aim at finding the smallest possible potentially visible set of voxels, but at the same time we need to ensure that the additional processing does not outweigh our gains. Hence, there is a trade-off between the computational cost of evaluating v and the number of voxels that are incorrectly classified as visible. Our approach for computing

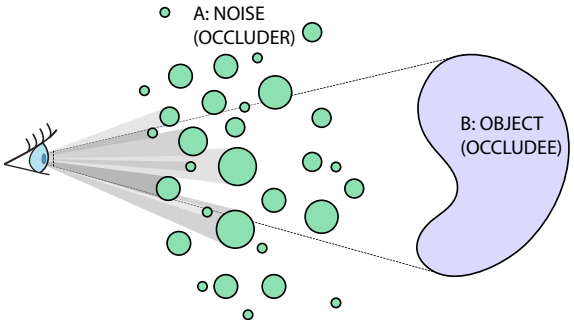

Figure 4: *Points in subset A (noise) are tagged as visible in the original dataset. They are surrounded by invisible voxels. A occludes subset B and according to the straight-forward solution, B would be tagged as invisible and will not be filtered. Data filtering will remove noise, then the subset A will become invisible. Later rendering will reveal B that was not filtered.*

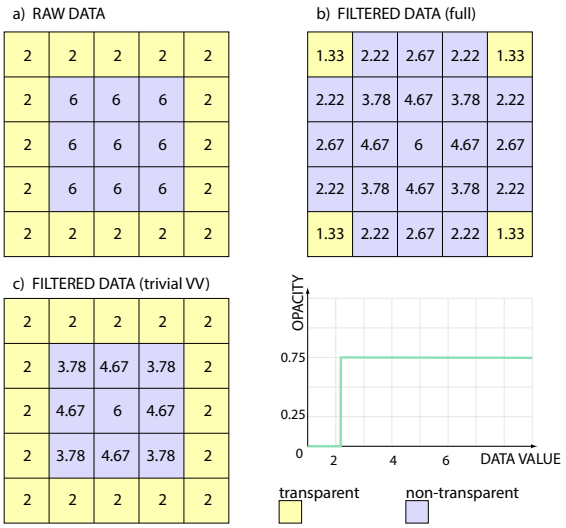

Figure 5: *a) is a 2D projection of a noisy object. We use a simple averaging kernel 3×3. b) Full filtering that includes also transparent regions (background) and we consider it the ground truth. c) The result if we filter on non-transparent regions.*

v is comprised of two basic steps described in Section 3.1: First, we obtain $\inf \Omega_p$ and $\sup \Omega_p$ for the neighborhood of the processing operation. We then perform a visibility pass which generates a binary volume that stores the values of v.

Based on the set of potentially visible voxels, we then generate a working set of the volume data which needs to be processed (see Section 3.2). Our pipeline uses a block-based volume layout which enables optimized memory access and volume traversal and also reduces the intermediate storage requirements for multiple processing steps. Only blocks which contain at least one non-transparent voxel need to be accessed during the processing operation. Similar to Hadwiger et al. [HBJP12] we use a virtual block table which is generated on-the-fly. After the volume has been processed, it is rendered using a standard GPU-based volume raycaster as briefly described in Section 3.4.

3.1. Potentially-Visible Voxel Determination

Our goal is to obtain a view-dependent set of potentially visible voxels (PVV) after application of the filtering operation which takes into account both transparency and occlusion. As our approach is designed to handle live streaming volume data, this set needs to be recomputed for every volume update. We need to consider two cases:

1. A region in the volume considered to be an occluder before the filtering operation has been applied, may get new values which are be mapped to opacity $\alpha = 0$ during the visualization stage. This means that it no longer acts as an occluder. This case is illustrated in Figure 4.
2. The filtering operation can also change data values in such a manner that previously invisible regions will be non-transparent after its application. An example of such a scenario is shown in Figure 5.

To take into account such potential visibility changes without actually executing the filtering operation, we make use of the two assumptions stated in the previous section, i.e., the filter has compact support and the result of the operation lies within the minimum and maximum of its neighborhood. To quickly determine the relevant opacity contributions for a particular neighborhood, we generate the following two lookup tables:

$$\alpha_{\min}(i,j) = \min_{u \in [i,j]} tf_{\alpha}(u) \quad (1)$$

$$\alpha_{\max}(i,j) = \max_{u \in [i,j]} tf_{\alpha}(u) \quad (2)$$

where α_{min} and α_{max} are the minimum and maxmimum, respectively, opacity values in the transfer function tf_{α} for all values in the interval $[i,j]$. Both $\alpha_{\min}$ and $\alpha_{\max}$ can be computed simultaneously and stored in a single 2D texture as two channels. Computation of these tables is straightforward and only consumes a negligible amount of time when the transfer function is modified. This is conceptually similar to pre-integrated volume rendering which employs a lookup table for accumulated colors and opacities along a viewing ray [EKE01]. An example of an opacity transfer function and its corresponding $\alpha_{\min}$ and $\alpha_{\max}$ are given in Figure 6.

Our approach for potentially visible voxel determination then proceeds as follows:

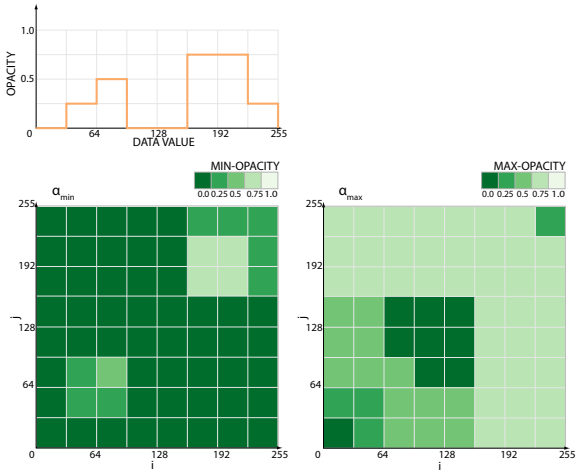

Figure 6: *An opacity transfer function and its corresponding* $\alpha_{\min}$ *and* $\alpha_{\max}$ *tables.*

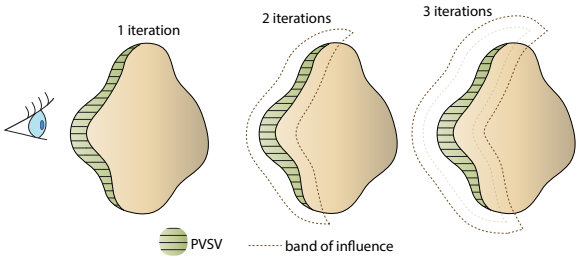

Figure 7: *If data enhancement consists of one operation only, expansion of the PVV is not necessary. If it runs in two iterations, values within the band of interest (BOI) must have been processed in the* 1^{st} *iteration since the* 2^{nd} *iteration relies on its results. If the enhancement runs in three iterations, the radius of the BOI is larger accordingly.*

Minimum/maximum computation. Unlike other approaches, which do not consider dynamically changing data, we can not rely on any preprocessing. Hence, in order to obtain information about the voxel neighborhood, it needs to be recomputed for every new volume. We compute, for each position p in the volume, the minimum $\inf\Omega_p$ and maximum value $\sup\Omega_p$ with a given neighborhood Ω_p determined by the support of the processing operation (i.e., the size of the kernel for convolution operations). We use an OpenCL kernel in a multi-pass approach which exploits the fact that the **min** and **max** filters are separable. While this is not a cheap operation, it is still significantly less costly than the types of filtering operations we want to support.

Visibility evaluation. The set of potentially visible voxels is then computed in a front-to-back visibility pass which outputs a binary volume. As one value needs to be generated for each voxel, we use axis-aligned traversal implemented as an OpenCL kernel. Similar to 2D texture-mapped volume rendering, we choose the slice axis as the major axis of the volume most parallel to the current viewing direction. The $\alpha_{\max}$ table is used to determine the visibility of a voxel, while $\alpha_{\min}$ is used to evaluate its contribution to the accumulated opacity in the following manner:

$$A_{p_i} = A_{p_{i-1}} + (1 - A_{p_{i-1}})\alpha_{\min}(\inf\Omega_{p_i}, \sup\Omega_{p_i}) \quad (3)$$

where A_{p_i} is the accumulated opacity at the i-th sample position p_i along a viewing ray, and $\inf\Omega_{p_i}$, $\sup\Omega_{p_i}$ are the minimum and maximum values, respectively, in the filter neighborhood. Accumulating with $\alpha_{\min}$ ensures that no viewing ray will be terminated too early. The final visibility prediction function, which is the characteristic function of the PVV set, is then defined as:

$$v(p_i) = \begin{cases} \alpha_{\max}(\inf\Omega_{p_i}, \sup\Omega_{p_i}) > 0 & \text{if } A_{p_{i-1}} < 1 \\ 0 \end{cases} \quad (4)$$

While the PVV gives us information about which voxel positions are potentially visible in the final image after the processing operation has been applied, we also need to make sure that the results of the processing operation are correct. Many data enhancement techniques require information about the neighborhood. Moreover, many approaches are carried-out in an iterative fashion, e.g., when filters are separable. If we rely only on the PVV that considers reduced occlusion and increased visibility discussed so far, iterations will obtain incorrect information from the neighborhood of the PVV in later passes. This problem is sketched in Figure 7.

Our solution to this problem is to dilate the visibility mask to obtain a working set of voxels (WSV). Then we define a band of influence (BOI) as WSV\PVV and BOI $\cap$ PVV= $\emptyset$. For example, similarly to the min-max computation, if a filtering operation has m iterations and in each iteration, the operation requires a neighborhood of radius n, then the visibility mask must be dilated by a radius $m \cdot n$. Using WSV for filtering calculations will ensure that the values of all voxels $\in$ PVV will be correct after the final iteration of the filter. In Figure 8, we show an example of a WSV for a kernel size of 9.

3.2. Working-Set Assignment

Many filters require additional buffers to store information. Lowest-variance filtering, for example, requires the direction along the lowest variance to be stored as a 3D vector

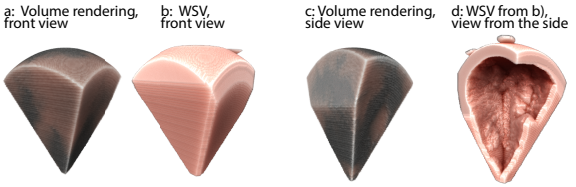

Figure 8: *Example of a working set of voxels (WSV): a) A rendering of a 3D ultrasound frustum. b) The WSV seen from the front, same as a). This WSV contains 22% of the voxels. In the bottom row, the view is rotated. c) The 3D frustum rendered from this viewing direction. d) The WSV is computed for the same viewing direction as in b), but it is rendered from the same viewing direction as in c) to show the empty space.*

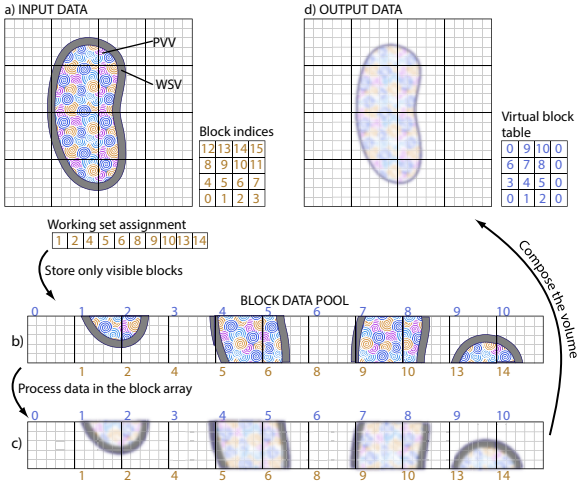

Figure 9: *Concept of data packing on a 2D example. A set of visible voxels determines which blocks from the input data (a) will be active and copied to the block data pool (b). Only blocks in the pool will be processed. After the processing operation is finished, the blocks in the data pool are assembled back to the output volume.*

field [vSHW*12]. Other operations such as anisotropic diffusion require an additional swap-buffer. With larger volumes, it can be challenging to keep such buffers in GPU memory simultaneously. As we are only filtering the visible data, we are only required to keep a fraction of the data on the GPU for processing. We can reduce memory usage by only storing buffers for the actually visible parts of the volume. For instance, lowest-variance filtering needs to store an additional 3D vector with float components for each voxel it processes. For the entire volume, this extends the memory requirements by a factor of 12 if the original volume is an 8-bit byte array and a 32-bit floating point representation is used for the vectors.

In our approach, we subdivide the volume into fixed-size blocks. We then need to only store and process those blocks that contain voxels which are part of the WSV. We can then process the array one block at a time, rather than the entire volume. The packing mechanism is illustrated in Figure 9. From the WSV we create the following data structures which are used during the execution of the filtering operations:

Block data pool. The block pool is a buffer which stores the data values for all blocks in the working set. Additionally, for reasons explained below, we store one *empty block* (filled with zero values) at index zero. The block size is set at compile-time and for all experiments in the paper we used a block size of $B_{x,y,z} = 32$.

Virtual block table. The virtual block table stores the locations of virtual blocks indices in the block data pool. For an input volume of size $V_x \cdot V_y \cdot V_z$, it has a size of $T_x \cdot T_y \cdot T_z$ with $T_{x,y,z} = V_{x,y,z}/B_{x,y,z} + 2$. The inflated parts of the virtual block table are initialized with zero, i.e., they point to the empty block. In this way, the subsequent processing passes can safely access locations beyond the volume boundaries while avoiding costly out-of-bounds checks. Note that this inflation is different from the common duplication of voxel values *within* each brick, which does not occur in our approach. For blocks which are not in the current working set, the virtual block table also contains the value zero.

Working set array. This array contains only the indices of blocks in the current working set. Hence, the working set array is a linearized version of the virtual block table where zero entries have been removed. It is used for volume traversal during the execution of the processing operation.

3.3. Visibility-Driven Filtering

After generating our working set data structures, we can now apply the actual filtering operation. Each processing pass takes as input the block data pool and the virtual block table, and writes its result into a new buffer which duplicates the structure of the working block pool. By traversing the working set array, each block can be processed independently. The virtual block table is used to resolve the locations of neighborhood voxels which lie outside the current block. For a voxel position x, y, z, we first compute its index i in the virtual block table $i = (x/B_x + 1) + (y/B_y + 1)T_x + (z/B_z + 1)T_xT_y$ which is then used to retrieve the corresponding block's offset o in the block data pool. The final location of the voxel in the block table pool is then $o + (x \bmod B_x) + (y \bmod B_y)B_x + (z \bmod B_z)B_xB_y$.

3.4. Volume Rendering

While our block-based representation could be directly used for rendering, the fact that we do not duplicate boundary

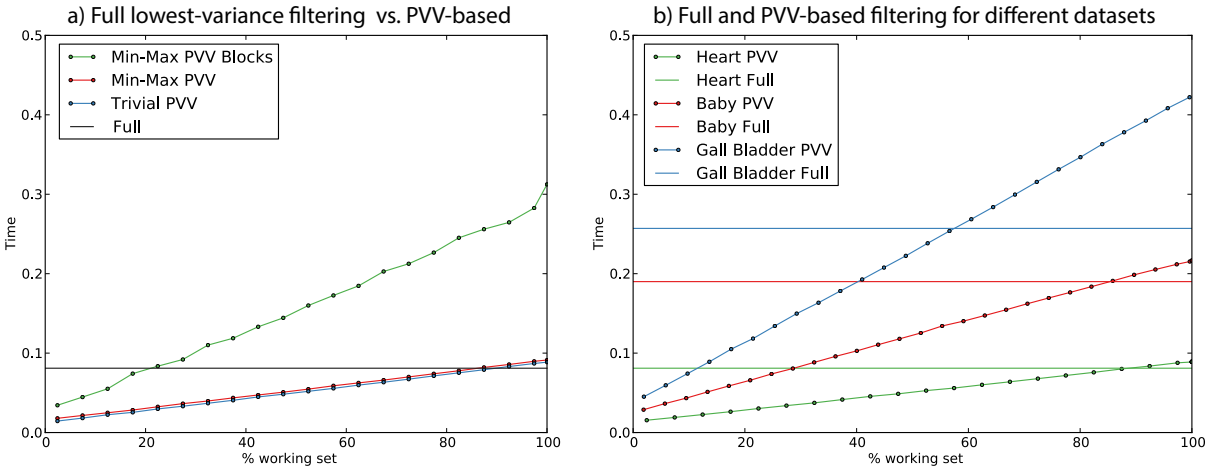

Figure 10: *Performance boost of visibility-driven processing compared to full processing with the lowest-variance filtering with radius 5 on a stream of 3D cardiac ultrasound datasets (128 × 100 × 128) – a snapshot is shown in Figure 2. b) illustrates the behavior of the same filter for the different dataset sizes: the same cardiac stream as in a). A streamed 3D ultrasound of a fetus (232 × 262 × 114) shown in Figure 1 and streamed 3D ultrasound of a gall bladder (256 × 256 × 256) as shown in Figure 11.*

voxels would prevent us from utilizing native trilinear interpolation which impacts the performance. Instead, however, we can exploit the recent *GL_ARB_sparse_texture* extension which natively supports rendering from sparse texture representations without duplicated block borders. While this implies an additional copy operation of the visible blocks into the sparse texture, we found that this option is preferable both in terms of performance and flexibility/ease-of-implementation. Our approach is not tied to any specific volume rendering algorithm, and in fact the current implementation of our pipeline flexibly integrates several different renderers including standard GPU-based ray casting and slice-based multidirectional occlusion volume rendering [vPBV10]. Furthermore, as block-based visibility information – as described in the previous sections – is already available, it can be straight-forwardly employed for rasterization-based empty-space skipping.

4. Evaluation

Our system was implemented in C++ using OpenCL for the realization of our processing pipeline. The volume rendering itself is performed in OpenGL. All benchmarks displayed in this section were performed on a PC with a 3.06 GHz CPU with NVIDIA GTX680 GPU with 2 GB of dedicated graphics memory running Windows 7. We investigated that our approach for culling invisible regions of the volume is correct in the sense that it can not affect the final image. Our goal was to show that if we solve the problems of reduced occlusion and increased opacity, the visualizations will be identical. One can prove that it can never occur that these visualization are not identical. To formally prove this, one can break down the main problem. Each subproblem represents a different situation with respect to transparent and non-transparent voxels. We refer the reader to the Appendix (part of the supplemental material) for the complete proof.

Figure 10 profiles the performance of our method. In 10a, we summarize the performance of for the lowest-variance filter. The black horizontal line shows the constant time that is needed to process the full volume. The blue line shows the dependency of the performance of the visibility-driven filtering on the amount of visible data, using a trivial set of visible voxels (trivial VV). This means that this set of voxels was defined only based on the immediate occlusion and opacity of the voxels, but not on the eventual change during filtering as we described in Section 3.1. The red line shows the performance with respect to the amount of visible data, but using the correct PVV as described in Section 3.1. We observe that the difference between the blue and the orange line is approximately constant and relatively small, also for other datasets for which we do not display the performance curves. Therefore, we can conclude that the correct PVV computation does not represent a significant performance overhead in the entire pipeline. Only if more than 85% of the data is visible, which is most likely not the case for cardiac 3D ultrasound, the visibility-driven filtering optimization would no longer pay off. We also measured performance using other filters, e.g., Perona-Malik anisotropic diffusion and bilateral filtering and they showed a very similar trend as lowest-variance filtering. In Figure 10b, we plot the performance gain for ultrasound datasets with different sizes using lowest-variance filtering.

Visibility-driven filtering requires a certain amount of

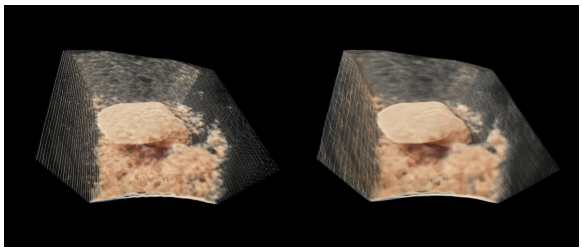

Figure 11: *Visualization US of a gall bladder: a snapshot of streamed data. Raw data visualization on the left and partially filtered (35% of the dataset) data visualization on the right.*

supporting buffers which consume memory. This will become an issue when handling larger datasets than ultrasound datasets used in obstetrics or 3D echocardiology. The examples shown in Figures 1 and 2 do not require memory optimization. For datasets that do not fit into the memory together with their supporting buffers, visibility-driven data packing becomes necessary. In Figure 10a, the green line plots the performance dependency if we use the data packing scheme with blocks. Obviously, the data packing scheme does not pay off for small datasets, but it is inevitable for large datasets that do not fit into the GPU memory together with the supporting buffers.

On modern ultrasound scanners with a 4D cardiac probe, ultrasound volumes are acquired at rates of 10-15 volumes per second [PVMd12], depending on the spatial extent of the acquired volume (sector). The larger the sector, the bigger is the volume and the lower is the acquisition rate. According to our own experience, larger sectors are acquired at approximately 15 volumes per second. Figure 2 shows that we allow for higher framerates which is relevant in-situ visualization for current 3D ultrasound acquisition capabilities.

5. Discussion and Limitations

The presented approach was designed to support complex filtering operations for live streaming volume data. Clearly, for very simple filters (such as, trivially, the **min** and **max** filters themselves), the overhead of minimum/maximum computation does not pay off, but in those cases filtering the whole volume is easily possible in real-time. Our goal was to enable application of filters that normally would be too expensive in such a scenario.

Furthermore, naturally our approach becomes less effective if a high percentage of the voxels remain visible. In this case, processing of the full volume might even be faster. However, if we detect a high percentage of visible voxels (e.g., due to a highly transparent transfer function) in one frame, we can simply disable our pipeline until the transfer function is modified again. This means that in most practical scenarios, the overall performance of our approach will always be as fast as processing of the entire volume.

Currently, there are some limitations in our implementation which result in sub-optimal performance and which we aim to address in the future:

- We start from an already reconstructed regular grid which we receive from the ultrasound scanner's software. However, it would also be possible to integrate our pipeline earlier to potentially eliminate or at least reduce some of the overhead. For, instance it would be easy to compute the minimum/maximum information during the resampling step from beam space to the regular grid.
- While using minimum/maximum neighborhood information to determine the set of potentially visible voxels is general enough to enable a wide range of practically useful filters, it is also quite conservative and can, in some cases, lead to considerable overestimation. For this, a closer analysis of the mathematical properties of individual filters may enable us to determine tighter bounds for special cases.
- While our block-based data packing scheme is effective in reducing memory overheads and enables efficient volume traversal, it sacrifices performance in processing operations which could exploit native trilinear filtering. If mechanisms similar to *GL_ARB_sparse_texture* were exposed in OpenCL, our approach could easily take advantage of them to remedy this drawback.

Despite these limitations, however, we have shown that our approach already results in a considerable reduction of processing times for realistic percentages of visible voxels and hence enables real-time filtering and, consequently, high-quality visualization of streaming volume data in cases where this was previously impossible.

6. Conclusions

In this paper, we presented a novel approach for integrated processing and visualization of streaming volume data. Our method conservatively estimates visibility information to limit processing operations to regions in the volume which can potentially contribute to the final image and hence can result in a considerable reduction of the processing load. By employing a memory efficient data packing scheme, our method also limits the amount of intermediate storage required for complex multi-pass operations. We have demonstrated that the resulting pipeline enables high-performance integrated filtering and visualization of steaming volume data such as 4D ultrasound.

Acknowledgments

This work has been carried out within the ISADAF project (In-Situ Adaptive Filtering, # 229352/O70) co-funded by the VERDIKT program of the Norwegian Research Council.

This project has been partially funded by the Vienna Science and Technology Fund (WWTF) through project VRG11-010 and by EC Marie Curie Career Integration Grant through project PCIG13-GA-2013-618680. The authors thank also the MedViz network in Bergen and GE Vingmed Ultrasound for the support and Matej Mlejnek for providing the dataset Anna.

References

[BBBV12] BIRKELAND Å., BRUCKNER S., BRAMBILLA A., VIOLA I.: Illustrative membrane clipping. *Computer Graphics Forum 31*, 3 (2012), 905–914. 4

[BHMF08] BEYER J., HADWIGER M., MÖLLER T., FRITZ L.: Smooth mixed-resolution gpu volume rendering. In *Proceedings of Point-Based Graphics 2008* (2008), pp. 163–170. 3

[BJE*11] BRUDER R., JAUER P., ERNST F., RICHTER L., SCHWEIKARD A.: Real-time 4d ultrasound visualization with the voreen framework. In *Proceedings of ACM SIGGRAPH 2011 Posters* (2011), pp. 74:1–74:1. 3

[BNS01] BOADA I., NAVAZO I., SCOPIGNO R.: Multiresolution volume visualization with a texture-based octree. *The Visual Computer 17*, 3 (2001), 185–197. 3

[BW03] BITTNER J., WONKA P.: Visibility in computer graphics. *Journal of Environment and Planning B: Planning and Design 5*, 30 (2003), 729–756. 3

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of Symposium on Interactive 3D Graphics and Games* (2009), ACM, pp. 15–22. 3

[COCSD03] COHEN-OR D., CHRYSANTHOU Y. L., SILVA C. T., DURAND F.: A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (2003), 412–431. 3

[DWS*88] DEERING M., WINNER S., SCHEDIWY B., DUFFY C., HUNT N.: The triangle processor and normal vector shader: A vlsi system for high performance graphics. *ACM SIGGRAPH Computer Graphics 22*, 4 (1988), 21–30. 3

[EEH*09] ELNOKRASHY A., ELMALKY A., HOSNY T., ELLAH M., MEGAWER A., ELSEBAI A., YOUSSEF A.-B., KADAH Y.: Gpu-based reconstruction and display for 4d ultrasound data. In *Proceedings of the IEEE International Ultrasonics Symposium 2009* (2009), pp. 189–192. 3

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EG Workshop on Graphics Hardware 2001* (2001), pp. 9–16. 5

[FNVV98] FRANGI A. F., NIESSEN W. J., VINCKEN K. L., VIERGEVER M. A.: Multiscale vessel enhancement filtering. In *Proceedings of Medical Image Computing and Computer–Assisted Intervention* (1998), pp. 130–137. 4

[FSK13] FOGAL T., SCHIEWE A., KRÜGER J.: An analysis of scalable GPU-based ray-guided volume rendering. In *IEEE Symposium on Large Data Analysis and Visualization* (2013). 3

[GM05] GOBBETTI E., MARTON F.: Far Voxels – a multiresolution framework for interactive rendering of huge complex 3d models on commodity graphics platforms. *ACM Transactions on Graphics 24*, 3 (2005), 878–885. 3

[HBJP12] HADWIGER M., BEYER J., JEONG W.-K., PFISTER H.: Interactive volume exploration of petascale microscopy data streams using a visualization-driven virtual memory approach. *IEEE Transactions of Visualization and Computer Graphics 18*, 2 (2012), 2285–2294. 3, 5

[HHSG05] HADWIGER M., HENNING SCHARSACH K. B., GROSS M.: Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum 24*, 3 (2005), 303–312. 3

[HKG00] HLADŮVKA J., KÖNIG A., GRÖLLER M. E.: Curvature-based transfer functions for direct volume rendering. In *Proceedings of Spring Conference of Computer Graphics* (2000), pp. 58–65. 3

[Ion10] IONESCU C.: The benefits of 3D-4D fetal echocardiography. *Maedica (Buchar) 5*, 1 (2010), 45–50. 2

[JBH*09] JEONG W.-K., BEYER J., HADWIGER M., VAZQUEZ A., PFISTER H., WHITAKER R. T.: Scalable and interactive segmentation and visualization of neural processes in em datasets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1505–1514. 3

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *Proceedings of ACM SIGGRAPH/EG Conference on Graphics Hardware* (2002), pp. 7–15. 3

[KLF05] KNISS J., LEFOHN A., FOUT N.: *Deferred Filtering: Rendering from Difficult Data Formats*. Addison Wesley, 2005, ch. 41, pp. 669–677. 3

[KWTM03] KINDLMANN G., WHITAKER R., TASDIZEN T., MOLLER T.: Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Visualization* (2003), pp. 513–520. 3

[Lev90] LEVOY M.: Efficient ray tracing of volume data. *ACM Transactions on Graphics 9*, 3 (1990), 245–261. 3

[LL94] LACROUTE P., LEVOY M.: Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of ACM SIGGRAPH 1994* (1994), pp. 451–458. 3

[LLY06] LJUNG P., LUNDSTRÖM C., YNNERMAN A.: Multiresolution interblock interpolation in direct volume rendering. In *Proceedings of EuroVis 2006* (2006), pp. 259–266. 3

[LV11] LEBIT F.-D., VLADAREANU R.: The role of 4d ultrasound in the assessment of fetal behaviour. *Maedica (Buchar) 6*, 2 (2011), 120–127. 2

[MJC02] MORA B., JESSEL J.-P., CAUBET R.: A new object-order ray-casting algorithm. In *Proceedings of IEEE Visualization 2002* (2002), pp. 203–210. 3

[NPH*00] NELSON T., PRETORIUS D., HULL A., RICCABONA M., SKLANSKY M., JAMES G.: Sources and impact of artifacts on clinical three-dimensional ultrasound imaging. *Ultrasound in Obstetrics & Gynecology 16*, 4 (2000), 374–383. 2

[PM87] PERONA P., MALIK J.: Scale-space and edge detection using anisotropic diffusion. In *Proceedings of IEEE Computer Society Workshop on Computer Vision* (1987), pp. 16–22. 4

[PVMd12] PERRI D., VASILYEV N., MARX G., DEL NIDO P.: Temporal enhancement of 3d echocardiography by frame reordering. *J Am Coll Cardiol Img 5*, 3 (2012), 300–304. 9

[TM99] TOMASI C., MANDUCHI R.: Bilateral filtering for gray and color images. In *Proceedings of International Conference on Computer Vision* (1999), pp. 839–846. 4

[vPBV10] ŠOLTÉSZOVÁ V., PATEL D., BRUCKNER S., VIOLA I.: A multidirectional occlusion shading model for direct volume rendering. *Computer Graphics Forum 29*, 3 (2010), 883–891. 8

[vSHW*12] ŠOLTÉSZOVÁ V., SÆVIL-HELLJESEN L. E., WEIN W., GILJA O. H., VIOLA I.: Lowest-variance streamlines for filtering of 3d ultrasound. In *Proceedings of EG Workshop on Visual Computing for Biomedicine* (2012), pp. 41–48. 2, 4, 7