Fused Multi-Volume DVR using Binary Space Partitioning

Stefan Lindholm†1,2, Patric Ljung2, Markus Hadwiger3 and Anders Ynnerman‡1

1Linköping University, Sweden
2Siemens Corporate Research, USA
3VRVis Research Center, Austria

Abstract

Multiple-volume visualization is a growing field in medical imaging providing simultaneous exploration of volumes acquired from varying modalities. However, high complexity results in an increased strain on performance compared to single volume rendering as scenes may consist of volumes with arbitrary orientations and rendering is performed with varying sample densities. Expensive image order techniques such as depth peeling have previously been used to perform the necessary calculations. In this work we present a view-independent region based scene description for multi-volume pipelines. Using Binary Space Partitioning we are able to create a simple interface providing all required information for advanced multi-volume renderings while introducing a minimal overhead for scenes with few volumes. The modularity of our solution is demonstrated by the use of visual development and performance is documented with benchmarks and real-time simulations.


1. Introduction

Fused rendering of multiple overlapping volumes is becoming an increasingly desired feature in visualization pipelines. Today, several established methods are used for the acquisition of medical data such as Magnetic Resonance Imaging (MRI) and Functional MRI (fMRI), single or dual source Computed Tomography (CT, DECT) and Ultrasound (US). Simultaneous visualization of this data can provide a greater sense of context, highlight relational aspects of different data or add functional information to a visualization [BHWB07]. Examples include high resolution scans of specific organs placed within a semi-transparent body, blood flow visualization through the model of a heart or regional brain activity visualized in the context of a human head. However, performing calculations on the combined visual impact of multiple spatially overlapping volumes is a complex and resource draining task that needs improved algorithms beyond regular DVR.

Without assumptions of aligned orientation or size, fused DVR is typically conformed to the intersection pattern of the involved volumes in order to limit processing of any volume to its own spatial extent. Support for large volumes through bricking further emphasizes the need for a true representation of the intersection pattern as brick boundaries typically cannot be traversed during a single rendering pass. To ensure proper blending of multiple semi-transparent objects a relative depth alignment among the regions is also essential.

In this paper we introduce a region based scene description that enables fusion in a DVR pipeline. To do this we use an efficient structure for representing regions in the intersection pattern with inherent depth order based on Binary Space Partitioning (BSP). This description represents all intersection regions as one or more convex polyhedra facilitating arbitrary fusion of bricked volumes. In our work the region based scene description is separated from the renderers increasing the modularity of our pipeline. The main contributions of this paper are

• Fusion of arbitrarily oriented volumes acquired from different modalities.
• Minimal overhead for static scenes during camera interaction.
• Real-time scene partitioning for limited numbers of animated volumes.
• Very efficient view-dependent sorting using a view-independent structure (BSP-tree).
• Efficient tailored shaders per combination of source volumes.

2. Related Work

BSP, and its data structure the BSP-tree, provides a method to recursively subdivide a conceptual space into subspaces by partitioning planes and was introduced for Computer Graphics purposes in 1980 by Fuchs et al. [FKN80]. It is commonly used in 3D graphics to ensure proper use of Painters Algorithm. Naylor et al. [NAT90] provides merging of multiple BSP-trees and discuss the use of set operations.

Engel et al. [EHK06] provides a comprehensive overview of the background, underlying problem and main ideas for single volume visualization. Jacq and Roux [JR97] performed multi-volume visualization but assume spatial alignment between data sets. Cai and Sakas [CS99] introduced image, accumulation and illumination levels of volume intermixing. The levels require different amount of flexibility in the pipeline, described briefly in the next section. Grimm et al. [GBKGO4] performed fusion of arbitrarily oriented volumes on the CPU. Their method of extracting depth knowledge on a per ray basis resembles hardware methods like depth peeling. The importance of distinguishing between costly overlapping regions and single volume areas is also noted. The notion of segments along the rendering ray is also introduced and is used in our work. Rössler et al. [RTF06] provided one of the earliest examples of multi-volume rendering performed on the GPU. The same assumptions or alignment as in [JR97] and limited blending options of samples due to separate slices for each volume prevents a generic pipeline. Beyer et al. [RTF06] provide an application oriented approach that stresses the usefulness of blending several modalities in a single scene.

Plate et al. [PHF07] demonstrate a full hardware supported pipeline for rendering of multiple arbitrarily aligned volumes. Bricking of large volumes is supported and as a result they emphasize the need to identify exact representations for all individual regions in the intersection pattern created by overlapping volumes. Their geometry based approach focus on specific overlapping parts of the scene where space is subdivided to region level. However, their approach requires expensive depth sorting of the resulting geometry prior to rendering which is performed by depth peeling on the GPU and the rendering is restricted to Texture Slicing. Rössler et al. [RBE08a] and [RBE08b] have constructed two similar hardware supported pipelines adapted for Texture Slicing and Raycasting. Common for the two publications is the presentation of an intermediate level of shader construction that enables visual shader generation. Also discussed is the importance and impact of instantiating multiple shaders for different volume combinations. The two pipelines contain expensive depth sorting algorithms or depth peeling techniques in order to assure correct blending. Maintaining separate solutions for Texture Slicing and Raycasting is also something we wish to avoid in order to maintain a generic pipeline. Brecheisen et al. [BPVtHR08] present a multi-volume rendering solution based on accelerated Raycasting. Their focus is primarily on flexibility in the pipeline to allow different modalities as well as tools, grids and widgets to contribute to the rendering. Similar to [PHF07] and [RBE08b] depth peeling is used to ensure correct blending. Intermixing schemes for sample blending is discussed and provide four variants which can be categorized using the levels introduced by Cai and Sakas [CS99]. Volumetric clipping using arbitrary polygonal models is also implemented and performed in the depth peeling stage of the pipeline.

3. Fusing Multiple Volumes

Regardless of rendering method, DVR corresponds to an iterative sequence of small color and opacity contributions along a ray. For blending purposes these contributions are constrained to be performed sequentially in the view direction. Fusion employs techniques that allow multiple volumes to contribute to the calculations while not invalidating the global order of their individual sample contributions. A ray can be segmented according to the intersection pattern of overlapping volumes and the segments rendered separately as long as they are blended in the correct order [GBKGO4]. All regions exhibiting different combinations of overlapping volumes can thus be found, represented and processed separately. These regions ensure that the number of volumes present on the GPU simultaneously is minimized and unnecessary sampling is avoided. Furthermore, these regions are typically forced to be convex to avoid depth conflicts during rendering.

Using Texture Slicing or Raycasting [EHK06] for fusing multiple overlapping volumes requires a scheme to choose how the volumes are to be intermixed that obey the requirement regarding sequential color composition. Image level intermixing schemes violates the necessary sequentiality by only taking into account samples from a single volume at any time. Accumulation level and illumination level intermixing, operating at data and classification level respectively, support arbitrary blending but require depth information regarding the involved volumes. This is the cause of the expensive operations such as depth sorting [PHF07], additional tesselation [RBE08a] or depth peeling [RBE08b] introduced in previous work.

Varying resolutions among multiple participating volumes can be a problem as it often causes visual errors.
These approximation errors originate in the presumption that a sample is given a certain length or size along the view direction. Different sampling schemes exist for dealing with this problem: Global sampling frequencies samples all volumes at the highest native resolution of any volume leading to expensive over sampling of low resolution volumes. Regional sampling frequencies samples the extent of a region at the highest resolution for any volume present in that region. However, variable sample frequencies alters the relative spatial extent between samples and can cause visual artifacts between neighboring pixels. The error can be minimized by the use of opacity correction [LNW07] but not completely removed. Interleaved sampling samples volumes on per volume grids. This scheme has been known to introduce unspecified ‘artifacts’ [RTF06] and ‘opacity errors’ [PHF07].

One known source for artifacts is that each samples spatial extent alters the depth order of contributions from different volumes within a small neighborhood of that sample. Accumulation level interleaving is supported by all three sampling schemes while illumination level interleaving requires synchronized sample positions found in global and regional but not in interleaved sampling. If used with the under operator however, the same synchronization either requires an internal order to be established among the volumes or a modified operator that includes multiple samples.

4. Binary Space Partitioning

The BSP used in this work is a non axis aligned scheme capable of producing accurate representations for intersections of multiple convex polyhedra [NAT90]. It also provides inherent depth order for relative regions removing the need for additional depth sorting.

All conceptual spaces in BSP are called cells, here denoted $C_v$, which are subdivided until a specified criteria is met. The initial space can be bounded or unbounded and each subspace in BSP can be represented by a node in a binary tree structure called BSP-tree. All internal nodes in the BSP-tree are associated with a partitioning and two children denoted $p_v$, $C_v^+$ and $C_v^-$ respectively. Since space is divided in a binary way and the plane normals are known, a view dependent order of the cells can be extracted from the BSP-tree in linear time without additional sorting. From a mathematical point of view each cell is a proper subset of its direct ancestors and the extent of a cell can be defined as all initial space combined with all partitioning planes in its direct ancestors applied in a top down order.

4.1. Geometrical Homogeneity and Complete Cells

Homogeneous qualities in cells are important aspects in BSP and often constitute the condition for terminating the recursive subdivision. Strict homogeneity as described in [FvDFH06] occur when no boundaries exists in the interior of a cell. This includes boundaries between one entity or another present within the cell and boundaries towards empty space. In this work the subdivision is driven with the objective to find intersecting regions amongst multiple convex polyhedra but no requirement exists to find boundaries between these polyhedra and empty space. This relaxation is introduced as geometric homogeneity.

In our work, cells do not have any geometrical representation of their own. Instead, they contain a set of polyhedra where a cell is said to be geometrically homogeneous if all polyhedra within the cell are equal and occupy the same space. It also means that cells occupied by less than two polyhedra are geometrically homogeneous by default. In figure 1 cells (a) and (c) are geometrically homogeneous by default and (b) by definition while cell (d) requires further subdivision. A geometrically homogeneous cell is called complete and is not subdivided further.

5. Region Based Scene Description using BSP

Regular volume-based scene descriptions often consist of representations on a per volume basis with properties such as spatial extent, transformations, bricking or polygonal boundary representations. On top of this we generate our region based scene description. In the case of non overlapping volumes the two scene descriptions will be equivalent while in the case of overlapping volumes the region based scene description will increase its granularity down to region level including information of volume occupation.

5.1. Volume Polyhedra

A volume polyhedron, $V_v$, provides a geometrical representation for a part of a volume defined as the intersection between the conceptual space of a cell, $C_v$, in the BSP-tree and the space occupied by the volume, $v$, as in equation 1.

$$V_v = C_v \cap v \quad (1)$$

Initially one polyhedron is created per volume representing the full extent of the volume in $\mathbb{R}^3$. All polyhedra are convex polygonal structures and are typically partitioned during

![Figure 1: Binary Space Partitioning is used on a test case in two dimensions to generate region representations for two polygons with the rightmost polygons containing a branched subdivision. Geometrical Homogeneity is introduced as a means to stop the recursive process. Cells (a), (b) and (c) are geometrically homogeneous.](image)
5.2. Generating Region Representations

The root node of a tree structure is first initiated with polyhedra from all volumes. BSP is then applied on the scene one plane at a time and the recursive procedure continues until all regions are represented by complete cells. If a cell exhibits polyhedra with open boundaries a partitioning plane is retrieved and the node subdivided. When subdividing a node by a given plane all polyhedra that do not intersect the plane are distributed to the children, equation 3. If a polyhedron intersects the plane it will be split into one small polyhedron in each child, equation 4.

\[ V_s \cap p_C = \emptyset \Rightarrow V_s \rightarrow V_s^+ \in C_s^+ \text{ or } V_s \in C_s^- \] (3)

\[ V_s \cap p_C \neq \emptyset \Rightarrow V_s \rightarrow V_s^+ \in C_s^+ \text{ and } V_s^- \in C_s^- \] (4)

This scheme is repeated on every level from polyhedra down to polygons and individual lines. During subdivision care must be taken to close the geometrical representations of the pieces created in the partitioning. In the case of polyhedra this means adding a polygon at the place of the intersection to close the hull. When the generation is completed, any region in the intersection pattern can be represented by a polyhedron present in the cell that corresponds to that region.

Our algorithm uses autopartition so the limited set of available partitioning planes is defined as all planes that coincide with boundaries for the original volumes. For the partitioning of a specific cell this set can be reduced to all planes that coincide with open boundaries of the cell polyhedra and choosing partitioning planes from this set provides a measurable way to reach a completed subdivision, equation 5.

\[ p_{\text{partition}} = \left\{ p : p = B_{\text{open}} \cup V_s \in C_s \right\} \] (5)

Introducing auxiliary planes could potentially improve performance by attaining reduced complexity but would have to be evaluated on a per case basis. The order in which the partition planes are applied can also be a significant factor in determining the overall complexity and thus affect the final rendering performance. However, we have discovered that while specific cases such as sparse scenes benefit from intelligent but costly choices of partitioning planes the general use case and our examples uses a simple low cost pick-any algorithm. Unlike many other applications built around tree structures we have no loss of performance for unbalanced trees as the trees are always traversed in full and not used for search purposes. For efficient polygon clipping in the partitioning process our implementation is based on the Sutherland-Hodgman algorithm as described in [FvDFH96] and the algorithm is extended to support splitting of polygons. An on-plane-threshold is also introduced to avoid nu-

---

**Figure 2:** BSP-tree (right) and Volume polyhedra (left). Our fusion solution uses Binary Space Partitioning to generate a region based scene description of multiple overlapping volumes by finding spatial representations for all individual regions in the intersection pattern. The resulting volume polyhedra are then used directly in the rendering or for the creation of intermediate proxy geometry.
merical inaccuracies. The thickness of the threshold is kept small within the range of the precision for the float data type.

In order to support bricking the algorithm needs to be extended so that each brick is treated as a separate volume and brick boundaries should be included in the set of eligible partitioning planes.

6. Region Based Raycasting

Our raycaster iterates over the regions and uses a dual buffer approach with a main buffer for the rendering and accumulation and a scratch buffer for per region storage of ray exit points. The accumulated alpha of the main buffer is also duplicated to the scratch buffer between passes to avoid read/write conflicts. Each region iteration includes the following passes; 1) Render back facing polygons of the region polyhedron to scratch buffer to store exit point location and duplicate alpha from main buffer by a lookup. 2a) Switch to main buffer and render front facing polygons from scratch buffer and perform rendering to main buffer. Only pixels that are covered by the active polyhedron are affected in each pass and none of the buffers needs clearing between passes. The adaption of raycasting to the non-cuboid rendering regions in region based scene description require three alterations relative to a standard implementation:

- **Non Cuboid Rendering Areas** - Entry and exit points for cuboid regions can be acquired procedurally. However, the non cuboid polyhedra defining the regional rendering areas in our approach requires additional computation. Alternation between two render targets is performed twice for each cell to extract exit points and to perform rendering.

- **Cell Based Rendering** - Ray entry positions for all cells must follow a global depth offset to avoid artifacts at cell boundaries. This offset is implemented as a manipulation of the ray entry point in the shader. Cell based rendering requires double buffers or pixel read-backs to progress the accumulated alpha between cell rendering passes.

- **Shader Instantiation** - To avoid costly conditional expressions [Fer04] multiple shaders for different volume configurations are used. We employ a simple instantiation scheme such that only a single shader source is written before being instantiated multiple times for different volume combination. Instantiation is performed using preprocessor conditionals (#ifdef, #endif).

The BSP-tree is traversed according to the camera position and all leaf cells are raycasted in visibility order. If all volumes fit on the GPU no memory transfers are necessary and entry and exit points are acquired per cell through proxy geometry. Information regarding volumes present in the cell is then used within the rendering module to select an appropriate shader. All shaders are designed such that a single ray is traversed in world space coordinates and texture lookup coordinates are then acquired by a transformation. Shaders that operate on multiple volumes simultaneously maintain one such transformation matrix and a distance to the next sample individually for each volume. Steps along the ray are taken according to the smallest distance to any volume and samples are accumulated one after the other according to the blending defined in the shader. Raycasting is performed with cell-compatible empty space leaping [KW03] individually per volume and early ray termination for individual rays. Empty space leaping is implemented using a small binary map per volume with a resolution relative to a user defined block size of the original volume. If a lookup in such a map is negative the distance to the next sample for that volume is increased so that the successive sample is taken in the next block but less than one step length on the other side of the boundary. Empty space leaping is performed by discarding alpha-saturated rays.

Support for arbitrary blending and composition schemes for individual volumes is only achieved when instantiating and compiling one shader per volume combination. This is cumbersome due to the high number of possible combinations but for cases with only a handful of volumes the cost is manageable. The cost of frequent shader switches in the GL API is lower than the overhead of conditionals and does not affect performance since they are enforced anyway by the ping-pong required by the non-cuboid rendering areas. All instantiation schemes use the same macro building so existing shader sources can be instantiated with a different scheme without change. Three shader instantiation schemes have been implemented:

- **Single Instantiation** - Instantiates a single shader for all volumes and require no shader switches but introduces costly conditionals [Fer04] in the shader. With increased support for conditionals in future hardware this scheme could remove the need for instantiation.

- **Instantiation by Count** - Instantiates n shaders for n volumes and require 2 shader switches per region. Different blending for different volume combinations is prohibited since there is no information in the shader at runtime about which volumes that are active.

- **Instantiation by Combination** - Instantiates one shader per volume combination and require 2\textsuperscript{nd} shaders...
for $n$ volumes and 2 shader switches per region. Different blending between different volume combinations is possible.

For Texture Slicing, the partitioning planes or the polygonal hulls can be used to generate proxy geometry. A z-offset for each plane is introduced to adapt for the cell based rendering and the instantiation schemes described above are used in the same way as in raycasting. The scene graph oriented pipeline is implemented as a plug-in for the open source eXtensible Imaging Platform (XIP) [caB], a visual programming environment partly developed at Siemens Corporate Research.

7. Results

Benchmark timings have been performed using VTune Performance Analyzer while application frame rates were measured directly in XIP. All tests were performed on a 3.2GHz Intel Xeon machine equipped with 2GB RAM and a single GeForce 8800 Ultra graphics accelerator with 768MB VRAM. Three test cases are presented in figure 4-6 demonstrating arbitrarily oriented overlapping volumes rendered with independent resolutions. None of the cases demonstrates any drop in overall rendering speed even if the BSP-tree is re-generated every frame. The only shading used is a simulated head light performed as a multiplication of channel RGB by A for each sample before accumulation.

CASE 1 (CT + fMRI) Skull and Brain - The head and brain data sets in figure 4 have resolutions of $256 \times 256 \times 256$ and $512 \times 512 \times 256$ respectively at 12 bits integer precision. The bone structure of the head is shaded using simulated self shading on top of a TF lookup while the brain is unshaded. Rendering is performed at 24-27Hz with negligible BSP-Tree generation time.

CASE 2 (3xCT) Upper body and Heart - The shoulder and chest data sets in figure 5 both have resolutions of $512 \times 512 \times 256$ and are sub sampled at 1 sample per two voxels while the $512 \times 448 \times 416$ resolution heart is super sampled at 2 samples per voxel in all directions. All three data sets are stored as 8 bit integer textures and raycasting delivers 37-45Hz rendering with simulated self shading on shoulders and chest.

CASE 3 (2xCT + US) Lower body and Baby - The hips and abdomen data sets in figure 6 share size and precision with the shoulder and chest data sets in Case 2. The ultrasound probe measures $200 \times 199 \times 135$ at 8 bits and is super sampled with 3 samples per voxel. Simulated self shading was used on all data sets at 40-43Hz using raycasting.

Total BSP-tree complexity relates roughly as $\text{nodes} \propto \text{volumes}$ as can be seen in table 1 with storage requirements for polygons following the same pattern. This growth rate is only apparent in worst-case scenarios where all volumes overlap each other without relative alignment. Table 2 shows a comparison of performance between scenes of varying complexity where a sparse placement directly translates to shorter BSP-tree generation times with a 45% - 70% gain (50% - 350% loss) for increasing number of involved volumes. Most recent work in the field of multi-volume visualization relies on hardware depth peeling in some form and thus view-dependent. BSP-trees on the other hand are view-independent and need not be recalculated unless the volumes are moved relative to each other. Table 3 illustrates where complexity is introduced.

Regardless if the BSP-trees are regenerated every frame due to volume animation the generation times listed in table 2 are low enough to not be noticeable in our test cases. We also compare the rendering performance of our three test cases to $\text{image level}$ intermixing where volumes are rendered separately before a final accumulation. The results are small, 3%-5%, drops in frame rates between single volume rendering and fusion for cases 1 and 2 while case 3 suffered a 18% drop.

8. Conclusions

We have presented a generic pipeline for fused multi volume DVR. Introducing a novel technique using BSP to find and represent regions in the intersection pattern for a set of volumes we provide the concept of a region based scene description enabling easy adaption of standard DVR methods for the rendering stage. Using the region descriptions provided in our BSP-tree structure and its inherent depth order
we not only bypass an expensive depth peeling but also accomplish a larger portion of the pipeline to be generic. A render scheme independent fusion module increases modularity in our framework as demonstrated by the possibility to switch between raycasting and Texture Slicing at runtime.

The three test cases presented display how fusion allows different modalities to compose a scene. Independent sampling density is used to maximize performance and shading is applied individually on each volume to guide visual perception. Our benchmarks imply high but necessary generation times for large number of volumes in extreme conditions while the performance overhead for scenes with few volumes is minimal. Furthermore, the BSP structure only needs to be updated in case volumes are moved in relation to each other and the exposed depth order could potentially be valuable in other fields of Computer Graphics.

9. Acknowledgements

We would like to thank Gianluca Paladini at Siemens Corporate Research for the initiative of this collaboration. The work in this publication is supported by the Knowledge Foundation grant no. 2007/402.

References


[caB] CABIG: The eXtensible Imaging Platform (XIP) project is an Open Source framework and platform for Medical Imaging. XIP is part of the caBIG initiative. http://www.openxip.org.


Table 1: BSP-tree complexity in terms of primitives. Only the number leaf cells and leaf polygons affects rendering while nodes, polyhedra and total polygon count affects BSP-tree generation.
Figure 4: Case 1: Visualizing an fMRI scan depicting brain activity inside the context providing CT scan of a human head at 24-27Hz. Using two separate volumes allows us to lower the sampling rate and apply different shading on the bone structure and the BSP-tree generation time was 0.08ms. Empty space leaping is applied individually on both volumes.

Figure 5: Case 2: Super-sampling of a high resolution heart within two low resolution body parts (shoulders and chest). Using two opposing clip planes to cut away redundant information we are able to sustain a high sampling rate of for the heart while still maintaining 37-45Hz with 0.14ms spent on BSP-tree generation.

Figure 6: Case 3: Visualizing a US probe in the context of two CT data sets depicting the lower body of a woman. The non-axis alignment of the modalities is evident and motion of the ultrasound probe gives no detectable difference in rendering speed. The visualization is performed at 40-43Hz at a cost of 0.17ms for generating the BSP-tree.