Vision, Modeling, and Visualization (2024)
L. Linsen and J. Thies (Eds.)

# SVDAG Compression for Segmentation Volume Path Tracing

Mirco Werner[1†], Max Piochowiak[1†], and Carsten Dachsbacher[1]

[1] Karlsruhe Institute of Technology

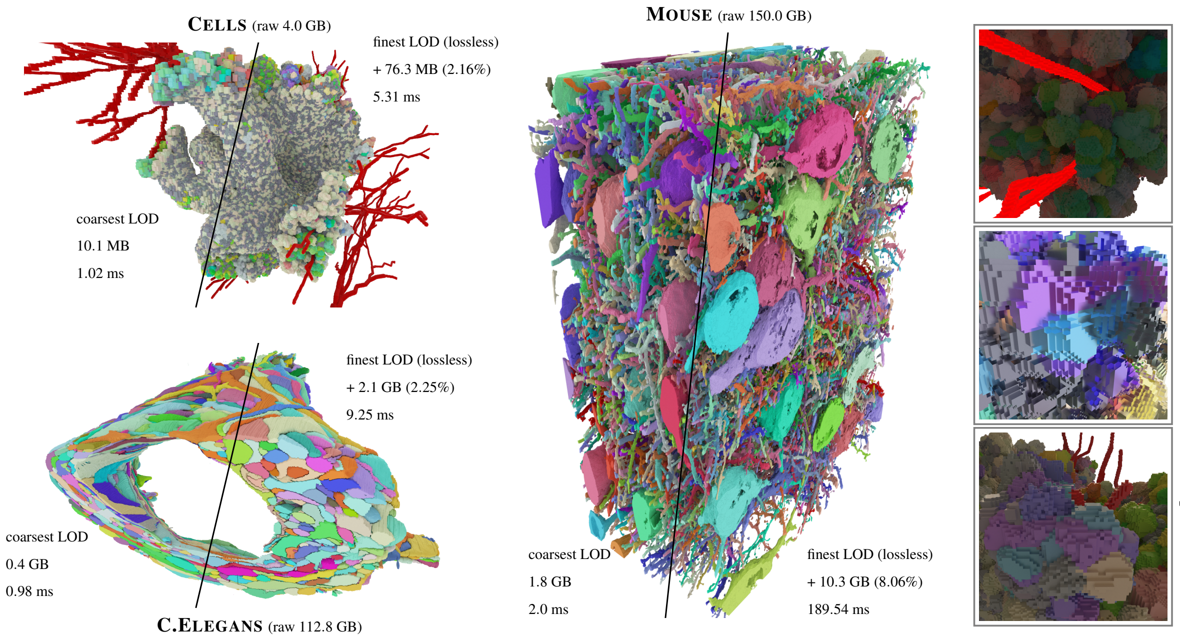


Figure 1: Our lossless compression technique for segmentation volumes achieves strong compression rates by decoupling labels from voxel occupancy. The 1920 × 1080 images show path tracing (32 bounces with Russian roulette) of our compressed volumes rendered with interactive frame rates at 1 sample per pixel (here accumulated over 1024 frames). Path tracing with physically based BRDFs enables a wide range of photorealistic effects (insets). Our technique supports rendering at different levels of detail (LOD). For each dataset, we report average render times per frame as well as required memory for the coarsest LOD and additional memory for fine LODs with compression rates in %.

**Abstract**

*Many visualization techniques exist for interactive exploration of segmentation volumes, however, photorealistic renderings are usually computed using slow offline techniques. We present a novel compression technique for segmentation volumes which enables interactive path tracing-based visualization for datasets up to hundreds of gigabytes: For every label, we create a grid of fixed-size axis aligned bounding boxes (AABBs) which covers the occupied voxels. For each AABB we first construct a sparse voxel octree (SVO) representing the contained voxels of the respective label, and then build a sparse voxel directed acyclic graph (SVDAG) identifying identical sub-trees across all SVOs; the lowest tree levels are stored as an occupancy bit-field. As a last step, we build a bounding volume hierarchy for the AABBs as a spatial indexing structure. Our representation solves a compression rate limitation of related SVDAG works as labels only need to be stored along with each AABB and not in the graph encoding of their shape. Our compression is GPU-friendly as hardware raytracing efficiently finds AABB intersections which we then traverse using a custom accelerated SVDAG traversal. Our method is able to path-trace a 113 GB volume on a consumer-grade GPU with 1 sample per pixel with up to 32 bounces at 108 FPS in a lossless representation, or at up to 1017 FPS when using dynamic level of detail.*

**CCS Concepts**

***• Human-centered computing*** $\rightarrow$ ***Scientific visualization; • Computing methodologies*** $\rightarrow$ ***Ray tracing;***

## 1. Introduction

Segmentation volumes store labels per voxel to encode object regions (colored in Fig. 1). These volumes are used in a wide range of domains [GZGZ23; WAL*14; BTB*22; MBB*19] in which visual analysis is essential for explorative analysis. Suitable interactive visualization software for segmentation volumes rarely supports photorealistic renderings. Current workflows to create high-quality rendering, e.g. for science communication, are slow [WPB*23]: First, data is visually explored in an interactive, non-photorealistic framework. Second, a scene configuration is exported to an offline path tracing renderer for high-quality rendering. Rendering can take hours and is usually CPU-based as large-scale volumes do not fit into GPU memory.

GPU path tracing is increasingly used for visual analysis of *qualitative* voxel data storing continuous signals [WBDM23; HMES20]. We propose a path tracing framework for *segmentation* volumes: Volumes are compressed with a new variant of sparse voxel directed acyclic graphs (SVDAGs) [KSA13]. Our three-level data structure decouples labels and voxel occupancy enabling a strong compression (Section 3). We show how the representation can be efficiently traversed during path tracing (Section 4). To summarize, our main contributions are:

- a hardware-accelerated path tracing framework for interactive rendering of segmentation volumes,
- a lossless SVDAG-based compression technique for segmentation volumes that decouples labels and voxel occupancy,
- a configurable level of detail scheme and occupancy field encoding to increase rendering performance.

## 2. Related Work

Visualization and compression of segmentation volumes is an active research topic [BTB*22; PD24]. We summarize related works:

**Segmentation Volume Compression** Compressing qualitative voxel data where each voxel directly stores attributes is well studied [BGI*14]. Minnen et al. compress raw images before the segmentation [MJB*21]. Few specialized solutions for the actual segmentation volumes exist, including brick-wise paletting [Goo16], bit-encoding of region boundaries through connected component analysis [MHL*17], graph-compression of hierarchical label distributions [AAS21], and operation-based brick encodings [PD24].

**SVDAG Compression** As introduced by Kämpe et al. [KSA13], sparse voxel directed acyclic graphs (SVDAGs) can effectively compress sparse voxel scenes. While SVDAGs originally operate on binary data, extensions enable compression of attributed data as well [Wil15; DKB*16; DSKA17]. Other works improve the compression rates by utilizing pointerless DAGs [VMB20], spatial [VMG16] or temporal [KRB*16] correlation, or lossy schemes [vdLSE20]. To the best of our knowledge, we are the first to apply SVDAG compression to segmentation volumes.

**Segmentation Volume Visualization** Segmentation volume rendering is commonly used for segmentation annotation or proofreading [BBB*17]. Displaying 2D slices [Goo16; AGL05; BSG*09; BBB*17] is fast but limited in conveying spatial layouts. Meshing label regions [QDB11; LC87] allows subsequent rasterization but is time-consuming. Recent works for direct, voxel-based rendering handle opacity management for dense volumes [LAB*24], semi-automatic transfer function design [AAA*22] or color filtering for coarse levels of detail [AAS21]. Voxel renderers capable of displaying large-scale volumes use culling [BMA*18], out-of-core methods [BAK*13], and compression [PD24].

**Path Tracing for Volume Visualization** Interactive path tracing is commonly used for non-segmented volumes. Hofmann et al. [HMES20; HHCM21] introduce denoising for semi-transparent volume visualization. Iglesias-Guitian et al. [IMM22] handle noisy guiding buffers. Compression can increase path tracing performance which is usually memory-bound [WHW22; WBDM23]. Recent works apply path tracing to unstructured volumes [ZWMW23; MSG*23]. Path tracing segmentation volumes is commonly handled in offline renderers [WPB*23; RBS20], e.g. Blender Cycles [Ble18]. We introduce an interactive path tracer for segmentation volumes that leverages hardware-accelerated ray tracing and a tailored SVDAG-based compression.

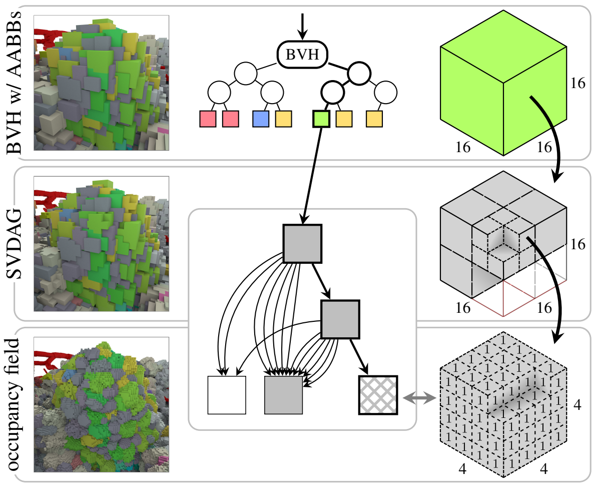


Figure 2: Our three-level compression technique decouples label information from voxel occupancy by grouping voxels with the same label in AABBs of size $16^3$. The voxel positions are compressed in an SVDAG that is shared between all AABBs. Leaf nodes are either entirely solid, empty, or contain an occupancy bitfield that encodes voxel positions in a $4^3$ region. SVDAG nodes that are unreachable from the green AABB are omitted for clarity.

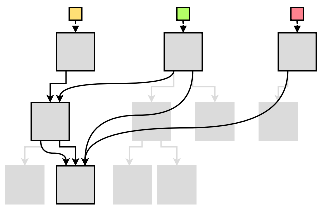


Figure 3: SVOs (flat) per AABB are compacted into one SVDAG by redirecting their child pointers. The AABBs (colored) track their own SVDAG entry node. Pointers to empty nodes are omitted.

$^{\dagger}$joint first authors

## 3. Compressed Segmentation Volumes

Segmentation volumes store one integer label per voxel. In the following sections, we explain the three levels of our compression method for such volumes and the resulting data structure (Fig. 2). Our method starts by covering each label region in the dataset with axis aligned bounding boxes (AABBs). As we use a fixed AABB size of $16^3$ voxels, larger regions are split into a grid of AABBs (Fig. 4). The first level of our data structure contains these AABBs, each containing voxels that have the same label only (Section 3.1). The AABBs are stored in a bounding volume hierarchy (BVH) for fast spatial access. In the second level, the actual voxel occupancy of all AABBs is compacted into one shared sparse voxel directed acyclic graph (SVDAG) to reuse duplicate voxel occupancy patterns (Section 3.2). We define regions of $4^3$ as leaf nodes in the SVDAG and encode empty and solid voxels in the form of an occupancy bitfield for faster rendering (Section 3.3).

### 3.1. Level 1: BVH with Label Region AABBs

The first level of our data structure separates labels from voxel occupancy. Segmentation volumes contain large uniform areas where each voxel is assigned to the same label (colored in Fig. 1). We exploit this property by subdividing the volume into AABBs of fixed size ($16^3$) that contain voxels with the same label only (Fig. 4). Regions that are larger than $16^3$ are covered by a grid of AABBs. To create this grid, a label region is recursively subdivided using techniques similar to BVH construction [MOB*21]. The placement of the AABB grid could be optimized to improve compression in the later stages. As this is time-consuming, we simply subdivide regions recursively along their longest axis until they are small enough. A voxel may only be assigned to a maximum of one AABB. AABBs belonging to different labels can overlap (Fig. 4). An AABB is defined by its position, its label, and its contained voxel occupancy. With this subdivision, the label information of each of the up to $16^3$ voxels in an AABB becomes redundant and is no longer stored per voxel. We build a BVH over all AABBs for fast spatial access which is used as an acceleration structure during rendering.

### 3.2. Level 2: Sparse Voxel Directed Acyclic Graphs

The second level of our data structure encodes the voxels inside the AABBs. Since we previously separated the labels, our goal here is to compress the voxel occupancy in a lossless fashion as much as possible while ensuring efficient rendering. To that end, we first construct sparse voxel octrees (SVOs) [LK10] in each AABB (Section 3.2.1). The SVOs are then converted to SVDAGs which again can be compressed to a single SVDAG that is shared between all AABBs to achieve high compression rates (Section 3.2.2).

#### 3.2.1. SVOs

Each SVO has $\log_2(16)+1=5$ levels. A single node of an SVO can be encoded in 2 bytes, where we use 12 bits to store a pointer to the first of the eight child nodes. The children of a node are stored consecutively in memory. SVOs achieve high compression when there is a lot of empty space [LK10]. In our case, however, all voxels have the same label, so it is possible to compress large solid areas as well. Therefore, we mark nodes as leaves by assigning them an invalid value as their child pointer to terminate traversal on a (potentially) higher level and exploit one unused bit to distinguish between solid (1) and empty (0) regions. The pointer to the root node of each SVO is stored in its corresponding AABB.

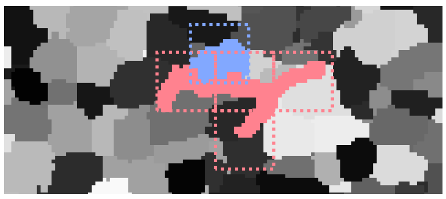


Figure 4: Slice of a segmentation volume. Each label region is covered by a grid of fixed-size $16^3$ AABBs (dashed). The AABBs serve as entry points for the following SVDAG voxel occupancy encoding.

#### 3.2.2. SVDAGs

The structures of the voxels in the AABBs often contain identical subregions. Therefore, we compress the SVOs to an SVDAG by redirecting all pointers that reference topologically identical subtrees to reference the same node (Fig. 3). Since our SVOs do not store any label information, it is not only possible to compress each SVO of an AABB to an SVDAG individually, but we can apply this compression across all SVOs. As the SVDAG compression reorders and merges individual nodes, child nodes are no longer stored consecutively in memory, so eight explicit child pointers (each 32 bits) per node are required. SVO nodes have to be transitioned to the new layout before applying SVDAG compression. The number of bytes per node increases from 2 to 32. However, this is more than compensated by the strong reduction in the number of nodes. To denote leaf nodes, we set their first child pointer to an invalid value. Another bit is used to distinguish solid and empty regions.

For SVDAG compression, we follow the in-place bottom-up approach of Kämpe et al. [KSA13]. It is possible to compress all SVOs simultaneously to an SVDAG or to work incrementally. For the latter, a few SVOs are first compressed to smaller SVDAGs which are compacted to a single shared SVDAG afterward. This way, large volumes can be handled piecewise when all SVOs require too much memory simultaneously. The SVDAG compression can be implemented on the GPU to improve performance.

Commonly, there are certain groups of labels, which we call *types*, that are always visualized together, e.g. individual cells in the CELLS dataset belong to tumor or healthy tissue types. Using this property, we can build separate SVDAGs per type during preprocessing. SVDAGs of types requested during visualization can then be quickly merged on the GPU at runtime which is faster than constructing the full shared SVDAG from scratch. Note that this type grouping only streamlines transfer function editing and does not limit or change the number of unique labels in the visualization.

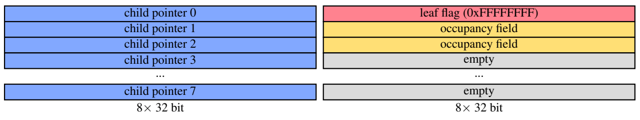


Figure 5: Our SVDAG memory layout of an inner node (left) and leaf node with stored occupancy field (right). Both are 32 bytes in total each. No additional attributes need to be stored per node.

### 3.3. Level 3: Occupancy Fields

Our last optimization does not change the compression rate but increases the rendering performance. As discussed in detail in Section 5, loading large (32 bytes) SVDAG nodes that are incoherently distributed in memory during ray traversal decreases performance. Hence, the number of nodes visited and consequently the number of memory accesses during the traversal should be minimized. To that end, we redefine SVDAG leaf nodes as a $4^3$ region and encode the voxel occupancy in the $4^3$ region directly in the leaf node using a bitfield. We refer to these bitfields as *occupancy fields*. The final memory layout of inner SVDAG nodes and leaves containing the encoded 64-bit occupancy field is visualized in Fig. 5. We choose $4^3 = 64$ voxels since $8^3 = 512$ bit entries would not fit into the 256 total bits of a node's child pointers. We encode the occupancy fields when transitioning the SVOs from the implicit (2 bytes) to the explicit (32 bytes) node layout. Since nodes that encode larger regions ($8^3$ or $16^3$) can be leaves as well, we set all bits of the occupancy field to either 1 or 0 in this case to distinguish completely solid or empty regions, respectively.

## 4. Rendering of Compressed Segmentation Volumes

In this section, we describe how our compressed segmentation volume (Section 3) is traversed during path tracing. We use modern hardware ray tracing as well as a custom accelerated SVDAG traversal to determine ray intersections with our data structure. Accessing the volume information on the GPU requires uploading three buffers containing the AABBs, the SVDAG nodes, and the shading materials.

### 4.1. AABB Traversal

We pass a list of AABB positions to the GPU BVH builder to construct a BVH for the volume. When we trace a ray, the GPU traverses the BVH and invokes a custom intersection shader to determine whether an intersection with the SVDAG inside the AABB occurs. We query the index of the AABB to load the corresponding pointer to the SVDAG root node and the label information.

### 4.2. SVDAG and Occupancy Field Traversal

Once the custom intersection is invoked, we manually traverse the SVDAG with a multi-level version of 3D-DDA [AW87]. When reaching a leaf node, the $4^3$ region encoded in the occupancy field is traversed using standard 3D-DDA. When an intersection with a solid voxel is found, the corresponding distance is committed, so that the GPU can determine the closest hit.

### 4.3. Level of Detail

Our data structure allows switching between different levels of detail (LODs) during rendering. In the coarsest LOD, the traversal is terminated after an intersection with an AABB is determined. Different LODs can be rendered by traversing to deeper levels of the SVDAG before terminating. Renderings with three different LODs corresponding to terminating at $16^3$, $4^3$, and $1^3$ voxels are shown in Fig. 2. It is possible to make a per-AABB decision for the LOD by calculating the minimum distance between the camera and the AABB. In contrast to using the distance to the current ray's AABB intersection for the LOD decision, this avoids rendering artifacts for neighboring pixels that hit the same AABB but at slightly different distances, resulting in potentially different LOD decisions.

## 5. Evaluation

We implemented our compression and rendering technique in Vulkan and provide its source code [Wer24].We evaluate our technique on three datasets using an NVIDIA RTX 4070 Ti SUPER GPU and an AMD Ryzen 5 2600 CPU. As our compression procedure works in stages and SVDAGs can be merged piecewise, we report numbers for different *modes* to show the effects on the compression rate and rendering performance:

- In SVOs (AABB), each AABB contains its own SVO.
- In SVDAGs (TYPE), all AABBs for labels that were grouped into a specific type share one SVDAG each (Section 3.2.2).
- In SVDAG (SHARED), one single SVDAG contains the shared voxel occupancy of all AABBs in the dataset.

In practice, we recommend always using a single shared SVDAG (SVDAG (SHARED)) during rendering. It is created by uploading and merging requested SVDAGs per type (SVDAGs (TYPE)) on the GPU given the visibility configuration from the transfer function.

### 5.1. Datasets

**CELLS** [RBS20] is a tumor growth simulation with a high number of labels. We visualize 118251 tumor cells and the blood vessels and treat other regions as empty. We group the 118251 uniquely labeled cells into 22 types and the blood vessels into one type, i.e. the mode SVDAGs (TYPE) contains 23 SVDAGs. For visualization purposes, we assign each of the cell types a color and add a slight color offset for different cells of a type. **C.ELEGANS** [WMM*21] is a segmentation of C. Elegans brains from eight development stages. We use the last stage containing 234 neurons. The 234 neurons map directly to 234 different types. **MOUSE** [MBB*19] is a segmented mouse cortex. We visualize all its 96 neuron regions and treat other label regions as empty. Again, the 96 neuron regions map directly to 96 different types. Note that as our data structure uses $16^3$ voxel regions as a basis and decouples labels from occupancy, higher label numbers would not alter the compression rates significantly.

As MOUSE and C.ELEGANS contain empty space at their borders, we treat the bounding box around solid voxels as the raw volume. The raw memory requirements are calculated assuming the volume is stored in a three-dimensional texture with 32-bit labels

| | dimensions | $\# \cdot 10^9$ | solid | mem. [MB] | # labels |
|---|---|---|---|---|---|
| CELLS | $1000 \times 1000 \times 1000$ | 1.0 | 15.39% | 4000.0 | 118251 |
| C.ELEGANS | $661 \times 11044 \times 15448$ | 112.77 | 5.45% | 112771.7 | 234 |
| MOUSE | $5446 \times 8381 \times 3286$ | 149.98 | 15.57% | 149982.65 | 96 |

Table 1: Datasets' dimension in voxels, total number of voxels, fraction of solid voxels, raw memory requirements as 32 / 8 / 8 bit volumes. As the original C.ELEGANS and MOUSE datasets contain plenty of empty space, the evaluated dimensions are sub-volumes of the original data containing all solid voxels. The number of labels is reported for completeness but does not significantly alter compression rates in our method.

| CSGV mem. [MB] | | CSGV CR | |
|---|---|---|---|
| 15.46 | (+544) | 0.39% | (13.99%) |
| 736.981 | (+768) | 0.65% | (1.33%) |
| 1910.17 | (+768) | 1.27% | (1.79%) |

Table 2: Memory and compression rate of CSGV [PD24]. Theoretical compression rate when including a GPU cache size to decompress visible regions to fitting levels of detail during rendering in ().

| | AABBs + BVH + | SVOs (AABB) | | SVDAGs (TYPE) | | SVDAG (SHARED) | |
|---|---|---|---|---|---|---|---|
| | mem. [MB] | mem. [MB] | CR | mem. [MB] | CR | mem. [MB] | CR |
| CELLS | $5.63 + 4.42$ | $+243.92$ | 6.35% | $+90.47$ | 2.51% | $+76.29$ | **2.16%** |
| C.ELEGANS | $248.13 + 186.24$ | $+13888.0$ | 12.7% | $+3724.4$ | 3.69% | $+2099.91$ | **2.25%** |
| MOUSE | $1032.11 + 786.44$ | $+61092.74$ | 41.95% | $+15598.79$ | 11.61% | $+10276.93$ | **8.06%** |

Table 3: Memory (mem) and compression rate (CR, final CR in bold) reported for different modes. Total memory consists of the AABBs, the BVH during runtime, and either the SVOs or SVDAG(s). The CR compares the total memory in a specific mode with the raw memory of the volume as reported in Table 1.

| | | | without occupancy field | | with occupancy field | | |
|---|---|---|---|---|---|---|---|
| | | SVOs (AABB) | SVDAGs (T) | SVDAG (S) | SVDAGs (T) | SVDAG (S) | CSGV |
| 1 bounce | CELLS | 8.19 (0.4) | 5.81 (0.42) | 5.87 (0.42) | 2.95 (0.51) | 2.95 (0.51) | 23.21 (2.28) |
| | C.ELEGANS | – | 12.76 (0.47) | 12.75 (0.47) | 5.33 (0.55) | 5.61 (0.55) | 87.18 (16.56) |
| | MOUSE | – | – | 134.72 (0.78) | – | 126.73 (0.98) | 124.89 (8.47) |
| 32 bnc. | CELLS | 11.42 (0.81) | 8.6 (0.84) | 8.66 (0.85) | 5.32 (1.01) | 5.31 (1.02) | 25.64 (2.29) |
| | C.ELEGANS | – | 35.06 (0.83) | 18.72 (0.83) | 9.01 (0.98) | 9.25 (0.98) | 97.63 (17.06) |
| | MOUSE | – | – | 203.65 (1.67) | – | 189.54 (2.0) | 136.16 (10.55) |

Table 4: Render times [ms] per frame of the scenes shown in Fig. 1. Averaged over 1024 frames when always using the finest or (coarsest) LOD. Last column shows timings for voxel grid traversal of cached CSGV bricks, again, either using always the finest or (coarsest) LOD.

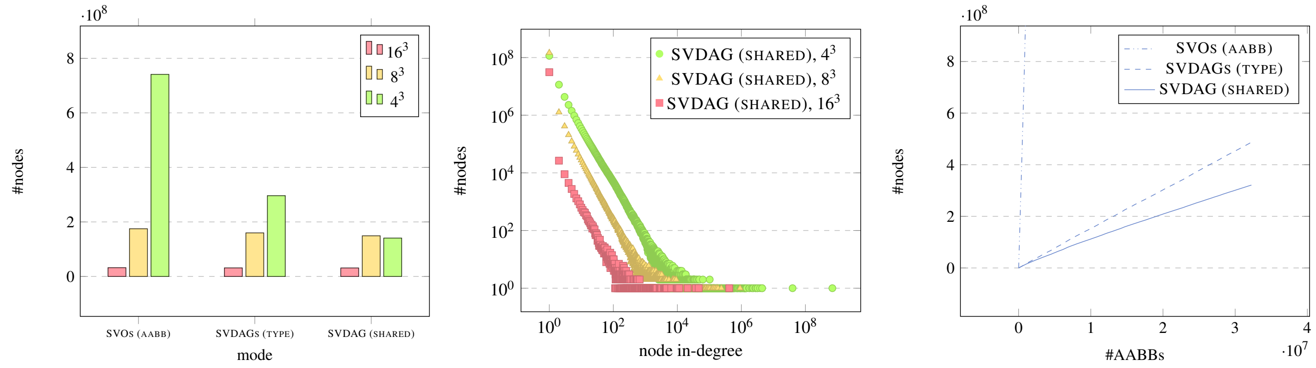


Figure 6: SVDAG node reuse and number of nodes in relation to merging steps for MOUSE. Most reuse of nodes occurs on the $4^3$ leaf node level (left). A small number of nodes is reused extremely often (middle). Compressing the voxel occupancy information into an SVDAG shared between AABBs theoretically has logarithmic growth as only $2^{64}$ reusable possibilities for the $4^3$ occupancy field exist. However, we initially observe almost linear memory growth when incrementally merging AABBs into the SVDAG (SHARED), but at a much slower rate than for the non-merged SVOs (AABB) or partially merged SVDAGs (TYPE) (right). This is due to the fact that a constant portion of leaf nodes is never reused and introduces a small additional memory footprint per new AABB (middle).

| | CELLS | C.ELEGANS | MOUSE |
|---|---|---|---|
| SVOs (AABB) CPU [s] | 5.97 | 905.26 | 4127.16 |
| SVDAGs (TYPE) CPU [s] | 2.24 | 117.92 | 683.45 |
| SVDAG (SHARED) CPU [ms] | 408.49 | 33089.85 | 147721.0 |
| SVDAG (SHARED) GPU [ms] | 101.96 | 2865.33 | – |

Table 5: Timings for each step to compress segmentation volumes: Computing AABBs and their SVOs from lists of solid voxel positions per label, merging them to SVDAGs (TYPE), and subsequently merging these partial SVDAGs to one SVDAG (SHARED). We provide a GPU implementation of the last step for fast merging of visualized types during rendering start-up or runtime.

for CELLS and 8-bit labels for C.ELEGANS and MOUSE. We report the dimensions, number of voxels, the fraction of solid voxels, and the raw memory requirements in Table 1.

## 5.2. Memory Consumption and Compression Rate

The memory consumption and compression rates are listed in Table 3. The total memory is composed of the AABBs, the BVH, and either the SVOs or SVDAG(s). By decoupling labels from voxel occupancy, we achieve strong compression rates. Although the SVDAG has an increased node size of 32 bytes compared to the SVO with 2 bytes per node, its compression rate is significantly better due to the reuse of node subtrees.

Merging smaller SVDAGs belonging to subsets of AABBs (SVDAGS (TYPE)) into a single large SVDAG (SVDAG (SHARED)) improves the compression rate further. We analyze this behavior in Fig. 6, where we visualize the required number of nodes for all modes depending on the number of AABBs for MOUSE. The other two datasets show similar behavior. Our technique can effectively compress empty space by building AABBs only around non-empty regions. Our SVDAG compression of non-empty space yields best results for volumes with label regions that are non-overlapping, have a size that is approximately equal to or a multiple of the AABB size, or share regions with a highly similar structure.

## 5.3. Rendering Performance

Fig. 1 shows converged path-traced renderings of the three datasets using our compression scheme and a path length of up to 32 bounces (paths may terminate earlier due to throughput-based Russian roulette). In Table 4, we report render times of a single frame for each of the camera positions seen in Fig. 1 with either 1 or 32 bounces, a constant coarsest or finest LOD, and across all modes. C.ELEGANS and MOUSE can only be rendered using SVDAGS (TYPE) as the SVOS (AABB) (and even SVDAGS (TYPE) for MOUSE) exceed GPU memory limitations.

Due to large volume sizes and incoherent access to nodes which minimizes cache hits, our technique is currently limited by VRAM throughput. For this reason, we gain a significant performance improvement by employing occupancy fields since fewer distributed SVDAG nodes have to be loaded from memory. Additionally, we observe thread divergence when threads of a workgroup traverse the SVDAG to different levels. Although MOUSE has only roughly four times as many solid voxels as C.ELEGANS, the render times are about 20 times higher. While C.ELEGANS has a rather closed surface and its neuron label regions are tightly packed, MOUSE contains a lot of free space between neurons, which is computationally much more expensive to traverse and prone to thread divergence.

For certain visualization purposes, a non-photorealistic rendering without indirect illumination is desired. Direct illumination or ambient occlusion corresponds to one-bounce path tracing which is significantly cheaper than full path tracing as can be seen in Table 4. We show a comparison between one bounce and full path tracing in Fig. 7a. Path tracing introduces image noise that quickly vanishes with diffuse materials and a constant environment map (Fig. 7b). To ensure interactive exploration, even in the case of the MOUSE dataset, we visualize only the coarsest LOD, i.e. the AABBs, while the camera is moving. When accumulating multiple frames, the fine LOD is used after the first frame, hence the first frame has to be discarded to avoid biased renderings. Reducing the LOD allows us to use more samples per pixel in the first frame to show a largely noise-free path-traced image in real time.

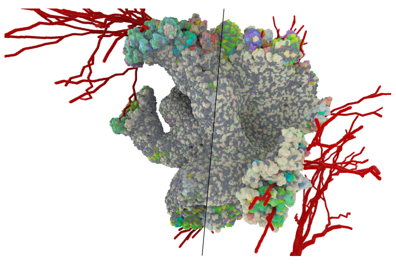
## 5.4. Compression Time

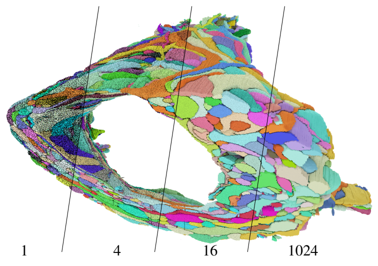
Table 5 shows timings for converting segmentation volumes into our compression scheme. Given lists of voxels per label, we report times for converting them to SVOS (AABB), and the SVOS (AABB) to SVDAGS (TYPE) as offline pre-processing steps on the CPU. Our implementation is single-threaded and processes one type after another. Since types share no dependencies in these steps, this process can be trivially parallelized. Merging SVDAGS (TYPE) to SVDAG (SHARED) can be performed as CPU offline pre-processing as well. Alternatively, only those types visible given the current transfer function can be merged using a GPU SVDAG builder on startup or during runtime.

## 5.5. Comparative Results

We compare our compression with the state-of-the-art compressed segmentation volumes (CSGV) from [PD24] using a brick size of $32^3$. Volume dimensions and labels as in Table 1 are compressed with voxels containing empty space set to zero. While CSGV compression rates are initially better by factors of 5 to 6 compared to our method (Table 2), compression domain traversal is not possible and additional memory for a cache of visible bricks is needed during rendering. For smaller volumes with many visible surfaces (CELLS), our method thus requires less overall GPU memory.

We compare our rendering performance with grid-based voxel traversal using the CSGV brick caching in Table 4. For always rendering the finest CSGV LOD, a 4 GB cache is allocated for decompressing CSGV bricks while our method requires no additional memory for volume traversal. CSGV rendering uses path tracing at 1 sample per pixel with Russian roulette, multi-level DDA [AW87],


(a)



(b)

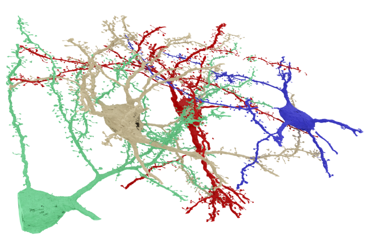

(c)

Figure 7: (a) Single bounce (left) and up to 32 bounces (right). (b) Increasing number of samples per pixel. (c) MOUSE rendered with only a subset of neurons. The transfer function can be updated during runtime by rebuilding the BVH to enable or disable certain types.

and empty brick skipping. Except for MOUSE at the finest LOD, our SVDAG raytracing method outperforms the voxel grid traversal rendering by factors of 2 to 15. This is mostly due to better handling of empty space and the missing overhead that the separate brick decompression stages introduce for CSGV.

### 5.6. Discussion

**AABB Size** We choose a fixed AABB size of $16^3$ as a compromise between compression rate and rendering performance: Smaller AABBs result in a larger BVH but smaller SVDAG depths. This increases performance as BVH traversal is hardware-accelerated. Larger AABBs result in a smaller BVH but deeper SVDAGs with better compression. In theory, it is possible to use arbitrary AABB sizes, e.g. by building AABBs for label regions without any subdivision. However, as this leads to arbitrary SVDAG depths, more thread divergence occurs, and SVDAG compression is reduced.

**Adjusting the Transfer Function** We can adjust the transfer function to enable or disable types (or labels) during runtime. For best rendering performance, we rebuild the BVH with the adjusted AABBs sets when a type is en- or disabled. Although it is possible to discard intersections with disabled types in the intersection shader, it introduces shader invocation and traversal overhead. Figure 7c shows the MOUSE dataset with only a subset of neurons enabled. Using the transfer function to change the appearance of a type neither requires adjusting the AABBs nor the SVDAG, but only an update of the buffer containing the shading materials.

## 6. Conclusion and Future Work

We have introduced a novel lossless SVDAG compression method and a hardware-accelerated path tracing framework for interactive rendering of segmentation volumes. Our technique achieves strong compression and rendering performance that outperforms voxel-based traversal. Future work may further improve compression rates by optimizing AABB placement or leveraging symmetry [VMG16]. As discussed in Section 5.3, the rendering performance of our technique is currently limited due to thread divergence, which may be solved by wavefront path tracing [LKA13]. Caching and freeing SVDAG nodes based on visibility could reduce memory demands during rendering. Finally, for more complex lighting configurations, storing learned light information inside the AABBs for path guiding [VHH*19] could be investigated.

**Funding**

This work has been supported by the Helmholtz Association (HGF) under the joint research school "HIDSS4Health – Helmholtz Information and Data Science School for Health" and through the Pilot Program Core Informatics.

## References

[AAA*22] AGUS, M., ABOULHASSAN, A., AL THELAYA, K., et al. "Volume Puzzle: visual analysis of segmented volume data with multivariate attributes". *Proc. IEEE Visualization and Visual Analytics*. 2022, 130–134. DOI: 10.1109/VIS54862.2022.00035 2.

[AAS21] AL-THELAYA, K., AGUS, M., and SCHNEIDER, J. "The Mixture Graph – A Data Structure for Compressing, Rendering, and Querying Segmentation Histograms". *IEEE Trans. on Vis. and Comp. Graph.* 27.2 (2021), 645–655. DOI: 10.1109/TVCG.2020.3030451 2.

[AGL05] AHRENS, J., GEVECI, B., and LAW, C. "ParaView: An End-User Tool for Large Data Visualization". *Visualization Handbook* (Jan. 2005). DOI: 10.1016/B978-012387582-2/50038-1 2.

[AW87] AMANATIDES, JOHN and WOO, ANDREW. "A Fast Voxel Traversal Algorithm for Ray Tracing". *Proc. Eurograph. - Technical Papers*. 1987. DOI: 10.2312/egtp.19871000 4, 6.

[BAK*13] BEYER, J., AL-AWAMI, A., KASTHURI, N., et al. "ConnectomeExplorer: Query-Guided Visual Analysis of Large Volumetric Neuroscience Data". *IEEE Trans. on Vis. and Comp. Graph.* 19.12 (2013), 2868–2877. DOI: 10.1109/TVCG.2013.142 2.

[BBB*17] BOERGENS, K. M, BERNING, M., BOCKLISCH, T., et al. "webKnossos: efficient online 3D data annotation for connectomics". *Nature Methods* 14.7 (2017), 691–694. DOI: 10.1038/nmeth.4331 2.

[BGI*14] BALSA RODRÍGUEZ, M., GOBBETTI, E., IGLESIAS GUITIÁN, J.A., et al. "State-of-the-Art in Compressed GPU-Based Direct Volume Rendering". *Computer Graphics Forum* 33.6 (2014), 77–100. DOI: 10.1111/cgf.12280 2.

[Ble18] BLENDER, ONLINE COMMUNITY. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: http://www.blender.org 2.

[BMA*18] BEYER, J., MOHAMMED, H., AGUS, M., et al. "Culling for Extreme-Scale Segmentation Volumes: A Hybrid Deterministic and Probabilistic Approach". *IEEE Trans. on Vis. and Comp. Graph. (Proc. IEEE Sci. Vis.)* 25.1 (2018). DOI: 10.1109/TVCG.2018.2864847 2.

[BSG*09] BRUCKNER, S., SOLTESZOVA, V., GRÖLLER, E., et al. "BrainGazer - Visual Queries for Neurobiology Research". *IEEE Trans. on Vis. and Comp. Graph.* 15.6 (2009), 1497–1504. DOI: 10.1109/TVCG.2009.121 2.

[BTB*22] Beyer, J., Troidl, J., Boorboor, S., et al. "A Survey of Visualization and Analysis in High-Resolution Connectomics". *Computer Graphics Forum* 41.3 (2022), 573–607. DOI: 10.1111/cgf.14574 2.

[DKB*16] Dado, B., Kol, T. R., Bauszat, P., et al. "Geometry and Attribute Compression for Voxel Scenes". *Computer Graphics Forum* 35.2 (2016), 397–407. DOI: 10.1111/cgf.12841 2.

[DSKA17] Dolonius, D., Sintorn, E., Kämpe, V., and Assarsson, U. "Compressing color data for voxelized surface geometry". *Proc. ACM SIGGRAPH Symposium on Interact. 3D Graph. and Games*. San Francisco, California: ACM, 2017. DOI: 10.1145/3023368.3023381 2.

[Goo16] Google Inc. *Neuroglancer*. https://github.com/google/neuroglancer. 2016 2.

[GZGZ23] Gao, S., Zhou, H., Gao, Y., and Zhuang, X. "BayeSeg: Bayesian modeling for medical image segmentation with interpretable generalizability". *Medical Image Analysis* 89 (2023), 102889. DOI: 10.1016/j.media.2023.102889 2.

[HHCM21] Hofmann, N., Hasselgren, J., Clarberg, P., and Munkberg, J. "Interactive Path Tracing and Reconstruction of Sparse Volumes". *Proc. ACM Comput. Graph. Interact. Tech* 4.1 (2021). DOI: 10.1145/3451256 2.

[HMES20] Hofmann, N., Martschinke, J., Engel, K., and Stamminger, M. "Neural Denoising for Path Tracing of Medical Volumetric Data". *Proc. ACM Comput. Graph. Interact. Tech* 3.2 (Aug. 2020). DOI: 10.1145/3406181 2.

[IMM22] Iglesias-Guitian, J. A., Mane, P., and Moon, B. "Real-Time Denoising of Volumetric Path Tracing for Direct Volume Rendering". *IEEE Trans. on Vis. and Comp. Graph.* 28.7 (2022), 2734–2747. DOI: 10.1109/TVCG.2020.3037680 2.

[KRB*16] Kämpe, V., Rasmuson, S., Billeter, M., et al. "Exploiting coherence in time-varying voxel data". *Proc. ACM SIGGRAPH Symposium on Interact. 3D Graph. and Games*. Redmond, Washington: ACM, 2016, 15–21. DOI: 10.1145/2856400.2856413 2.

[KSA13] Kämpe, V., Sintorn, E., and Assarsson, U. "High resolution sparse voxel DAGs". *ACM Transactions on Graphics* 32.4 (July 2013). DOI: 10.1145/2461912.2462024 2, 3.

[LAB*24] Lesar, Ž., Alharbi, R., Bohak, C., et al. "Volume conductor: interactive visibility management for crowded volumes". *The Visual Computer* 40.2 (2024), 1005–1020. DOI: 10.1007/s00371-023-02828-8 2.

[LC87] Lorensen, W. E. and Cline, H. E. "Marching cubes: A high resolution 3D surface construction algorithm". *Proc. ACM SIGGRAPH* 21.4 (1987), 163–169. DOI: 10.1145/37402.37422 2.

[LK10] Laine, S. and Karras, T. "Efficient sparse voxel octrees". *Proc. ACM SIGGRAPH Symposium on Interact. 3D Graph. and Games*. New York, NY, USA: ACM, 2010, 55–63. DOI: 10.1145/1730804.1730814 3.

[LKA13] Laine, S., Karras, T., and Aila, T. "Megakernels considered harmful: wavefront path tracing on GPUs". *Proc. of the High-Performance Graphics Conference*. New York, NY, USA: ACM, 2013, 137–143. DOI: 10.1145/2492045.2492060 7.

[MBB*19] Motta, A., Berning, M., Boergens, K. M., et al. "Dense connectomic reconstruction in layer 4 of the somatosensory cortex". *Science* 366.6469 (2019), eaay3134. DOI: 10.1126/science.aay3134 2, 4.

[MHL*17] Matejek, B., Haehn, D., Lekschas, F., et al. "Compresso: Efficient Compression of Segmentation Data For Connectomics". *Medical Image Computing and Computer-Assisted Intervention*. Cham: Springer, 2017, 781–788. DOI: 10.1007/978-3-319-66182-7_89 2.

[MJB*21] Minnen, David, Januszewski, Michał, Blakely, Tim, et al. "Denoising-based Image Compression for Connectomics". *bioRxiv* (2021). DOI: 10.1101/2021.05.29.445828 2.

[MOB*21] Meister, Daniel, Ogaki, Shinji, Benthin, Carsten, et al. "A Survey on Bounding Volume Hierarchies for Ray Tracing". *Computer Graphics Forum* 40.2 (2021), 683–712. DOI: 10.1111/cgf.142662 3.

[MSG*23] Morrical, N., Sahistan, A., Güdükbay, U., et al. "Quick Clusters: A GPU-Parallel Partitioning for Efficient Path Tracing of Unstructured Volumetric Grids". *IEEE Trans. on Vis. and Comp. Graph.* 29.1 (2023), 537–547. DOI: 10.1109/TVCG.2022.3209418 2.

[PD24] Piochowiak, M. and Dachsbacher, C. "Fast Compressed Segmentation Volumes for Scientific Visualization". *IEEE Trans. on Vis. and Comp. Graph.* 30.1 (2024), 12–22. DOI: 10.1109/TVCG.2023.3326573 2, 5, 6.

[QDB11] Quey, R., Dawson, P., and Barbe, F. "Large-scale 3D random polycrystals for the finite element method: Generation, meshing and remeshing". *Computer Methods in Applied Mechanics and Engineering* 200 (Apr. 2011), 1729–1745. DOI: 10.1016/j.cma.2011.01.002 2.

[RBS20] Rosenbauer, J., Berghoff, M., and Schug, A. "Emerging Tumor Development by Simulating Single-cell Events". *bioRxiv* (2020). DOI: 10.1101/2020.08.24.264150 2, 4.

[vdLSE20] Van der Laan, R., Scandolo, L., and Eisemann, E. "Lossy Geometry Compression for High Resolution Voxel Scenes". *Proc. ACM Comput. Graph. Interact. Tech* 3.1 (May 2020). DOI: 10.1145/3384541 2.

[VHH*19] Vorba, J., Hanika, J., Herholz, S., et al. "Path guiding in production". *Proc. ACM SIGGRAPH Courses*. Los Angeles, California: ACM, 2019. DOI: 10.1145/3305366.3328091 7.

[VMB20] Vokorokos, L., Madoš, B., and Bilanová, Z. "PSVDAG: Compact Voxelized Representation of 3D Scenes Using Pointerless Sparse Voxel Directed Acyclic Graphs". *Computing and Informatics* 39.3 (2020), 587–616. DOI: 10.31577/cai_2020_3_587 2.

[VMG16] Villanueva, A. J., Marton, F., and Gobbetti, E. "SSVDAGs: symmetry-aware sparse voxel DAGs". *Proc. ACM SIGGRAPH Symposium on Interact. 3D Graph. and Games*. New York, NY, USA: ACM, 2016, 7–14. DOI: 10.1145/2856400.2856420 2, 7.

[WAL*14] Weissenböck, J., Amirkhanov, A., Li, W., et al. "FiberScout: An Interactive Tool for Exploring and Analyzing Fiber Reinforced Polymers". *2014 IEEE Pacific Visualization Symposium*. 2014, 153–160. DOI: 10.1109/PacificVis.2014.52 2.

[WBDM23] Wu, Q., Bauer, D., Doyle, M. J., and Ma, K. "Interactive Volume Visualization Via Multi-Resolution Hash Encoding Based Neural Representation". *IEEE Trans. on Vis. and Comp. Graph.* (2023), 1–14. DOI: 10.1109/TVCG.2023.3293121 2.

[Wer24] Werner, M. https://github.com/MircoWerner/SegmentationVolumeCompression. 2024 4.

[WHW22] Weiss, S., Hermüller, P., and Westermann, R. "Fast Neural Representations for Direct Volume Rendering". *Computer Graphics Forum* 41.6 (2022), 196–211. DOI: 10.1111/cgf.14578 2.

[Wil15] Williams, B. "Moxel DAGs: Connecting material information to high resolution sparse voxel DAGs". *California Polytechnic State University*. 2015. DOI: 10.15368/theses.2015.112 2.

[WMM*21] Witvliet, D., Mulcahy, B., Mitchell, J. K., et al. "Connectomes across development reveal principles of brain maturation". *Nature* 596.7871 (2021), 257–261. DOI: 10.1038/s41586-021-03778-8 4.

[WPB*23] Winding, M., Pedigo, B. D., Barnes, C. L., et al. "The connectome of an insect brain". *Science* 379.6636 (2023), eadd9330. DOI: 10.1126/science.add9330 2.

[ZWMW23] Zellmann, S., Wu, Q., Ma, K., and Wald, I. "Memory-Efficient GPU Volume Path Tracing of AMR Data Using the Dual Mesh". *Computer Graphics Forum* 42.3 (2023), 51–62. DOI: 10.1111/cgf.14811 2.