Supplemental Material

Additional performance analysis results

In Table 1, we list the timing for all three evaluated steps (linearization and shader recompilation, visibility mask computation, and rendering) for all graphics cards on all the datasets. On the Nvidia Quadro RTX 8000, the DOS method still runs at around 2 frames per second (FPS) at a 2048×2048 resolution with 1024 slices, and at 4 to 7 FPS at a 1024×1024 resolution with 512 slices even for the largest datasets (fibers and mitochondria).

	Frame size	Number of slices 256	512	1024
Integrated graphics				
Fibers	512 × 512	6 / 468 / 429	5 / 468 / 786	6 / 468 / 1502
	1024 × 1024	6 / 469 / 1012	7 / 468 / 1917	6 / 468 / 3722
	2048 × 2048	9 / 471 / 2736	10 / 471 / 5210	9 / 472 / 10170
Pores	512 × 512	5 / 545 / 464	6 / 556 / 839	5 / 545 / 1629
	1024 × 1024	6 / 533 / 934	7 / 535 / 1760	6 / 542 / 3439
	2048 × 2048	9 / 551 / 2618	8 / 551 / 4990	9 / 545 / 9772
Mitos	512 × 512	6 / 546 / 487	6 / 545 / 898	5 / 546 / 1734
	1024 × 1024	5 / 547 / 985	7 / 546 / 1839	6 / 545 / 3655
	2048 × 2048	8 / 550 / 2642	8 / 552 / 5036	8 / 548 / 9825
Commodity desktop graphics				
Fibers	512 × 512	9 / 89 / 221	11 / 89 / 421	15 / 89 / 805
	1024 × 1024	12 / 89 / 215	15 / 89 / 373	12 / 89 / 689
	2048 × 2048	14 / 89 / 405	15 / 89 / 617	13 / 89 / 1203
Pores	512 × 512	7 / 56 / 161	15 / 47 / 287	18 / 52 / 542
	1024 × 1024	21 / 48 / 172	12 / 52 / 294	19 / 50 / 523
	2048 × 2048	13 / 48 / 356	15 / 48 / 597	15 / 49 / 1068
Mitos	512 × 512	9 / 26 / 101	9 / 29 / 187	11 / 27 / 327
	1024 × 1024	9 / 27 / 134	12 / 27 / 215	15 / 27 / 383
	2048 × 2048	17 / 27 / 340	17 / 27 / 575	18 / 29 / 1018
Professional graphics				
Fibers	512 × 512	5 / 85 / 34	7 / 85 / 55	5 / 85 / 100
	1024 × 1024	5 / 85 / 81	5 / 85 / 138	4 / 85 / 248
	2048 × 2048	4 / 86 / 204	4 / 87 / 324	4 / 91 / 580
Pores	512 × 512	5 / 47 / 26	5 / 47 / 41	4 / 47 / 72
	1024 × 1024	4 / 47 / 60	4 / 46 / 92	4 / 46 / 156
	2048 × 2048	4 / 46 / 162	4 / 46 / 231	4 / 50 / 368
Mitos	512 × 512	5 / 26 / 39	6 / 26 / 67	5 / 25 / 123
	1024 × 1024	4 / 25 / 91	4 / 25 / 160	4 / 25 / 294
	2048 × 2048	4 / 25 / 203	4 / 25 / 326	4 / 25 / 574

Table 1: Performance evaluation of the volume conductor. The three numbers are the times in milliseconds to linearize the predicates and recompile the shader, recompute the visibility mask, and render the image.

We also present additional graphs for integrated graphics (Fig. 1) and commodity desktop graphics (Fig. 2).

Algorithms

Algorithms Algorithm 1 and Algorithm 2 describe the instance sparsification and visibility mask generation processes in detail.

Online repository and examples

The source code and all examples with synthetic data are available online at `http://lgm.fri.uni-lj.si/portfolio-view/volume-conductor`. The online demo is implemented in WebGL 2.0 Compute. While the API is unsupported in stable releases of web browsers, the demo can be tested in

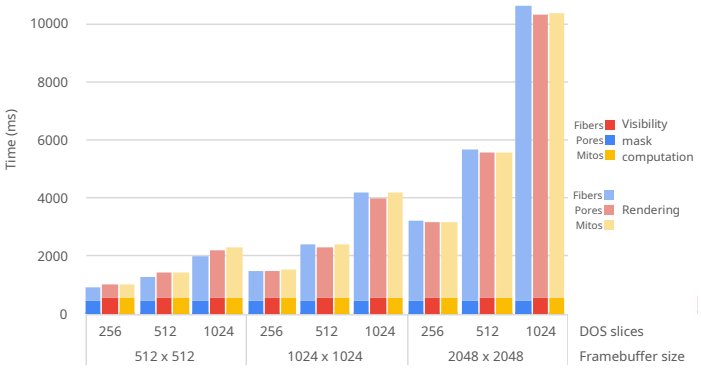

Fig. 1: Performance evaluation on a laptop computer with integrated graphics. Linearization and shader recompilation times are negligible and not depicted in the graph.

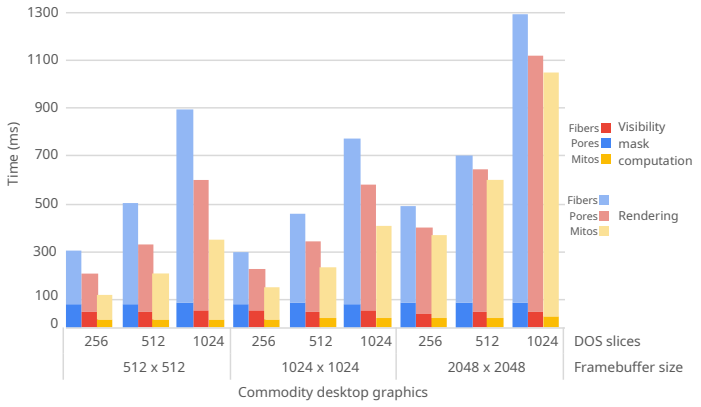

Fig. 2: Performance evaluation on a commodity desktop computer with an Nvidia GTX 1060 graphics card. Linearization and shader recompilation times are negligible and not depicted in the graph.

Chromium Dev build 87 available at `https://commondatastorage.googleapis.com/chromium-browser-snapshots/index.html`.

Additional examples

This section contains additional examples of the features of the volume conductor. Fig. 3 presents additional renders of the fibers, pores, and mitochondria. Fig. 4 shows the effects of different sparsification functions on a synthetic dataset. In Fig. 5 and Fig. 6, we used another synthetic dataset containing boxes, spheres, and ellipsoids of various sizes. The distribution of most instances is uniform, except for ellipsoids, which are placed along one of the spatial diagonals of the volume. A large sphere is placed in the center of the volume, which is not visible without sparsification. The volume conductor is designed to be independent of the rendering method; thus, we include a comparison of path tracing and directional occlusion shading (Fig. 7).

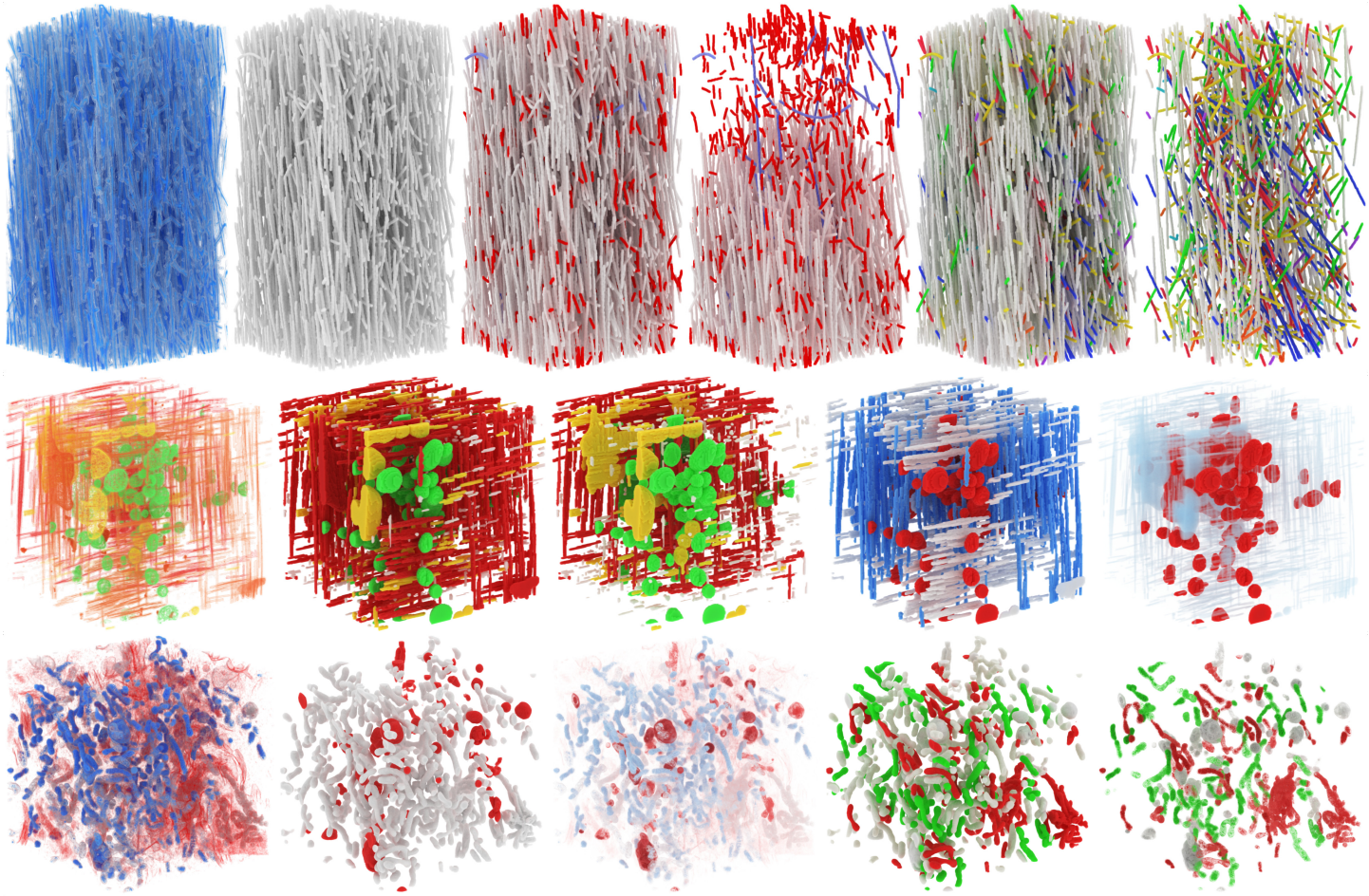

Fig. 3: **Top row, fibers:** raw data, segmentation, colorization of short (red) and bent (blue) instances, same as previous but with 50 % of the remaining instances hidden, colorization by orientation, the same as previous with 80 % of the vertical fibers hidden. **Middle row, pores:** raw data, colorization by roundedness, the same as previous with 50 % of the needle-shaped pores hidden, colorization of needle-shaped pores by orientation, and ghosting. **Bottom row, mitochondria:** raw data, colorization of type, the same as previous but with added blending with raw data, colorization by branching and thinning, same as previous with opacity transfer. Path tracing was used to generate the images.

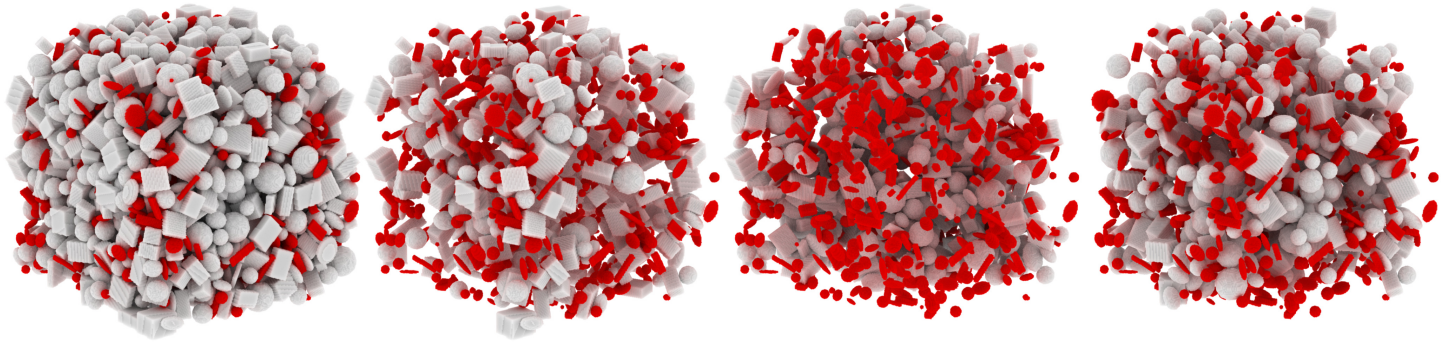

Fig. 4: Demonstration of sparsification functions. From left to right: no sparsification, random, depth-based, and context-preserving sparsification. Instances with a small volume are colored red. Path tracing was used to generate the images.

Algorithm 1 Sparsification with temporal coherency.

```
for all i ∈ instances do
    hidden_i ← false
end for
for all r ∈ predicates do
    G ← instances(r)
    n_hidden ← 0
    n_toHide ← ⌊visibilityRatio(r) · |G|⌋
    sort(G)
    for all i ∈ G do
        if not hidden_i and not visible_i then
            n_hidden ← n_hidden + 1
            hidden_i ← true
            if n_hidden = n_toHide then
                break
            end if
        end if
    end for
    for all i ∈ G do
        if hidden_i then
            continue
        else if n_hidden ≤ n_toHide then
            n_hidden ← n_hidden + 1
            visible_i ← false
        else
            visible_i ← true
        end if
    end for
end for
```

Algorithm 2 Visibility mask generation.

```
for all i ∈ instances do
    for all r ∈ predicates do
        if r(i) then
            if visible_i then
                group_i ← r
                maskValue_i ← maskValue(r)
            else
                group_i ← 0
                maskValue_i ← maskValue(0)
            end if
        end if
    end for
end for
for all v ∈ voxels do
    i ← id(v)
    maskValue_v ← maskValue_i
end for
```

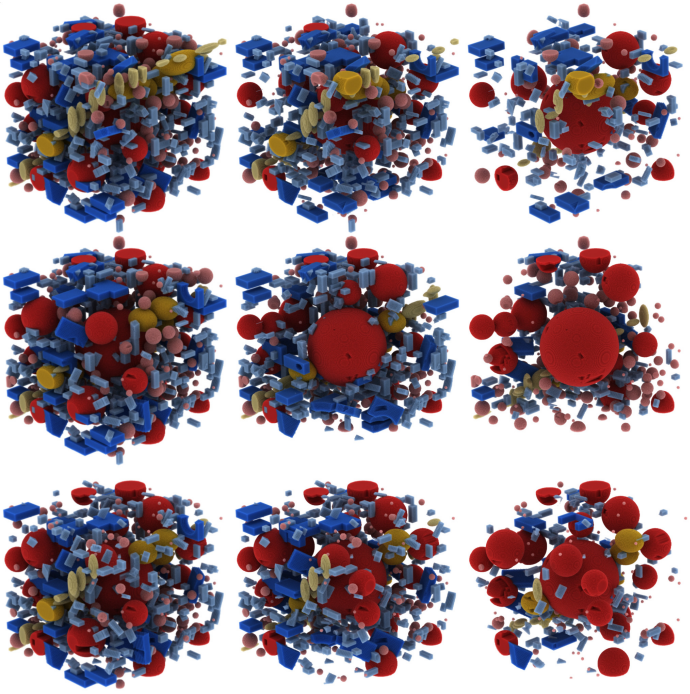

Fig. 5: From left to right: increasing degrees of uniform (top row), depth-based (middle row), and context-preserving (bottom row) sparsification.

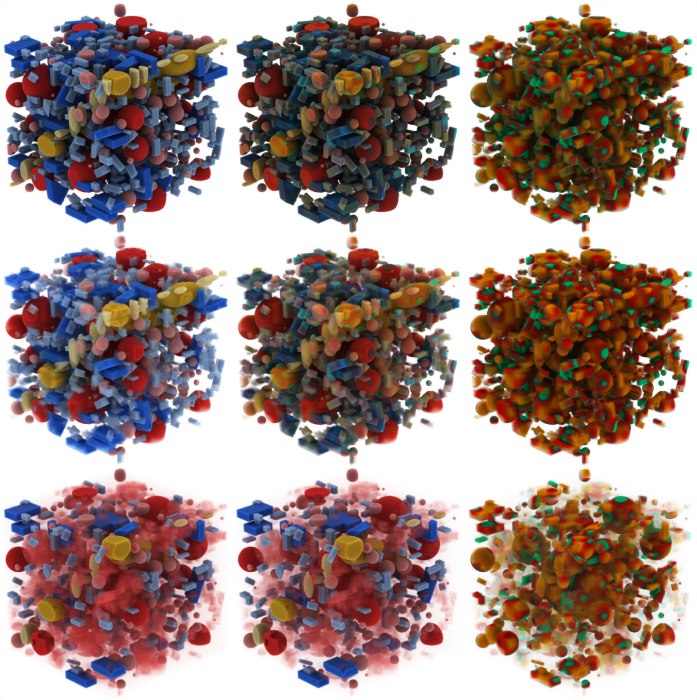

Fig. 6: Top row: different colorization as a result of blending the raw data and segmentation data transfer functions. Middle row: the same colors as above, but with the opacity transferred from the raw data transfer function, enabling the user to employ both the sparsification features of the volume conductor and the opacity set by the transfer function. Bottom row: ghosting of the instances as a result of blending the opacity after sparsification.

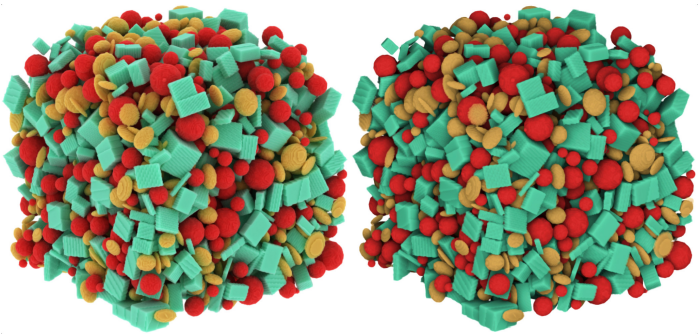

Fig. 7: Left: Rendering comparison of path tracing; Right: directional occlusion shading.