High Fidelity Visualization of Large Scale Digitally Reconstructed Brain Circuitry with Signed Distance Functions

Jonas Karlsson*, Marwan Abdellah, Sebastien Speierer,
Alessandro Foni, Samuel Lapere and Felix Shürmann†
Blue Brain Project (BBP), Ecole Polytehnique Fédérale de Lausanne (EPFL), Switzerland

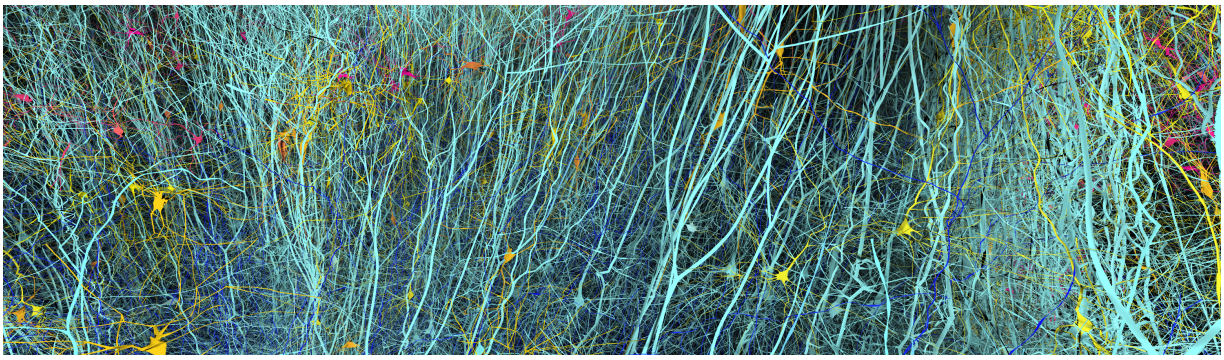


Figure 1: A close up view showing a digital circuit with thousands of neocortical neurons rendered with signed distance functions.

ABSTRACT

We explore a first proof-of-concept application for visualizing large scale digitally reconstructed brain circuitry using signed distance functions. The significance of our method is demonstrated in comparison with using *implicit geometry* that is limited to provide the natural look of neurons or *explicit geometry* that requires huge amounts of memory and has limited scalability with larger circuits.

Index Terms: Signed distance function—Neuron—Ray-marching—Visualization

1 INTRODUCTION

The recent years have witnessed a growing interest in applying visual computing methods to neuroscience research making it possible to accelerate our understanding of the brain, the most complicated phenomenon in the *known* universe. Conventional *in vivo* and *in vitro* experiments are not sufficient to unravel the complexity of the brain. A little rat brain contains nearly a hundred billion neurons and ten times more of glial cells.

The quantum leap in computing technologies was accompanied with great potentials to use the digital revolution to understand the brain [18]. Simulation-based neuroscience is introduced to complement traditional wet lab experiments, making it possible to perform computer simulations relying on digital models to test competing hypotheses about the brain [19]. Markram *et al.* have presented a first draft digital circuit reconstructed from the somatosensory cortex of a young rat [20, 25] allowing to perform a series of biological experiments in a computer simulation or *in silico*. The scale of the experiments is so huge that it is extremely challenging to build convenient tools to visualize and analyze the resulting outcomes of the simulations. Tens of Terabytes of simulations, hundreds of thousands of neuronal morphologies, millions of synapses and other kinds of large scale data that must be visualized to accomplish the mission. In this context, we present a high fidelity visualization method that is capable of rendering those circuits using signed distance functions.

*jonas.karlsson@epfl.ch
†felix.schuermann@epfl.ch

1.1 Background & Related Work

Various techniques have been used to visualize three-dimensional digital circuits of neurons starting from individual cells and up to complex circuits with several thousand or even hundreds of thousands of neurons. The most notable application, that has been used to visualize simulated electrical activity RTNeuron [15], is based on rasterization. It is implemented on top of OpenSceneGraph [21]. It can use implicit geometry (cones, spheres and cylinders) or even polygonal surface meshes created a priori with domain specific meshing libraries [16, 6, 4] . RTNeuron has added support to parallel rendering of relatively dense circuits using Equalizer [9, 8, 14]. Nevertheless, this scalability has certain limits owing to using rasterization in addition to relying on highly tessellated polygonal meshes.

Garcia *et al.* have developed a convenient method to generate adaptive meshes from morphological skeletons on-the-fly using tessellation shaders [11]. Their approach is convenient for visualizing larger circuits, but the scale of the circuit is also limited to the memory size of the GPU. Moreover, reducing the tessellation factor to visualize more neurons would significantly reduce the visual quality of the scene.

Other approaches have used volume rendering to visualize even larger circuits to simulate in silico imaging experiments [3, 2]. Unfortunately, this approach involves three expensive steps: 1) generating piecewise watertight mesh models per neuron, 2) voxelizing the membrane of each neuron using surface voxelization and 3) filling the intracellular space of the neurons using solid voxelization to ultimately reconstruct a single volume model that represents the entire neuronal circuit. Depending on the complexity and density of the circuit, this offline process might take several hours to create a huge volume with decent resolution at which we can recognize individual arbors of very thin and long axons.

Our approach uses *signed distance functions* (SDF) to visualize the circuits, enabling us to overcome the technical limitations of all previous techniques. In contrary to using explicit geometric representations (or polygonal surface meshes) where complexity

scales with memory consumption, SDF trades memory usage with a little overhead in computational complexity by evaluating a set of simple functions to render neurons. In our context, this is a desirable trade-off as we are dealing with a huge number of neurons to visualize.

1.2 Signed Distance Functions

A signed distance function (or signed distance field for its discrete form) is a mathematical function that returns the shortest distance between a given point and an implicit surface [23]. Therefore, the zero-level set of that function can be used to represent the underlying implicit surface. Such representations are very attractive since the level-of-detail of that surface will only depend on the complexity of the mathematical function. For this reason, SDF has many applications in various domains.

Fuhrmann *et al.* leverage the simplicity of collision detection with SDF to improve the performance of physically-based cloth simulation [10]. In computer vision, SDF have long been used for accurate 3D reconstructions from RGB-D data, which can be used in robotics for both mapping and planning [22]. Bærentzen uses SDF in volumetric solids manipulation and sculpting [5].

Another use of signed distance fields is found in GPU-based font rendering, where precomputed multi-channel distance fields from vector shapes can be used in real-time to efficiently reproduce the symbol shapes [12].

In the context of fractal rendering, Hart *et al.* introduced a rendering algorithm similar to ray-tracing called *ray-marching* [13]. As no close-formula could be found for computing the ray intersection with the implicit fractal surface, this method uses the SDF representation to step along the camera rays until it reaches the surface to find the intersection location.

Quilez established many of the now common distance field effects in the demo scene domain [1] . Later, modern game engines started to use SDF and ray-marching to approximate rendering effects such as soft shadows and ambient occlusion as presented in [26].

2 Our Approach

As described above, the SDF is a function which given a geometric point returns a signed distance to a surface. The sign is determined by whether the point is inside or outside of the surface. In this paper, we let a negative sign represent that the point is inside the surface. Algorithm 1 is a simple example of an SDF representing a sphere with a specific center and radius.

Algorithm 1 Sphere SDF

p = position to evaluate
c = sphere center
r = sphere radius
procedure SPHERESDF(p, c, r)
    return $length(p - c) - r$

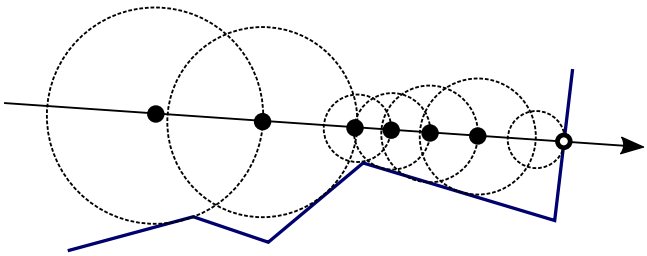


Figure 2: A conceptual diagram illustrating the ray-marching algorithm.

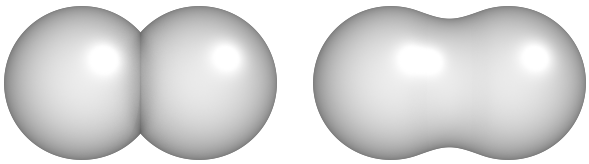
Algorithm 2 Cone pill SDF

p = position to evaluate
p_b = base position
r_b = base radius
p_t = tip position
r_t = tip radius

procedure CONEPILLSDF(p, p_b, r_b, p_t, r_t)
    $v = p_t - p_b$
    $w = p - p_b$
    $d_1 = w \cdot v$ ▷ Distance to p_b along pill axis
    if $d_1 \leq 0$ **then return** sphereSDF(p, p_b, r_b)
    $d_2 = ||v||$
    if $d_2 \leq d_1$ **then return** sphereSDF(p, p_t, r_t)
    $t = d_1/d_2$ ▷ Closest t-value along pill axis
    $p_p = p_b + t \cdot v$ ▷ Closest point on pill axis
    $t_s = 6b^5 - 15b^4 + 10x^3$ ▷ Sigmoid-shape adjusted t-value
    $r_p = lerp(r_b, r_t, t_s)$ ▷ Linear interpolated radius
    return sphereSDF(p, p_p, r_p)

The core algorithm we present uses a ray-tracing technique called ray-marching, see Figure 2. In ray-marching, the whole scene or some parts of it is defined in terms of one or several signed distance functions. Finding the intersection between the launched ray and the scene is done by evaluating the SDF at certain points along the ray. At each point the distance to the surface is given and we use this distance to step closer. By iteratively evaluating the SDF and stepping closer we can conclude within some error margin if a ray is intersecting the geometry or not. The main benefit of using ray-marching is that you can describe complex shapes using only mathematical functions. Compared to meshing, the same shape using SDF can be rendered with equivalent quality and lower memory usage. Another benefit is that it provides an infinite resolution since evaluating the SDF with high enough precision is always an option.

One important property of SDF is that since they are just mathematical functions, combining them into another function is trivial. Using this fact, we can combine two SDF using some formulas to make it appear as the two geometries are blended together. This is a very important technique used throughout this paper which we will refer to as *blending*, see Figure 3.

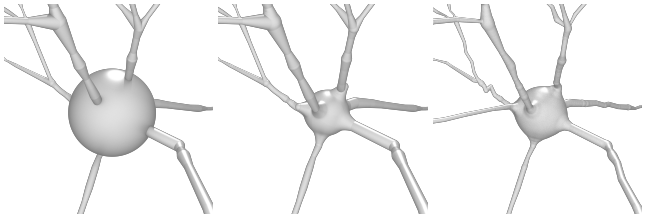


(a) (b)

Figure 3: A comparison between rendering two spheres using ray-marching without (a) and with blending (b).

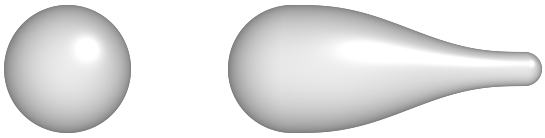
Today we use two different techniques to visualize morphologies. The first one uses implicit geometries with a sphere for the soma and cones and spheres for the segments of the arbors. The other technique uses meshes reconstructed from these implicit geometries, see Figure 4. The meshed version will look more realistic but uses much more memory.

To render the morphology using ray-marching, we use sphere and cone pill primitives represented as SDF functions as described by Algorithms 1 and 2. The cone pill is a pill with different radii at the base and tip. It also provides a variant with a sigmoid function-like shape for the radius, as shown in Figure 5.



(a) (b) (c)

Figure 4: A zoom-in of a neuron soma rendered using implicit geometry(a), SDF (b) and a triangle mesh (c).



(a) Sphere (b) Cone pill with sigmoid shape

Figure 5: The two types of primitives used, sphere and cone pill. The cone pill is the sigmoid function variant.

With these shapes we can visualize a good approximation of the morphology. If two primitives are to be evaluated and blended together using the blending function, we call these primitives *neighbours*. To improve performance and only blend connected geometries, a global buffer of the primitives' neighbours is kept.

For each primitive, two parameters are stored: 1) an index into the global buffer and 2) the number of neighbours this primitive has. If the primitive is not to be blended then it has no index into the buffer.

When evaluating a primitive, all the neighbouring primitives' functions are also evaluated and blended together as described by Algorithms 3 and 4.

Algorithm 3 Intersection algorithm

p = intersected primitive
b = intersected bounding box
r = ray

procedure INTERSECT(p, r, b)
  r_o = origin of r
  r_d = direction or r
  t_0 = t-value entry point of intersection between b and r
  t_1 = t-value exit point of intersection between b and r
  $t = t_0$
  for $i \leftarrow 0$ to $MAXSAMPLES$ **do**
    $d = blendedDistance(p, r_o + r_d * t)$
    if $d < \varepsilon$ or $t > t_1$ **then**
      break
    $t = t + d$;
  if $t > t_1$ **then return** -1
  return t

The soma is modeled using several cone pills connected with one end at the soma center and the other at the first segment of the arbors. The pill has two different radii, the base matches the average radius of the soma and the radius of the tip is set to the radius of the first segment of the connecting arbor. A sigmoid function is used for the radius evaluation to give it a more organic or realistic look, refer to Figure 4.

The arbors, including axons, apical and basal dendrites are represented as regular cone pills without any blending to maintain performance. However, the primitives part of the bifurcations of the arbors are blended together to make the branching more natural. By only blending at the soma and the bifurcations, we can reduce the needed computations significantly for efficient rendering.

Algorithm 4 Primitive blended distance algorithm

$prim$ = primitive
p = position to evaluate
f = blend factor
$distance()$ = primitive specific geometric distance function
$blend()$ = blending function

procedure BLENDED DISTANCE($prim, p$)
  $d = distance(prim, p)$;
  N = number of neighbours to $prim$
  for $i \leftarrow 0$ to N **do**
    $prim_i$ = primitive with neighbour index i
    $d_i = distance(prim_i, p)$
    $d = blend(d, d_i, f)$
  return d

3 Results & Discussion

Our algorithm was implemented in Brayns [7] using the OSPRay engine [27]. We visualized multiple digital circuits reconstructed based on the ecosystem presented by Markram *et al* [20].

The performance was benchmarked on a cluster node using a-dual socket CPU with Intel Xeon Gold 6140 CPUs running at 2.30 GHz. We loaded a single neocortical column with 0.1%, 1.0% and 10.0% of the total geometry (∼31,000 neurons with thousands of samples per neuron) using SDF, meshes and implicit geometry one at a time. To see the average performance, we rendered two views: a close-up view and a global view where the entire circuit can be seen. Moreover, two techniques have been used to render the scene: basic ray-tracing and physically-based path tracing with one sample per pixel. We believe that this combination is mandatory to assess an average performance. The comparative results are illustrated in Figure 6 including the size of each scene and the rendering time per frame.

3.1 Memory

In terms of memory requirements, implicit geometry always have the lowest overhead. Using explicit geometry requires at least five folds of memory to render the same scene. Meanwhile, SDF has slightly higher, yet comparable, memory requirements to implicit geometry. Nevertheless, it has a comparable visual quality to using the meshes as shown in Figure 7. These numbers will vary depending on the resolution of the mesh and the specific neuron but it gives a good approximation of what to expect for each geometry type.

3.2 Performance

As shown in Figure 6, implicit geometry has the best performance in all scenes. For global views (looking from far away distance), the performance of SDF becomes comparable to implicit geometry. However, it gets worse the more we render closer frames. This is likely due to the fact that the SDF geometries in the soma are more expensive to render and when they cover a bigger area of the view the performance will degrade.

We also see that at the 1% and 10% scenes the SDF starts to perform better than the meshes. This is where the overhead of using meshes starts to become a problem, likely due to the higher memory usage causing slow-downs in the BVH traversal. At this scale, using SDF becomes the only viable option for more realistic renderings owing to memory and performance issues with using meshes. With this we see that SDF fills an important gap and can

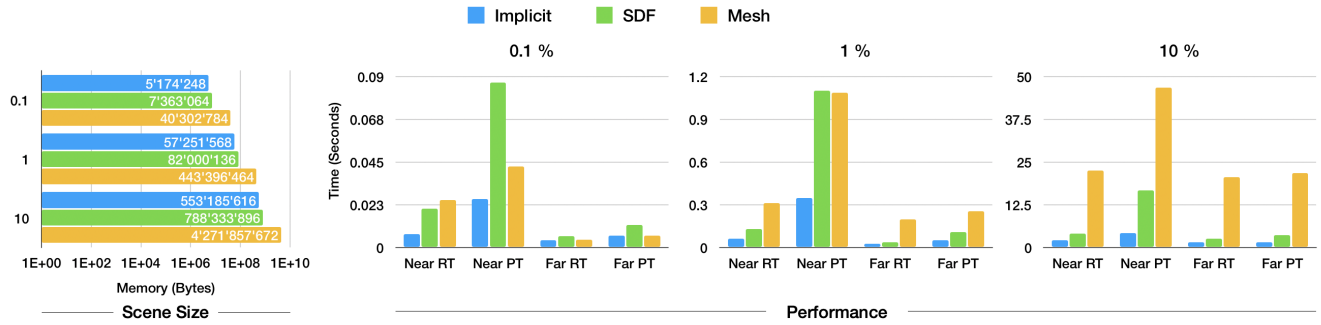


Figure 6: The performance and scene size of a single neocortical column using 0.1%, 1.0% and 10.0% of the whole data set for implicit, SDF and mesh geometry. The performance is measured using the average rendering time for a 1920*x*1080 frame. *Near* means the scene was rendered close-up and *Far* means it was rendered from a far. *RT* and *PT* means that the images are rendered with ray-tracing and path-tracing respectively.

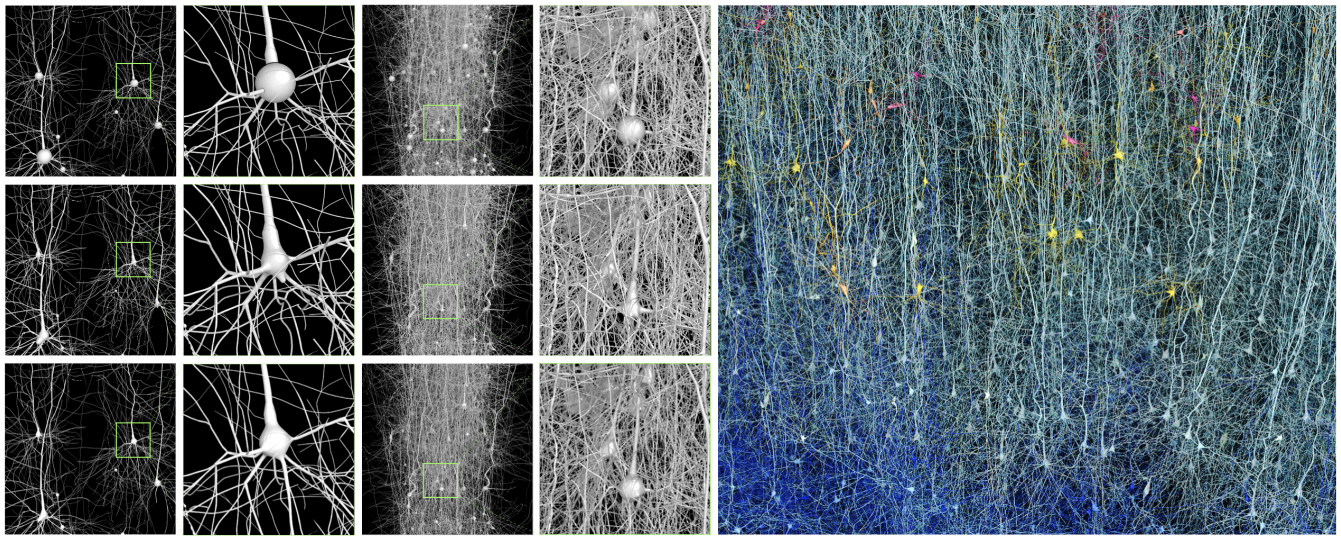


Figure 7: Rendering digital neocortical circuits with different cell densities. From the left, a circuit containing tens of neurons (and a close up on one neuron), hundreds of neurons (and a close up on a group of layer V pyramidal neurons), and a relatively large circuit containing tens of thousands of neurons. The scenes used for benchmarking (to the left) are rendered with implicit geometry (top), SDF (middle) and explicit geometry (bottom). The image on the right shows a collection of neurons color coded by their morphological types.

be considered as the best option for large scale realistic rendering of neural morphologies.

4 Conclusion & Future Work

Visualization of large scale neuronal circuits is essential for simulation-based neuroscience. It gives us the opportunity to analyze various structural and functional aspects of the outcomes of *in silico* experiments. We presented a novel approach that uses signed distance functions to render neuronal morphologies from their digital reconstructions taking into account several factors, including memory consumption, performance and visual quality. The significance of our approach is demonstrated with a comparison to using implicit geometric models, which have less memory requirements but reduced visual quality and explicit models that have huge memory requirements. Relying on this technique we were able to render extremely large scale circuits (a sampled isocortex with 200,000 neurons on a single compute with 720 Gigabyte) which were unfeasible to visualize before with the same visual quality.

We are currently working on several extensions to further improve the visual appearance of the neurons and the rendering performance. The surface of the neurons is being improved by applying a random noise function to the SDF evaluation giving it a more textured and organic look.

We are also investigating the possibility to adapt streamlines rendering [17] to create high quality branching based on bézier curves with differing radii along the curve which will give perfectly smooth arbors and branching. This would allow us to synthesize high quality connected arbors with lower memory requirements. Another idea is to slightly shorten the individual segments of the arbors at each end and apply blending functions between the preceding and following segments to make the arbors smoother. We can extend the rendering to also include synapses, which are mushroom-like objects connecting to the arbors. These could easily be modeled using SDF and blended with the connecting segments of the arbors.

In terms of performance, we are implementing our algorithm on the GPU using Nvidia's rendering engine, OptiX [24].

REFERENCES

[1] Rendering Worlds with Two Triangles with raytracing on the GPU in 4096 bytes. www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf. Accessed: 2019-06-11.

[2] M. Abdellah. In silico brain imaging physically-plausible methods for visualizing neocortical microcircuitry. Technical report, EPFL, 2017.

[3] M. Abdellah, J. Hernando, N. Antille, S. Eilemann, H. Markram, and F. Schürmann. Reconstruction and visualization of large-scale volumetric models of neocortical circuits for physically-plausible in silico optical studies. *BMC bioinformatics*, 18(10):402, 2017.

[4] M. Abdellah, J. Hernando, S. Eilemann, S. Lapere, N. Antille, H. Markram, and F. Schürmann. Neuromorphovis: a collaborative framework for analysis and visualization of neuronal morphology skeletons reconstructed from microscopy stacks. *Bioinformatics*, 34(13):i574–i582, 2018.

[5] J. A. Bærentzen. Manipulation of volumetric solids with applications to sculpting. Citeseer, 2002.

[6] J. P. Brito, S. Mata, S. Bayona, L. Pastor, J. DeFelipe, and R. Benavides-Piccione. Neuronize: a tool for building realistic neuronal cell morphologies. *Frontiers in neuroanatomy*, 7, 2013. doi: 10.3389/fnana.2013.00015

[7] Cyrille Favreau et al. Brayns: Visualizer for large-scale and interactive ray-tracing of neurons . `https://github.com/BlueBrain/Brayns`, 2019. Online; accessed 13 June 2019.

[8] S. Eilemann, A. Bilgili, M. Abdellah, J. Hernando, M. Makhinya, R. Pajarola, and F. Schürmann. Parallel rendering on hybrid multi-gpu clusters. In *Eurographics Symposium on Parallel Graphics and Visualization*, number EPFL-CONF-216016, pp. 109–117. The Eurographics Association, 2012. doi: 10.2312/EGPGV/EGPGV12/109-117

[9] S. Eilemann and R. Pajarola. *The Equalizer parallel rendering framework*. Citeseer, 2007.

[10] A. Fuhrmann, G. Sobotka, and C. Groß. Distance fields for rapid collision detection in physically based modeling. In *Proceedings of GraphiCon 2003*, pp. 58–65. Citeseer, 2003.

[11] J. J. Garcia-Cantero, J. P. Brito, S. Mata, S. Bayona, and L. Pastor. Neurotessmesh: A tool for the generation and visualization of neuron meshes and adaptive on-the-fly refinement. *Frontiers in neuroinformatics*, 11:38, 2017.

[12] C. Green. Improved alpha-tested magnification for vector textures and special effects. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pp. 9–18. ACM, New York, NY, USA, 2007. doi: 10.1145/1281500.1281665

[13] J. C. Hart, D. J. Sandin, and L. H. Kauffman. Ray tracing deterministic 3-d fractals. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '89, pp. 289–296. ACM, New York, NY, USA, 1989. doi: 10.1145/74333.74363

[14] J. Hernando, J. Biddiscombe, B. Bohara, S. Eilemann, and F. Schürmann. Practical parallel rendering of detailed neuron simulations. In *EGPGV*, pp. 49–56, 2013.

[15] J. Hernando, F. Schurmann, and L. Pastor. Towards real-time visualization of detailed neural tissue models: View frustum culling for parallel rendering. In *Biological Data Visualization (BioVis), 2012 IEEE Symposium on*, pp. 25—32, October 2012.

[16] S. Lasserre, J. Hernando, S. Hill, F. Schuermann, P. de Miguel Anasagasti, G. A. Jaoudé, and H. Markram. A neuron membrane mesh representation for visualization of electrophysiological simulations. *IEEE Transactions on Visualization and Computer Graphics*, 18(2):214–227, 2012. doi: 10.1109/TVCG.2011.55

[17] L. Li and H.-W. Shen. Image-based streamline generation and rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):630–640, 2007.

[18] H. Markram. The Blue Brain Project. *Nature Reviews Neuroscience*, 7(2):153–160, 2006. doi: 10.3389/fnana.2013.00001

[19] H. Markram. The human brain project. *Scientific American*, 306(6):50–55, 2012.

[20] H. Markram, E. Muller, S. Ramaswamy, M. W. Reimann, M. Abdellah, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–492, 2015. doi: 10.1016/j.cell.2015.09.029

[21] P. Martz. *OpenSceneGraph Quick Start Guide: a quick introduction to the cross-platform open source scene graph API*. Skew Matrix Software, 2007.

[22] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart. Signed distance fields: A natural representation for both mapping and planning. In *RSS 2016 Workshop: Geometry and Beyond - Representations, Physics, and Scene Understanding for Robotics*. University of Michigan, 2016. RSS 2016 Workshop: Geometry and Beyond - Representations, Physics, and Scene Understanding for Robotics; Conference Location: Ann Arbor, MI, USA; Conference Date: June 19, 2016. doi: 10.3929/ethz-a-010820134

[23] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer Verlag, 2003.

[24] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, et al. Optix: a general purpose ray tracing engine. In *Acm transactions on graphics (tog)*, vol. 29, p. 66. ACM, 2010.

[25] S. Ramaswamy, J.-D. Courcol, M. Abdellah, S. R. Adaszewski, N. Antille, S. Arsever, G. Atenekeng, A. Bilgili, Y. Brukau, A. Chalimourda, et al. The neocortical microcircuit collaboration portal: a resource for rat somatosensory cortex. *Frontiers in neural circuits*, 9, 2015. doi: 10.3389/fncir.2015.00044

[26] T. Stachowiak. Advances in Real-Time Rendering in Games.

[27] I. Wald, G. P. Johnson, J. Amstutz, C. Brownlee, A. Knoll, J. Jeffers, J. Günther, and P. Navrátil. Ospray-a cpu ray tracing framework for scientific visualization. *IEEE transactions on visualization and computer graphics*, 23(1):931–940, 2016.