Eurographics Conference on Visualization (EuroVis) 2019
M. Gleicher, H. Leitte, and I. Viola
(Guest Editors)

Volume 38 (2019), Number 3

Ray Tracing Generalized Tube Primitives: Method and Applications

Mengjiao Han[†1], Ingo Wald[2,3], Will Usher[1,2], Qi Wu[1,4], Feng Wang[1], Valerio Pascucci[1], Charles D. Hansen[1], and Chris R. Johnson[1]

[1]SCI Institute, University of Utah [2]Intel Corporation [3]NVIDIA [4]University of California, Davis

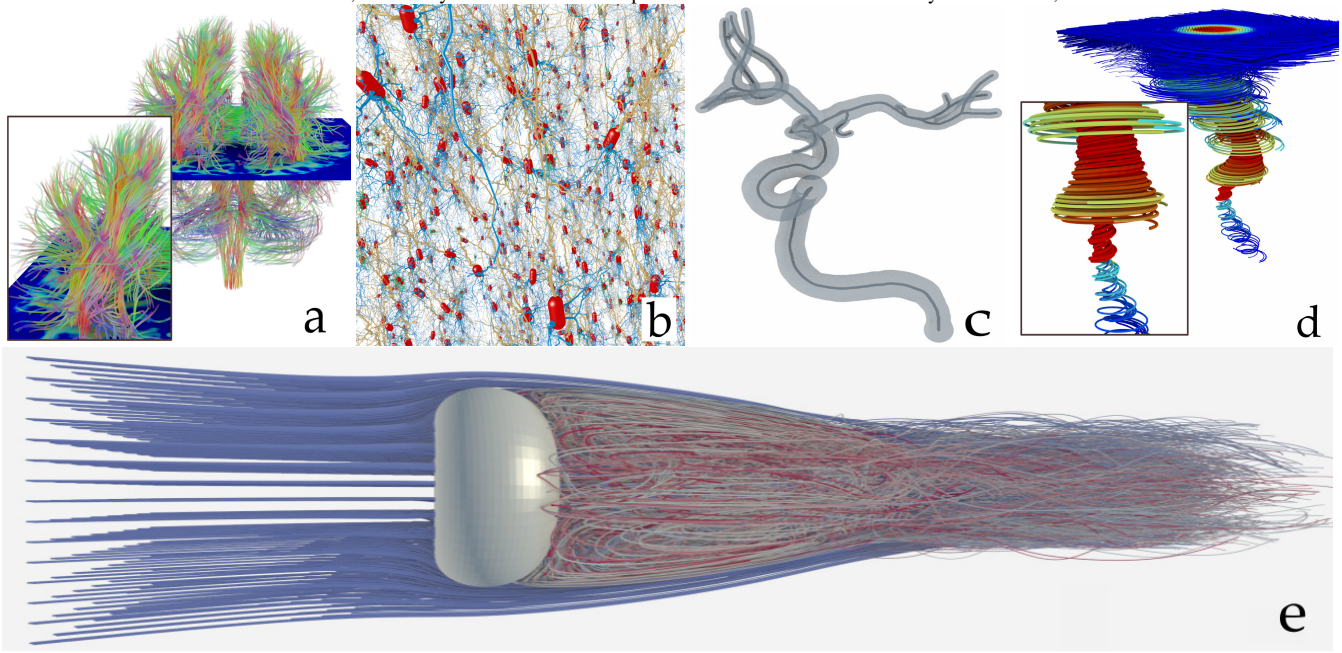


Figure 1: *Visualizations using our "generalized tube" primitives. (a): DTI tractography data, semi-transparent fixed-radius streamlines (218K line segments). (b): A generated neuron assembly test case, streamlines with varying radii and bifurcations (3.2M l. s.). (c): Aneurysm morphology, semi-transparent streamlines with varying radii and bifurcations (3.9K l. s.) and an opaque center line with fixed radius and bifurcations (3.9K l. s.). (d): A tornado simulation, with radius used to encode the velocity magnitude (3.56M l. s.). (e): Flow past a torus, fixed-radius pathlines (6.5M l. s.). Rendered at: (a) 0.38FPS, (b) 7.2FPS, (c) 0.25FPS, (d) 18.8FPS, with a 2048^2 framebuffer; (e) 23FPS with a 2048×786 framebuffer. Performance measured on a dual Intel® Xeon® E5-2640 v4 workstation, with shadows and ambient occlusion.*

Abstract

We present a general high-performance technique for ray tracing generalized tube primitives. Our technique efficiently supports tube primitives with fixed and varying radii, general acyclic graph structures with bifurcations, and correct transparency with interior surface removal. Such tube primitives are widely used in scientific visualization to represent diffusion tensor imaging tractographies, neuron morphologies, and scalar or vector fields of 3D flow. We implement our approach within the OSPRay ray tracing framework, and evaluate it on a range of interactive visualization use cases of fixed- and varying-radius streamlines, pathlines, complex neuron morphologies, and brain tractographies. Our proposed approach provides interactive, high-quality rendering, with low memory overhead.

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing;***

1. Introduction

Visualization focuses on helping scientists explore or explain data through software systems that provide static or interactive visual representations. Creating a visualization typically requires two steps: choosing the best *representation* to convey the data visually and then efficiently *rendering* this representation. Although often viewed as separate stages, the two are tightly intertwined. Constraints imposed in the second stage—particularly the primitives and model sizes supported by the rendering system—influence the choices of visual representations made in the first stage.

In this paper, we are concerned with high-performance and

† mengjiao@sci.utah.edu

high-fidelity rendering of data represented as 3D line primitives. Such line primitives are used to represent data in a range of scientific domains, such as fluid dynamics (e.g., streamlines and pathlines) [Ste00, MTHG03, STH*09, GGTH07, Mer12], medical imaging (e.g., diffusion tensor imaging) [RBE*06, MSE*06, ZDL03], and vector field visualization (e.g., magnetic or vector fields) [PVH*02, CYY*11, MCHM10]. Additional attributes can be encoded along the line by varying the line color, thickness [LMSC11], or opacity [WVDLH05, GRT13, KRW18]. This same type of geometry—long, thin lines with varying thickness—is also useful for representing other data, such as ganglions in neuron datasets [Mar06] or vessels in aneurysm visualization [SSV*14], although such data further requires the method to support acyclic graph structures.

To visualize such line primitives, much of the visualization community has focused on tessellating their surfaces and rasterizing the resulting primitives, leveraging the high triangle rasterization performance of GPUs. However, it is difficult to support transparent geometries, ambient occlusion, and global illumination effects in a rasterizer. Ray tracing provides a direct method for rendering non-polygonal geometries, such as tubes, streamlines, etc., by directly computing ray-surface intersections with the objects. A ray tracer naturally supports effects such as transparency, ambient occlusion, and global illumination, allowing for high-quality visualization.

Line primitives have been widely employed in visualization, and several open-source applications exist for ray tracing them, with varying levels of support for bifurcations, transparency, and varying radius (e.g., Embree [WWB*14], OSPRay [WJA*17], "Brayns" [Blu19]). Prior work has addressed, in part, features such as varying radii [SGS05], transparency [SZH97, ZSH96, MTHG03, KRW18], and bifurcations [TWHS05, TWSH02, TAC*13, KP17, SSV*14]. However, no single method supports all three features in combination, making the implementation of general visualization software and its use by scientists more challenging, as special purpose methods must be used for each domain.

In this paper, we explore the use of ray tracing to efficiently visualize a class of data that is best represented as 3D line primitives. We propose a new rendering primitive, the "generalized tube", that supports varying radii, bifurcations, and correct transparency, and is applicable to any ray tracer. Moreover, our technique provides high-quality interactive rendering, with low memory overhead. We implement our method as a module in the OSPRay [WJA*17] ray tracer and evaluate it on a range of datasets. Our contributions are:

- A new method for rendering 3D line primitives, the "generalized tube", supporting varying radii, bifurcations, and correct transparency;
- An efficient CSG-based intersection approach that enables our primitive to support correct transparency with interior surface removal;
- Demonstration of our approach on a range of datasets, from scalar and vector fields, to neuron morphologies and topological structures;
- Implementation of our approach as an open-source module in OSPRay [WJA*17], to allow use of in a range of visualization packages.

2. Background and Related Work

In this section, we summarize recent work on rendering 3D line primitives (Section 2.1) and related work on ray tracing non-polygonal surfaces (Section 2.2).

2.1. Rendering Line Primitives

The majority of work in visualization has focused on GPU-based approaches to render 3D line primitives. Early work by Zöckler et al. [ZSH96] proposed to render the streamlines as illuminated line primitives. Schussman and Ma [SM02] proposed self-orienting surfaces (SOS). SOS renders view-aligned triangle strips that are shaded using fixed-function illumination and bump mapping. SOS formed the basis of later imposter-based streamline and streamtube methods, where view-aligned triangle strips [PFK07] or a combination of strips and point sprites [SKH*04, MSE*06] are rasterized, and ray-cylinder and ray-sphere intersections are computed in the fragment shader. Bhagvat et al. [BJCW09] defined a conical frustum representation for line segments and rendered it via GPU ray casting of the relief-mapped frusta. Oeltze et al. [OP05] used convolution surfaces, which have varying-radius and bifurcations, to visualize vasculature. Stoll et al. [SGS05] presented an approach for rendering stylized line primitives based on imposters that is able to support varying radii of the control points. Melek et al. [MMYK06] presented an approach based on a GPU implementation of SOS for visualizing neuronal fibers. Kanzler et al. [KRW18] recently proposed a voxel-based GPU ray-casting method for rendering 3D line primitives with transparency, shadows, and ambient occlusion. However, as the approach is based on re-sampling the data to a grid, the resulting line quality is inherently dependent on the chosen grid resolution. Eichelbaum et al. [EHS13] presented an improved 3D line rendering approach to enhance structural perception by providing a novel ambient occlusion method. Recent work by Lindow et al. [LBLH19] proposed a hybrid rasterization and raycasting approach for ribbon and stick rendering of DNA and RNA.

Although domain-specific tools exist that support efficient methods for rendering streamlines [BSG*09, GKM*15], off-the-shelf visualization tools, such as ParaView [Aya15] and VisIt [CBW*12], default to tessellating them. For example, in the visualization toolkit (VTK) [SLM04], the default method for rendering streamlines is to tessellate them. Similarly, in the field of neuroscience, we are aware of at least one major project that originally rendered large neuron datasets by tessellating them [BMB*13], and dealt with the large number of triangles produced using parallel rendering [Eil13]. However, as dataset size grows, tessellation can require the use of numerous powerful GPUs to fit the data in memory and achieve interactive framerates.

2.2. Ray Tracing Non-Polygonal Primitives

Parker et al. [PSL*98] proposed one of the first interactive applications of ray tracing non-polygonal primitives to visualize implicit isosurfaces. Following this work, a large body of visualization research has explored ray tracing for rendering non-polygonal or implicit geometry [DPH*03, GIK*07, BPL*12, KWN*13, WKJ*15, WKI*17]. Today, the most common applications of ray tracing non-polygonal primitives are the rendering of spheres to represent

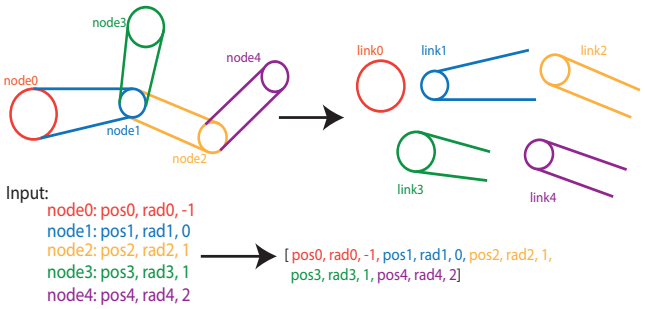


Figure 2: *Illustration of the input data structure. We make a list of control points, each with a position, radius, and predecessor index. Each control point and its cylinder or cone stump connection to its predecessor is refered to as a "link".*

particle data [GIK*07, WKJ*15] and combinations of spheres and cylinders for ball-and-stick models [KWN*13, Sto98] or streamlines [WJA*17].

OSPRay's current streamline geometry [WJA*17] is implemented as a combination of sphere primitives linked together with cylinders. This approach is simple to implement in a ray tracer and produces high-quality images for opaque, fixed radius streamlines. However, this method inherently lacks support for varying radii along the streamline and does not support transparency or bifurcations.

Favreau's "Brayns" ray tracer [Blu19] employs a combination of sphere, cylinder, and cone stump primitives in a manner similar to our own for interactive ray tracing of large neuron assemblies. Our work, although developed independently, has been motivated by similar challenges when visualizing such large-scale neuron data.

Outside visualization, the most common application of ray tracing non-polygonal surfaces, is found in movie rendering, in particular for memory-efficient rendering of subdivision surfaces [BBLW07, BWN*15], hair [WBW*14], and curve or ribbon primitives [BK85]. Recently, Embree [WWB*14] has introduced support for curves with varying radii by adding support for varying-radii features to their Bézier and B-Spline curve primitives. These primitives have also been made available in OSPRay, which builds on top of Embree. Such curves are visually pleasing, but they are expensive to render and do not support bifurcations or varying radii.

3. Method Overview

We represent our generalized tubes with a combination of spheres to represent the control points, cylinders for fixed-radius links, and cone stumps for varying-radii links. In the following sections, we describe the input data structure to specify these primitives (Section 3.1) and how we compute the appropriate spheres, cylinders, and cone stumps to represent the tubes (Section 3.2).

3.1. Input Data Structure

Although more general representations of lines or tubes are possible, for the purposes of this work we consider only input data in the form of linearly connected control points. The input data is specified as a list of control points, each with a position and radius, along with a connectivity attribute, which specifies how the control points are connected (Figure 2). For the sake of simplicity, we consider only acyclic graphs, where each point can have at most one predecessor. Although simple, we have found this input structure sufficient to represent all the datasets used in our evaluation. We note that a generalization to cyclic graphs is straightforward.

With these assumptions, we can view our input as being simply a set of what we call "links". Each link specifies a control point and a reference to the control point preceding it, or "-1" if the link is the starting point of the streamline. Bifurcations are then simply cases where two links connect to the same predecessor. Figure 2 shows an illustration of a set of tube primitives with constant and varying-radii links and a bifurcation.

Depending on the application domain, it sometimes makes sense to talk *logically* about entire segments of links (e.g., an entire ganglion in a neuron, a particle trace). However, as each such logical segment can be reduced to a series of links, we leave this higher level semantic information to the application, and from the point of a ray tracer consider only individual links.

3.2. Choice of Representation

Given this input data structure, the next step toward rendering it in a ray tracer is to break it up into smaller geometric primitives, for which ray-surface intersections can be more easily formulated.

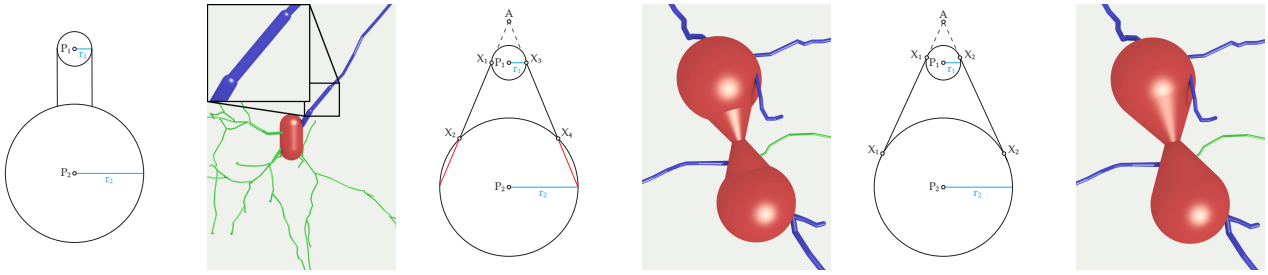
In OSPRay's current implementation of streamlines with a fixed radius, each control point is internally represented as a sphere and the links between points as cylinders. The cylinder composes the bulk of the streamline, and the spheres round off the corners where two cylinders meet. As all the radii are the same, these primitives will always fit perfectly together, creating the appearance of a single connected streamline. Implementing this approach is straightforward: ray-sphere and ray-cylinder intersections are well described in the literature [Dra99, PJH16], and building an acceleration structure on these primitives can be left to Embree.

For our generalized tube primitives, we follow a similar approach; however, properly handling the varying radii of the control points requires some modifications, as illustrated in Figure 3.

3.2.1. Linking with Cylinders and Naïve Cone Stumps

To solve the problem of choosing correct representation, we compared two existing approaches first: the existing OSPRay's [WJA*17] streamlines and connected cones from the Blue Brain project [Mar06]. The first prototype is a trivial extension of OSPRay's [WJA*17] existing streamlines, where we simply chose the cylinder's radius to be that of the smaller control point. This approach prevented any holes from appearing, but the images produced were quickly judged unacceptable (Figure 3a). Clearly, the proper geometric primitive to linearly connect two spheres with different radii is a cone stump, not a cylinder. Similar to "Brayns" [Blu19], we next computed cone stumps linking the control points, whose caps were centered at P_1 and P_2, with radius r_1 and r_2, respectively, oriented along $\vec{P_1P_2} = P_2 - P_1$ (Figure 3b).

Although this naïve way of computing the cones gives acceptable results in many cases, it produces noticeable banding artifacts in sections where the radius changes rapidly (Figure 3b). Similar to



(a) *Cylinder links produce clear discontinuities.* **(b)** *Naïve cone links result in banding artifacts.* **(c)** *We compute correct tangent cone links.*

Figure 3: *When linking control points of varying radii, cylinders are clearly the wrong choice (a); however, incorrectly chosen cones will also produce artifacts (b). To smoothly link the control points, we compute cones that are tangent to the spheres at their intersection (c).*

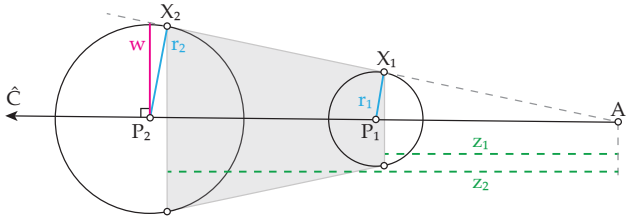


Figure 4: *Our method for computing a tangent cone stump to connect control points of varying radii.*

sweeping a sphere along a trajectory [VW85], the *real* shape that linearly connects two spheres is a slightly different cone stump than the one produced using the computation described above. Specifically, the naïve cone is not tangent to the sphere where the two meet (X_i's in Figure 3b). As a result, the larger sphere protrudes through the cone stump, and at the thinner end there is a visible, sharp change in surface curvature.

3.2.2. Computing Properly Tangential Cones

The desired cone, which smoothly connects the control points—the one tangential to the spheres at the points X_i—is shown in Figure 3c. A cone is described by its apex (A), orientation ($\hat{C}$), and radius (w). To clip the infinite cone to a cone stump, we will also require the clipping plane locations z_1 and z_2 along the axis of revolution $\hat{C}$. An illustration of the tangential cone computation is given in Figure 4. Our computation is somewhat similar to the silhouette computation of Gumhold [Gum03], although differs in the properties we require in the end, and thus we include it for completeness. The cone's orientation is given by

$$\hat{C} = \frac{P_2 - P_1}{||P_2 - P_1||} \tag{1}$$

Defining $p_1 = ||P_1 - A||$ and $p_2 = ||P_2 - A||$, we find from the theorem of intersecting tubes that

$$\frac{r_2}{r_1} = \frac{p_2}{p_1}$$

Substituting $p_2 = ||P_2 - P_1|| + p_1$, we can solve for p_1

$$\frac{r_2}{r_1} = \frac{||P_2 - P_1|| + p_1}{p_1}$$
$$p_1 = ||P_2 - P_1|| \frac{r_1}{r_2 - r_1}$$

Thus, we find the apex at

$$A = P_1 - p_1 \hat{C} \tag{2}$$

Next, we compute the locations of the clipping planes z_1 and z_2. Due to congruence and the theorem of intersecting lines, we know that

$$\frac{p_1 - z_1}{r_1} = \frac{r_1}{p_1}$$

which we solve for z_1.

$$z_1 = p_1 - \frac{r_1^2}{p_1}$$

We proceed similarly for the second clipping plane location z_2.

$$z_2 = p_2 - \frac{r_2^2}{p_2}$$

Finally, to compute the width of the cone at P_2, we first define $x_2 = ||X_2 - A||$. From the Pythagorean theorem, $x_2 = \sqrt{p_2^2 - r_2^2}$. Again, using the theorem of intersecting lines we can find w.

$$\frac{x_2}{p_2} = \frac{r_2}{w}$$
$$w = \frac{p_2 r_2}{x_2}$$

Once the modified cone stump's coordinates are known, we can compute ray-cone stump intersections (Section 4.1). The intersection computation is the same as for a naïve cone stump; the only difference is in the cone parameters. Our approach links the geometry correctly, though does not ensure the normals are continuous where the cone and sphere meet (Figure 3c).

4. Implementation

We represent the control points as spheres and link them with either cylinders or cone stumps. When the control points have the same radii, it is sufficient to link them with a cylinder; however, if the radii differ, we must use a cone stump. Ray-sphere, ray-cylinder, and ray-cone intersections are well described in the ray tracing literature [PJH16, Dra99]. Bhagvat et al. [BJCW09] presented a similar approach for ray-conical frusta intersection; however, our definition of a cone *stump* is not identical to a conical frusta. As an understanding of this operation is key to reproduce this paper, we briefly summarize our ray-cone *stump* intersection.

4.1. Ray-Cone Stump Intersection

Following Dodgson's discussion [Dra99], we consider the infinite dual-sided cone of which our cone stump is a part, and construct a

transformation that transforms this cone into the unit cone, with the apex at the origin, the z-axis as the axis of rotation, and a slope of 1. To do this, we compute the position of the non-truncated cone's apex A (Equation (2)) and an orthonormal basis $\hat{v_x}, \hat{v_y}, \hat{v_z}$ that transforms $\hat{z}$ to $\hat{C}$. The vectors $\hat{v_x}$ and $\hat{v_y}$ are then scaled by w/p_2, to span the larger cap and transform the cone to one with slope 1. The matrix M that transforms our cone stump to the unit coordinate system is thus given by Equation (3).

$$M = \begin{bmatrix} \frac{w}{p_2}\hat{v_x} & \frac{w}{p_2}\hat{v_y} & \hat{C} & A \end{bmatrix}^{-1} \tag{3}$$

This unit coordinate system places the larger cap at $z = 1$ by design, whereas the smaller cap position is found by

$$z_{\text{cap}} = \frac{z_1}{z_2}$$

We can now see our cone stump as the intersection of the slab $[z = z_{\text{cap}}, z = 1]$ with the infinite unit cone $X^2 + Y^2 = Z^2$, and we can formulate our ray-cone stump intersection accordingly. Given a ray $r(t) = o + t\hat{d}$, we transform the ray into the cone's coordinate system by applying M^{-1}, yielding $r'(t)$. We can then insert the transformed ray into the unit cone equation and solve the resulting quadratic. Solving this quadratic yields the (possibly empty) interval $[t_{c0}, t_{c1}]$ where the ray intersects the unit cone. If this interval is empty, or outside the valid ray interval $[t_{r0}, t_{r1}]$, there is no intersection and we can exit.

If an intersection with the infinite unit cone is found, we then compute the interval $[t_{z0}, t_{z1}]$ where the ray overlaps the slab $[z = z_{\text{cap}}, z = 1]$. This ray-slab interval is then intersected with the previously computed ray-cone interval to find $[t_{s0}, t_{s1}]$, which is the interval where the ray overlaps the cone stump.

Given the ray-cone stump interval $[t_{s0}, t_{s1}]$, the final step depends on what exactly we need. In Section 4.3.1, we will need the actual overlap interval between the ray and the cone stump, which is the intersection of the ray-cone stump interval $[t_{s0}, t_{s1}]$ and the valid ray interval, $[t_{r0}, t_{r1}]$. If we are interested only in finding the ray's intersection with the cone stump's surface, we need only the nearest of $[t_{s0}, t_{s1}]$, which is also inside the valid ray interval.

4.2. Acceleration Data Structure and Primitive Type

We use Embree [WWB*14] for the acceleration structure and traversal kernels. How we use Embree to build a bounding volume hierarchy (BVH) over our primitives can significantly influence performance and/or memory consumption, we discuss a few options and their trade-offs in the following sections.

4.2.1. Individual Primitives vs. Complete Links

The first choice is whether we build our Embree BVH over the individual link components (i.e., the spheres, cylinders, and cone stumps), or over logical "link" primitives, which would then internally perform intersections with their components. In the former case, we can implement three separate Embree geometries (one for spheres, one for cylinders, and one for cone stumps) and have dedicated intersection routines for each. Embree will then automatically build a single BVH over the different primitives. In the latter case, we have a single Embree geometry with a much more complex intersection routine. The first approach could result in a poorer quality BVH, with more BVH nodes and overlap between them, increasing both memory use and traversal cost compared to the latter. However, in the case of long, thin links with less overlap, it is likely that most rays will intersect only the cylinder or cone primitives, resulting in potentially higher performance in the first approach, compared to the latter's more costly primitive intersection.

The trade-offs between these two options are multi-faceted and non-obvious, and can be concluded only by an experiment, which we conduct in Section 6.1.

4.2.2. Precomputed vs. On-the-Fly Primitives

A second important choice is how much information we are going to pre-compute for the primitives. On one extreme, we can keep memory consumption low by not pre-computing anything, in which case we can describe each link by as little as a pointer to its control points; all other data—cone parameters, transformation matrices, etc.—can be computed on the fly for every intersection test.

At the other extreme, we could conclude that re-doing these computations millions of times per image is a waste, and could pre-compute the cone coordinates and/or up to two transformation matrices (the ray to object and object to world transforms) and store these pre-computed attributes with the primitives. This is a clear memory-vs-speed trade-off, which we will quantify with experiments in Section 6.1.

4.2.3. Embree Integration

Regardless of the final implementation we choose based on the experiments, our Embree integration is the same. To allow Embree to build a BVH over our primitives and intersect rays with them, we need to provide two methods for each primitive type. The first method computes the bounds of the primitive, and the second intersects a ray with the primitive. Depending on our choice of implementation, these primitives will be the individual spheres, cylinders, and cone stumps, or the entire links.

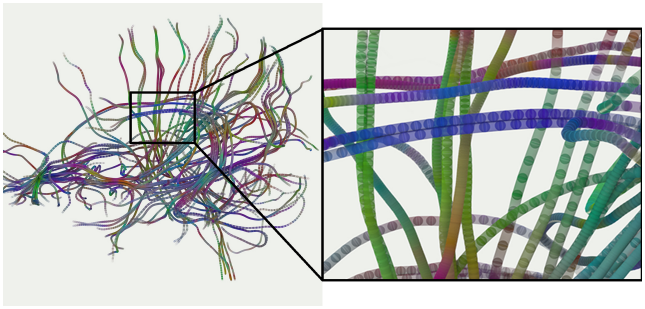
4.3. Transparency

Our current description of our generalized tubes can readily be used to render opaque lines with both bifurcations and varying radii, which were lacking in prior work. However, a third limitation of prior work also applies to our description so far—artifacts when rendering with transparency (Figure 5a).

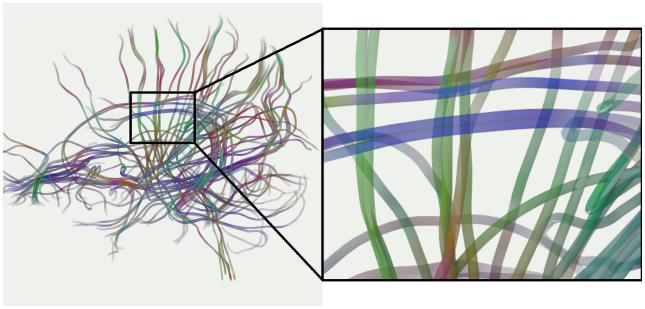
These artifacts result from the fact that, whereas logically we want our tubes primitive to be what in constructive solid geometry (CSG) terms would be called the *union* of the base primitives, we have actually implemented them as the *sum* of these primitives, resulting in interior surfaces. Therefore, a naïve approach to transparency will find and shade intersections with these interior surfaces as well, producing visible artifacts.

4.3.1. Removing Interior Surfaces via CSG Intersection

The simplest approach to remove these interior surfaces is to borrow ideas from constructive solid geometry, and properly treat our



(a) *Interior surfaces shining through.*



(b) *Our method removes interior surfaces correctly.*

Figure 5: *(a) Without our CSG interior surface removal approach, interior surfaces can be seen, producing visual artifacts. (b) Our CSG intersection computation correctly finds only exterior surfaces*

geometry as a *union* of the base primitives. Rather than finding the closest ray-surface intersection with any base primitive, we can instead compute all the intervals where the ray overlaps each primitive. We can then sort these intervals and traverse them front to back, counting the number of entry and exit events.

This incremental entry and exit counting tells us, at any point along the ray, how many of these intervals we are currently overlapping. Each time we transition from 0 to 1, we are entering the object, and at each transition from 1 to 0, we are exiting. All other transitions are interior surfaces and can be ignored. Note that to handle the case where rays start inside a tube, we must modify the ray start interval and set $t_{r0} = -\infty$ before intersecting the primitives.

4.3.2. Implementation via Intersection Filters

At first, Embree seems badly suited to this operation: like most ray tracers, it is primarily targeted at first- and any-hit ray traversal. However, Embree also supports so-called "intersection filters", which can be used to implement multi-hit ray traversal [AGGW15, GWA16]. Using an intersection filter, we can implement exactly the algorithm described above.

In Embree, an intersection filter is a callback function that is called after each ray-primitive intersection is encountered. The intersection filter can then decide whether to accept or reject the hit and modify additional per-ray data. To implement the algorithm described above, each time Embree calls our intersection filter we compute the ray-primitive overlap interval and store it in an auxiliary buffer attached to each ray. We then reject the hit to force Embree to discard the intersection and continue traversal, eventually iterating through all the primitives overlapped by the ray.

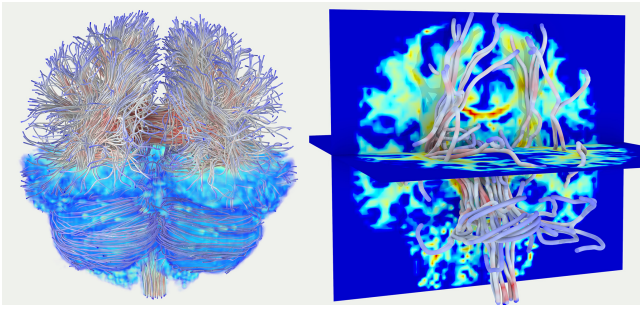


Figure 6: *Our geometry module integrated into OSPRay can be combined with volumes (left, 9.4 FPS) or other geometry (right, 22.8 FPS) to create interactive, high-quality visualizations.*

Some care must be taken when implementing this approach within OSPRay, as we want to apply the intersection filter only to our tube primitives. To achieve this, our OSPRay geometry internally builds a separate Embree scene over the base tube primitives and applies our intersection filter to this scene. Our OSPRay geometry then reports the Embree scene bounds to OSPRay as its bounds, and in its intersection method forwards the ray on to traverse its Embree scene and collects the ray intervals. After the ray intervals have been collected, they are sorted and the closest exterior surface is found and returned as the hit point.

This method can correctly remove interior surfaces from being reported incorrectly as hits, and can therefore handle transparency correctly (Figure 5b). However, this method comes at significant cost, due to the overhead in finding, storing, and sorting the ray-primitive intervals, along with the partial loss of early ray termination, as we must now find all intervals along the ray. We quantify this performance impact in Section 6.3.

5. Applications

In Figure 1, we show several sample visualization applications enabled by our module within OSPRay, ranging from DTI tractography, flow visualization, and vessel morphology to large-scale neuron assemblies. Our method can provide high-fidelity results at interactive framerates. Figure 6 shows the DTI tractography dataset in different visualization use cases. On the left in Figure 6, the full set of tracts is shown in the context of the underlying DWI volume to provide an overview visualization. On the right in Figure 6, a sub-set of the tracts is shown along with two slices of the DWI volume to focus on a specific region of the brain. Both visualizations are rendered with OSPRay's *scivis* renderer, which can render combined volumetric and surface data with high-quality shading effects such as shadows and ambient occlusion.

Figure 7 shows an illustrative visualization of neuron activity, similar to those used by the Blue Brain Project [Mar06], rendered with OSPRay's *path tracer* renderer. An emissive material is applied to the neurons to indicate the firing of electrical signals throughout the assembly.

6. Experiments and Results

We first quantify the different implementation choices discussed in Section 4.2 with a set of benchmarks to find a suitable default

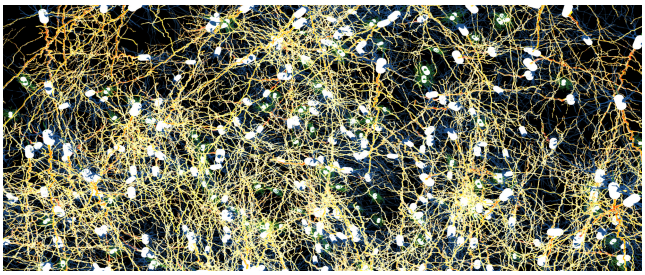


Figure 7: *An illustrative visualization of neuron activity rendered using OSPRay's* path tracer *with emissive materials.*

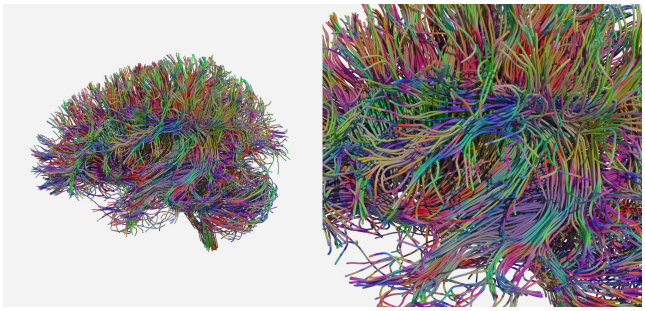


Figure 8: *The far and near views used for benchmarks on the DTI dataset, with ambient occlusion and shadows.*

implementation (Section 6.1). We then focus our evaluation on two key aspects of our method: the absolute performance achieved when rendering opaque geometry (Section 6.2) and the impact of the CSG interior surface removal method (Section 6.3). Finally, we compare the performance, rendering quality, and memory consumption of our method against Embree's existing curve primitive (Section 6.4).

Our evaluations are done using our method implemented as a module within OSPRay 1.7.2, built with Embree 3.2.0 and ISPC 1.9.1. We ran our benchmarks on three machines, *Desktop*, with an Intel® i7-5930K CPU (12 logical cores at 3.7 GHz) and 32GB RAM; *Workstation*, a dual socket workstation with two Intel® Xeon® E5-2640 v4 CPUs (40 logical cores at 2.4 GHz) and 128GB RAM; and *FSM*, a quad socket workstation with four Xeon E7-8890 v3 CPUs (144 logical cores at 2.5 GHz) and 3TB RAM.

We conducted our benchmarks on four representative datasets at varying levels of model complexity to evaluate typical use cases of our generalized tubes. The first is a diffusion tensor imaging (DTI) tractography dataset [WTBJ19] consisting of 220,711 nodes and 218,637 cylinder links with a fixed radius (Figure 8).

The second dataset is a representative model of the neuron assemblies used in neuron simulations, such as those of the Blue Brain Project [Mar06]. To generate these models, we wrote a tool that creates an assembly of neurons by placing N randomly or manually chosen base neurons (Figures 9a to 9d) at random locations within a properly scaled bounding box. Using the assembly generation program, we created datasets ranging in size from 4^3 to 20^3 neurons (far view: Figures 9e to 9g; near view: Figure 1b), in total consisting of 28,032 spheres, 2,496 cones, and 25,472 cylinders; up to 9.4M spheres, 2.4M cones, and 7M cylinders. To provide an accurate representation of this data, where each neuron is unique, we do not use OSPRay's instancing features, and instead render actual transformed copies of the base neurons.

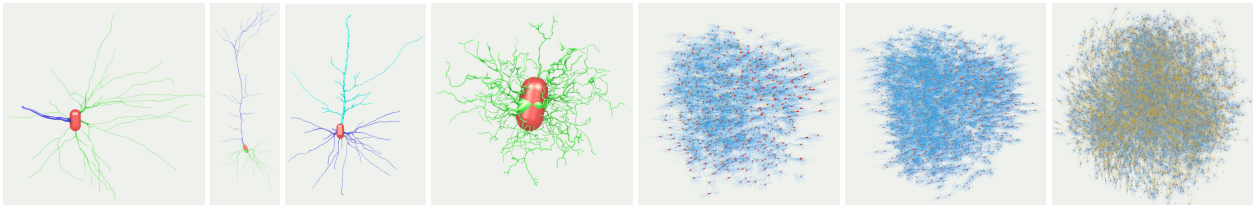
The third dataset consists of different sub-sets of pathlines extracted from a tornado simulation (Figure 1d). The first sub-set, "Tornado 1M", consists of 4096 pathlines and 947,872 fixed radius links. The second sub-set, "Tornado 6.5M", consists of 24,576 pathlines and 6.5M links, where we encode the velocity using the pathline radius. The third sub-set, "Tornado 35.9M", consists of 0.13M lines with 35.9M fixed-radius links. The last dataset used for benchmarking is the Torus Flow simulation (Figure 1e), consisting of 263,144 pathlines with 6.5M fixed-radius links. This range of datasets captures a variety of use cases for pathlines in practice. The DTI, Torus Flow, and Tornado data is represented with a dense distribution of long, thick lines; the neuron assemblies contain almost random, bifurcating, and highly intersecting lines with varying radii. On the DTI and Tornado datasets, we also use the line radius to encode additional attributes, such as fractional anisotropy (FA), on the DTI data, and velocity, on the Tornado 6.5M sub-set.

In the evaluation, we benchmark rendering performance using three renderers in OSPRay: the *ray casting* renderer is a basic primary ray-only renderer; the *scivis* renderer computes common secondary effects useful in scientific visualization (e.g., ambient occlusion and shadows); and the *path tracing* renderer is a photorealistic global illumination renderer. We render with one sample per pixel with all the renderers and use OSPRay's progressive refinment to refine the image. We configure the *scivis* renderer to take one sample for ambient occlusion when shading. Unless otherwise specified, benchmarks were run on the *Workstation* with a 1024×1024 framebuffer.

6.1. Quantification of Implementation Choices

In this section, we quantify the trade-offs of the different implementation choices discussed in Section 4.2 on six datasets. In addition to the Brain DTI tractographies and neuron assemblies (10^3, 14^3 and 20^3), we also evaluate the Tornado 1M dataset and the Torus Flow. We evaluate the four possible implementation choices discussed in Section 4.2: (a) separate sphere, cylinder, and cone stump primitives with on-the-fly transform computations; (b) separate primitives as in (a), but this time with the transforms pre-computed; (c) combined link primitives with on-the-fly transform computations; and (d) combined link primitives with pre-computed transforms.

Table 1 shows the performance and memory consumption for each option. As expected, the overall performance and memory consumption of (b) are higher than those of (a), due to pre-computing and storing the transformation matrices of the primitives, thereby avoiding redundant computation. Interestingly, the performance difference between (a) and (b) is not as large on datasets with a denser distribution of pathlines (e.g., the Torus and neuron assemblies). In these datasets, although we pre-compute transformation matrices for all primitives, we are likely intersecting only a small sub-set of them, given our fixed viewpoint. Similar results are seen when comparing on the fly vs. pre-computation on the combined link primitives. We find option (d) provides better performance at the cost of more memory use than (c) for most datasets; again, the performance difference becomes smaller on the denser datasets.



(a) (b) (c) (d) (e) (f) (g)

Figure 9: *(a-d) The base neurons used to build the neuron assembly benchmark scenes, from NeuroMorpho.org [ADH07]. The base neurons consist of: (a) 438 nodes, 39 cone links, and 398 cylinder links [JSP*01]; (b) 1176 nodes, 645 cone links, and 530 cylinder links [AA09]; (c) 2140 nodes, 320 cone links, and 1819 cylinder links [KP17]; (d) 955 nodes, 206 cone links, and 748 cylinder links [VPRK02]. (e-g) Examples of the generated neuron assemblies used in the benchmarks, rendered interactively with ambient occlusion. The assemblies are generated by randomly placing the base neurons N times within a scaled box. The assemblies have: (e) 10^3, (f) 14^3, and (g) 20^3 neurons.*

Table 1: *Performance and memory use comparison of the four implementation choices, shown as FPS / MB, benchmarked with the* scivis *renderer. We find that option (d) provides the best balance of performance and memory use.*

Implementation	DTI ($r = 0.25$)	Tornado 1M	Torus Flow	10^3 neurons	14^3 neurons	20^3 neurons
(a) separate, on-the-fly	32.3 / 120.3	8.0 / 292.9	56.2 / 1861.9	22.8 / 166.8	14.3 / 374.0	0.9 / 2666.0
(b) separate, pre-computed	37.7 / 163.4	9.2 / 485.5	56.3 / 3189.0	27.4 / 257.4	17.1 / 618.0	1.2 / 4584.4
(c) combined, on-the-fly	34.0 / 99	9.0 / 197.0	52.7 / 1102.0	22.7 / 134.3	14.1 / 235.2	0.9 / 1533.0
(d) combined, pre-computed	43.1 / 122.9	11.5 / 297.3	67.3 / 1797.6	27.8 / 184.8	17.1 / 355.0	1.2 / 2534.2

When comparing the separate primitive options (a, b) with the combined link primitive options (c, d), we find that the combined links have lower memory consumption and tend to have better rendering performance. The combined link primitives reduce memory use by sharing the control point data among the sphere and cone or cylinder primitives, and also reduce the total number of primitives Embree must build the BVH over, potentially leading to a shallower BVH with fewer nodes. With the combined link primitive, we find performance improvements on sparser data (DTI, Tornado) and the Torus. On these datasets, the individual link primitives are relatively short, and thus the rays are likely to intersect both the cylinder or cone stump link and the sphere for the control point. However, we find less performance improvement of the completed link primitives on the neuron assemblies. On the neuron assemblies, the individual links are longer, and therefore rays are more likely to require traversing only the cylinders or cone stumps. Overall, we find that (d), combined link primitives with pre-computed transformation matrices, provides the best memory-performance trade-off, and we use this implementation throughout the rest of the benchmarks.

6.2. Performance on Opaque Geometry

To evaluate overall performance and how our primitive scales with the model configuration and complexity, we examine the effect on performance of several single neuron morphologies, the DTI tractography data with several different radii, and increasing the number of neurons in the neuron assemblies.

We find that our method achieves high framerates when rendering small to medium datasets, such as the neuron morphologies and DTI tractographies, on a typical desktop system (Table 2). On the DTI tractography data we render at multiple radii and observe that for opaque geometry increasing the radius improves the performance. With very thin lines the rays must traverse further through the data, whereas thicker lines lead to more occlusion and thus require less traversal to find a hit. We find that even for the most expensive rendering method evaluated, path tracing, we still achieve interactive framerates.

Table 2: *Performance on the* Desktop *with a* 1024^2 *framebuffer.*

Dataset	Frame Rate (FPS)		
	Ray Casting	SciVis	Path Tracing
Neuron (a)	94.9	90.0	47.7
Neuron (b)	118.8	111.0	76.2
Neuron (c)	107.9	95.4	66.5
Neuron (d)	87.3	52.2	15.6
DTI ($r = 0.05$mm)	37.8	13.1	2.1
DTI ($r = 0.15$mm)	44.7	16.6	2.3
DTI ($r = 0.30$mm)	50.6	16.9	2.8

We benchmark the neuron assemblies from a viewpoint that displays the entire assembly (Figures 9e to 9g) on the *Workstation* at a 1024×1024 framebuffer (Figure 10). Even for extremely large neuron assemblies, our method is able to provide interactive rendering at high quality, achieving 41FPS on the 14^3 neuron assembly with the ambient occlusion renderer. When employing the most expensive rendering method, path tracing, we still reach 7FPS on the 14^3 assembly. Finally, we perform a large-scale stress test and generate a neuron assembly with 1 billion links. Our method remains interactive even at a 2400×600 framebuffer, achieving 22.5FPS on *FSM*.

6.2.1. Comparison to Tessellation

To perform a rough comparison between our method and the tessellation approach, which is similar to the approach that is commonly employed in tools such as VTK and ParaView, we create triangulated models of our data. These models are created by tessellating the sphere, cylinder, and cone stump primitives into 960, 124, and 124 triangles, respectively. Although this coarse tessellation leaves some gaps at the connections between the primitives,

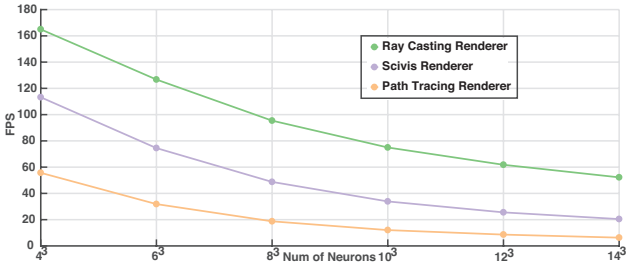


Figure 10: *Rendering performance on the generated neuron assemblies (Figure 9). Our method performs well even at large scales (14^3, 1.2M links) with ambient occlusion.*

Table 3: *Triangulated models (Triangles) compared to our non-polygonal generalized tubes (GT) on the* Workstation *(top) and* FSM *(bottom). $*$ indicates out of memory. GT consumes far less memory and provides higher framerates.*

	Memory Use (GB)		Framerate (FPS)	
Dataset	Triangles	GT	Triangles	GT
DTI	35.4	0.13	38.9	131.2
Torus	*	1.8	*	134.5
10^3 Neurons	69.8	0.18	23.03	74.9
14^3 Neurons	*	0.36	*	52.3
Tornado 6.5M	*	1.7	*	79.2
Tornado 35.9M	*	8.8	*	33.5
DTI	35.6	0.16	117.6	259.4
Torus	678.0	1.8	31.29	271.2
10^3 Neurons	70.1	0.2	65.5	151.4
14^3 Neurons	191.8	0.36	38.5	107.78
Tornado 6.5M	673.1	1.8	12.7	171.4
Tornado 35.9M	*	9.0	*	75.8

it is a reasonable approximation to the models produced by VTK and ParaView. We compare rendering performance and memory consumption of our method against the tessellated models using the ray casting renderer (Table 3). Similar to previous results in molecular visualization [FKE13, GKM*15, Sto98, HDS96], we find significant performance and memory improvements when using our non-polygonal geometry.

6.3. Performance Impact of CSG Intersection

As discussed previously in Section 4.3.1, the CSG intersection method required to remove interior surfaces for correct transparency comes at a significant cost. To quantify this cost, we compare the rendering performance of the first-hit ray traversal, suitable for opaque geometry, with our all-hit CSG traversal, suitable for transparency. In both cases, we render opaque geometry, to avoid including other performance impacts inherent in rendering with transparency, thus isolating the impact of the CSG traversal method.

We measure this overhead on four datasets: the DTI tractography data at $r = 0.05$ and $r = 0.25$, and the 10^3 and 14^3 neuron assemblies at a near viewport (Figure 11). As expected, the CSG traversal decreases rendering performance; however, we find that for all but the most expensive renderer (path tracing), the CSG traversal remains interactive. We further observe that the CSG traversal has a greater impact on the Brain DTI data than on the neuron assemblies, and that the impact is greater as the radius increases on the DTI data.

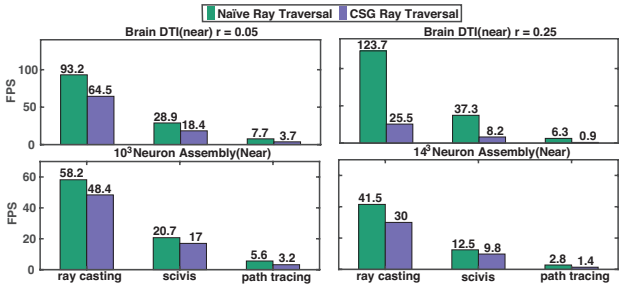


Figure 11: *Performance impact of the CSG ray traversal required for correct transparency. Benchmarks were performed rendering opaque geometry in both cases, with only the traversal method switched. Although the CSG traversal comes with a performance impact, it remains interactive in most cases.*

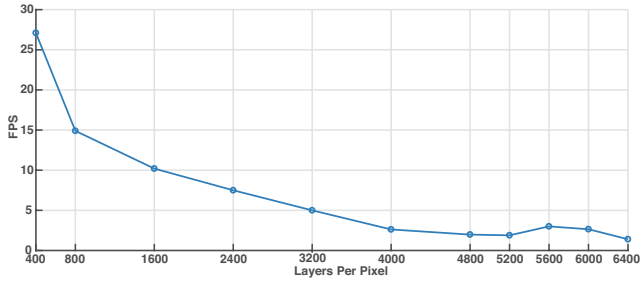


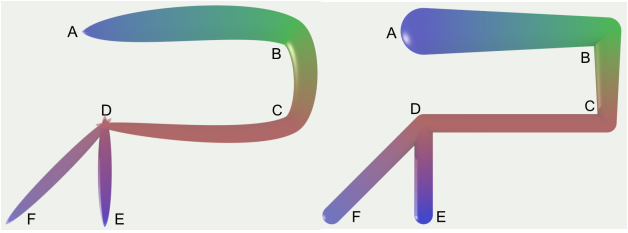
Figure 12: *Performance impact of the CSG ray traversal required for correct transparency. Benchmarks were performed by increasing the number of layers of semi-transparent geometry per pixel. Our method remains interactive, even at 6000 layers of transparency.*

In the case of the DTI data, the number of tracts overlapped by each ray is higher than on the neuron assemblies, where the individual lines are quite thin when viewed from far away. As the radius of the tracts increases, the number of tracts overlapping each ray increases correspondingly, translating to a more expensive CSG traversal. We also evaluate how our method scales with the number of layers of transparency per-pixel (Figure 12). Each layer's opacity is set to 0.5, with randomly generated RGB colors. Even at a large number of transparent layers, our method remains interactive.

6.4. Smooth Curves vs. Linear Links

Although our focus in this work is on using linear links between control points, we note that Embree's Bézier curve primitive, which also supports varying radii and transparency, was recently made available in OSPRay to represent streamlines. Bifurcations can also be emulated with Embree's curves by duplicating the start point of the branches, although the transparency at the bifurcation will be incorrect. We compare our generalized tube with Embree's Bézier curve using a test case with three key features: varying radius, a bifurcation, and transparency (Figure 13).

Compared to our generalized tube, Embree's curve provides smoother bends along the curve (at points B and C), giving a visually pleasing result. However, Embree's curve primitive loses information encoded using the line radius, which could result in users misinterpreting the data. Finally, bifurcations must be faked by duplicating the start point to create the branches (lines DF and



(a) *Embree's Bézier curve primitive.* **(b)** *Our generalized tube.*

Figure 13: *Although Embree's curve primitive (a) provides a visually pleasing representation, it loses information encoded in the radius and exhibits artifacts at bifurcations.*

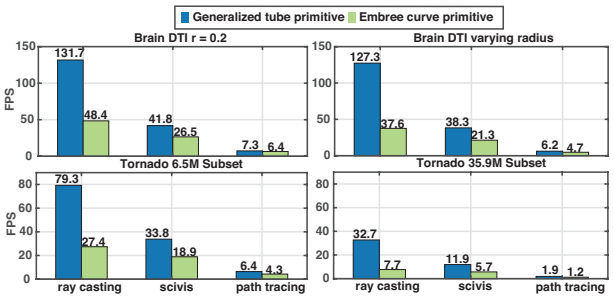


Figure 14: *Comparison of rendering performance of our generalized tubes and Embree's curve primitive. We find our method is up to* $2\times$ *to* $4\times$ *faster for scientific visualization style use cases.*

DE), resulting in artifacts at the bifurcation point, D. Due to the duplication of D, interior surfaces can be seen in the overlap at the bifurcation point, and the bifurcation does not round-off at the point.

For non-bifurcating lines, Embree's curve primitive provides images roughly similar to those rendered by our method. In these cases, we can perform a quantitative comparison and examine the rendering performance (Figure 14) and memory use of the two methods (Table 4). We evaluate the methods on four datasets: the Brain DTI data, with a fixed radius ($r = 0.25$) and varying radii, encoding the fractional anisotropy, and the two Tornado sub-sets.

We find that the smoothness of Embree's Bézier curves comes with a performance cost compared to our simpler method (Figure 14). On all datasets, we find better rendering performance with our method, with the exception of path tracing on the Tornado sub-sets, where our method performs similar to Embree. Finally, we observe similar results in memory cost when using our generalized tube, compared to Embree (Table 4). However, the implementation choice we used in benchmark is not one that saves the most memory. As discussed in Section 4.2 and Section 6.1, our method can reduce memory usage and still be faster than Embree's curve primitive.

Table 4: *Average memory consumption of our generalized tube and Embree's curve primitive. In all cases, our method consumes memory similar to that for Embree's curve primitive.*

Dataset	Embree Curve	Generalized Tubes
DTI ($r = 0.25$)	0.13GB	0.13GB
DTI (varying r)	0.13GB	0.13GB
Tornado 6.5M	1.4GB	1.6GB
Tornado 35.9M	7.6GB	8.8GB

7. Discussion and Conclusion

In this paper, we have presented a new method for rendering generalized tube primitives that supports varying radii and bifurcations. This primitive type is applicable to a wide range of datasets, such as flows, scalar or vector fields, neuron morphologies, and topological structures. Furthermore, we used an efficient CSG-based intersection approach that enables correct transparency by removing interior surfaces. Our approach provides high-performance rendering with low memory overhead for up to billions of primitives. Furthermore, our method is general enough to embed into any ray tracing framework, such as Nvidia Optix [PBD*10], or production film renderers such as Cycles [Fou] and Arnold [KCSG18].

Some challenges remain to be addressed in our proposed approach. Specifically, proper handling of transparency is important for applications that require this feature; although our current CSG method for removing interior surface has provided a solution, it strongly impacts performance. Although we offer the faster non-transparent traversal mode, it is up to the application to select this mode, which, if past experience is any guide, will likely mean applications will pick the "slow but correct" mode by default. Future work along this line to explore faster methods for transparency will be valuable for end users of the primitive.

Finally, whereas the integration into tools such as ParaView should in theory be simple, any such integration always uncovers at least some missing or mismatched features that may require additional modifications. Eventually, it is also worth considering the broader question of whether it would make sense to add the geometry type and algorithms described in this paper to other ray tracers, such as OptiX [PBD*10], taking advantage of Geforce RTX [nvi], and if so, whether there is a need for some standardization of what exactly a line primitive type would have to support in any given ray tracer.

Despite these open issues, we believe our approach will be a useful addition to the arsenal of geometric primitive types in visualization. Although the applicability of our primitive is somewhat specific, for applications that do need such primitives, ours will significantly improve users' ability to visualize and understand their data.

Acknowledgements

This work was supported in part by the NIH (Grant P41 GM103545-18). Additional support comes from the Intel Parallel Computing Centers Program, NSF:CGV: Award 1314896, NSF:IIP: Award 1602127, NSF:ACI: Award 1649923, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375 and NSF:OAC: Award 1842042. The authors wish to thank Ally Warner for the brain DTI dataset, Steve Petruzza for the torus flow and tornado datasets, and Attila Gyulassy for the jet flame Morse-Smale complex dataset. The authors also thank the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing access to Stampede2.

References

[AA09] ACKER C. D., ANTIC S. D.: Quantitative Assessment of the Distributions of Membrane Conductances Involved in Action Potential

Backpropagation Along Basal Dendrites. *Journal of neurophysiology* (2009). 8

[ADH07] ASCOLI G. A., DONOHUE D. E., HALAVI M.: NeuroMorpho. Org: A Central Resource for Neuronal Morphologies. *Journal of Neuroscience* (2007). 8

[AGGW15] AMSTUTZ J., GRIBBLE C., GÜNTHER J., WALD I.: An Evaluation of Multi-Hit Ray Traversal in a BVH using Existing First-Hit/Any-Hit Kernels. *Journal of Computer Graphics Techniques (JCGT)* (2015). 6

[Aya15] AYACHIT U.: *The Paraview Guide: A Parallel Visualization Application*. Kitware, Inc., 2015. 2

[BBLW07] BENTHIN C., BOULOS S., LACEWELL D., WALD I.: Packet-based Ray Tracing of Catmull-Clark Subdivision Surfaces. *SCI Institute, University of Utah, Technical Report* (2007). 3

[BJCW09] BHAGVAT D., JESCHKE S., CLINE D., WONKA P.: GPU Rendering of Relief Mapped Conical Frusta. In *Computer Graphics Forum* (2009), Wiley Online Library. 2, 4

[BK85] BRONSVOORT W. F., KLOK F.: Ray Tracing Generalized Cylinders. *ACM Transactions on Graphics (TOG)* (1985). 3

[Blu19] BLUEBRAIN: BlueBrain/Brayns, 2019. URL: https://github.com/BlueBrain/Brayns. 2, 3

[BMB*13] BRITO J., MATA S., BAYONA S., PASTOR L., DEFELIPE J., BENAVIDES PICCIONE R.: Neuronize: a tool for building realistic neuronal cell morphologies. *Frontiers in neuroanatomy* (2013). 2

[BPL*12] BROWNLEE C., PATCHETT J., LO L.-T., DEMARLE D., MITCHELL C., AHRENS J., HANSEN C.: A Study of Ray Tracing Large-Scale Scientific Data in Parallel Visualization Applications. In *Proceedings of the Eurographics Workshop on Parallel Graphics and Visualization, EGPGV* (2012). 2

[BSG*09] BRUCKNER S., SOLTESZOVA V., GROLLER E., HLADUVKA J., BUHLER K., JAI Y. Y., DICKSON B. J.: BrainGazer-Visual Queries for Neurobiology Research. *IEEE transactions on visualization and computer graphics* (2009). 2

[BWN*15] BENTHIN C., WOOP S., NIESSNER M., SELGRAD K., WALD I.: Efficient Ray Tracing of Subdivision Surfaces using Tessellation Caching. In *Proceedings of the 7th Conference on High-Performance Graphics* (2015), ACM. 3

[CBW*12] CHILDS H., BRUGGER E., WHITLOCK B., MEREDITH J., AHERN S., PUGMIRE D., BIAGAS K., MILLER M., HARRISON C., WEBER G., ET AL.: VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. High Performance Visualization-Enabling Extreme-Scale Scientific Insight. *Insight* (2012). 2

[CYY*11] CHEN C.-K., YAN S., YU H., MAX N., MA K.-L.: An Illustrative Visualization Framework for 3D Vector Fields. In *Computer Graphics Forum* (2011), Wiley Online Library. 2

[DPH*03] DEMARLE D. E., PARKER S., HARTNER M., GRIBBLE C., HANSEN C.: Distributed Interactive Ray Tracing for Large Volume Visualization. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2003. PVG 2003.* (2003), IEEE. 2

[Dra99] DRAKOS N.: Some Mathematics for Advanced Graphics, 1999. URL: https://www.cl.cam.ac.uk/teaching/1999/AGraphHCI/SMAG/node2.html. 3, 4

[EHS13] EICHELBAUM S., HLAWITSCHKA M., SCHEUERMANN G.: LineAO—Improved Three-Dimensional Line Rendering. *IEEE Transactions on Visualization and Computer Graphics* (2013). 2

[Eil13] EILEMANN S.: *Equalizer Programming and User Guide: The official reference for developing and deploying parallel, scalable OpenGL applications using the Equalizer parallel rendering framework*. Eyescale Software GmbH, 2013. 2

[FKE13] FALK M., KRONE M., ERTL T.: Atomistic Visualization of Mesoscopic Whole-Cell Simulations Using Ray-Casted Instancing. In *Computer Graphics Forum* (2013), Wiley Online Library. 9

[Fou] FOUNDATION B.: Cycles Open Source Production Rendering. URL: https://www.cycles-renderer.org/. 10

[GGTH07] GARTH C., GERHARDT F., TRICOCHE X., HAGEN H.: Efficient Computation and Visualization of Coherent Structures in Fluid Flow Applications. *IEEE Transactions on Visualization and Computer Graphics* (2007). 2

[GIK*07] GRIBBLE C. P., IZE T., KENSLER A., WALD I., PARKER S. G.: A Coherent Grid Traversal Approach to Visualizing Particle-Based Simulation Data. *IEEE Transactions on Visualization and Computer Graphics* (2007). 2, 3

[GKM*15] GROTTEL S., KRONE M., MÜLLER C., REINA G., ERTL T.: MegaMol—A Prototyping Framework for Particle-Based Visualization. *IEEE transactions on visualization and computer graphics* (2015). 2, 9

[GRT13] GÜNTHER T., RÖSSL C., THEISEL H.: Opacity Optimization for 3D Line Fields. *ACM Transactions on Graphics (TOG)* (2013). 2

[Gum03] GUMHOLD S.: Splatting Illuminated Ellipsoids with Depth Correction. In *VMV* (2003). 4

[GWA16] GRIBBLE C., WALD I., AMSTUTZ J.: Implementing Node Culling Multi-Hit BVH Traversal in Embree. *Journal of Computer Graphics Techniques Vol* (2016). 6

[HDS96] HUMPHREY W., DALKE A., SCHULTEN K.: VMD: Visual Molecular Dynamics. *Journal of molecular graphics* (1996). 9

[JSP*01] JACOBS B., SCHALL M., PRATHER M., KAPLER E., DRISCOLL L., BACA S., JACOBS J., FORD K., WAINWRIGHT M., TREML M.: Regional Dendritic and Spine Variation in Human Cerebral Cortex: a Quantitative Golgi Study. *Cerebral cortex* (2001). 8

[KCSG18] KULLA C., CONTY A., STEIN C., GRITZ L.: Sony Pictures Imageworks Arnold. *ACM Transactions on Graphics (TOG)* (2018). 10

[KP17] KOVÁCS A., PÁL B.: Astrocyte-Dependent Slow Inward Currents (SICs) Participate in Neuromodulatory Mechanisms in the Pedunculopontine Nucleus (PPN). *Frontiers in cellular neuroscience* (2017). 2, 8

[KRW18] KANZLER M., RAUTENHAUS M., WESTERMANN R.: A Voxel-based Rendering Pipeline for Large 3D Line Sets. *IEEE transactions on visualization and computer graphics* (2018). 2

[KWN*13] KNOLL A., WALD I., NAVRÁTIL P. A., PAPKA M. E., GAITHER K. P.: Ray Tracing and Volume Rendering Large Molecular Data on Multi-Core and Many-Core Architectures. In *Proceedings of the 8th International Workshop on Ultrascale Visualization* (2013), ACM. 2, 3

[LBLH19] LINDOW N., BAUM D., LEBORGNE M., HEGE H.-C.: Interactive Visualization of RNA and DNA Structures. *IEEE transactions on visualization and computer graphics* (2019). 2

[LMSC11] LEE T.-Y., MISHCHENKO O., SHEN H.-W., CRAWFIS R.: View Point Evaluation and Streamline Filtering for Flow Visualization. In *2011 IEEE Pacific Visualization Symposium* (2011), IEEE. 2

[Mar06] MARKRAM H.: The Blue Brain Project. *Nature Reviews Neuroscience* (2006). 2, 3, 6, 7

[MCHM10] MARCHESIN S., CHEN C.-K., HO C., MA K.-L.: View-Dependent Streamlines for 3D Vector Fields. *IEEE Transactions on Visualization and Computer Graphics* (2010). 2

[Mer12] MERZKIRCH W.: *Flow Visualization*. Elsevier, 2012. 2

[MMYK06] MELEK Z., MAYERICH D., YUKSEL C., KEYSER J.: Visualization of Fibrous and Thread-like Data. *IEEE Transactions on Visualization and Computer Graphics* (2006). 2

[MSE*06] MERHOF D., SONNTAG M., ENDERS F., NIMSKY C., HASTREITER P., GREINER G.: Hybrid Visualization for White Matter Tracts using Triangle Strips and Point Sprites. *IEEE Transactions on Visualization and Computer Graphics* (2006). 2

[MTHG03] MATTAUSCH O., THEUSSL T., HAUSER H., GRÖLLER E.: Strategies for Interactive Exploration of 3D Flow Using Evenly-spaced Illuminated Streamlines. In *Proceedings of the 19th spring conference on Computer graphics* (2003), ACM. 2

[nvi] NVIDIA GeForce RTX. URL: https://www.nvidia.com/en-us/geforce/20-series/rtx/. 10

[OP05] OELTZE S., PREIM B.: Visualization of Vasculature With Convolution Surfaces: Method, Validation and Evaluation. *IEEE Transactions on Medical Imaging* (2005). 2

[PBD*10] PARKER S. G., BIGLER J., DIETRICH A., FRIEDRICH H., HOBEROCK J., LUEBKE D., MCALLISTER D., MCGUIRE M., MORLEY K., ROBISON A., ET AL.: OptiX: A General Purpose Ray Tracing Engine. In *ACM transactions on graphics (Tog)* (2010), ACM. 10

[PFK07] PETROVIC V., FALLON J., KUESTER F.: Visualizing Whole-Brain DTI Tractography with GPU-based Tuboids and LoD Management. *IEEE transactions on visualization and computer graphics* (2007). 2

[PJH16] PHARR M., JAKOB W., HUMPHREYS G.: *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2016. 3, 4

[PSL*98] PARKER S., SHIRLEY P., LIVNAT Y., HANSEN C., SLOAN P.-P.: Interactive Ray Tracing for Isosurface Rendering. In *Proceedings Visualization'98 (Cat. No. 98CB36276)* (1998), IEEE. 2

[PVH*02] POST F. H., VROLIJK B., HAUSER H., LARAMEE R. S., DOLEISCH H.: Feature Extraction and Visualization of Flow Fields. *Eurographics 2002 State-of-the-Art Reports* (2002). 2

[RBE*06] REINA G., BIDMON K., ENDERS F., HASTREITER P., ERTL T.: GPU-based Hyperstreamlines for Diffusion Tensor Imaging. In *EuroVis* (2006), Citeseer. 2

[SGS05] STOLL C., GUMHOLD S., SEIDEL H.-P.: Visualization with stylized line primitives. In *VIS 05. IEEE Visualization, 2005.* (2005), IEEE. 2

[SKH*04] SCHIRSKI M., KUHLEN T., HOPP M., ADOMEIT P., PISCHINGER S., BISCHOF C.: Efficient Visualization of Large Amounts of Particle Trajectories in Virtual Environments Using Virtual Tubelets. In *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry* (2004), ACM. 2

[SLM04] SCHROEDER W. J., LORENSEN B., MARTIN K.: *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Kitware, 2004. 2

[SM02] SCHUSSMAN G., MA K.-L.: Scalable Self-Orienting Surfaces: A Compact, Texture-Enhanced Representation for Interactive Visualization of 3D Vector Fields. In *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.* (2002), IEEE. 2

[SSV*14] SANGALLI L. M., SECCHI P., VANTINI S., ET AL.: AneuRisk65: A dataset of three-dimensional cerebral vascular geometries. *Electronic Journal of Statistics* (2014). 2

[Ste00] STEINMAN D. A.: Simulated pathline visualization of computed periodic blood flow patterns. *Journal of Biomechanics* (2000). 2

[STH*09] SHI K., THEISEL H., HAUSER H., WEINKAUF T., MATKOVIC K., HEGE H.-C., SEIDEL H.-P.: Path Line Attributes - an Information Visualization Approach to Analyzing the Dynamic Behavior of 3D Time-Dependent Flow Fields. In *Topology-Based Methods in Visualization II*. Springer, 2009. 2

[Sto98] STONE J. E.: An Efficient Library for Parallel Ray Tracing And Animation. 3, 9

[SZH97] STALLING D., ZOCKLER M., HEGE H.-C.: Fast Display of Illuminated Field Lines. *IEEE transactions on visualization and computer graphics* (1997). 2

[TAC*13] THOMANETZ V., ANGLIKER N., CLOËTTA D., LUSTENBERGER R. M., SCHWEIGHAUSER M., OLIVERI F., SUZUKI N., RÜEGG M. A.: Ablation of the mTORC2 component rictor in brain or purkinje cells affects size and neuron morphology. *J Cell Biol* (2013). 2

[TWHS05] THEISEL H., WEINKAUF T., HEGE H.-C., SEIDEL H.-P.: Topological Methods for 2D Time-Dependent Vector Fields Based on Stream Lines and Path Lines. *IEEE Transactions on Visualization and Computer Graphics* (2005). 2

[TWSH02] TRICOCHE X., WISCHGOLL T., SCHEUERMANN G., HAGEN H.: Topology tracking for the visualization of time-dependent two-dimensional flows. *Computers & Graphics* (2002). 2

[VPRK02] VUKŠIĆ M., PETANJEK Z., RAŠIN M. R., KOSTOVIĆ I.: Perinatal Growth of Prefrontal Layer III Pyramids in Down Syndrome. *Pediatric neurology* (2002). 8

[VW85] VAN WIJK J. J.: Ray Tracing Objects Defined by Sweeping a Sphere. *Computers & Graphics* (1985). 4

[WBW*14] WOOP S., BENTHIN C., WALD I., JOHNSON G. S., TABELLION E.: Exploiting Local Orientation Similarity for Efficient Ray Traversal of Hair and Fur. In *High Performance Graphics* (2014). 3

[WJA*17] WALD I., JOHNSON G. P., AMSTUTZ J., BROWNLEE C., KNOLL A., JEFFERS J., GÜNTHER J., NAVRÁTIL P.: OSPRay-A CPU Ray Tracing Framework for Scientific Visualization. *IEEE transactions on visualization and computer graphics* (2017). 2, 3

[WKI*17] WU K., KNOLL A., ISAAC B. J., CARR H., PASCUCCI V.: Direct Multifield Volume Ray Casting of Fiber Surfaces. *IEEE transactions on visualization and computer graphics* (2017). 2

[WKJ*15] WALD I., KNOLL A., JOHNSON G. P., USHER W., PASCUCCI V., PAPKA M. E.: CPU Ray Tracing Large Particle Data with Balanced P-k-d Trees. In *2015 IEEE Scientific Visualization Conference (SciVis)* (2015), IEEE. 2, 3

[WTBJ19] WARNER A., TATE J., BURTON B., JOHNSON C. R.: A High-Resolution Head and Brain Computer Model for Forward and Inverse EEG Simulation. *bioRxiv* (2019). 7

[WVDLH05] WÜNSCHE B., VAN DER LINDEN J., HOLMBERG N.: DTI volume rendering techniques for visualising the brain anatomy. In *International Congress Series* (2005), Elsevier. 2

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Transactions on Graphics (TOG)* (2014). 2, 3, 5

[ZDL03] ZHANG S., DEMIRALP C., LAIDLAW D. H.: Visualizing Diffusion Tensor MR Images Using Streamtubes and Streamsurfaces. *IEEE Transactions on Visualization and Computer Graphics* (2003). 2

[ZSH96] ZOCKLER M., STALLING D., HEGE H.-C.: Interactive Visualization of 3D-Vector Fields Using Illuminated Stream Lines. In *Proceedings of Seventh Annual IEEE Visualization'96* (1996), IEEE. 2