Curve Complexity Heuristic KD-trees for Neighborhood-based Exploration of 3D Curves

Yucheng Lu, Luyu Cheng, Tobias Isenberg, Chi-Wing Fu, Guoning Chen, Hui Liu, Oliver Deussen, Yunhai Wang

EUROGRAPHICS 2021 / N. Mitra and I. Viola
(Guest Editors)

Volume 40 (2021), Number 2

Curve Complexity Heuristic KD-trees for Neighborhood-based Exploration of 3D Curves

Yucheng Lu,[1] Luyu Cheng,[1] Tobias Isenberg,[2] Chi-Wing Fu,[3] Guoning Chen,[4] Hui Liu,[5] Oliver Deussen,[6] Yunhai Wang[1†]

[1]Shandong University, Qingdao, China
[2]Université Paris-Saclay, CNRS, Inria, LISN, France
[3]The Chinese University of Hong Kong
[4]University of Houston, USA
[5]Shandong University of Finance and Economics, China
[6]University of Konstanz, Germany; SIAT Shenzhen, China

Abstract

We introduce the curve complexity heuristic (CCH), a KD-tree construction strategy for 3D curves, which enables interactive exploration of neighborhoods in dense and large line datasets. It can be applied to searches of k-nearest curves (KNC) as well as radius-nearest curves (RNC). The CCH KD-tree construction consists of two steps: (i) 3D curve decomposition that takes into account curve complexity and (ii) KD-tree construction, which involves a novel splitting and early termination strategy. The obtained KD-tree allows us to improve the speed of existing neighborhood search approaches by at least an order of magnitude (i. e., 28× for KNC and 12× for RNC with 98% accuracy) by considering local curve complexity. We validate this performance with a quantitative evaluation of the quality of search results and computation time. Also, we demonstrate the usefulness of our approach for supporting various applications such as interactive line queries, line opacity optimization, and line abstraction.

CCS Concepts

• ***Human-centered computing*** → ***Scientific visualization;*** • ***Theory of computation*** → ***Nearest neighbor algorithms;***

1. Introduction

Line data consisting of dense curves arises in many applications.[†] Streamlines [MLP*10] derived from flow data by trajectory computation, e. g., convey the structure of the underlying vector fields, fiber tracts traced from DTI-MRI data (e. g., [HS15]) reveal the brain's white matter structure in medical imaging, and trajectories summarized as trails or bundles (e. g., [ZCL08a, LHT17]) of graph edges describe object motion. As our computational capabilities increase, so do the sizes of the line datasets generated by simulations [Sar09] and sensing [BMZA12]; e. g., many DTI datasets contain several thousands of fiber tracts or more, comprising millions of point samples overall. Large line datasets often cover the entire 2D or 3D domain densely, a direct visualization thus not only leads to occlusion and clutter but also often obscures important structures. Only displaying a few curves, however, may let researchers or practitioners miss structures of interest. To solve these problems, we need efficient exploration techniques for massive line data.

Several visual simplification methods [MVVW05, YWSC12, Ise15] have been proposed for the effective visualization of dense curves. We can roughly group them into *curve selection* and *curve abstraction* approaches. Curve selection carefully picks representative curves using local importance measures per curve [MCHM10, GRT13], or clustering all curves with different line similarity metrics [YWSC12, OLK*14]. However, almost all existing methods do not allow users to interactively query curves based on spatial proximity. Yet a substantial demand exists for such interactive line data exploration, e. g., for showing streamlines around user-specified vortex cores [VB96, Sca11, CLSW14]. We thus need means to use local neighborhoods in *interactive* curve selection.

Unlike curve selection, curve abstraction [BCP*12, Ise15, VI18] directly works with nearest neighbors. Everts et al. [EBB*15], e. g., iteratively displaced curve samples based on nearby curves, such that the resulting curve abstraction highlights the overall structure with minimal deformation. This approach needs the nearest neighbors of a curve within a given radius, which they found by brute force with a time complexity of $O(m^2n^2)$ for m curves (n is the average number of samples per curve). This kind of complexity hinders interactive exploration of such abstracted line data—especially for data sets with millions of samples.

These issues motivated us to explore efficient data structures for nearest neighbor searches for line data to facilitate interactive curve

† The corresponding author

† We interchangeably use the word "line" and "curve" in our discussion. Wherever we mean a straight line we explicitly say so.

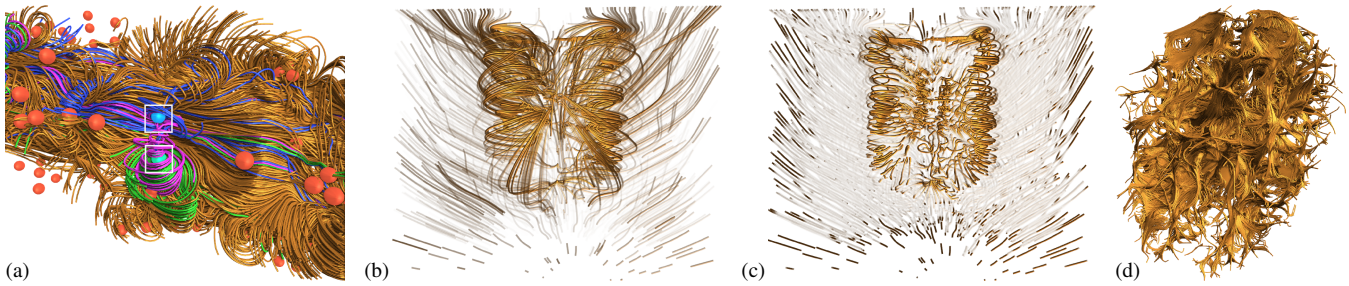

Figure 1: *Neighborhood-based exploration of 3D line data. (a) Selecting nearest lines with a radius r around two query points (white boxes in (a)). For the top query point, the nearest lines are shown in blue, for the bottom query point in green. Lines in violet are close to both query points; (b) varying opacity for the line segments is specified by using a measure based on the nearest line segments as importance. This allows us to better display large-scale vortex structures than a measure based only on local curvature, as shown in (c); and (d) abstraction of DTI fiber tracts at the scale of 5 mm based on our neighborhood search, only major paths are shown.*

selection and abstraction. We characterize the search of nearest neighbors from given curves as the problem of *nearest curve* (NC) search. It can be solved with two straightforward solutions using KD-trees, which have been widely used for nearest point searches in 2D or 3D space [WH06, ZHWG08, SBMN16]. That is, either we sample many points of each curve and use a KD-tree to find the nearest points from distinct lines, or we build a KD-tree for all curve segments between every pair of adjacent points and then find the nearest curve segment(s) for the query point based on the point-to-line distance. The former strategy is faster but might not be able to find the accurate nearest curves, while the latter finds the correct nearest curves, but might be slower. Since they do not consider the curve complexity, it is unknown if they are fast enough for NC search in a large line field.

Inspired by the strategy of using the surface area heuristic for constructing KD-trees in ray tracing [GS87, WH06, ZHWG08], we approach this problem using a carefully-designed KD-tree, namely a *CCH KD-tree*, with the curve complexity heuristic (CCH). Our CCH KD-tree is constructed in two steps. First, we approximate each input 3D curve individually by a set of straight-line segments constructed based on the curve complexity. Second, we construct an efficient KD-tree by adopting the CCH to determine the split plane and to compute the node split cost. Integrated with an early termination strategy, our CCH KD-tree can be constructed in less than a few seconds for data with 10K curves and a million point samples on average. For DTI fiber data of such size, we demonstrate that our method runs in an order of magnitude faster than the straightforward solutions, while recall and precision (i. e., search accuracy) are both higher than 98%. Our method supports the *radius-nearest curve* (RNC) search (i. e., to find all curves located within a given radius r from the query point, and to also find the associated nearest point samples on each output curve) as well as the *k-nearest curve* (KNC) search (i. e., to find the k nearest curves from the given query point). Specifically, we achieve a 20× speed-up for KNC and a 15× speed-up for RNC in general with 98.5% curve recall/precision.

With our fast approximate approach, we can thus now tackle various neighborhood analysis tasks on massive line data sets that were previously outside of our reach. Our RNC search allows users to explore lines around critical points and vortex cores, facilitating an interactive identification of patterns of interest. The fast KNN graph enables us to interactively abstract large line datasets to identify higher-level structures [EBB*15] (cf. Fig. 1(c)). In summary, our main contributions are:

- we characterize the problem of nearest curve search for exploring line data (Sec. 3) and propose a new strategy for constructing an efficient KD-tree (Sec. 5), which we call CCH KD-tree,
- we quantitatively evaluate the quality and performance of CCH KD-tree and show that it significantly improves the curve search performance while maintaining the search accuracy (Sec. 6), and
- we introduce a neighborhood-based analysis of line data and demonstrate it in various applications, including interactive line queries, opacity optimization, and line abstraction.

2. Related Work

We begin by reviewing previous work related to curve selection, abstraction, segmentation, and nearest neighbor search problems.

2.1. Curve Selection

Given a large set of curves, which densely cover a domain, the goal of curve selection is to choose a few representatives that convey the main features of the data [SBGC20]. In general, there are two classes of methods: importance-based and clustering-based selection.

Importance-based selection pre-computes local properties of all given curves, then selects new curves based on their relation to the already selected curves, often in a greedy manner. Marchesin et al. [MCHM10] looked at the screen-space footprint of the already selected lines, and used local properties such as linear and angular entropy to convey the respective amount of velocity and angular variation along the streamlines. Günther et al. [GBWT11] measured the screen coverage of each curve to select curves with a higher screen area. Since a single view might not sufficiently show the characteristics of every curve, Lee et al. [LMSC11] further computed the screen-space entropy of each line to select curves and viewpoints. Tao et al. [TMWS13] suggested to simultaneously select curves and viewpoints with their proposed streamline information, which indicates the dependence between the streamlines and views.

Rather than selecting a subset of curves for visualization, Günther et al. [GRT13] proposed a global curve selection approach by means

of opacity optimization. By rendering curves with optimal opacity per segment, this approach aims at showing curves with high importance, without omitting those with low importance. Later, Günther et al. [GTG17] provided an analytic solution to opacity optimization, enabling interactive exploration of large line data. Though the method allows users to define application-specific importance measures, the importance is often defined using local properties such as curvature or curve length, which might not faithfully characterize the global structures.

So far, however, existing methods select curves based mainly on local curve properties. Their shortcoming is that without considering the spatial relationship between neighboring curves, these methods might miss interesting structures in the data (see the vortex structures in Fig. 1(b),(c)). In contrast, our nearest curve search selects curves according to the spatial proximity, and enables us to define local properties in the neighborhood range, such that we can better characterize structural information in the line data.

Clustering-based selection picks representative curves from curve clusters found in the data. This approach is commonly used for DTI data exploration [MVVW05], where individual fibers are grouped into relevant bundles. Based on the spatial proximity between every pair of curves, Zhang et al. [ZCL08a] applied hierarchical clustering to produce fiber bundles for interactive exploration. Moberts et al. [MVVW05] evaluated different hierarchical clustering methods and distance measures for the problem, and found that combining single-link hierarchical clustering using mean closest-point distances produces the best results. Later, Yu et al. [YWSC12] applied this finding to streamlines, and created a hierarchy of streamline bundles. Oeltze et al. [OLK*14] discussed different clusterings and different metrics for clustering streamlines of blood flow data. Recently, Kanzler et al. [KFW16] constructed a fully balanced line hierarchy based on a variant of hierarchal line clustering and then used this hierarchy to guide the representative line selection. Kern and Westermann [KW19] discussed the clustering of complex line geometry and used the bounding volume hierarchies (BVHs) to compute the vectors to the closest point volumes of lines on a regular grid. While these methods reveal interesting patterns, they might not show patterns related to specific geometric properties of line segments of multiples curves, e. g., the ring structure in magnetic field lines (see Fig. 12). We, in contrast, advocate the clustering of curve samples to efficiently group curve segments of similar patterns.

2.2. Curve Abstraction & Segmentation

Curve abstraction [EBB*15] creates a visually abstracted [VI18, VCI20] representation of large DTI fiber sets without generating explicit fiber bundles. After finding the nearest neighbors within a given radius of a query point, the method iteratively contracts the fiber segments by drawing fiber samples closer to the similar ones. This way, the method produces a *visual abstraction* [VI18] of the fibers to facilitate a better understanding of structures in the white matter of the brain. However, finding the exact nearest neighbors for all samples is expensive, e. g., it takes approx. 15 minutes for 70K tracts on four Intel Xeon X7350 processors [EBB*15] using four threads, this limits the applicability of the method for exploring large data sets. Moreover, finding exact nearest neighbors is not needed for such an application as long as the approximation is of high quality. Our CCH KD-tree does precisely that—we find the nearest neighbors for the same data set with millions of point samples with our probabilistic method in around 5s on a single thread of a similar machine, and thus enable interactive data exploration.

Curve segmentation divides the curves in line data into short segments, and is often taken as a pre-processing step for curve selection [GRT13] and abstraction [EBB*15]. Using segment-based local properties such as curvature histograms and total curvature, Lu et al. [LCL*13] and Wang et al. [WESW14] decomposed streamlines into segments of different lengths to facilitate search for similar streamlines or similar streamline segments. We also base our CCH KD-tree on line segments that we pre-generate using a variant of the Ramer-Douglas-Peucker algorithm [Ram72, DP73, VW90], which decomposes and approximates a curve by fewer line segments.

2.3. Nearest Neighbor Search

A few algorithms [HS92, EMSN12] have been proposed to accelerate (approximate) nearest neighbor search in 3D space, where KD-trees [AMN*98] and their variants are still the leading approach. By partitioning the data points recursively over different planes, a KD-tree can efficiently perform a nearest neighbor search in logarithmic time and linear space. Due to these computational advantages, it has been widely used for accelerating many graphics algorithms and applications, such as ray casting [ZSL*18], isosurface rendering [WWW*18], and particle tracing [ZGH*17].

To improve the query performance, numerous efforts have been made to optimize the KD-tree structure. A classic example is the surface area heuristic (SAH) [GS87] for splitting the KD-tree, which is widely used in ray tracing. Rather than naively selecting split planes using the spatial median along one axis, an SAH KD-tree determines proper split by estimating the node split cost in terms of the surface area (see Sec. 4.2 for details). By doing so, the amount of ray intersection tests can be greatly reduced, and thus this heuristic has become the basis for many fast ray-tracing methods. Likewise, CCH KD-tree explores the curve complexity to choose proper split planes for 3D curves to accelerate nearest neighbor queries. Nonetheless, nearest neighbor search for line data is very different from ray tracing, e. g., the node split cost in our case has to consider also the cost of backtracking [HS12], which helps explore nearby nodes, since the true nearest neighbor may not lie in the query cell.

3. Problem Definition

To explore the spatial relations in a line data set we have to find the closest curves to a query point in a potentially large set. To formalize this task, we first define the notation we use in this paper:

- $\mathbf{C} = \{C_1, C_2 \cdots, C_m\}$ denotes a large set of m curves, which is the input data to the problem; these 2D or 3D curves could be straight lines or bended curves;
- $\mathbf{P}_i = \{\mathbf{p}_{i,1}, \cdots, \mathbf{p}_{i,n_i}\}$ denotes the set of n_i point samples along C_i;
- $\mathbf{P} = \cup_i \mathbf{P}_i$ denotes the set of all point samples in $\mathbf{C}$;
- $n = \sum_i n_i$ denotes the total number of point samples in $\mathbf{P}$, so the average number of point samples in each curve is n/m;
- $\mathbf{q}$ denotes a query point located in the same space as $\mathbf{C}$; and

- $\mathbf{G} = \{g_1, \cdots, g_k\}$ denotes the set of curves in $\mathbf{C}$ that are the k nearest to $\mathbf{q}$, so $\mathbf{G}$ is the query solution.

Problem. Given query point $\mathbf{q}$, our goal is to efficiently find $\mathbf{G}$. Representing each curve (C_i) as a set of points $\{\mathbf{P_i}\}$, we define the (shortest) distance from $\mathbf{q}$ to C_i as

$$D(\mathbf{q}, C_i) = \min_{\mathbf{x} \in C_i} d(\mathbf{q}, \mathbf{x}) \,, \tag{1}$$

where $d(\mathbf{q}, \mathbf{x})$ denotes the Euclidean distance from $\mathbf{q}$ to point $\mathbf{x}$ on C_i. Since $d(\mathbf{q}, \mathbf{x})$ is the distance to the segments, $\mathbf{x}$ might not be any point sample $\mathbf{p}_{i,j}$. Hence, the nearest curve (NC) problem can be defined as

$$\mathrm{NC}(\mathbf{q}, \mathbf{C}) = \underset{i \in [1,m],\ i \neq i_\mathbf{q}}{\arg\min} \ D(\mathbf{q}, C_i) \,,$$

where $i_\mathbf{q}$ is the index of the curve to which $\mathbf{q}$ belongs, i.e., if $\mathbf{q}$ is a point sample of a certain curve in $\mathbf{C}$, say $C_{i_\mathbf{q}}$; otherwise, $i_\mathbf{q}$ is null.

Often, we want to find a set of nearest curves rather than just a single one. Hence, we extend the NC problem in two ways:

(i) Radius Nearest Curve (RNC) search, to find all the nearest curves within a distance of r from query point $\mathbf{q}$, and
(ii) K-nearest Curve (KNC) search, to find the k nearest curves to $\mathbf{q}$.

In the case of KNC, we should obtain exactly k distinct nearest curves, i. e., $|\mathbf{G}| = k$, and the query solution $\mathbf{G}$ should satisfy

$$\forall g_i \in \mathbf{G},\ l \in \mathbf{C} \setminus \mathbf{G},\ \ D(\mathbf{q}, g_i) \leq D(\mathbf{q}, l). \tag{2}$$

In the case of RNC, $|\mathbf{G}| = k$ is not needed but we require $\mathbf{G}$ to satisfy the following condition for parameter r:

$$\forall g_i \in \mathbf{G} \,,\ \ D(\mathbf{q}, g_i) \leq r \,. \tag{3}$$

Fig. 2 illustrates RNC and KNC searches using query points $\mathbf{q}_1$ and $\mathbf{q}_2$, respectively. In particular, point samples associated with the nearest curves for RNC and KNC must lie on distinct data lines. This requirement fundamentally differs from RNN and KNN [AMN*98, ML09]: we have to consider the curves associated with the nearest point samples, and we cannot simply look for the nearest point samples in the whole sample point set. Some of the nearest points from $\mathbf{q}$ may likely come from the same data line.

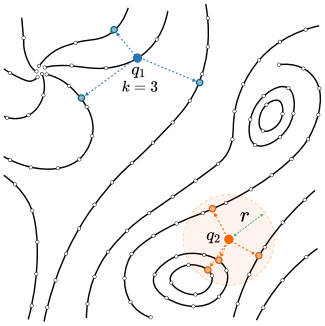

Figure 2: *A 2D illustration of k-nearest curve (KNC) and radius nearest curve (RNC) search problems with a set of uniformly-sampled points along streamlines. For a query point $\mathbf{q}_1$ using KNC we find the three nearest curves shown above subject to a count threshold $k = 3$, while for query point $\mathbf{q}_2$ using RNC, we find the four nearest curves shown above subject to a distance threshold r. Note that there are three instead of four curves for $\mathbf{q}_1$, since we do not report the curve that $\mathbf{q}_1$ belongs to.*

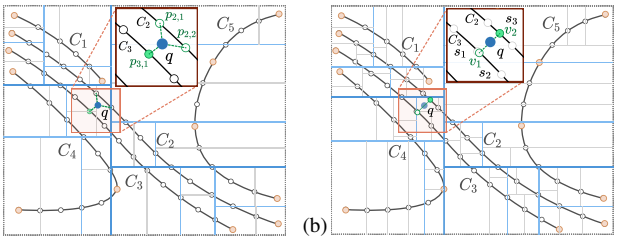

Figure 3: *Nearest curve search for query point $\mathbf{q}$ using two variants of KD-trees. (a) A point-based KD-tree finds the nearest point sample $p_{3,1}$ and returns C_3 as the nearest curve. (b) A segment-based KD-tree finds the nearest point v_2 on C_2 and returns the correct nearest curve C_2.*

4. Analyzing Existing Approaches

Before presenting our full method in Sec. 5, we briefly describe how existing nearest neighbor search techniques such as KD-trees can be adopted for NC search, review the SAH KD-tree, and discuss how it can be applied to curves.

4.1. Approaches for Neighbor Curve Search

We now describe three general approaches for nearest curve search, given query point $\mathbf{q}$ and curve set $\mathbf{C}$, and analyze their time complexity.

A brute-force approach. We may exhaustively compute the minimum distance from $\mathbf{q}$ to every curve in $\mathbf{C}$, and select the k nearest curves accordingly. Since $k \ll n$, the time complexity is $O(n)$ by using a heap to keep the k distinct nearest lines, where n is the total number of point samples in the data (Sec. 3). However, since n is in the order of millions for most DTI fibers and streamlines, such an approach could take more than an hour, thus inhibiting an interactive data exploration (e. g., [EBB*15]).

Point-based nearest curve (NC) search. To accelerate the search, we could also collect all point samples in $\mathbf{C}$ and use an advanced nearest neighbor search (NNS) such as a KD-tree, FLANN [ML09], or libnabo [EMSN12] to keep the k nearest points using a heap. With a hash table, we can avoid points from the same curve, and keep adding nearest points to the heap until we obtain k nearest points from the distinct curves.

Suppose a KD-tree is used and $k \ll n$, the overall time complexity is $O(\log n)$ for a KNC search. Similarly, we can achieve an RNC search by taking a distance threshold r from $\mathbf{q}$ as a pruning condition when traversing the KD-tree, so the time complexity is bounded by $O(n^{2/3})$ [BF79]. Regardless of the NNS method being used, this approach might not always find the correct nearest curve, because representing each curve as a set of discrete point samples ignores their geometric continuity. Fig. 3(a) shows an example, where the nearest point sample found by the point-based KD-tree was located on curve C_3 but the nearest point is located on curve C_2 instead.

We can also build a point-based KD-tree for each curve in the data and perform the query for each curve. Obviously, the query time of this scheme is proportional to the number of curves and the overall time complexity is $O(m \log n/m)$ for a KNC search. Thus, this method is much slower than the point-based KD-tree.

Segment-based KD-tree. For each curve, say C_i, we collect $n_i - 1$ curve segments between pairs of adjacent point samples. Now we can build a segment-based KD-tree (see Fig. 3(b)) by recursively subdividing the space with so-called "spatial median" splitting planes [WH06]. During the subdivision, a segment might be divided into two parts by the splitting plane. To ensure segment completeness, we duplicate this segment into two sub-nodes. To avoid excessive subdivision, we stop the splitting of nodes with N segments, if one of the resulting sub-nodes also contains no less than N segments; otherwise, the search efficiency would be lower than just linear search with N segments. By doing so, the problem of nearest curve search converts into a nearest line segment search, in which the point-to-point distances required for nearest neighbor search are replaced by the point-to-line distances.

Since the number of segments is close to the number of point samples n, the search complexities of KNC and RNC become the same as the ones for point-based KD-trees (see Fig. 3(b)). Unlike point-based KD-trees, segment-based KD-trees always return correct results (since all segments are straight lines). Hence, we will use them to find the ground truths for evaluating our CCH KD-tree. However, such KD-trees do not take curve complexity into account and divide nearly straight curves into many small segments, causing poor performance for NC queries; see our quantitative comparison results in Sec. 6.

4.2. Background: SAH KD-tree

A KD-tree is often used for ray-triangle intersection tests in ray tracing of static scenes. To minimize the time for KD-tree traversal and ray intersection steps, the surface area heuristic (SAH) [GS87, WH06] is often used for building an efficient KD-tree. This heuristic tries to divide the scene in a way that a cost function is minimized. Given a static scene S, an SAH KD-tree is built by recursive subdivision. That is, we recursively subdivide S into two sub-scenes at a time by measuring the SAH costs of all potential split planes and applying the one with the lowest cost. To model the SAH cost, rays are typically assumed to be uniformly distributed in space. Hence, given a random ray that passes through scene S, the probability for the ray to pass through sub-scene $S_{sub} \subset S$ is

$$P(S_{sub}|S) = \frac{SA(S_{sub})}{SA(S)}, \tag{4}$$

where $SA(S)$ denotes the total surface area of scene S.

Given a potential split plane h, which subdivides S into sub-scenes S_l and S_r, the expected time cost C of intersecting a random ray with the tree resulted by plane h is given by

$$C(h) = T_{\text{traverse}} + P(S_l|S)\ T_{\text{intersect}}(S_l) + P(S_r|S)\ T_{\text{intersect}}(S_r) \tag{5}$$

where T_{traverse} is the time to traverse the internal node associated with h, while $T_{\text{intersect}}(S_l)$ and $T_{\text{intersect}}(S_r)$ are the times to perform intersection tests for S_l and S_r, respectively. Since $T_{\text{intersect}}(S_l)$ and $T_{\text{intersect}}(S_r)$ can only be measured after the whole tree is built, finding a global optimal tree is intractable. So, a local greedy approximation is often used to approximate $C(h)$ by assuming the resulting children are leaves:

$$C(h) = T_{\text{traverse}} + \left[\frac{SA(S_l)}{SA(S)} N_l + \frac{SA(S_r)}{SA(S)} N_r\right] t_{\text{intersect}} \tag{6}$$

where N_l and N_r are the number of triangles in S_l and S_r, $t_{\text{intersect}}$ is the time to perform an intersection test with a single triangle. Since both S_l and S_r are likely to be further subdivided, this approximated solution is not globally optimal, but works well in practice.

As discussed in Sec. 4.1, a KD-tree can be used for accelerating NC search. To build the best tree with the minimal traversal cost, we can design curve-based heuristics as an SAH KD-tree. Unlike ray tracing surfaces with SAH, our nearest curve search method works within the whole volume of related KD-tree nodes instead of the surface area in SAH.

5. CCH KD-tree for 3D Curves

Our key idea is to fit each curve in the data with a small number of straight-line segments and build a line-segment-based KD-tree for fast and as accurate as possible NC search. However, it is inefficient to build and search within a tree, in which each grid cell contains only a single segment defined by two adjacent point samples. Hence, we propose *curve complexity heuristic* (CCH) KD-trees that are generated by exploiting curve complexity to optimize KD-trees for NC search. Fig. 4 shows the pipeline of our method, which consists of two stages: *offline preprocessing* and *online query*.

Offline preprocessing. Each curve in a given dataset consists of an ordered list of densely and uniformly sampled points. At the beginning, we divide each curve into a set of segments, such that each segment is not excessively bent (see Fig. 4(a)). Then, we construct the CCH KD-tree using these segments (see Fig. 4(b)) and represent each curve segment by a set of straight lines (see Fig. 4(c)). Each leaf node in the KD-tree stores the start and end points, and the curve index of the corresponding line segment; each internal node stores the split dimension and split plane.

Online query. Once a CCH KD-tree is built, we can perform an NC search similar to a nearest point search with a traditional KD-tree for a given query point. The major difference is that we compute the distances from the query point to the line segments stored in the leaf nodes instead of simply using point to point distances.

We now detail the three steps of the offline pre-processing: *curve partitioning*, *CCH KD-tree construction*, and *curve fitting*, before we discuss the efficiency of our approach in Sec. 6.

5.1. Curve Partitioning

Although existing curve segmentation methods that are based on local properties (e. g., total curvature) can preserve important features to a certain extent, they do not aim to approximate curves by straight line segments with small errors. Rather than using expensive optimizations [Sto61, HC94], we employ the Ramer-Douglas-Peucker algorithm [VW90] to iteratively divide each given curve into segments. Since each curve segment might be further divided during

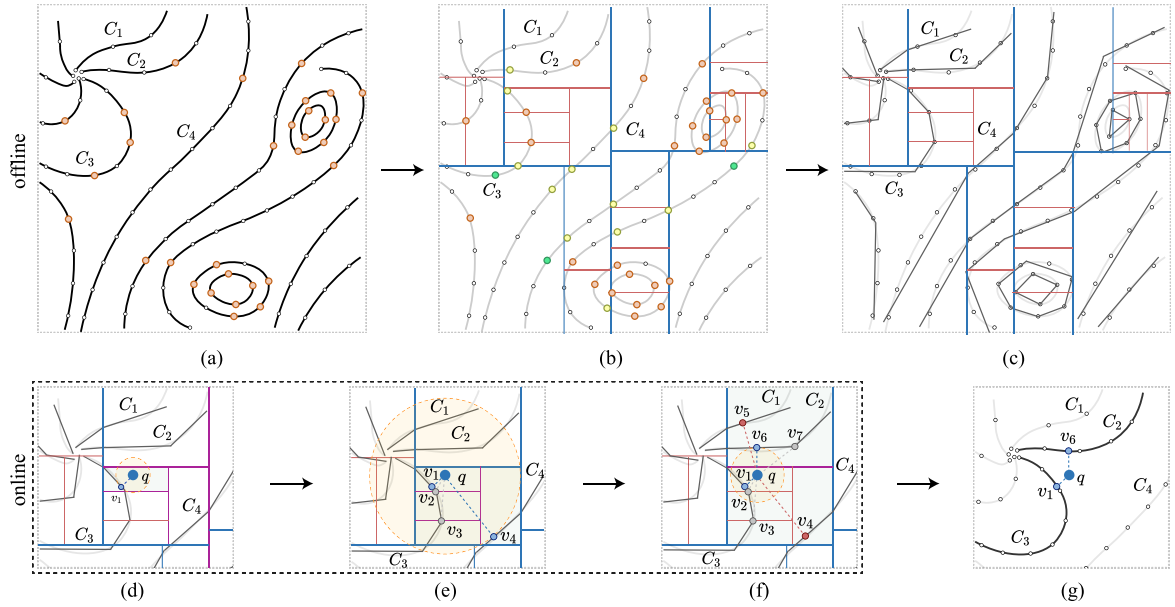

Figure 4: *Overview of our two-stage method—offline pre-processing (a–c) and online query (d–g): (a) set of input curves with point samples (white) and split points (orange); (b) the CCH KD-treebuilt for the input curves with the removed split points (green) and the newly added split points (yellow), the split axes' (blue) thickness indicates the level, where the thicker the lines are the higher levels of the KD-tree; (c) the curve segment in each grid cell is approximated by a straight-line segment; (d,e,f) three steps in retrieving the two nearest curves $k = 2$ for the given query point* $\mathbf{q}$*, where the related split axes are highlighted in purple; (d) the first nearest curve C_3 with the nearest sample v_1 is found from the leaf node; (e) backtracking to the upper level and finding the two nearest samples v_2 and v_3, which are from the same curve as v_1, continued backtracking gives us a new nearest sample v_4 from curve C_4; (f) further backtracking to find the nearest samples in the circle with radius from* $\mathbf{q}$ *to v_4. A closer sample v_6 is found, resulting in the fact that v_4 and v_5 (in red) are rejected; (g) the nearest curves are C_2 and C_3 and the corresponding nearest samples are v_1 and v_6 for query point* $\mathbf{q}$.

the construction of the KD-tree, we do not explicitly segment the curves but just record the split points in this step.

Each input curve C_i with point samples $\{\mathbf{p}_{i,j}\}$ $(j = 1, ..., n_i)$, is divided into a set of segments using the following procedure:

(i) construct a straight line l from $\mathbf{p}_{i,1}$ to $\mathbf{p}_{i,n_i}$;
(ii) find the point sample $\mathbf{p}_{i,c}$ furthest away from l, the distance being larger than the threshold $\theta * \overline{ls}$, use it as a *split point*; and
(iii) divide C_i into segments using the found split point and repeat the above steps for each divided curve segment.

Here, $\overline{ls}$ denotes the averaged sample interval distance: $\overline{ls} = \sum_{i=1}^{m} \text{len}(C_i)/(n-m)$, where $\text{len}(C_i)$ is the length of curve C_i. Fig. 5 shows a running example to illustrate the procedure.

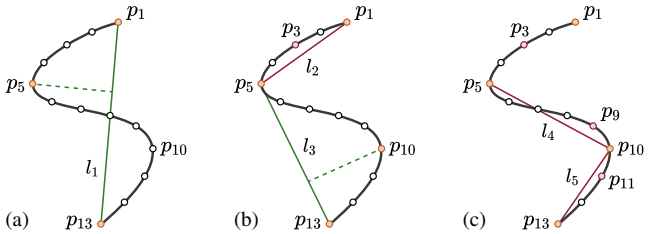

Figure 5: *Curve segmentation step: Given the curve in (a), we find the point sample with maximum distance from l_1 ($\mathbf{p}_1$ to $\mathbf{p}_{18}$), i. e., $\mathbf{p}_5$; the distance value is larger than threshold θ, so we treat it as a split point; (b) after splitting the curve using $\mathbf{p}_5$, we further find a split point $\mathbf{p}_{10}$ and produce the curve segments shown in (c); when no more split points can be found we stop the partitioning.*

5.2. CCH KD-tree Construction

Next, we build our CCH KD-tree by recursively subdividing the axis-aligned bounding box (AABB) of the input curves. Inspired by SAH, the CCH KD-tree takes into account the traversal cost for tree construction and also positions the split plane with the spatial median [WH06]. For each non-leaf (internal) node, the space subdivision proceeds as follows:

(i) compute the cost of all three split planes in three dimensions and pick the one with the lowest cost;
(ii) split the node into two child nodes and divide the corresponding curves; and
(iii) duplicate the curve segments divided by the split plane into two sub-trees and update the associated split points.

Assuming that the x-dimension is selected for a split, we position the split plane by using the spatial median so as to evenly divide the point samples among the two children. In the following, we will detail two essential components in our procedure: *split plane cost*, and *curve segment adjustment*.

5.2.1. Split Plane Cost

Using the aforementioned split plane, the input curves can be split until each sub-grid cell contains only a single curve segment. However, searching in such a KD-tree may be inefficient compared to a brute-force search. The additional segments introduced by the node splits may introduce higher costs for traversing the sub-trees and also increase the backtracking cost as the tree depth increases. For producing an efficient KD-tree, we thus measure the cost of the split plane following Eq. 6. If the cost is larger than the one for a brute-force search, we stop the split of the current node. For searching KNC with $k > 1$ and RNC, we further provide an early termination criterion to determine if the split of the current node needs early termination before evaluating the cost of the split plane. Next, we provide details about the traversal and backtracking costs as well as the early termination criterion.

Traversal Cost. Suppose a node has n curve segments, and the split plane divides them into l segments in the left sub-tree and r segments in the right sub-tree. Since segments that cut through the split plane have to be further divided, we might have $l + r \geq n$. Assuming that candidates of nearest neighbors are uniformly distributed in space [San04], the probability P of searching the left sub-tree T_l of tree T is

$$P(T_l|T) = \frac{Vol(T_l)}{Vol(T)}. \tag{7}$$

where $Vol(T)$ is the volume of the AABB of the curve segments associated with tree T. Note that, in NC search, we derive the probability with the AABB volume of the KD-tree node T, instead of the surface area in SAH for ray tracing. Due to the axis-aligned split plane, the probability for the right sub-tree T_r is thus $1 - P(T_l|T)$.

Computing the global optimal traversing cost is infeasible without completely building the entire tree. We thus follow the SAH KD-tree to estimate the cost with a locally greedy approximation. We assume all nodes in T_l and T_r to be leaves, so we define the cost as

$$C_{traversal}(T) = T_{\text{traverse}} + \big[P(T_l|T)l + P(T_r|T)r\big]T_{\text{dist}}, \tag{8}$$

where T_{dist} is the cost of distance comparison in a leaf node. We empirically found that T_{traverse} is about 20% of T_{dist}. Thus, we set $T_{\text{traverse}} = 0.2$ and $T_{\text{dist}} = 1$.

Backtracking Cost. During the nearest neighbor search, backtracking is often required as shown in Fig. 4. The number of nodes to be visited during backtracking is usually proportional to the tree depth. The backtracking cost is thus proportional to the depth of the sub-tree [RD95].

Given again a current node with n segments (l segments for the left child and r segments for the right). Without loss of generality, we assume $l \geq r$. Let $\rho = \frac{l}{n}$ and $\tau = \frac{r}{n}$ to be the split ratio of the segments for the left and right nodes of the current sub-tree and all its subsequent sub-trees. We can then estimate the maximum depth of this sub-tree as $\log_{\frac{1}{\rho}}(n)$, while the minimum depth of the other sub-tree is $\log_{\frac{1}{\tau}}(n)$. Thus, the average cost of the possible maximum and minimum depths of the sub-tree is

$$C_{backtrack}(T) = \lambda(\log_{\frac{1}{\rho}}(n) + \log_{\frac{1}{\tau}}(n))/2\,, \tag{9}$$

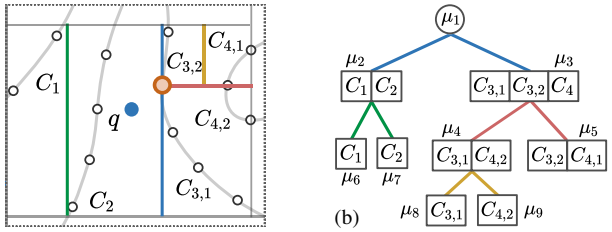

Figure 6: *Termination criteria for query point* $\mathbf{q}$ *using KNC: (a) grid cells generated by decomposing the space covered by the input curves with the KD-tree in (b), where each node except the root stores the indices of all contained curves.*

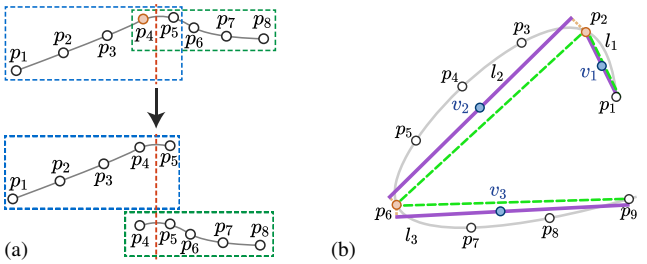

Figure 7: *Curve segment adjustment and fitting. (a) Adjusting and distributing curve segments resulting from a node split: (top) the split axis (dashed line) divides the input curve into two segments (blue and green boxes), (bottom) both segments are lengthened to include an extra point sample and distributed into two sub-nodes; after that, the orange split point* $\mathbf{p}_4$ *is no longer a split point. (b) Fitting each curve segment with a purple straight line segment (solid) using PCA, instead of connecting the two ends to form the green straight line (dashed).*

where λ is a user-specified weight.

The node split should generally make the query more efficient than a brute-force search. We thus define the split plane's cost as

$$C(T) = C_{traversal}(T) + C_{backtrack}(T) - nT_{\text{dist}}, \tag{10}$$

where nT_{dist} is the cost of brute-force search. If the costs for all split planes to be created are larger than zero, we stop T's subdivision.

Termination Criterion. For KNC search with a large k, a sub-tree may not contain sufficient distinct curves, and thus, the query often needs backtracking to its parent nodes for checking the other sub-trees. In such cases, further subdivision might bring more costs than not splitting at all. We thus stop splitting a node if its left or right child contains less than $k/2$ distinct curves. As illustrated in Fig. 6 with searching within four NCs, further subdivision of nodes μ_2 and μ_3 requires us to check six line segments and traverse five nodes, whereas without splitting we only need to check five line segments and traverse a single node.

This process is similar for RNN. Given a search radius r, we terminate splitting the KD-tree, if the length of either sub-region with an axis orthogonal to the split plane is less than $r/2$.

5.2.2. Curve Segment Adjustment.

If a curve segment is divided into two parts by a split plane, we distribute each newly-generated segment into the respective child node. Since the intersection between split plane and curve may not exactly be at the point samples of the curve, we lengthen each divided segment by also including the adjacent point sample beyond the split plane; see Fig. 7(a) for an illustrative example. In addition, we have to update the distance of the split points in the new segment and check if it is still larger than the threshold $\theta * \overline{ls}$ as mentioned in Section 5.1. If not, we remove the split points. For example, the orange split point $\mathbf{p}_4$ in the top of Fig. 7(a) is removed at the bottom of Fig. 7(a).

5.3. Curve Fitting

Once a CCH KD-tree is built, each grid cell contains one or multiple curve segments. To fit each segment with a straight line, a straightforward method is to connect the adjacent split and end points (see the green dashed lines in Fig. 7(b)). However, this method might introduce a large error for fitting the original curves. To address this issue, we fit each curve segment by projecting 2D point samples to a 1D line with principal component analysis (PCA) [WEG87]. Since PCA finds the projection axis with minimal reconstruction error, it leads to better piecewise approximations of the curve segments than simply connecting the two endpoints as in the Ramer-Douglas-Peucker algorithm. In doing so, we fit each segment by the line along the principal direction and determine the two ends of this line by projecting the starting and ending samples to it. Fig. 7(b) shows two examples, where the curves are fitted by the purple solid lines.

6. Evaluation

We implemented our CCH-KD-tree and a segment-based KD-tree and point-based KD-tree in C++. The implementation of our point-based KD-tree is based on the optimized state-of-the-art implementation, FLANN [ML09]. We used the single KD-tree option for the computation in the FLANN method; the other options (e. g., randomized KD-tree forest) might work well for high-dimensional data but are unsuitable for our low-dimensional data. We tested and compared these methods on a PC with an Intel Core™ i7-6700HQ CPU @ 2.6GHz and 8 GB memory. The data sets used for the evaluation is shown in Table 1.

6.1. Quantitative Measures

Ideally, the KNC/RNC search should achieve high accuracy (i. e., search results close to those obtained using a brute-force nearest neighbor search), and with faster computation. Therefore, we measure the speed of an approach in terms of *query time* and assess the search accuracy in terms of *curve recall* and *curve precision*, which together assess the accuracy of a search algorithm.

To investigate KNC/RNC, we let the set of curves retrieved with a specific search approach for query point $\mathbf{q}$ be $\mathbf{A}(\mathbf{q})$ and the set returned by the brute-force approach (i. e., the ground truth) be $\mathbf{I}(\mathbf{q})$.

Curve recall measures the amount of ground-truth of nearest curves $\mathbf{I}(\mathbf{q})$ that a given method correctly identifies in $\mathbf{A}(\mathbf{q})$. A value close to 1 means that most ground-truth nearest curves are found.

$$curve\ recall(CR) = \frac{|\mathbf{A}(\mathbf{q}) \cap \mathbf{I}(\mathbf{q})|}{|\mathbf{I}(\mathbf{q})|} . \tag{11}$$

Curve precision measures the amount of correctly-found curves in the retrieved results $\mathbf{A}(\mathbf{q})$. A value close to 1 indicates that most nearest curves found by the current method are true nearest curves.

$$curve\ precision(CP) = \frac{|\mathbf{A}(\mathbf{q}) \cap \mathbf{I}(\mathbf{q})|}{|\mathbf{A}(\mathbf{q})|} . \tag{12}$$

Table 1: *Datasets used for the comparison.*

Data	Domain	# Curves	# Samples
aneurysm	Medical	531	364005
combustion	Physics	3000	768545
borromean	Chemistry	3930	1489125
laminar	Fluid dynamics	1000	552340
cylinder	Fluid dynamics	944	540051
heli_flight	Aerodynamics	1001	567678
heli_descent	Aerodynamics	1031	901402
rings	Synthesized	446	243476
Square Cylinder	Fluid dynamics	3112	3545135
tornado	Fluid dynamics	3000	856720

6.2. Parameter Analysis

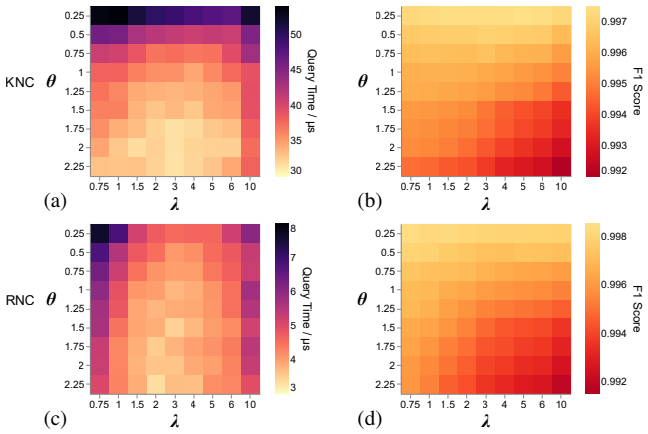

Figure 8: *Heatmaps with query time (a,c) and F1 scores (b,d) of grid search, for parameters θ and λ in KNC ($k = 25$) and RNC ($r = 0.02$) searches.*

Next, we analyze two parameters in our CCH KD-tree: partitioning curve threshold θ and backtracking cost parameter λ. The parameter θ determines whether a curve should be subdivided at a candidate split point. A larger θ leads to coarser partitioning, yielding fewer line segments, which reduces not only the search time but also the accuracy. The parameter λ affects the depth of the obtained KD-tree. Larger λ values lead to an earlier termination of KD-tree splitting, smaller λ values lead to more splits during the construction of the KD-tree. Thus, we need to find optimal parameter values to optimize both query accuracy and query time.

We performed a grid search over the parameters θ and λ in the ranges of [0.25,2.25] and [0.75,10], respectively. Since we want to maximize both recall and precision, we measure the query accuracy by using F1 score [Bis06]:

$$F1 = 2\frac{CR * CP}{CR + CP}, \tag{13}$$

which ranges from 0 to 1. For each data set listed in Table 1 and each value pair of (θ, λ), we randomly sample 50K data points to perform KNC (k=25) and RNC (r=0.02) search and then average the query time and F1 score of all data sets. The heatmaps in Fig. 8 summarize the results of KNC and RNC search. To find the optimal parameter values resulting in less query time and high query accuracy, we combine these two aspects together by calculating $Time/F1^2$ for each value pair, where the squared F1 score is used for the more emphasis in query accuracy. In doing so, the optimal (θ, λ) are (2.25, 3) and (2.25, 2.0) for KNC and RNC searches for all data sets, respectively.

6.3. Quantitative Comparison

Next, we conducted a comparative evaluation of the three methods by performing RNC and KNC searches using M randomly-generated query points for each data set. Here, we set M as 50000 and tested different k values, i.e., $\{5, 10, 15, \cdots, 50\}$, for the KNC search. For the RNC search, we tested different r values, i.e., $\{0.01, 0.02, \cdots, 0.1\}$ since the dataset values have been normalized. For a comprehensive evaluation, we collected ten 3D line fields from a wide variety of application domains and structures (see Table 1). We took the results returned by the segment-based KD-tree as our ground truth, since they always return the correct results (see Sec. 4.1).

Result analysis. To quantitatively assess the three methods, we computed the averaged *CP*, *CR*, and *query time* over the searches with M query points for each dataset. The boxplots shown in Fig. 9(a,b,c) summarize the three measures for the three methods to perform KNC ($k = 25$) and RNC ($r = 0.02$) searches. The results show that our CCH KD-tree achieves an accuracy of 98.5% in terms of CP and CR, while the point-based KD-tree results in 0.1%–0.5% higher accuracies than our CCH KD-tree. In terms of the performance, the CCH KD-tree achieves a 529% speedup for the KNC search, while achieving a 232.8% speedup for the RNC search. Compared to the segment-based KD-tree, it is around 28× for KNC search and 12× faster for RNC search. CR and CP are equal for all three methods for KNC search, as the cardinalities of $\mathbf{A}(\mathbf{q})$ and $\mathbf{I}(\mathbf{q})$ both are k.

For both searches, the query time of CCH KD-tree is less than 50 µs, facilitating interactive neighborhood-based curve exploration. CR and CP increase with an increasing k for KNC search, and increase with an increasing radius r for RNC search, which remains above 98% in most cases.

Fig. 10 shows a comparison of the methods for different search parameters k and r on the *aneurysm* data, where the query time in Fig. 10(c,d) is shown in log scale. For both KNC and RNC, our CCH KD-tree achieves 100%–500% speedup over the point-based KD-tree and is 8–15 × faster than the segment-based KD-tree, while maintaining a CR between 0.95 and 1.0. For both searches, the speedup of our CCH KD-tree over the segment-based KD-tree increases with increasing k or r. After carefully checking, we observed

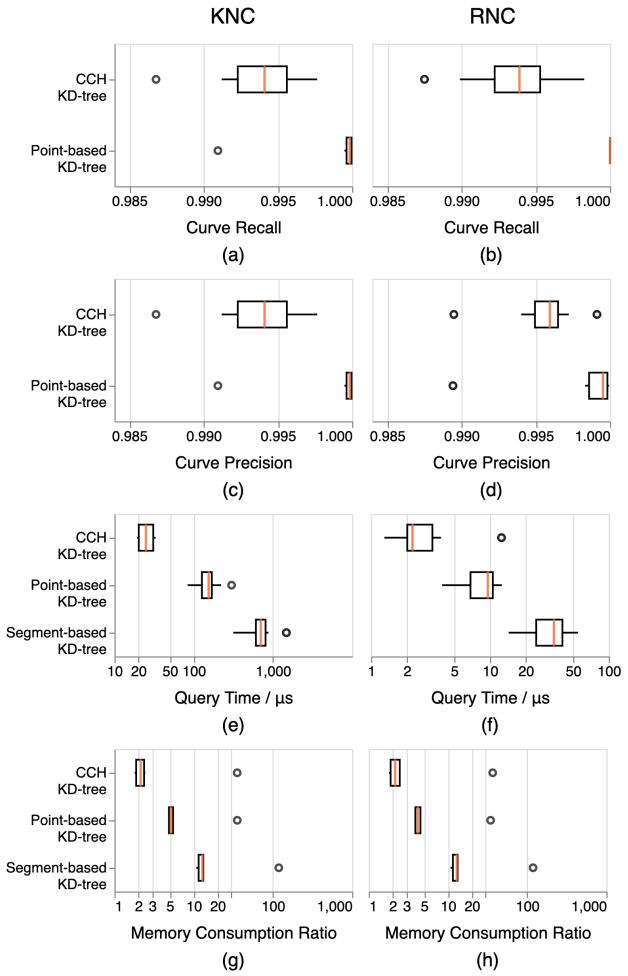

Figure 9: *Boxplots summarizing curve recall (a,b) and precision (c,d), query time (e,f) and memory consumption ratio (g,h) for KNC search (k=25) (a,c,e,g) and RNC search (r=0.02) (b,d,f,h) on all datasets using the three methods. Since the results retrieved by segment-based KD-trees are ground truth, we did not show them in (a,b,c,d).*

that many curves intersect in the data and thus the retrieved curves are almost the same for RNC search with different rs. Note that the curve recall of the point-based KD-tree in RNC search is 1.0, which is consistent with the results in Fig. 9(b).

The boxplots in Fig. 9(d) summarize the memory consumption ratio of each method, which is defined as the memory consumption to the original data size. We can see that each method performs similarly in KNC and RNC search, while our CCH KD-tree requires the least memory, followed by the point-based KD-tree, while the segment-based KD-tree is the worst. The corresponding median consumption ratios of the three methods (our CCH KD-tree, point-based KD-tree, and segment-based KD-tree) are 2.14, 4.81, and 12.92, respectively. Note that the outlier in each of the three methods is the *tornado* data, which consists of many intertwined curves and is hard to be segmented and fit with straight lines.

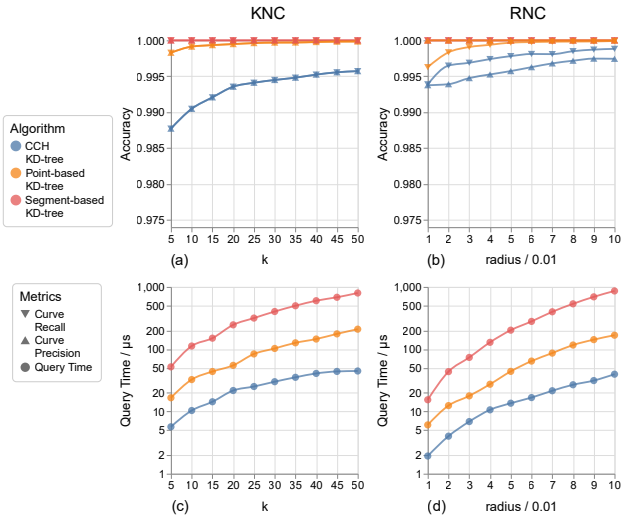

Figure 10: *Comparing curve recall (downward triangles), curve precision (upward triangles), and query time (circle marks) of the three methods for performing KNC searches (a,b) and RNC searches (c,d) within the* aneurysm *dataset. Note that the lines of curve recall and curve precision overlap together in KNC search (a), while they might not overlap in RNC search (b).*

In our experiment, we found that PCA-based curve fitting (Sec. 5.3) does improve the precision and recall of the KNC accuracy both from 98.79% to 99.20%, RNC precision from 98.91% to 99.23%, and RNC recall from 98.76% to 99.49%, respectively.

7. Selected Application Examples

We now discuss potential applications in which a fast curve search facilitates interactive data analysis techniques that have been out of reach in the past. Nonetheless, these examples only represent tasks in curve-based data exploration, other applications may exist.

7.1. Structure-aware line Selection

Most 3D user interfaces select a subset of curves in 3D line data by encircling curves with a lasso [ZCL08b], 3D brushing [JCK12], or checking the co-linearity [JLS*13]. With our fast curve search we can now support interactive, *structure-aware* line selection, related to approaches for particle clouds and volumes [YEII12, YEII16]. In contrast to those techniques whose challenge is how to find a meaningful 3D subvolume based on the 2D input and the data's 3D structure, our key challenge for 3D line data is how to select meaningful subsets of lines in 3D space based on a selected point or a center line, with a given k or radius r. Our CCH KD-tree facilitates efficient search queries: a single KNC search takes only around 20 µs and only 40 µs for an RNC search within real data (see Fig. 9).

By using Our CCH KD-tree for NC search, we provide two modes: point-based mode and line-based mode for users to perform structure-aware line selection. The former finds a set of nearest curves returned from an NC search based on $\mathbf{q}$, while the latter first finds the nearest curve $g_\mathbf{q}$ from $\mathbf{q}$ then performs an RNC search for all the samples along curve $g_\mathbf{q}$ and returns the union of all the search results. Once the search is done, the output curves can be highlighted, while the remained curves are shown with transparency. To facilitate the exploration of dense line fields, users can combine the query results from the two modes using various set operators [WS06]: intersection, union, and difference as we demonstrate in Fig. 11.

Our CCH KD-tree also facilitates line selections using feature points in data, e. g., critical points in a vector field, from which the line data was originally extracted. A common requirement in vector field visualization is to explore streamlines around some critical points, which can be readily fulfilled by our CCH KD-tree. Fig. 1(a) shows an example of streamlines around two critical points in red. Hence, the user can promptly identify interesting flow structures.

7.2. Structure-aware Opacity Specification

The usually heavy occlusions in 3D make specifying proper opacity values crucial for revealing important structures in 3D line fields. Günther et al. [GRT13, GTG17] formulated opacity specification as an optimization problem, which strives to balance occlusion avoidance and information presentation. By showing that a pixel-based formulation can lead to analytical solutions [GTG17], the varying opacity for each line segment can be automatically obtained to highlight important structures. For more detail about that optimization refer to Günther et al.'s [GTG17] discussion.

The importance of each point sample can be defined by various properties, such as line length, point curvature, linear entropy [FI08], angular entropy [MCHM10], and domain-specific attributes. Günther et al. [GRT13] showed that using both length and curvature reveals major structures in data. These two properties, however, are line-based, i. e., they cannot capture larger-scale structures. We thus propose the use of *point saliency* by exploiting our CCH KD-tree approach to better characterize major structures in 3D line fields.

Point saliency. Image saliency has been extensively studied in computer vision [Sze10]. Among them, patch-based methods [CMH*15, GZMT12, LYS*11] characterize global image structures by estimating the saliency based on the similarity between image patches. Benefits have been demonstrated for many applications, including image retargeting, summarization, and segmentation. Similar to this approach, we define the point saliency based on its surrounding line segments, rather than based on individual surrounding point samples.

Given a query point $\mathbf{q}$, we first find the k nearest curves (denoted as g_i) from $\mathbf{q}$, then compute the segment-based similarity between $\mathbf{q}$ and each g_i using the point-to-streamline distance proposed by Chen et al. [CCK07] (denoted as $d_{sim}(\mathbf{q}, g_i)$), and finally, aggregate the similarities to define the point saliency in a range of $[0, 1]$:

$$s_q = 1 - \exp\left(-\frac{1}{k}\sum_i d_{sim}(\mathbf{q}, g_i)\right). \tag{14}$$

If the line segment through $\mathbf{q}$ has more dissimilar segments with the nearest lines, $\mathbf{q}$ would have a larger point saliency; and vice versa.

Opacity specification. By taking point saliency as importance, the opacity optimization framework [GTG17] can automatically find

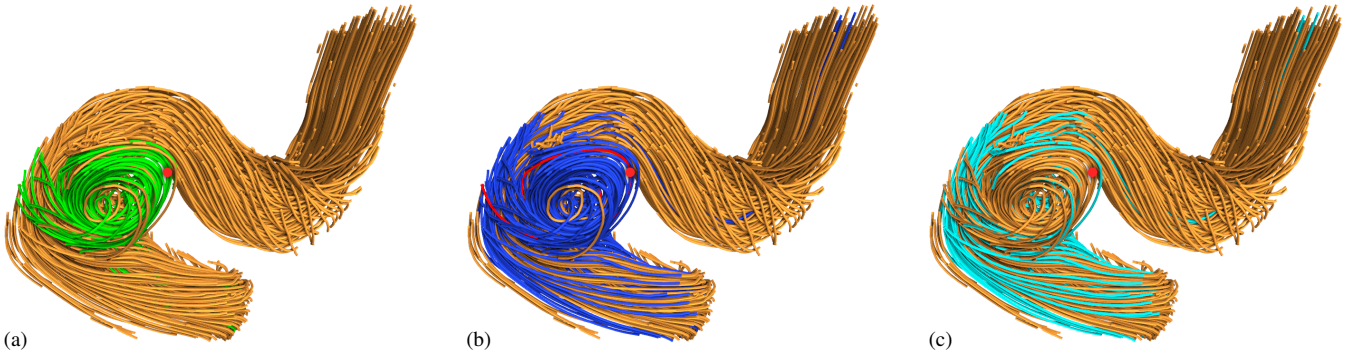

Figure 11: *Selecting lines by combining point-based and line-based mode. (a) The nearest curves (green, denoted as S_a) selected by the red query point; (b) the nearest curves (blue, denoted as S_b) selected in line-based mode, using the red curve (through the red point) as the query curve; and (c) the difference between the two line sets ($S_b - S_a$), showing the curves that are near the query curve but not near the query point.*

Figure 12: *Opacity optimization results for different importance notions: (a) curvature; (b) linear entropy; and (c) point saliency.*

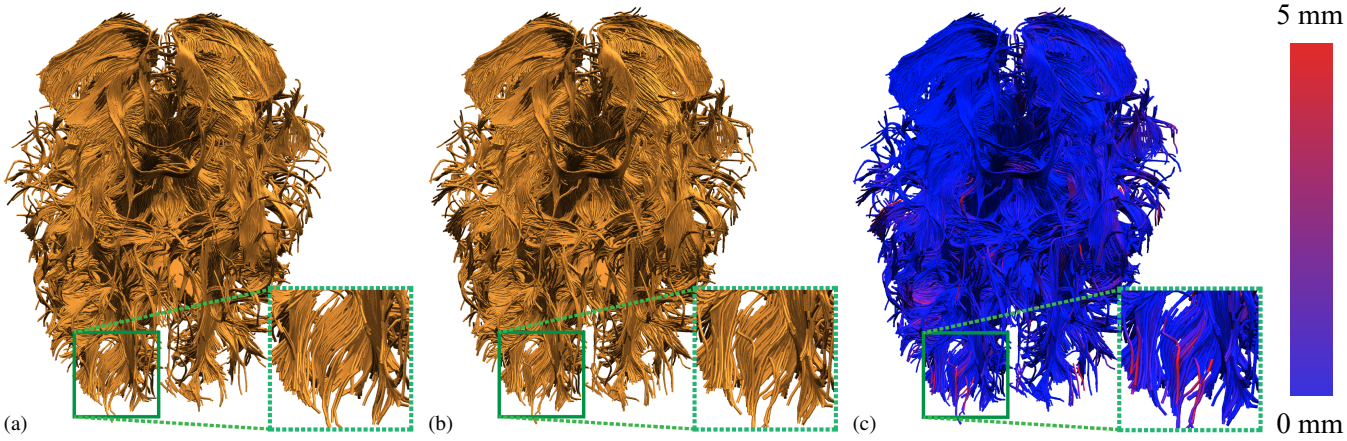

Figure 13: *DTI fiber tract contraction results at the scale of 5 mm by using (a) segment-based KD-tree and (b) our CCH KD-tree. (c) The color coding of the 3D position difference between the two results in (a,b), and the result in (a) is taken as the ground truth as Fig. 9.*

proper opacities for the input 3D lines, while emphasizing the large-scale structures. Fig. 12 compares the results generated by using curvature, line entropy, and our point saliency. We can see that our measure captures large-scale vortex structures to a better extent (see Fig. 12(c)), whereas the other methods mainly show small-scale features (a) or loose major structures (b). Moreover, the intertwined structures are clearly shown in Fig. 12(c), whereas only the interior ring structures are shown in Fig. 12(b).

Although our CCH KD-tree does not directly affect the rendering results shown in Fig. 12(c), its identified nearest curves are an indispensable ingredient of the point saliency computation. As shown in Sec. 6, it is faster than other methods in NC search while pre-

Table 2: *Computing times of the DTI contraction for a dataset consisting of 77,389 tracts with a total of 1,944,700 vertices at all the five-scale levels between the original implementation [EBB*15] and our three different nearest curve search techniques.*

Method	Phase 1 (min.)	Phase 2 (min.)	Total (min.)
Everts et al.'s [EBB*15]	39.45	2.09	41.54
Segment KD-tree	4.79	2.05	6.85
Point KD-tree	1.69	2.06	3.75
CCH KD-tree	0.58	1.43	2.01

serving the accuracy and thus we believe that it is a cornerstone of structure-aware opacity specification.

7.3. Fast Multi-scale Fiber Tract Contraction

The third application is to use RNC search to compute the fiber tract contraction introduced by Everts et al. [EBB*15]. The whole computation has two phases: (i) neighborhood graph construction and (ii) iterative contraction. Both phases are very time-consuming; as indicated by the computational complexity, interactive fiber contraction was considered as not possible at that time.

We compare Everts et al.'s [EBB*15] implementation to our method using the segment-based KD-tree, point-based KD-tree, and CCH KD-tree on the same machine (Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz, 6 cores) for the same dataset with five levels (1 mm–5 mm). Table 2 shows the computing time of four implementations for the two phases, we observe an 8.2 × speedup from the segment-based KD-tree in phase 1 and 3.4 × speedup in total and a 3 × speedup from the deterministic point-based KD-tree in phase 1 and 1.8 × speedup in total. Compared to Everts et al.'s method, our CCH KD-tree has a 20 × speedup. Although such a performance still cannot allow for an interactive line contraction, our CCH KD-tree tremendously reduces the running time and greatly improve the efficiency of DTI abstraction. Note that the almost 1.8 × speedup of the CCH KD-tree compared to the point-based KD-tree can mainly be attributed to the high sparsity of this dataset (i. e., a few regions with low data densities).

Fig. 13 compares fiber tract contraction results from the segment-based KD-tree and from using our CCH KD-tree, at a scale level of 5 mm. The contraction results are virtually identical, only at higher contraction distances, we can see some differences for single tracts. The overall shape of the major pathways, however, does not change. To improve the overall contraction performance, we added a linear weight to the vertices during contraction. It allows us to assign a value of 1 to points at the search center and a weight of 0 to points at the edge of the search radius. Our visual and timing results demonstrate that we are now also able to further explore our technique for the interactive exploration of abstracted fiber tract data, as envisioned by Everts et al.'s [EBB*15].

8. Conclusion

In this paper, we presented the problem of nearest curve (NC) search, including the radius-nearest curve (RNC) search and k-nearest curve (KNC) search for 3D line data. To address this problem, we proposed a technique that uses a curve complexity heuristic for a KD-tree and utilizes curve complexity to customize KD-trees for NC search. We conducted a comprehensive evaluation using various 3D line datasets, and quantitatively demonstrated that our technique can largely improve NC search performance with only small potential errors. The speed-up we achieved allows certain NC-search-based applications, such as structure-aware selection and contraction, which used to be computationally unfeasible, to now be used interactively. It also allows us to perform interactive NC queries to augment various visualizations with less effort, e. g., opacity specification.

There are still some limitations, which we will address in future work. First, our method requires considerable memory to store the point samples and the KD-trees and thus we will develop an out-of-core version to handle data with more than 200K curves. Second, inspired by the GPU construction algorithm for SAH KD-tree [ZHWG08] and GPU-based spatial data visualization [EGG*15], we will explore a GPU version of our CCH KD-tree as a potential avenue for future research. Last, we will explore more neighborhood-based visualization applications. The nearest samples of different lines, for instance, might be grouped into the same cluster, and we will thus investigate neighborhood-based spectral clustering for exploring flow data.

Acknowledgements

We thank Maarten H. Everts for assisting us with the comparison to his original results and Jian Zhang, Yinqi Sun, and Xiao Jiang for discussions on algorithm design. This work was supported by the grants of the NSFC (61772315, 61861136012), the Open Project Program of State Key Laboratory of Virtual Reality Technology, and Systems, Beihang University (No.VRLAB2020C08). It was also partly funded by the Deutsche Forschungsgemeinschaft (DFG Project DE 620/27-1), a CAS grant (GJHZ1862) and the Special Project of Science and Technology Innovation Base of Key Laboratory of Shandong Province for Software Engineering (No. 11480004042015).

References

[AMN*98] ARYA S., MOUNT D. M., NETANYAHU N. S., SILVERMAN R., WU A. Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM 45*, 6 (Nov. 1998), 891–923. doi:10.1145/293347.293348. 3, 4

[BCP*12] BRAMBILLA A., CARNECKY R., PEIKERT R., VIOLA I., HAUSER H.: Illustrative flow visualization: State of the art, trends and challenges. In *Eurographics State-of-the-Art Reports* (2012), The Eurographics Association, Goslar, Germany, pp. 75–94. doi:10.2312/conf/EG2012/stars/075-094. 1

[BF79] BENTLEY J. L., FRIEDMAN J. H.: Data structures for range searching. *ACM Computing Surveys 11*, 4 (Dec. 1979), 397–409. doi:10.1145/356789.356797. 4

[Bis06] BISHOP C. M.: *Pattern Recognition and Machine Learning*. Springer, New York, 2006. URL: https://www.springer.com/gp/book/9780387310732. 9

[BMZA12] BONOMI F., MILITO R., ZHU J., ADDEPALLI S.: Fog computing and its role in the internet of things. In *Proc. MCC* (2012), ACM, New York, pp. 13–16. doi:10.1145/2342509.2342513. 1

[CCK07] CHEN Y., COHEN J., KROLIK J.: Similarity-guided streamline placement with error evaluation. *IEEE Transactions on Visualization and Computer Graphics 13*, 6 (Nov./Dec. 2007), 1448–1455. doi:10.1109/TVCG.2007.70595. 10

[CLSW14] CHAUDHURI A., LEE T.-Y., SHEN H.-W., WENGER R.: Exploring flow fields using space-filling analysis of streamlines. *IEEE Transactions on Visualization and Computer Graphics 20*, 10 (Oct. 2014), 1392–1404. doi:10.1109/TVCG.2014.2312009. 1

[CMH*15] CHENG M.-M., MITRA N. J., HUANG X., TORR P. H., HU S.-M.: Global contrast based salient region detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 37*, 3 (Mar. 2015), 569–582. doi:10.1109/TPAMI.2014.2345401. 10

[DP73] DOUGLAS D. H., PEUCKER T. K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization 10*, 2 (Dec. 1973), 112–122. doi:10.3138/FM57-6770-U75U-7727. 3

[EBB*15] EVERTS M. H., BEGUE E., BEKKER H., ROERDINK J. B. T. M., ISENBERG T.: Exploration of the brain's white matter structure through visual abstraction and multi-scale local fiber tract contraction. *IEEE Transactions on Visualization and Computer Graphics 21*, 7 (July 2015), 808–821. doi:10.1109/TVCG.2015.2403323. 1, 2, 3, 4, 12

[EGG*15] ELSHEHALY M., GRAČANIN D., GAD M., ELMONGUI H. G., MATKOVIĆ K.: Interactive fusion and tracking for multi-modal spatial data visualization. *Computer Graphics Forum 34*, 3 (June 2015), 251–260. doi:10.1111/cgf.12637. 12

[EMSN12] ELSEBERG J., MAGNENAT S., SIEGWART R., NÜCHTER A.: Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics 3*, 1 (Mar. 2012), 2–12. doi:10.6092/JOSER_2012_03_01_p2. 3, 4

[FI08] FURUYA S., ITOH T.: A streamline selection technique for integrated scalar and vector visualization. In *Posters of IEEE Visualization* (2008). URL: http://itolab.is.ocha.ac.jp/~itot/paper/ItotRICPPE13.pdf. 10

[GBWT11] GÜNTHER T., BÜRGER K., WESTERMANN R., THEISEL H.: A view-dependent and inter-frame coherent visualization of integral lines using screen contribution. In *Proc. VMV* (2011), The Eurographics Association, Goslar, Germany, pp. 215–222. doi:10.2312/PE/VMV/VMV11/215-222. 2

[GRT13] GÜNTHER T., RÖSSL C., THEISEL H.: Opacity optimization for 3D line fields. *ACM Transactions on Graphics 32*, 4 (July 2013), Article 120, 8 pages. doi:10.1145/2461912.2461930. 1, 2, 3, 10

[GS87] GOLDSMITH J., SALMON J.: Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications 7*, 5 (May 1987), 14–20. doi:10.1109/MCG.1987.276983. 2, 3, 5

[GTG17] GÜNTHER T., THEISEL H., GROSS M.: Decoupled opacity optimization for points, lines and surfaces. *Computer Graphics Forum 36*, 2 (May 2017), 153–162. doi:10.1111/cgf.13115. 3, 10

[GZMT12] GOFERMAN S., ZELNIK-MANOR L., TAL A.: Context-aware saliency detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 34*, 10 (Oct. 2012), 1915–1926. doi:10.1109/TPAMI.2011.272. 10

[HC94] HAMANN B., CHEN J.-L.: Data point selection for piecewise linear curve approximation. *Computer Aided Geometric Design 11*, 3 (June 1994), 289–301. doi:10.1016/0167-8396(94)90004-3. 5

[HS92] HOEL E. G., SAMET H.: A qualitative comparison study of data structures for large line segment databases. In *Proc. SIGMOD* (1992), ACM, New York, pp. 205–214. doi:10.1145/130283.130316. 3

[HS12] HE K., SUN J.: Computing nearest-neighbor fields via propagation-assisted KD-trees. In *Proc. CVPR* (2012), IEEE Computer Society, Los Alamitos, pp. 111–118. doi:10.1109/CVPR.2012.6247665. 3

[HS15] HOTZ I., SCHULZ T. (Eds.): *Visualization and Processing of Higher Order Descriptors for Multi-Valued Data*. Springer, Berlin/Heidelberg, 2015. doi:10.1007/978-3-319-15090-1. 1

[Ise15] ISENBERG T.: A survey of illustrative visualization techniques for diffusion-weighted MRI tractography. In *Visualization and Processing of Higher Order Descriptors for Multi-Valued Data*, Hotz I., Schulz T., (Eds.). Springer, Berlin/Heidelberg, 2015, ch. 12, pp. 235–256. doi:10.1007/978-3-319-15090-1_12. 1

[JCK12] JACKSON B., COFFEY D., KEEFE D. F.: Force Brushes: Progressive data-driven haptic selection and filtering for multi-variate flow visualizations. In *EuroVis Short Paper Proc.* (2012), The Eurographics Association, Goslar, Germany, pp. 7–11. doi:10.2312/PE/EuroVisShort/EuroVisShort2012/007-011. 10

[JLS*13] JACKSON B., LAU T. Y., SCHROEDER D., TOUSSAINT JR. K. C., KEEFE D. F.: A lightweight tangible 3D interface for interactive visualization of thin fiber structures. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (Dec. 2013), 2802–2809. doi:10.1109/TVCG.2013.121. 10

[KFW16] KANZLER M., FERSTL F., WESTERMANN R.: Line density control in screen-space via balanced line hierarchies. *Computers & Graphics 61* (Dec. 2016), 29–39. doi:10.1016/j.cag.2016.08.001. 3

[KW19] KERN M., WESTERMANN R.: Clustering ensembles of 3D jet-stream core lines. In *Proc. VMV* (2019), The Eurographics Association, Goslar, Germany, pp. 81–88. doi:10.2312/vmv.20191321. 3

[LCL*13] LU K., CHAUDHURI A., LEE T.-Y., SHEN H.-W., WONG P. C.: Exploring vector fields with distribution-based streamline analysis. In *Proc. PacificVis* (2013), IEEE Computer Society, Los Alamitos, pp. 257–264. doi:10.1109/PacificVis.2013.6596153. 3

[LHT17] LHUILLIER A., HURTER C., TELEA A.: State of the art in edge and trail bundling techniques. *Computer Graphics Forum 36*, 3 (June 2017), 619–645. doi:10.1111/cgf.13213. 1

[LMSC11] LEE T.-Y., MISHCHENKO O., SHEN H.-W., CRAWFIS R.: View point evaluation and streamline filtering for flow visualization. In *Proc. PacificVis* (2011), IEEE Computer Society, Los Alamitos, pp. 83–90. doi:10.1109/PACIFICVIS.2011.5742376. 2

[LYS*11] LIU T., YUAN Z., SUN J., WANG J., ZHENG N., TANG X., SHUM H.-Y.: Learning to detect a salient object. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 2 (Feb. 2011), 353–367. doi:10.1109/TPAMI.2010.70. 10

[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 16*, 6 (Nov./Dec. 2010), 1578–1586. doi:10.1109/TVCG.2010.212. 1, 2, 10

[ML09] MUJA M., LOWE D. G.: Fast approximate nearest neighbors with automatic algorithm configuration. In *Proc. VISAPP* (2009), vol. 1, INSTICC Press, Setubal, Portugal, pp. 331–340. doi:10.5220/0001787803310340. 4, 8

[MLP*10] MCLOUGHLIN T., LARAMEE R. S., PEIKERT R., POST F. H., CHEN M.: Over two decades of integration-based, geometric flow visualization. *Computer Graphics Forum 29*, 6 (Sept. 2010), 1807–1829. doi:10.1111/j.1467-8659.2010.01650.x. 1

[MVVW05] MOBERTS B., VILANOVA A., VAN WIJK J. J.: Evaluation of fiber clustering methods for diffusion tensor imaging. In *Proc. Vis* (2005), IEEE Computer Society, Los Alamitos, pp. 65–72. doi:10.1109/VISUAL.2005.1532779. 1, 3

[OLK*14] OELTZE S., LEHMANN D. J., KUHN A., JANIGA G., THEISEL H., PREIM B.: Blood flow clustering and applications invirtual stenting of intracranial aneurysms. *IEEE Transactions on Visualization and Computer Graphics 20*, 5 (May 2014), 686–701. doi:10.1109/TVCG.2013.2297914. 1, 3

[Ram72] RAMER U.: An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing 1*, 3 (Nov. 1972), 244–256. doi:10.1016/S0146-664X(72)80017-0. 3

[RD95] RONTOGIANNIS A., DIMOPOULOS N. J.: A probabilistic approach for reducing the search cost in binary decision trees. *IEEE Transactions on Systems, Man, and Cybernetics 25*, 2 (Feb. 1995), 362–370. doi:10.1109/21.364828. 7

[San04] SANTALÓ L. A.: *Integral Geometry and Geometric Probability*, 2nd ed. Cambridge University Press, UK, 2004. doi:10.1017/CBO9780511617331. 7

[Sar09] SARITAŞ E. U.: *High-resolution Diffusion MRI of Targeted Regions*. PhD thesis, Stanford University, USA, 2009. URL: https://searchworks.stanford.edu/view/12303425. 1

[SBGC20] SANE S., BUJACK R., GARTH C., CHILDS H.: A survey of seed placement and streamline selection techniques. *Computer Graphics Forum 39*, 3 (June 2020), 785–809. doi:10.1111/cgf.14036. 2

[SBMN16] SCHAUER J., BEDKOWSKI J., MAJEK K., NÜCHTER A.: Performance comparison between state-of-the-art point-cloud based collision detection approaches on the CPU and GPU. *IFAC–PapersOnLine 49*, 30 (Nov. 2016), 54–59. doi:10.1016/j.ifacol.2016.11.125. 2

[Sca11] SCAGLIARINI A.: Geometric properties of particle trajectories in turbulent flows. *Journal of Turbulence 12* (June 2011), Article N25, 11 pages. doi:10.1080/14685248.2011.571261. 1

[Sto61] STONE H.: Approximation of curves by line segments. *Mathematics of Computation 15*, 73 (Jan. 1961), 40–47. doi:10.2307/2003089. 5

[Sze10] SZELISKI R.: *Computer Vision: Algorithms and Applications*. Springer, London, 2010. doi:10.1007/978-1-84882-935-0. 10

[TMWS13] TAO J., MA J., WANG C., SHENE C.-K.: A unified approach to streamline selection and viewpoint selection for 3D flow visualization. *IEEE Transactions on Visualization and Computer Graphics 19*, 3 (Mar. 2013), 393–406. doi:10.1109/TVCG.2012.143. 2

[VB96] VASSILICOS J., BRASSEUR J. G.: Self-similar spiral flow structure in low Reynolds number isotropic and decaying turbulence. *Physical Review E 54*, 1 (July 1996), 467–485. doi:10.1103/PhysRevE.54.467. 1

[VCI20] VIOLA I., CHEN M., ISENBERG T.: Visual abstraction. In *Foundations of Data Visualization*, Chen M., Hauser H., Rheingans P., Scheuermann G., (Eds.). Springer, Berlin/Heidelberg, 2020, ch. 2, pp. 15–37. doi:10.1007/978-3-030-34444-3_2. 3

[VI18] VIOLA I., ISENBERG T.: Pondering the concept of abstraction in (illustrative) visualization. *IEEE Transactions on Visualization and Computer Graphics 24*, 9 (Sept. 2018), 2573–2588. doi:10.1109/TVCG.2017.2747545. 1, 3

[VW90] VISVALINGAM M., WHYATT J. D.: The Douglas-Peucker algorithm for line simplification: Re-evaluation through visualization. *Computer Graphics Forum 9*, 3 (Sept. 1990), 213–225. doi:10.1111/j.1467-8659.1990.tb00398.x. 3, 5

[WEG87] WOLD S., ESBENSEN K., GELADI P.: Principal component analysis. *Chemometrics and Intelligent Laboratory Systems 2*, 1–3 (Aug. 1987), 37–52. doi:10.1016/0169-7439(87)80084-9. 8

[WESW14] WANG Z., ESTURO J. M., SEIDEL H.-P., WEINKAUF T.: Pattern search in flows based on similarity of stream line segments. In *Proc. VMV* (2014), The Eurographics Association, Goslar, Germany, pp. 23–30. doi:10.2312/vmv.20141272. 3

[WH06] WALD I., HAVRAN V.: On building fast kd-trees for ray tracing, and on doing that in *O(N log N)*. In *Proc. Symposium on Interactive Ray Tracing* (2006), IEEE Computer Society, Los Alamitos, pp. 61–69. doi:10.1109/RT.2006.280216. 2, 5, 6

[WS06] WOODRING J., SHEN H.-W.: Multi-variate, time varying, and comparative visualization with contextual cues. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (Sept./Oct. 2006), 909–916. doi:10.1109/TVCG.2006.164. 10

[WWW*18] WANG F., WALD I., WU Q., USHER W., JOHNSON C. R.: CPU isosurface ray tracing of adaptive mesh refinement data. *IEEE Transactions on Visualization and Computer Graphics 25*, 1 (Jan. 2018), 1142–1151. doi:10.1109/TVCG.2018.2864850. 3

[YEII12] YU L., EFSTATHIOU K., ISENBERG P., ISENBERG T.: Efficient structure-aware selection techniques for 3D point cloud visualizations with 2DOF input. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (Dec. 2012), 2245–2254. doi:10.1109/TVCG.2012.217. 10

[YEII16] YU L., EFSTATHIOU K., ISENBERG P., ISENBERG T.: CAST: Effective and efficient user interaction for context-aware selection in 3D particle clouds. *IEEE Transactions on Visualization and Computer Graphics 22*, 1 (Jan. 2016), 886–895. doi:10.1109/TVCG.2015.2467202. 10

[YWSC12] YU H., WANG C., SHENE C.-K., CHEN J. H.: Hierarchical streamline bundles. *IEEE Transactions on Visualization and Computer Graphics 18*, 8 (Aug. 2012), 1353–1367. doi:10.1109/TVCG.2011.155. 1, 3

[ZCL08a] ZHANG S., CORREIA S., LAIDLAW D. H.: Identifying white-matter fiber bundles in DTI data using an automated proximity-based fiber-clustering method. *IEEE Transactions on Visualization and Computer Graphics 14*, 5 (Sept./Oct. 2008), 1044–1053. doi:10.1109/TVCG.2008.52. 1, 3

[ZCL08b] ZHOU W., CORREIA S., LAIDLAW D. H.: Haptics-assisted 3D lasso drawing for tracts-of-interest selection in DTI visualization. In *Posters of IEEE Visualization* (2008). URL: http://vis.cs.brown.edu/docs/pdf/g/Zhou-2008-HAL.pdf.html. 10

[ZGH*17] ZHANG J., GUO H., HONG F., YUAN X., PETERKA T.: Dynamic load balancing based on constrained K-D tree decomposition for parallel particle tracing. *IEEE Transactions on Visualization and Computer Graphics 24*, 1 (Jan. 2017), 954–963. doi:10.1109/TVCG.2017.2744059. 3

[ZHWG08] ZHOU K., HOU Q., WANG R., GUO B.: Real-time KD-tree construction on graphics hardware. *ACM Transactions on Graphics 27*, 5 (Dec. 2008), Article 126, 11 pages. doi:10.1145/1409060.1409079. 2, 12

[ZSL*18] ZELLMANN S., SCHULZE J. P., LANG U., CHILDS H., CUCCHIETTI F.: Rapid k-d tree construction for sparse volume data. In *Proc. EGPGV* (2018), The Eurographics Association, Goslar, Germany, pp. 69–77. doi:10.2312/pgv.20181097. 3