Leopold Franzens University
Innsbruck
Department of Computer Science
Interactive Graphics and Simulation Group

PhD Thesis

Geometric Reconstruction and Visualization of Point Clouds by Second Order Tensor Neighborhoods

Dipl.-Ing. Marcel Ritter, BSc

supervised by
O. Univ.-Prof. Dr. Matthias Harders
O. Univ.-Prof. Dr. Sabine Schindler

February 18, 2021

Abstract

Point cloud data structures are widely used due to their application in data capturing, e.g. by depth cameras and laser scanners, and in physically based particle simulation. Typically, point cloud algorithms operate heavily in local neighborhoods and employ accelerating spatial data structures, such as grids or trees. In this thesis a second order tensor view on point cloud neighborhoods, similar to a covariance, is introduced and analyzed. Based thereon, computational performance was optimized, continuous geometries, such as lines, were reconstructed, and point cloud visualization was enhanced.

Strategies to optimize the tensor computation were explored by utilizing different spatial data structures on CPU and/or GPU: kd-tree, octree, uniform grid cell hashing and sorting, curvilinear grid filling, and a screen space based technique. These are tailored to different use cases, i.e. full numerical computation, visualization of fine grain point clouds, and visual optimization by vertex clustering. The grid hashing yielded a speed up of about 500 compared to a naïve parallel tree based implementation, also taking advantage of heterogeneous hardware systems.

An algorithm to reconstruct lines in noisy point clouds was developed and analyzed in depth. An eigenvector streamline integration of a vector field – derived from mixing eigenvectors at optimal neighborhood radii and angular weighted directions – reconstructs lines by merging multiple line lets. Therefore, a multi scale tensor was introduced to identify optimal radii, to detect start points for the integration, and to compute a novel noise rate. A comparison to a recent reconstruction method showed that it can compete in terms of quality as well as performance and, moreover, supports 3D line reconstructions.

A new visualization method for the multi scale tensor via color mapped images is utilized to reveal and optimize different choices within the tensor computation, i.e. weighting functions and points of reference. Further, point cloud data were explored by a single scale tensor visualization approach stemming from diffusion tensor magnetic resonance imaging. Examples are demonstrated on data of airborne light detection and ranging laser scans and particle simulations of an evolving cosmos including star birth modeling in gaseous regions of galaxy clusters.

Contents

1 Introduction

A point cloud is a set of points in spacetime. Point clouds are used nowadays as digital spatial representation of geometric objects, mostly in 3D space, e.g. in numerical simulations or as output from sensor devices. In simulation, particle methods have become widely used due to the improvements in parallel GPU computing. Pure particle methods, such as the smoothed particle hydrodynamics introduced by Monaghan (1988) have been picked up by the graphics community, see e.g. Solenthaler and Pajarola (2009). The position based dynamics presented by Macklin and Müller (2013) and, e.g. Bender et al. (2014) also operates on particles only. Hybrid methods mix particles with grid structures and were introduced by the physics community, as e.g. the particle in cell method 'pic flip' by Brackbill et al. (1988). They have also been used in computer graphics, e.g. by Zhu and Bridson (2005). Further, the material point method makes use of a particle in cell approach; see Sulsky et al. (1994) for an early engineering publication and Wang et al. (2020b) for recent advances by the graphics community. Those simulation approaches operate heavily on point clouds. In observation, points result from scanner devices: light detection and ranging (LiDAR) or camera based stereo matching methods generate point clouds from real world geometry. In LiDAR laser scanning points are computed by the known sensor position, laser direction, and time of flight of a laser echo see, e.g Toth and Jóźkòw (2016) and Foix et al. (2011). The Earth sciences and industry uses such devices for surveying or as sensors for autonomous cars. In camera based stereo matching, the 3D positions of correlating pixels in a stereo pair of images are computed; similar to the human visual system – each image representing one eye. Thus, 3D point clouds are created from images. Even, mobile phones come with multiple cameras to enable depth capturing. The stereo matching process is presented e.g. by Scharstein et al. (2001).

Point clouds are especially interesting in 3D (or higher dimensional) space. In 2D many algorithms can operate on images, and do so very effectively. In 3D the amount of data to represent information, e.g., in dense voxels becomes quickly unfeasible when domains get larger or resolutions higher. Here, sparse data representations have to be used, such as a point cloud. Data is stored only at certain explicitly specified locations. Thus, they also have been targeted for data compression of spatial data, e.g. to compress vector fields by Skala (2017) or for image compression by Daropoulos et al. (2020). Further, current research is carried out on point cloud compression itself, see Schwarz et al. (2019) or Wang et al. (2020a). For grid based images several algorithms exist to compute local features, such as edges or corners. They are employed in more complex tasks such as tracking or object detection. Here, deep learning has overtaken the performance for higher level tasks via abstracting important local and multi scale image features automatically, see e.g. Wei Shen et al. (2015) or Liu et al. (2020). In deep learning point clouds have not been as intensively targeted for investigation as images. But they are increasingly employed as data sources. For instance, graph based networks have been applied to enable recognition tasks on point based data e.g. by Qi et al. (2017) and Sheshappanavar and Kambhamettu (2020).

Independent of the creation of points clouds, often the point neighborhood information is either lost, insufficient, or not available (e.g. not stored or samples reordered). However, the neighborhood can be used to enrich the point cloud information, i.e. to enhance the visualization by highlighting geometric properties or by enabling lighting computation via an estimated “surface” normal. Especially when working with captured data the reconstruction of the original geometry from point sets is of interest, e.g., in the application of surveying and special effects. In the former, maps of landscapes, or the mapping of buildings and construction sites, and in the latter the creation of virtual equivalents of real world scenes, objects, or characters is of interest.

In this work, a novel method to analyze point cloud neighborhoods has thus been introduced and its application investigated. More precisely, a local 3D multi scale feature was derived, visualized, analyzed, and applied to a higher level task: 3D curve reconstruction. This work direction evolved from earlier work dealing with spacetime curvature visualizations, see Ritter and Benger (2010), as well as curvature computations on polyhedral meshes, see Mathews et al. (2010).

Due to this, a tensor formulation capturing the directions of point neighbors has been introduced, which is an equivalent of a covariance matrix. Further, weighting is employed to enhance numerical stability in noisy data. The method is then extended to multiple scales, especially to reduce user parameters and strengthen the automation of algorithms. The multi scale tensors are then used for visualization and geometric reconstruction, via eigenvector stream lines. The eigenvector stream line approach was inspired by techniques used in the medical imaging domain of fiber tracking of neurons in the human brain. A line reconstruction algorithm is developed and analyzed. Generally, neighborhood computations of unstructured data are costly in terms of run time performance. A nearest neighbor search by a brute force distance query, a naive implementation, results in a run time of $\mathcal{O}(n^2)$ for a 3D point cloud. Thus, approaches have been developed to improve the performance in the past by inventing several data structures and algorithms to speed up spatial searches. In this thesis, a number of performance optimizations are implemented and analyzed for the case of the tensor neighborhood also using modern GPU/CPU architectures. In short, the main contributions of the thesis are:

1. A multi scale 3D tensor based point cloud neighborhood formulation.
2. An analysis of different centroids and weighting functions used in the tensor computation.
3. An algorithm to reconstruct 3D lines from point sets based on neighborhood analysis and an eigenvector based stream line integration.
4. An interactive visualization, analysis, and reconstruction tool, using an extracted core library, also provided open source.
5. Performance optimizations of the neighborhood search using the OpenGL pipeline and a heterogeneous GPU/CPU pipeline via OpenCL.
6. Employing tensor splats to enhance the visualization of point cloud data.
7. An application to reconstruct power cables of airborne LiDAR laser scans.

The main research questions – striven to be answered – are:

1. Is an eigenvector streamline based method on top of a neighborhood tensor feasible to reconstruct lines in point cloud data?
2. How sensitive is the line reconstruction algorithm to noise? How is it made robust?
3. What is the effect of using different centroids and weighting functions in the tensor computation?
4. What geometric features and details can be reconstructed?
5. How far can user dependent control parameters be removed to allow a (nearly) fully automated reconstruction process?
6. What is the computational bottleneck and can it be optimized or tackled by GPU computation?
7. Can a tensor visualization method stemming from spacetime curvature be employed to enhance point clouds visually by highlighting its neighborhood properties?

The structure of the thesis is organized in a cumulative style. Paper publications are categorized and grouped in three main chapters: line reconstruction method, performance optimization, and application. Each chapter starts with an own introduction into the topic, called preliminary, and then presents the original paper

publications. Each paper provides its own method, related works, results, discussion and conclusion. The line reconstruction method chapter of the thesis is extended by an additional section showing material that was not added to any publication due to page count limitations, lesser relevance, or not fitting optimally to any paper's focus. After this introduction follows an overview section of related work presenting the most influencing works and utilized technologies of this research activity. After the main contents, the presented results and methods are critically discussed. The thesis is then finalized by a conclusion and future work section.
Following paper publications build the cumulative part of the thesis:

Ritter et al. (2021a)~ Ritter M., Schiffner D., Harders M. (2021) *Robust Reconstruction of Curved Line Structures in Noisy 2D/3D Point Clouds.* Visual Informatics, *under review.*

Content: A point cloud reconstruction method is presented for 3D curves. A hybrid vector field – computed based on geometric measures of a second order tensor multi scale neighborhood – is integrated by eigenvector stream lines. Different parameters are tested on examples and automated parameter runs using developed error measures. Results are compared to a recent reconstruction method and an application is demonstrated on LiDAR data.

Contribution: conceptualization (80%), methodology (85%), software (80%), validation (100%), investigation (100%), visualization (100%), data preparation (100%), draft writing (80%).

Ritter et al. (2021b)~ Ritter M., Schiffner D., Harders M. (2021) *Visual Analysis of Point Cloud Neighborhoods by Multi-Scale Geometric Measures.* Eurographics Conference, *conditionally accepted.*

Content: A visualization technique for multi scale second order tensor based geometric features of point cloud neighborhoods is introduced. Color maps of shape factors are used to illustrate test geometries as well as a LiDAR example data set. The influence on the geometric measures by employing different weighting functions and points of reference are presented as well as the effect of noise.

Contribution: conceptualization (90%), methodology (90%), software (70%), validation (100%), investigation (100%), visualization (100%), data preparation (100%), draft writing (90%).

Grasso et al. (2015)* Grasso I., Ritter M., Cosenza B., Benger W., Hofstetter G., Fahringer T. (2015) *Point Distribution Tensor Computation on Heterogeneous Systems.* Procedia Computer Science 51, p. 160–169, Elsevier

Content: The neighborhood tensor was optimized for computation on a heterogeneous compute system utilizing CPU and GPU resources. A grid hashing and bitonic sorting was chosen and implemented in OpenCL. Numerical results were compared to the reference implementation. Three methods for scheduling compute jobs to the devices were investigated. A speed up of about 500 was achieved, as compared to the reference implementation (GPU vs. CPU).

Contribution: conceptualization (25%), methodology (25%), software (30%), validation (30%), investigation (10%), visualization (50%), data preparation (100%), draft writing (20%).

Schiffner et al. (2014)* Schiffner D., Ritter M., Steinhauser D., Benger W. (2014) *Using Curvilinear Grids to Redistribute Cluster Cells for Large Point Clouds.* In: Proceedings of SIGRAD 2014, Visual Computing

Content: A curvi linear grid based point clustering for raw point clouds was derived employing accumulation per cell in an OpenGL compute shader. Tensor based shape factors are used in a point re-distribution step. Then, only cell representatives are drawn. The method reduces the point data for available GPU resources in real time, demonstrated on simulation and LiDAR data.

Contribution: conceptualization (50%), methodology (50%), software (10%), validation (50%), investigation (33%), visualization (80%), data preparation (75%), draft writing (40%).

Schiffner et al. (2013)* Schiffner D., Ritter M., Benger W. (2013) *Fast Normal Approximation of Point Clouds in Screen Space.* In: WSCG 2013 Conf. on Computer Graphics, Visualization and Computer Vision Communication Paper Proceedings, p. 21–28

Content: Three methods to compute normal vectors of raw point clouds in an OpenGL fragment shader in real time are investigated. A method for obtaining the covariance from a multi sample depth buffer is derived. Its minor eigenvector is employed as a normal vector. Results are validated on artificial point sets as well as on LiDAR data.

Contribution: conceptualization (50%), methodology (25%), software (10%), validation (33%), investigation (25%), visualization (40%), data preparation (25%), draft writing (38%).

Ritter et al. (2012)* Ritter M., Benger W., Biagio C., Pullman K., Moritsch H., Leimer W. (2012) *Visual Data Mining Using the Point Distribution Tensor.* In: VisGra – ICONS 2012, IARIA, p. 218–222

Content: The tensor splat visualization method was applied to the neighborhood covariance. First, artificial point clouds were shown. Then, data stemming from geoscience – airborne LiDAR and coast line shapes – were investigated. Finally, a concept for parallelization employing CPU and GPU resources was introduced.

Contribution: conceptualization (50%), methodology (40%), software (80%), validation (50%), investigation (60%), visualization (100%), data preparation (60%), draft writing (60%).

Ritter and Benger (2012)* Ritter M., Benger W. (2012) *Reconstructing Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field.* Journal of WSCG 20, p. 223–230

Content: A line reconstruction method was introduced. Its concept is based on fiber tracking in diffusion tensor imaging. It was adjusted and extended. A line was created by stream line integration in an eigenvector field of a weighted covariance. Integration schemes and weighting functions were tested on artificial point sets and, finally, a power cable was reconstructed in a LiDAR scan.

Contribution: conceptualization (70%), methodology (90%), software (90%), validation (10%), investigation (100%), visualization (100%), data preparation (100%), draft writing (95%).

The works marked by a superscript tilde ~ are *submitted manuscripts* and have not been published, yet. Similarly, a superscript asterisk * marks published material. In the following text, any citations to these papers are indicated with those symbols.

2 Related Work

Each publication denoted above holds its own section of related work. Here, a selected subset of the most important and influencing ones is presented, and the relation to our work indicated.

Diffusion Tensor Imaging: The 3×3 covariance, or tensor neighborhood, can be decomposed in three eigenvectors and eigenvalues. Westin et al. (1997) introduced three shape factors computed based on the tensor's eigenvalues: linearity, planarity, and sphericity. These describe the shape of an ellipsoid which can be used to visually represent eigenvectors and eigenvalues. They form a barycentric coordinate system – always summing up to 1.0 – directly applicable in visualization algorithms. In Westin's work the second order tensors describe diffusion directions of water in the human brain; derived from magnetic resonance imaging (MRI). He uses gray-scale image slices enriched with colored ellipsoids to visualize diffusion directions of the brain. Brain regions can be classified by those directions, i.e. white matter and grey matter; Kolb and Whisaw (2015). White matter mainly consists of elongated axons and has a dominant diffusion direction. In contrast, gray matter consist of cell bodies and other structures leading to quite uniform diffusion directions. Thus, sphericity is high in gray matter and linearity is high in white matter with axons going in similar direction. Either planarity or sphericity is high at axon crossings. In Westin et al. (2002) an additional arrow and marker is added to the ellipsoids. Also, brain data is analyzed at larger scales by summing up the tensors over larger spatial regions. They argue, that large scale structures can then be classified because summing up diffusion tensors yields better directional representations, instead of summing up the direction vectors. In this thesis, this idea is employed to the case of point set neighborhoods in 3D space: from a central point of reference the directions to the neighbors are converted to second order tensors and then summed up to an averaged tensor; Ritter et al. (2012)* and Benger et al. (2012) show visualizations of such point distribution tensors on astrophysical and geoscientific point set data. The there employed visualization technique is an extension of tensors splats introduced in Benger and Hege (2004). Tensor splats are an enhancement of drawing ellipsoid as second order tensor representatives. They allow visual blending/merging by using transparency, coloring, and texturing. Our proposed technique uses Gabor functions (Bergeaud and Mallat (1998)) for improved transparency scaling; highlighting dominant eigenvector directions by visually combining point wise splats to elongated fibers.

In the analysis of the diffusion tensor MRI (DT-MRI) the so called fiber tractography was introduced to reconstruct connected regions with high linearity of the diffusion tensor field. A line following the direction of the major eigenvector of the diffusion tensor, thus, reconstructs paths of axons in the human brain; Basser (1998) and Basser et al. (2000). Basser uses the major eigenvector of the diffusion tensor as a tangent to the line. By solving an initial value problem, starting from an initial location, lines are traced along the tangent vector field. Algorithmically, this is an eigenvector streamline; a first order integration line. Basser states that without a priori knowledge fiber tracts cannot overcome regions of crossings reliably but larger scale features can be reconstructed well. He uses a Runge-Kutta method for integration and has the advantage to operate on a uniform grid discretization where higher order interpolation schemes can be applied in a straight forward way. In this thesis linear structures in point clouds are reconstructed by employing such an eigenvector streamline integration to follow dominant eigenvector directions. Also, Runge-Kutta schemes were implemented. However, the mesh free structure of the point set data required different approaches for interpolation. Kernel based distance weighting was employed for interpolating point based data; similar to the method used in smooth particle hydrodynamics Monaghan (1988). Ritter and Benger (2012)* reconstructs lines discretized by point sets of LiDAR data via eigenvector streamlines, Runge-Kutta integration, and mesh free interpolation.

Geometric reconstruction – surfaces: Hoppe et al. (1992) is an early work on reconstructing surfaces from an unordered set of points. They employ covariance matrices and estimate the tangent planes. They define a piece wise signed distance function approximating the surface. Therefore, the orientation of the tangential plane need to by consistently aligned, which is achieved using a graph structure. The distance function itself can be discontinuous, but isosurface based reconstruction compensates for that. They shortly mention an effect of choosing different neighborhood sizes for the covariance (k-neighbors) dependent on noise in the data but do not go further in that direction. In this thesis, a weighted variant of the covariance matrix is employed. The need for consistent orientation is avoided by locally flipping eigenvectors during stream line integration. Berkmann and Caelli (1994) employ the 3D covariance as a basis to compute 2D covariances of normal vectors projected onto the tangential planes to create a second fundamental form, equivalent to a Gauss map. They use those to detect jumps, edges as well as planar, parabolic, and curved regions in range image data. They also point out that this improves noisy data handling in contrast to direct curvature or surface normal definitions. As the data targeted in this thesis may be very noisy the chosen covariance approach was further improved by distance weighting. No consistent parameterization on the geometry was required on the simple line geometry, and thus no work in the direction of mappings compatible with differential geometry was carried out. Taubin (1995) estimates principal curvature and principal directions of triangular meshes based on computing the eigenvectors and eigenvalues of a matrix representation of a curvature tensor. Though not directly related to point cloud reconstruction, the publication motivated the approach to encode point neighborhood directions into a tensor representation. McIvor and Valkenburg (1997) compare different methods to estimate normal vectors and curvatures on surfaces. They were interested in the performance with varying amplitudes of added noise, comparing several methods; among them finite differences and quadratic fitting variants. They also introduce a covariance technique but do not investigate it further. This thesis also uses a controlled test bed with added noise to quantify the resulting performance. Alexa et al. (2001) approximate a surface from a set of points by small local patches using a moving least square projection, i.e. the projection plane related to the covariance. They enable interactive rendering of smooth surfaces. In under sampled regions new points are created on the surface approximation. Thus, they visually blend points to a smooth surface using the rendering pipeline. They experiment with the bandwidth radius or patch size of the approximation and create different levels of smoothness. They state that the patch size could be dependent on local feature sizes or noise levels, but do not investigate further. In this thesis the feature size in noisy data is of special interest. The optimal scale selection is a key in the line reconstruction approach and the visual analysis of the multi scale images. Their idea of changing smoothness levels was further extended in other work. The proposed sphericity feature was used before in geometry reconstruction, e.g. in Pauly et al. (2002), where it is related to surface curvature. They use the measure to steer the number of points for a low resolution resampling of a high resolution point based surface. More points are generated in high curvature regions. Further, they use a moving least squares (MLS) surface for local surface estimation, which originally employs a static neighborhood size. They extend this towards a dynamic size by employing the k-nearest radius divided by 3 as radius for a local surface neighborhood. Different strategies on choosing the k-neighborhood are presented, e.g. using hierarchical subdivision. Also, an error measure to compare point sets via MLS surface resampling by average and maximum (Hausdorff) distances is introduced. The dynamic size is extended to a multi scale space of the sphericity measure in Pauly et al. (2003) which allows to control the smoothness of the reconstruction. Further in Pauly et al. (2006), a multi scale decomposition of the surface representation is carried out: summing finer details to a smooth base representation. Regarding these works, the targeted point clouds stem from sensor data, but come with low noise rates. This enables the linkage of the sphericity with the surface curvature. With larger amounts of noise, an initial filtering could become necessary. Otherwise, more points will be placed in high noise regions during resampling. In this thesis the same shape factor idea is employed to estimate the noise rate in the data by analyzing multiple scales. Using fixed k-nearest neighborhoods was avoided as they denote a dependency on the sample rate with respect to geometric feature sizes. Still, a minimum sampling should be available for robust automated reconstruction. Average and Hausdorff distances on sampled line geometries are employed as part of the proposed error metrics. Stream lines are integrated by a constant step size and the ground truth data sampled by a similar resolution. Ohtake et al. (2005) also reconstruct from point sets. They finally create a triangulated mesh based on covering spheres. First, data is preprocessed to compensate for varying sampling densities and measurement errors. Normals are compute by utilizing Hoppe et al. (1992). Then an adaptive sphere coverage is created and auxiliary meshes based on sphere intersections. A quadratic error function based on a weighted covariance is minimized to compute the adaptive radii of the covering spheres. The method is tested on point clouds with low noise. A limitation is sparsely sampled regions. In this thesis the SPH kernels are of a similar shape. Berger et al. (2013) summarize and benchmark ten point based surface reconstruction methods. They show that there was no superior method found yet with respect to all test cases including noise and data holes. Global methods are more robust in case of noise, while local methods perform better without. The work derives error measures via a point to surface correspondence. Thereof average and maximum distance and angular errors are computed. Those measures are utilized to evaluate the test cases. In this thesis similar error measures are developed for line reconstruction; and further extended with criteria to describe the completeness of reconstructions. The latter was not relevant for Berger et al. since some of the tested reconstruction methods required a closed surface (implicit surface formulation) and objects were valid manifolds with little noise and good sample coverage. Similarly, in this thesis well defined test cases were set up for exploration, automated parameter runs, and method comparison. Among the ten reconstruction methods is the work by Guennebaud and Gross (2007). They introduced a sphere fitting approach by algebraic instead of geometric spheres. Thus, a sphere is represented as isosurface of a scalar field. They include a smooth neighborhood size function and a normal vector into a constraint minimization problem. The adaptively fitted spheres locally reconstruct the surface. Based on this geometric representation Mellado et al. (2012) investigated point clouds, not only for an adaptive surface reconstruction, but also with respect to multi scale features. They compute the fitted algebraic sphere over multiple scales and create a scale space, including positive and negative curvature, a fitness value (based on all distances towards the fitted sphere in the neighborhood), the point distance, and the normal vector. Further, they investigate derivatives in the scale space and introduce a variation measure, enabling a simpler bandwidth or feature scale identification. They compare their scale space to a covariance based scale space introduced by Pauly et al. (2003). The scale space derived in this thesis is also a continuous covariance scale space, but it includes linearity (or planarity) together with sphericity. The fitness function of Mellado et al. (2012) is related to sphericity, but this is not directly included in their scale space visualization. Also, they require normals on the unordered points, which can be costly to compute, and which becomes increasingly difficult at high noise rates or for 3D curves. They show feature images of the scale space, which show some similarity to the multi scale feature images (MSFIs) in Ritter et al. (2021b)~. However, these are different, as they show positive and negative curvature, whereas the three shape factors are visualized in the MSFIs. Moreover, all shape factors are related to curvature but also to noise. The fitness function they derive is related to sphericity and noise, but not directly included in the images. In contrast, our MSFIs do not include any information about the sign of a possible curvature. They employ their scale space to highlight features on surfaces, and also show an application of smoothing a curve-like point cloud with varying noise (via bandwidth selection guided by the multi scale space). This is related to the optimal radius and direction selection in Ritter et al. (2021a)~, however, the demonstrated case is on a densely sampled smooth 2D curve with quite low jitter noise. In contrast, this thesis investigates considerably higher noise rates, and examines geometric features on 2D and 3D curves. Lejemble et al. (2020) extends on Mellado et al. (2012) for planar segmentation and surface reconstruction at multiple scales. They use the previously introduced scale space to cluster points at different scales. They link those clusters to reconstructed surface components to build a hierarchy. Components present over multiple scales are the main features of the object. They investigate the stability of the method, and develop an interactive tool, for visualization of stable structures, interactive segmentation, and reconstruction. They employ a multi scale analysis for an interactive 3D surface reconstruction tool. This relates to Ritter et al. (2021a)~ , where an interactive tool based on a multi scale analysis is developed, but there the focus is on 3D curve reconstruction in high noise.

Geometric reconstruction – lines: Devore et al. (2013) focused on surface reconstruction of data stemming from LiDAR surveying. Besides describing error metrics, they work on planar and quadratic fitting in cube subspaces. Cubes are found by an octree data structure, with further subdivision based on error and size criteria. They reconstructed different scenes/objects, such as a mountain, a building, an urban street scene, or a light pole. Moreover, they state that the topic of surface reconstruction did not get much attention on this kind of surveying data, yet. This observation by Devore et al. was another motivation for our own investigation. Moreover, they attempted to reconstruct poles with quadratical fitting surfaces (see their Figures 13 and 14), which does not appear best suited. In general, a line, maybe including an additional "cylinder" radius, would be a more compact representation in that case; or a volumetric representation more fitting for trees and bushes. Nevertheless, their work is related to the line reconstruction method developed in this thesis, i.e. in Ritter and Benger (2012)* and Ritter et al. (2021a)~. An extension of the geometric representation from a surface to lines and volumes, is especially fitting for the properties captured by the point distribution tensor's shape factors, see Ritter et al. (2021b)~. In addition to this, the cube clustering approach of Devore et al.– selecting locally reconstructable regions – also relates to the real time clustering approach presented in Schiffner et al. (2014). There the forming of "planar" curvilinear grid cells is encouraged by moving point references to neighbored cells to increase planarity. This could, e.g. improve on the "intersections" of the ground with the buildings of Devore et al. (2013) by better adjusting to the not axis aligned edges; without any over smoothing (as visible in their Figures 15 and 18).

Next, a few essential publications of line reconstructions from a set of points are surveyed. Zeng et al. (2008) introduce the parameter free algorithm DISCUR to reconstruct lines from a given 2D set of points. Points are directly connected, by realizing two main criteria. First, close points are more likely to be connected and, secondly, smooth lines are preferred. Algorithmically, a Delaunay triangulation is performed and, edges selected and merged into lines based on statistical criteria. They are able to reconstruct multiple lines, as well as lines with sharp corners, and open or closed curves. Further, they provide a comparison to three other methods: CRUST, NN-CRUST, and GATHAN. As they focus on connecting points to reconstruct lines, the approach falls into a different class of algorithms. The point sets are rather sparse and are more susceptible to noise. However, the illustrated examples only include distribution noise. They demonstrate the algorithm performance by first sampling ground truth lines, which are then reconstructed. In Ritter et al. (2021a)~ artificial geometries are created in a similar fashion; but extended towards additional options for noise distortions. Further, points are not connected directly in our introduced method. Only the initial seeds of stream line integration are chosen from the set of input points. Lin et al. (2005) use B-splines to reconstruct noisy 2D point clouds. They first compute a number of rectangles to divide the point set into sub spaces based on a uniform grid. A sampling radius is evaluated by Delaunay triangulation. The investigated point clouds are densely sampled with a large amount of out-of-axis noise. For each fitted rectangle quasi centric points are found. A "left" and "right" border of the point cloud is constructed as B-splines, and by taking opposite control points of the borders a final central interval B-spline curve is created. While the data sets are quite noisy locally, they have very defined borders; there is no outlier noise present in the data sets. However, they are able to reconstruct 2D curves with varying sample rates and varying jitter noise. The B-spline approach provides a smooth curve of the point cloud axis. The algorithm differs considerably from the approach in this thesis; but still, some similarities exist. Their subdivision into rectangular subspaces provides a local view; in this thesis, the development started in a very small local scale, which was later extended to larger overlapping subspaces for analysis. Global methods, such as the Delaunay triangulation utilized by Lin et al. (2005) and Zeng et al. (2008) were avoided, to preserve scalability for large data and allow fine grain parallelization. Philsu and Hyoungseok (2010) introduce a natural distance metric – a Gauss kernel – to grow from an initially selected point. This creates a reconstructed line. They use a principal component analysis to compute the initial direction from the neighborhood. They use a factor to control the "diffusion" and provide limits to the factor. Essentially, distance and the weighted off axis assist candidate selection, to grow from an initial point. In Ritter et al. (2021a)~ weighting functions are also employed in many aspects. The off axis weighting of Philsu and Hyoungseok (2010) relates to the angular weighted direction proposed in our work. However, the latter is independent of distance, within a close neighborhood. Moreover, in our work distance dependent directions are provided, according to the best multi scale major eigenvector directions, adjusted by a radial distance weighting function. Öztürk and Hasirci (2013) introduce a bandwidth selection for varying noise. They are able to optimize the regression of a third order polynomial to reconstruct the axis of a point cloud with varying thickness and curvature. They use a covariance matrix to analyze direction and "elongation"; also, they normalize the eigenvalues by dividing with their sum. A local nearest neighbor KNN search is performed until the normalized major eigenvalue exceeds a threshold of 0.9. Points are then ordered according to the eigenvector, and the regression lines computed. Subsequently, the point cloud is thinned and reconstructed. The method allows to handle crossings as well as point clouds with varying noise. Their elongation measure directly relates to the linearity of the shape factors of the point distribution tensor used in our work. Their normalization is similar to our normalization of the shape factors. However, the approach of Westin et al. (1997) creates a barycentric relation of the shape factors. Moreover, in very noisy data a limitation of the elongation with the 0.9 threshold might lead to an improper bandwidth selection. In the proposed method scoring function were preferred to decide on "good" elongated bandwidths (radii for optimal linearity) to enable relative comparisons and selections. Ohrhallinger and Wimmer (2019) recently presented a method extending their earlier HNN-CRUST line reconstruction. Their *Fitconnect* connects points in noisy 2D point clouds resulting in a clean line manifold. Different scales are considered in the local view to identify the size of noisy clusters. The algorithm is designed for 2D manifold geometries. Its core – preferring smoothness – was extended to close sharp corners in a second step. They successfully reconstruct 2D point clouds with varying noise and varying feature scales. However, in contrast to our proposed method they cannot handle crossings, use only a ground truth distance metric (coverage or completeness are not considered), and did not extend their method to 3D. The work in Ohrhallinger and Wimmer (2018) further extends the (later published) ideas in Ohrhallinger and Wimmer (2019). The result of *Fitconnect* is additionally smoothed, and points of the reconstruction are constrained to their normals and shifted to optimize for a minimal global angle. As a result, the zigzaggy nature of the reconstructions is improved.

Compared to the mentioned related work for line reconstruction Ritter et al. (2021a)~ further extends on applied noise types and error measures. Further, a different way to analyze for optimal feature scales via multi scale geometric features based on weighted covariances is proposed. The general approach of stream line integration separates the line representation from the underlying point cloud geometry; but still relates to some of the above utilized criteria for connecting points based on distances and angles. Further, our proposed method works in 2D and 3D, which is demonstrated on several test geometries.

Local point neighborhoods: Natale et al. (2010) utilize similar shape factors in 3D point clouds of LiDAR flash images and analyze local geometric properties (see their Figure 16). Based on line, plane, and volume criteria, a neural network decides on feature classification. The network comprises several layers, each representing a growing neighborhood size. The illustration is similar to the multi scale geometric measures proposed in Ritter et al. (2021a)~ ; extended to multi scale feature images in Ritter et al. (2021b)~. Lin et al. (2014) also worked on point cloud classification; with a focus on airborne LiDAR laser scanning. They make use of three equivalent eigenvalue-based shape factors for classification in a covariance neighborhood. They propose a different formulation normalizing with respect to the largest eigenvalue. Also, they introduce a weighting related to point cloud density to stabilize the features. Moreover, they train a support vector machine classifier to label point clouds. Besides employing similar shape factors, they replace the mean point of reference in their neighborhood by the geometric median and achieve a 3% improvement for classification with respect to a classical PCA. The geometric median was also proposed for multi scale feature images in our work in Ritter et al. (2021b)~ as it improves smoothness and reduces noise. Further, a weighted extension of the geometric median also showed better performance in the line reconstruction algorithm on LiDAR data. Weinmann et al. (2015) compare different classifiers on LiDAR data and employ 21 geometric features; among them the same shape factor as in Lin et al. (2014). They also include the sphericity features as presented in this work. Further, they adjust the neighborhood sizes to optimize for the “eigenentropy”; a Shannon entropy applied on eigenvalues of a PCA. Finally, they suggest to use a random forest classifier with geometric features in the size of the optimal neighborhood for the best trade off between run time and classification quality. Lu et al. (2017) propose the geometric median to improve noisy point clouds. They are able to preserve geometric features by updating point positions and normals based on the L1 median. Especially, they are able to better preserve edges and smooth regions. A similar preservation feature was observed in our work, when employing the geometric median as centroid for the covariance matrix computation. The multi scale features images exhibit smoother regions and better defined edges. Next, Wendland (2005) proposes to use polynomial radial distance functions with a compact support, i.e. the influence of the function is limited by a maximum radius. The final approximation of a function is obtained as a sum of the weighted compactly supported kernels. The compact support thus transforms a global optimization problem into a local one. Also Skala (2012) investigated using RBFs for reconstruction. He states that in many applications, functions, such as images or surfaces, are often over sampled. This fact can be exploited to reduce the size of the system of linear equations to be solved for an RBF (or CRBF) approximation. The resulting problem can be solved via least squares or a singular value decomposition instead. This is demonstrated on examples of image inpainting and surface reconstruction.

In this thesis, polynomial weighting functions were also included and investigated as candidates for reconstruction optimization.

Neighborhood search: Husselmann and Hawick (2012) employ grid hashing and sorting on agent based data (points). Their method performs well in multi GPU systems. For sorting, Merge sort and Radix sort were performed on the GPU. Grasso et al. (2015)* builds on this idea and generates a sorted and grid hashed point cloud to speed up the range queries for neighborhood tensor computation. The Radix sort is replaced by a Bitonic sort (Peters et al. (2011)); further optimizing on the previous method. Willmott (2011) extends to vertex clustering in triangular meshes. His method enhances previous work by taking attributes on the meshes into account. Vertices are not clustered if they do not share the same attributes; e.g. the same material ID. This relates to our vertex clustering approach presented in Schiffner et al. (2014)*. Especially, the idea of attribute constraints led to including additional parameters, such as the planarity in the point redistribution of cell clusters.

Data and visualization frameworks: The methods in Ritter et al. (2012)* and Ritter and Benger (2012)*, as well as in Benger et al. (2012) were implemented in C++, on top of the vish visualization framework (see Benger et al. (2004)). In vish, algorithms are organized as a graph of nodes. Each node is a software module which can be reused by other algorithms, and be combined by visual programming or scripting. This permits to concentrate on certain aspects or sub-tasks, with also a number of existing – mostly visualization related – nodes provided. Further, it allows to develop spatio-temporal algorithms independent of the underlying spatial discretization. The visualization framework itself is built upon the fiber-bundle data model (see Benger (2004)). It is a scientific data model, especially dedicated to spatio-temporal data.

A HDF5 based format captures the fiber-bundle data model in files – the so-called F5 format (see Ritter (2009)). HDF5 2020 (accessed 2020) is an open format that is widely used in the academic field[1]. It stems from the high performance computing community, targetting efficient and scalable data storage, easy access, exchange, and sustainability. It is realized as a container format, in the back end of academic software, e.g. for Matlab's standard '.mat' files[2]. The F5 format adds a scheme to the basic container layout dedicated to spatio-temporal data. It is organized in seven hierarchical layers - similar to directories in a file system – and follows principles of topology, differential geometry, and geometric algebra. The F5 stand-alone version was used in our work in Grasso et al. (2015)* for computational investigations on heterogeneous platforms, that share multiple CPUs as well as GPU cores for common computational tasks. The remaining publications: Ritter et al. (2021a)~, Ritter et al. (2021b)~ Schiffner et al. (2013)*, and Schiffner et al. (2014)*, use smaller stand-alone prototypes in OpenGL and C++.

[1]https://www.hdfgroup.org/solutions/hdf5
[2]https://www.mathworks.com/products/matlab.html (from version 7.3)

3 Curved Line Reconstruction by Tensor Neighborhoods

3.1 Preliminaries

3.1.1 Second Order Neighborhood Tensor

An n-dimensional tensor $\underset{\sim}{\boldsymbol{\theta}}$ of order m maps the Cartesian product of m n-dimensional vectors to a scalar:

$$\underset{\sim}{\boldsymbol{\theta}} : \underbrace{\mathbb{R}^n \times ... \times \mathbb{R}^n}_{m} \to \mathbb{R}. \tag{3.1}$$

A tensor provides a multi-linear mapping[1] and holds its own coordinate base. Any freely chosen base can be transformed into that of a tensor, and vice versa. This is especially useful for physical quantities as they can be represented with a tensor independently of the global or other coordinate systems. Tensors are differentiated by their order m: 0 – scalar, 1 – vector, 2 – matrix (second order), etc. When tensors are dependent on space(-time) they are referred to as a tensor field. For example, temperature on Earth can be represented as a tensor field of order 0 on a sphere geometry. A displacement vector field on a finite element mesh is a tensor field of order 1.

Second order tensors are usually represented in matrix form, with the dimension related to the space(-time) they live in. In continuum mechanics, e.g., a second order stress tensor, represented as 3×3 matrix, holds inner force magnitudes dependent on their directions. The related strain field follows the same representation. In general relativity a second order spacetime curvature tensor denotes a 4×4 matrix, capturing stretched and compressed distances – also dependent on direction – induced by gravitation. A fourth order tensor is e.g. used as a mapping between stress and strain tensors. Here, it represents material properties and is able to model non-homogenous behavior, as e.g. in wood. Note that the encoded direction information in a second order tensor has, per-se, no orientation.

[1]Equation (3.5) demonstrates an example with dimension $n = 3$ and order $m = 2$.

In the stress tensor a negative magnitude usually stands for pressure and a positive one for tension. But pressure (and tension) is not pointing in a specific direction, there is rather only an axis of the effect. The flux of force can be interpreted as bidirectional connection of points sharing the same type of stress (e.g. principal stresses).

Within this thesis, a key idea is to use a second order tensor to encode local orientations and magnitudes (distances) of point neighbors within point sets; inspired by prior work on the visualization of spacetime curvature tensor fields. Here, the tensor field is embedded into a global 'standard' 3D Euclidean space. A neighborhood tensor can, thus, be computed at any arbitrary location without the need for local coordinates or spacetime dependent coordinate transformations. Refer to e.g. Pahl and Damrath (2000) or Benger (2004) for an in-depth introduction of tensor properties and tensor analysis.

The neighborhood tensor in point clouds is computed as follows: from a point of reference, vectors pointing towards 'close' neighbors are uplifted to a higher tensor order (first to second order), summed up, and normalized. The dyadic product is used to uplift the tensor order, formally expressed as:

$$\boldsymbol{v}_i = \boldsymbol{p}_i - \boldsymbol{c} \tag{3.2}$$

$$d_i = |\boldsymbol{v}_i|/r \tag{3.3}$$

$$\underset{\sim}{\boldsymbol{t}} = \frac{1}{\sum_{i=1}^{N} \omega_T(d_i)} \sum_{i=1}^{N} \omega_T(d_i)(\boldsymbol{v}_i \otimes \boldsymbol{v}_i), \tag{3.4}$$

with $\boldsymbol{c}$ the point of reference (centroid), $\boldsymbol{p}_i$ the points of a neighborhood of size N and radius r, $\boldsymbol{v}_i$ the vectors to the neighbors, d_i the respective distances to the neighbors, $\otimes$ the dyadic (or tensor) product, and $\omega_T(x)$ a weighting function ($\mathbb{R} \rightarrow \mathbb{R}$). The function allows to weight closer neighbors to contribute more into the tensor sum and, thus, the tensor's properties. Figure 3.1 further illustrates the process.

The magnitude of the neighborhood tensor with respect to a certain direction can be computed by the product:

$$\ell = \boldsymbol{d}^T \underset{\sim}{\boldsymbol{t}} \, \boldsymbol{d}, \tag{3.5}$$

with $\ell \in \mathbb{R}$ a scalar value, $\boldsymbol{d} \in \mathbb{R}^3$ a normalized direction vector and $\underset{\sim}{\boldsymbol{t}}$ the neighborhood tensor. Thus, $\underset{\sim}{\boldsymbol{t}}$ provides the mapping:

$$\underset{\sim}{\boldsymbol{t}} : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}. \tag{3.6}$$

The neighborhood tensor is symmetric by construction and holds six distinct values. Thus, the tensor is positive definite and ℓ will always be equal or greater than zero,

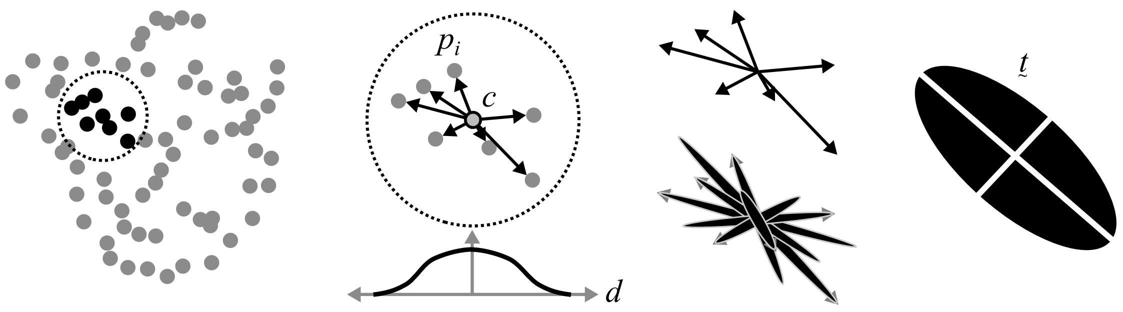


Figure 3.1: From a point set, a subset is selected (left). Then, direction vectors to the selected neighbors are computed, and possibly weighted by distance. Direction vectors are converted into second order tensors and summed up to one tensor (right); the point distribution tensor (or co-variance) is illustrated as an ellipsoid.

Note that the derived matrix is a co-variance matrix and its eigenanalysis denotes the principal component analysis, which is frequently used to reduce dimensionality of data, see e.g. Pourahmadi (2013). The formulation via tensors can be considered as a generalization, permitting application of tensor field algorithms: e.g. geodesics or tensor field visualization techniques.

3.1.2 Vector Field Integration

For the reconstruction of points being distributed in a line like formation, the method of streamline integration is employed. A streamline 'follows' vectors in a vectors field; i.e. the line's tangent is controlled by a vector field:

$$\frac{d}{ds}\boldsymbol{q}(s) = \mathcal{V}(\boldsymbol{q}(s)), \tag{3.7}$$

with $\boldsymbol{q}$ a mapping from the curve parameter $s \in \mathbb{R}$ to a point on the manifold M in $\mathbb{R}^3$, and $\mathcal{V}$ a vector field providing a tangential vector $v \in \mathbb{R}^3$ at each point on M. Throughout this thesis we focus on the Euclidean 3D space ($\mathbb{R}^3$); for a more general and precise introduction to metric spaces, tangential spaces, and integral curves, see e.g. Ritter (2011).

Generally, a streamline can be found by solving an initial value problem. Here, explicit numerical integration is employed on the major eigenvectors of the second order neighborhood tensor field, as illustrated in Figure 3.2 (Left). From a starting position and direction, a line is traced along the eigenvector field. An initial starting direction is required since the bidirectional eigenvectors provide no orientation. During integration, eigenvectors may have to be 'flipped' to be aligned with the starting (or current integration) direction.

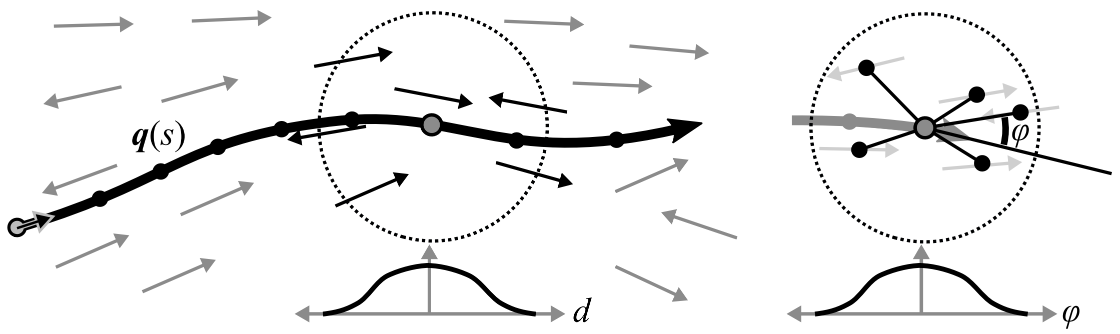


Figure 3.2: (Left:) An eigenvector streamline follows the vector field from a starting position. Opposing vectors are flipped during integration. A distance weighting function ω_I is employed to interpolate in-between the point based data. (Right:) An additional direction defined by angular weighting with respect to the current integration direction was employed as an extension in the line reconstruction.

In this thesis, the symplectic Euler, as well as several explicit Runge-Kutta (RK) schemes have been utilized for numerical integration: RK32, RK38, RK4, Fehlberg45, Merson45, Dormand-Prince5, and Dormand-Prince853 (Dormand and Prince, 1980; Hairer et al., 1993). With the exception of the latter, these methods were implemented as numerical solvers employing butcher tables for configuration. Concerning the RK methods, the first number in each name denotes the main order (e.g. RK32 is of order 3 and Fehlberg45 of order 4). A second number usually denotes the order of internal error estimation and step size control. In Ritter and Benger (2012)*, the reconstruction results could be further improved with the highly precise 8^{th} order DOP853 method. However, in the end the simpler and faster RK32 was finally chosen as the standard for our integration. Since the second order tensors are computed at given discrete points of a point set, interpolated values need to be provided for the streamline integration. Also in this process a weighting function ω_I is utilized, to interpolate the eigenvectors close to the current streamline position (indicated by the stippled circle in Figure 3.2 (Left)). Next, as an extension in our streamline integration, we also propose to employ the angles between the current integration direction and close neighbors, for finding the next direction. This idea is hinted at in Figure 3.2 (Right); see Ritter et al. (2021a)~ below for further details and the precise definition of the vector field. Also here, another weighting function ω_A is employed to weight normalized directions. Due to this, and the other weightings above, the selection and analysis of weighting functions has been a focus in the work of this thesis.

3.1.3 Weighting Functions

A number of weighting functions have been tested for Equation 3.4. Twenty-eight kernel style functions were selected: ranging from linear ramps, over polynomials up to order five, to exponential and sine based variants. In addition, parameter-free kernels originating from smooth particle hydrodynamics (Monaghan, 2005), compactly supported radial basis functions (Wendland, 1995; Skala, 2012), and animation easing functions (Penner, 2020) were employed and tested. Figure 3.3 (Left) illustrates a subset.

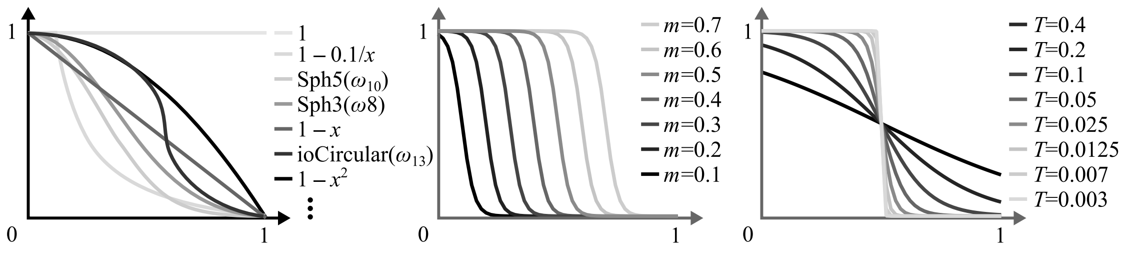


Figure 3.3: (Left:) Subset (7 of 28) of weighting functions employed as $\omega(x)$ in Equation 3.4, as well as for data interpolation. A simplified version of the Fermi-Dirac function showed best performance (see Equation 3.8). The shape can be optimized via two parameters. (Center:) Variation of m, (Right:) Variation of T.

As another kernel option, the Fermi-Dirac function was investigated. It originates from quantum statistic, see e.g. Reif (1965), with parameters m and T:

$$\omega_{\text{fermi}}(x) = \frac{1}{e^{(x-m)/T} + 1}. \tag{3.8}$$

Note that any physical constants having been removed, since the function is only used as a normalized weighting kernel. The parameter m shifts the function along the abscissa and T blends the shape smoothly from a step function to a linear function (see Figure 3.3). This allowed to optimize the weighting function's shape by selecting its parameters, according to error measures in a quantitative evaluation.

A preliminary evaluation on a subset of seven weighting functions has been carried out in Ritter and Benger (2012)*. Later, a test bed for automated parameter variation was set up and the full set of functions employed. Testing was performed on an artificial setup on the reconstruction of a circle and a rectangle; the final reconstruction error was measured, and optimized through varying the parameters (see Ritter et al. (2021a)~ and Section 3.3.5 below). Based on this analysis, in the final line reconstruction setup two Fermi-Dirac functions were employed: I) $(m = 0.6, T = 0.1)$ and II) $(m = 0.05, T = 0.35)$; I) for ω_T in Equation 3.4 and II) for interpolation (ω_I and ω_A). Besides better error measures, the Fermi-Dirac function also showed better results in the visualization of multi scale shape factors. A comprehensive list of all weighting functions and their equations is provided in Appendix A.4. Next to the weighting functions, also the selection of centroids for the tensor computation was examined.

3.1.4 Neighborhood Centroids

Different strategies were tested for selecting the point of reference $\mathbf{c}$ in the tensor computation in Equation 3.4. One option is to directly employ the points of the point set itself. In that case we proposed to denote the tensor as the point distribution tensor (PDT). Other common choices are: mean $\overline{\boldsymbol{c}}$, weighted mean $\overline{\boldsymbol{c}}_\omega$, geometric median $\boldsymbol{c}_{L1}$, and weighted geometric median $\boldsymbol{c}_{L1\omega}$ (see also Figure 3.4). The centroids are formally obtained as:

$$\overline{\boldsymbol{c}} = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{p}_i \tag{3.9}$$

$$\overline{\boldsymbol{c}}_\omega = \frac{1}{\sum_{i=1}^{N}\omega_C(|\boldsymbol{p}_i-\boldsymbol{p}_1|/r)}\sum_{i=1}^{N}\omega_C(|\boldsymbol{p}_i-\boldsymbol{p}_1|/r)\ \boldsymbol{p}_i \tag{3.10}$$

$$\boldsymbol{c}_{L1} = \arg\min_{\boldsymbol{x}}\sum_{i=1}^{N}|\boldsymbol{p}_i-\boldsymbol{x}| \tag{3.11}$$

$$\boldsymbol{c}_{L1\omega} = \arg\min_{\boldsymbol{x}}\sum_{i=1}^{N}\omega_C(|\boldsymbol{p}_i-\boldsymbol{x}|/r)\,|\boldsymbol{p}_i-\boldsymbol{x}|, \tag{3.12}$$

with $\boldsymbol{p}_i$ the points in a neighborhood of size N around a point $\boldsymbol{p}_1$, r the neighborhood radius, ω_C a normalized radial weighting function, and $\boldsymbol{x}$ an arbitrary point within the neighborhood radius. The first point $\boldsymbol{p}_1$ of the set $\boldsymbol{p}_i$ is used as center in a radial range query.

The mean centroid $\overline{\boldsymbol{c}}$ is employed in the standard principal component analysis (PCA). The geometric median is a point within a (neighborhood) point set, where the distances to all other points are minimal. It is also known as $L1$ median, and has been found to be a robust global center of point sets (see e.g. Huang et al. (2013)). We compute c_{L1} with an iterative algorithm originally proposed by Weiszfeld (1937), in an implementation following Burt et al. (2009). Note that according to Beck and Sabach (2015), the original Weiszfeld algorithm was later also rediscovered by Kuhn and Kuenne (1962).

In the examined reconstruction of real-world point clouds generated with LiDAR, the weighted variants of the mean and median showed better behavior. The weighted mean stood out in automated abstract parameter runs (see Ritter

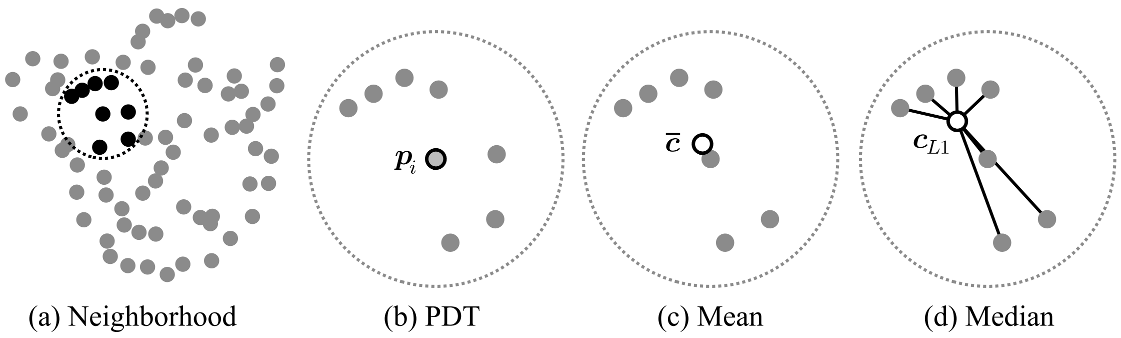


Figure 3.4: Different points of reference for the neighborhood analyzes were employed: a point itself (b), the mean (c), and the geometric median (d). The geometric median is defined as the point that minimizes the sum of all distances (indicated by the lines). The weighted variants are usually located in between the mean and the median (see Section 3.3.6).

et al. (2021a)~ for more detail). For visual exploration, the classical geometric median provided smoother and better defined multi scale images of geometric measures (see Sections 3.1.5 and 3.3.6 below, as well as Ritter et al. (2021b)~).

3.1.5 Multi Scale Geometric Measures

To select 'good' neighborhood sizes, especially with respect to a line reconstruction, geometric measures of the neighborhood tensor were computed, as shape factors according to Westin et al. (1997). They describe the shape of a tensor ellipsoid with three values:

$$\begin{aligned} C_S &= 3\lambda_1/L, \\ C_P &= 2(\lambda_2 - \lambda_1)/L \\ C_L &= (\lambda_3 - \lambda_2)/L \end{aligned} \tag{3.13}$$

$$L = \lambda_1 + \lambda_2 + \lambda_3, \tag{3.14}$$

with λ_i the eigenvalues of an eigenanalysis of $\underset{\sim}{\boldsymbol{t}}$, $\lambda_1 \leq \lambda_2 \leq \lambda_3$, and C_L denoting linearity, C_P planarity, and C_S sphericity. The three geometric measures form a barycentric coordinate system and, thus, are suited for tensor visualization (see Section 5.1.1). In Figure 3.5, three geometric shapes corresponding to a value of 1.0 for each of the three shape factors are illustrated, in a triangular arrangement.

High linearity in a tensor indicates that the points in the neighborhood are located on a line. Respectively, a high planarity indicates that points are distributed on a plane and high sphericity that points are homogeneously located in

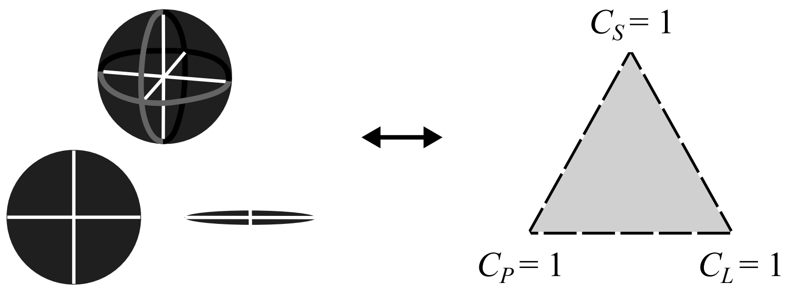


Figure 3.5: The shape factors span a barycentric coordinate system and, thus, can be arranged and interpolated over a triangle. The extreme cases – each single shape factor being 1.0 – are then located at the corners of a triangle. There, the other shape factors are 0.0, respectively.

3D space. However, for sampled points of a line, when curvature become high, e.g. at a corner of a planar rectangle, the shape factor at the corner is also dominated by planarity. The same happens at line crossings. Analogously, sphericity becomes non vanishing along an edge of a (point sampled) 3D cube. Inspecting the tensors of nearby points could help to decide in such cases. Clearly, the shape factors depend on the chosen radius.

To enrich the neighborhood information the tensor and its shape factors are computed over multiple scales, i.e. the geometric measures become functions of the neighborhood radius r: $C_S(r)$, $C_P(r)$, and $C_L(r)$. Figure 3.6 illustrates an example of the multi scale measures. The point of reference is located on a point sampled rectangle close to the corner (left). The radius dependent measures (center) show that for small radii the linearity is dominant. As soon as the radius grows over the corner the planarity fraction increases and, respectively, linearity decreases. The

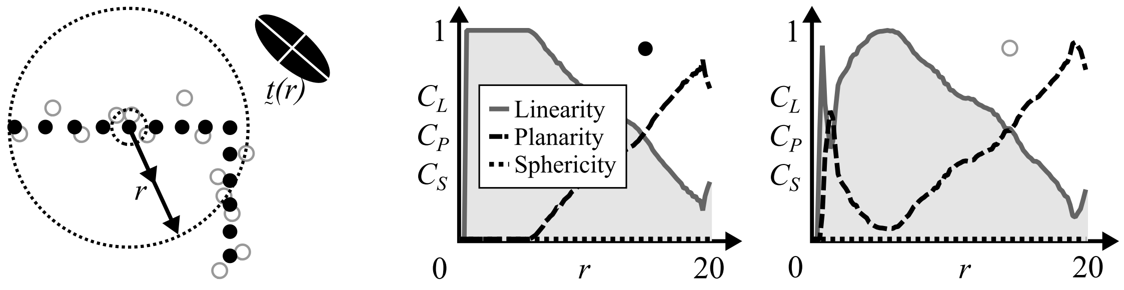


Figure 3.6: (Left:) Tensor computation with increasing radius illustrated in a point set with (white dots) and without noise (black dots). (Center:) shape factors as function over the radius r. At a small radius linearity is dominant. As the radius grows over the corner it decreases and planarity increases. (Right:) Shape graphs for the noisy point set.

graph on the right depicts the same scenario but with the sampled point locations being distorted by noise (light grey circles, left). Note the peak at the small radius scale, here linearity is high as only two points are within the radius. Then linearity quickly decreases to a local minimum and increases again, to a strong maximum. Here, the small local minimum directly visualizes the magnitude of the added noise. The strong maximum marks a radius, where the major eigenvector will be a feasible direction of local linear structure.

The multi scale geometric measures were explored and employed in several aspects within the thesis:

1. They were used for the evaluation of tensor computation parameters. They revealed the influence of the weighting functions: overall scale of the (local) minima and maxima as well as graph smoothness. Further, they were employed to show properties of the different centroids. This became especially apparent when the three graphs were mapped by a color map and used to illustrate spatially neighboring graphs. Ritter et al. (2021b)~ introduces these multi scale geometric measures and presents the measure for different weighting functions.
2. The multi scale measure graphs were utilized in the geometric reconstruction to identify 'good' candidate locations to start the eigenvector streamline integration. A scoring function is introduced, where a main component is an integral approximation of the linearity graph.
3. Further, linearity graphs were analyzed with respect to their local maxima to find radii of the tensor where the major eigenvector is a plausible direction to reconstruct a line. Therefore, a second scoring function is introduced.
4. Similar to the observation related to the noise amplitude in Figure 3.6, a noise rate estimator was developed operating on the multi scale measure graphs, analyzing the graphs with respect to local and global maxima and minima.

Ritter et al. (2021a)~ introduces the last three aspects in detail. The extension Section 3.3.4 shows more multi scale graphs dependent on employed weighting functions than presented in Ritter et al. (2021b)~.

3.1.6 Reconstruction and Visualization

Reconstruction: The proposed line reconstruction starts with a geometric analysis employing the multi scale measures as a pre-process on the point cloud. Therein, optimal linear directions and neighborhood radii are selected. Next, good candidates to start line integration are selected, also based on the multi scale measures, but adding a distance term. For each selection procedures a score function was designed. From multiple start points lines are integrated employing the RK32 scheme in parallel, forwards and backwards. Such line lets are merged and intersected as they are evolving. Different stop criteria have been developed to end line integration. A final pruning step deletes orphaned lines; yielding a final line reconstruction.

Error measures with respect to ground truth data were developed to evaluate the technique. A Hausdorff and mean distance metric, and criteria to decide on the completeness of a reconstruction were introduced. A set of artificial test cases was created in 2D and 3D; capturing important features of sampled lines: varying curvature, sharp corners, and crossings. Additionally, noise was added in a controlled way by adding: jitter, uniform speckle, distribution noise, and data holes. The error measures are evaluated on parameter runs, and in comparisons to recently introduced line reconstruction methods. Finally, a real world data set serves as a practical feasibility study, by automated detection and reconstruction of cables in a LiDAR scan. Ritter et al. (2021a)~ describes the reconstruction process in detail, below.

Visualization: Ritter et al. (2021b)~ delves into the visual exploration of point clouds employing the multi scale geometric measures. Visualizations thereof are used to explain optimization choices on the finally utilized weighting functions, centroids, and the exponentially growing radius scale. A color map and the overall visualization method by using multi-scale measure images on line-probes is introduced and demonstrated on small artificial test cases and, finally, on a LiDAR data set.

Robust Reconstruction of Curved Line Structures in Noisy Point Clouds

Marcel Ritter*[a,b], Daniel Schiffner[c], Matthias Harders[a]

[a]*Interactive Graphics and Simulation Group, Department of Computer Science, University of Innsbruck, Austria*
[b]*Airborne Hydromapping GmbH, Innsbruck, Austria*
[c]*DIPF | Leibniz Institute for Research and Information Education, Frankfurt, Germany.*

Abstract

Point-based geometry representations have become widely used in numerous contexts, ranging from particle-based simulations, over stereo image matching, to depth sensing via light detection and ranging. Our application focus is on the reconstruction of curved line structures in noisy 3D point cloud data. Respective algorithms operating on such point clouds often rely on the notion of a local neighborhood. Regarding the latter, our approach employs multi-scale neighborhoods, for which weighted covariance measures of local points are determined. Curved line structures are reconstructed via vector field tracing, using a bidirectional piecewise streamline integration. We also introduce an automatic selection of optimal starting points via multi-scale geometric measures. The pipeline development and choice of parameters was driven by an extensive, automated initial analysis process on over a million prototype test cases. The behavior of our approach is controlled by several parameters – the majority being set automatically, leaving only three to be controlled by a user. In an extensive, automated final evaluation, we cover over one hundred thousand parameter sets, including 3D test geometries with varying curvature, sharp corners, intersections, data holes, and systematically applied varying types of noise. Further, we analyzed different choices for the point of reference in the co-variance computation; using a weighted mean performed best in most cases. In addition, we compared our method to current, publicly available line reconstruction frameworks. Up to thirty times faster execution times were achieved in some cases, at comparable error measures. Finally, we also demonstrate an exemplary application on a real-world 3D light detection and ranging dataset, extracting power-line cables.

Keywords: Computational geometry, noisy point clouds, line reconstruction, automatic, adaptive control

1. Introduction

Datasets based on points as geometric primitives have become very popular in recent years. Noisy point cloud data are, for example, obtained from different capturing devices, such as depth cameras, during stereo matching, or in light detection and ranging (LiDAR) scanners (e.g. [1, 2]). Moreover, such data are also common in particle-based simulations (e.g. [3]). In this context, we currently focus on the automatic reconstruction of curved line structures from noisy 2D/3D point cloud data. To this end, we investigated the development of geometric measures for analyzing such data, as well as the automatic choice of optimal local point neighborhoods for further processing.

Our application area is the reconstruction of airborne scans. In these, data come with high noise, uncertainty, geometric diversity, as well as also at high volumes. As stated in [4], the reconstruction of airborne scans has received lesser attention than that of small-sized objects. Related work often focuses on small and well-defined objects, for which normal vectors may even be known (see e.g. [5]). This notion is also supported in the review in [6], where the authors point out the lack of a comprehensive evaluation specific for different sub-classes of reconstruction algorithms.

In our framework curved line structures are extracted based on a mesh-free streamline reconstruction strategy (e.g. [7]). Starting from automatically determined initial seed positions, important and dominant directions are identified in point cloud neighborhoods. The latter are based on weighted second order tensors of local point covariance. Size variations in the geometric features are captured through analysis over multiple scales. Based on the determined geometric measures, we automatically set algorithm parameters: the selection of starting points and integration directions, the radii of the directional search neighborhoods, as well as the direction evaluation. A large set of varying 2D and 3D geometries serves as a test-bed for automatic analysis, including different types of noise and data holes. Reconstruction errors are quantified with metrics that compare to the original undistorted point cloud geometry. Our heuristic framework is capable of robustly reconstructing 3D curved lines from distorted point sets; with the following main contributions:

- Analysis of geometric measures in point clouds to

*Corresponding author
Email addresses: `marcel.ritter@uibk.ac.at` (Marcel Ritter*), `Schiffner@dipf.de` (Daniel Schiffner), `matthias.harders@uibk.ac.at` (Matthias Harders)

automatically set neighborhood radii.

- Scoring functions to automatically select integration start points and optimized integration radii.
- Use of a hybrid vector field, for reconstruction via streamline integration.
- Noise rate estimation via geometric measures.
- New error measures for line reconstruction evaluation.
- Extensive automated analysis of parameter choices, such as covariance centroids and weighting factors.

After presenting the related work, we first provide an overview of the complete reconstruction pipeline in Section 3. Thereafter, we introduce the key elements of the framework in Section 4. First, the distance-weighted geometric measures are addressed; next, we present the hybrid vector field employed for streamline integration, based on Eigenvectors and angular-weighted directions; and finally, the use of multiple line-lets for streamline integration is introduced. In Section 5 we focus on the automation of the method: detecting good start point candidates and identifying optimal radii for Eigenvector pre-computation. Finally, performance evaluation and curved line reconstruction results are presented in Section 6. We introduce adequate error metrics, compare to recently proposed alternative techniques, and indicate the real-world application. In the next section we will cover prior research results, separated into several associated domains.

2. Related Work

Line Reconstruction in Points Clouds: In [8] an algorithm based on a Voronoi diagram of the point cloud was introduced; nearest neighbors were connected using an angle-to-Voronoi edge ratio and a topological condition. They were able to connect irregular point samples with sharp corners; however, existing points were connected directly, which is not appropriate for the noisy and densely sampled geometry targeted in our work. A Voronoi-based approach was also followed in [9], augmented with a human vision inspired criterion, directly connecting points. They also provide an overview of curve reconstruction algorithms, such as CRUST, or NN; and outperformed these with their DISCUR algorithm. They successfully performed line reconstructions of small point sets, including sharp corners, boundaries, and multiple components. However, they could not handle large and very noisy data, interpolation between samples, or 3D reconstructions. A method based on a uniform grid was developed in [10]. A sequence of fitting rectangles was computed containing points of the cloud. The center points of the rectangles were then connected and used along with border intersections to control a B-spline as curve approximation. While the method could handle jittered data, it was not robust against speckle noise; also, branches or sharp corners were not supported. Other works focused on fitting polynomials to noisy point clouds. In [11] a noise-adaptive smoothing term was added to the curve fitting; they employed a principal component analysis (PCA) in a pre-processing step for segmentation, and constructed piecewise curves, supporting branches and crossings. While our approach follows a different direction, similarities exist in the use of a weighted PCA, as well as the piecewise reconstruction strategy. Nevertheless, they only provided examples for 2D; and the method was hampered by outliers and data holes. Fitted B-splines were employed in [12], where the authors extended line and surface reconstructions to avoid user-defined regions. Nevertheless, for 3D they focused on surfaces, not on lines; and branches or crossings were not targeted in their work. A line reconstruction algorithm based on half disks and an angle-weighted probability function was suggested in [13]. From a start point, additional ones were selected and concatenated into a line. The method could handle line crossings; further, when noise was introduced, lines could still be reconstructed. However, the result would always be located at the outermost border of a curved point cloud. In our approach we also employ an angle-weighted directional component. In [14] lines were reconstructed via a PCA, using an adaptive radius selection. They employed a normalized maximal Eigenvalue to decide on an optimal radius, and then generated 3^{rd} order polynomials to reconstruct a line. Albeit, their method was limited to smooth lines and could not handle crossings, branches, or data holes. They extended their work in [15] by applying a minimal Euclidean spanning tree for point thinning. This enabled robust support for branches and crossings, but also introduced gaps at the intersections. In a different context, transmission lines were reconstructed in [16], via segmentation and piecewise regression. They focused on an automatic, robust, and precise method dealing with noise induced by wind; they compared to and outperformed the Hough Transform.

Most recent publications on 2D line reconstructions of point clouds can be found in [17], [18], and [19]. First the CRUST and NN approaches were extended, focusing on very sparsely sampled data. They select a local nearest neighbor and the opposite half-space neighbor, and prove that this permits connecting up to 60° sharp corners with a larger ϵ-sampling than previous methods. The authors further extended their introduced HNN-CRUST algorithm to *FitConnect*, by first estimating a local feature size and generating new points via blending in noisy regions. An extensive analysis on many examples is provided and compared to other work on line reconstruction. They were able to successfully deal with sharp corners and varying noise. A further extension improves line smoothness. However, their focus was on 2D-manifolds; T-junctions, crossings, and 3D reconstructions were not covered.

Surface Reconstruction in Point Clouds: In the early 1990s, tangent planes computed via covariance analysis to define implicit surface functions were introduced in [20].

Using these, they reconstructed surfaces with the Marching Cubes iso-surface algorithm. However, the covariance was not weighted and, thus, more prone to noise. In [21], several methods for local surface and normal estimations were compared. They analyzed the estimation performance on artificial geometries and added growing noise. They focused on quadratic fitting and mentioned a covariance-based technique, but did not include it in their comparison. A benchmark for surface reconstruction algorithms was provided in [6], comparing ten state-of-the-art approaches for point clouds with defined normals; among them: compactly supported radial basis functions (CSRBF)[22, 23], simple point set surfaces (SPSS), implicit moving least squares (IMLS), multilevel partition of unity (MPU). The results showed that there was no superior, general technique for surface reconstruction. Polynomial basis functions for distance weighting, as introduced e.g. for CSRBF in [23], were also included as weighting candidates in our work. In [24] they also aimed at the reconstruction of point sets, automatically adjusting to locally varying sampling and noise properties. A noise adaptive distance function was employed and an implicit function was found as a zero iso-line (or surface). The method yielded robust results to outliers and jittered point sets, but differed in that it required closed smooth shapes as prerequisite.

Tensor and Covariance Techniques: An algorithm to compute Gaussian and mean curvature on polygon meshes was presented in [25], where a tensor product of direction vectors to neighboring vertices was introduced to estimate a surface curvature. They constructed a curvature represented by a 2×2 matrix in the local surface tangent plane. The method is limited to meshes and not applicable to 3D point cloud data. In [26] a 2^{nd} fundamental form and a Gauss map estimation of surfaces was introduced, based on covariance in a local neighborhood. They detected regions in depth images and distinguished between planar, parabolic, and curved segments. They indicated that their method was robust to noise, due to the use of covariances, instead of the closed fundamental form. However, this only holds for low noise ratios. A similar weighted product to compute a covariance matrix in point cloud neighborhoods was proposed in [27] to estimate normals. The method operates on a point cloud, but the points are known to originate from a surface geometry, again, with low noise in the data. In [28] a voting technique was applied to make the normal vector estimation robust against noise. They employed shape factors for coloring and as a confidence measure. In our work, the shape factors are defined differently; our method achieves robustness by weighted covariance.

Line Following in Tensor Fields: Lines extracted from diffusion tensor fields have been used in the medical domain to visualize features in magnetic resonance images. Here, the connection between different regions of the brain are of interest, illustrated by fiber tracking. Early approaches employed Eigenvector streamlines following the major Eigenvector of the 2^{nd} order tensor [29], or a vector tensor multiplication to compute the next step direction [30]. Crossing fibers could attract streamline integration to switch to a different fiber. Thus, extensions have been proposed to favor the original direction, when at a crossing [31]. We also investigate direction selection strategies, but operate on mesh-less data instead of the uniform grids in diffusion tensor works.

Point Cloud Classification: In [32] shape factors of a PCA of indoor 3D flash LiDAR images were analyzed by a decision network for point classification. They parameterized shape factors by the number of neighbors in rather small neighborhoods. We use different shape factor definitions and parameterize by radius. Similarly, also in [33] shape factors were employed as input to a random forest classifier. They improved their true positives by 3% via relying on the geometric median in the data.

3. Reconstruction Pipeline Overview

Our framework for the extraction of curved line structures in noisy point cloud data is composed of several individual key steps (see also Figure 1):

1. **Geometric measures**: Geometric measures are locally computed via weighted covariances, in multi-scale neighborhoods of the point cloud data. An optimal neighborhood radius is determined in a point-wise fashion, based on these local measures.
2. **Start points**: Seed positions for the piecewise streamline integration are automatically selected based on the previously computed measures.
3. **Grow lines**: From the start points line-lets are grown in parallel, following streamlines in a hybrid vector field. They follow point samples, which are likely to have been sampled from continuous line geometry.
4. **Prune lines**: The growing line-lets are tested for crossings and corners, and extended to overcome data holes. Finally, they are pruned, and then concatenated into curved line structures as output.

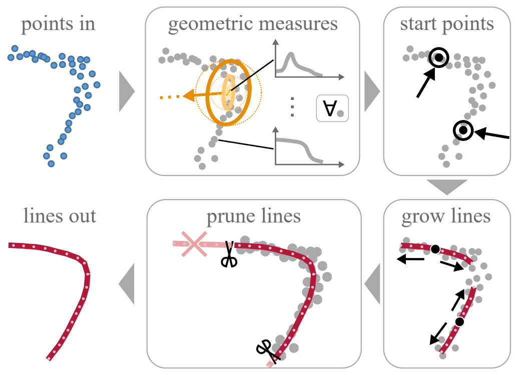


Figure 1: Main processing steps of the reconstruction pipeline.

Each pipeline step involves a number of parameters for control of the outcome. For most of these, we have carried out extensive automated initial tests to determine and fix best performing parameters. A few parameters remain that can be manually adapted, if desired, providing still some user-control over the pipeline outcome (see Section 5.3). In the following, we will first describe the individual steps in more detail, and then address the automation.

4. Reconstruction of Linear Structures

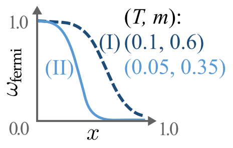
4.1. Geometric Measures

A central element of our reconstruction framework is numerical *geometric measures*, characterizing local *linearity*, *planarity*, and *sphericity*; following Westin et al. [34]. To obtain these measures for any location in a point cloud we first calculate weighted second order tensors $\underset{\sim}{t}$, which are constructed from all direction vectors within a neighborhood:

$$\underset{\sim}{t} = \frac{1}{\sum_{i=1}^{N} \omega(d_i)} \sum_{i=1}^{N} \omega(d_i)(\boldsymbol{v}_i \otimes \boldsymbol{v}_i), \quad \text{with} \tag{1}$$
$$\boldsymbol{v}_i = \boldsymbol{p}_i - \boldsymbol{c}, \quad d_i = |\boldsymbol{v}_i|/r,$$

where $\boldsymbol{c}$ is a specifically determined 3D center location (denoted below as *centroid*), $\boldsymbol{p}_i$ a point in the local neighborhood of $\boldsymbol{c}$ within neighborhood radius r, N the number of neighbor points for that radius, d_i normalized distances, and $\omega(x)$ a radial distance weighting function. Note that without using the weighting function and employing the mean position for $\boldsymbol{c}$, we would obtain standard covariance matrices of the local point cloud as these tensors, on which then a principal component analysis could be carried out. Also note that distance queries and computations in the point cloud are accelerated with an octree data structure.

We tested thirty different normalized scalar functions for the weighting in Equation (1). Possible candidates were, for instance, quadratic, cubic, compactly-supported RBF, or SPH smoothing kernels. In an initial controlled, automated study we computed reconstruction errors for these candidate functions; varying parameters such as neighborhood radius, number of points, jitter noise strength, etc. In total, 1.35 million parameter combinations were automatically tested, on a circle and a rectangle test shape, in about 7 hours of run time. In the end, the best performance was found for a weighting function similar to the Fermi-Dirac distribution (originating from quantum statistics, but here used without physical constants) [35]:



$$\omega_{\text{fermi}}(x) = \frac{1}{e^{(x-m)/T} + 1}. \tag{2}$$

The shape of the graph is controlled by two parameters $T \in [0.0, 0.4]$ and $m \in [0.0, 1.0]$. The former blends from a step to a linear function, while the latter shifts the function along the x-axis. In our automated experiments, we found weighting curve (I), with parameters $(0.1, 0.6)$ to be best for the tensor computation (also see plot above). Note that further below we will also make use of a similar weighting function for mesh-less interpolation; in that context, curve (II), with parameters $(0.05, 0.35)$ performed best. These values resulted from analyzing scatter-plots of the best reconstructions in the automatic testing mentioned above.

Further stabilization against noise can be achieved by choosing a suitable method for determining the centroid $\boldsymbol{c}$. Several strategies in the tensor computation above have been proposed in the past. Especially the presence of noise has a strong influence on the computation, which makes using the mean (MN) not necessarily the optimal choice. One option is to instead employ the geometric median (MD), which has been found to be robust against noisy data [36, 37]. Further, we also investigated the option to introduce a weighting into the mean and median computation (WMN, WMD); the centroid locations then vary dependent on the chosen weight function. For computing the geometric median we employ the Weiszfeld algorithm [38], which makes use of an exponential function in its iterations. Adding a weight function in the algorithm controls the strength of the iterative shift. Such weighted centroid variants yield locations in-between the geometric median and the mean. Finally, a further option could be to use the actual points of the point cloud itself as the "centroids" in the tensor estimation, thus skipping any mean or geometric median computation; this yields the so-called point distribution tensor (PDT) [39]. We compared these different options in performance tests with varying 3D test geometries (see Section 6.5). As a generally well-performing candidate, WMN with quadratic inverse stood out.

Employing the WMN as centroid and the Fermi-Dirac weighting function in the tensor calculation, we finally obtain our geometric measures. An Eigenvector decomposition of $\underset{\sim}{t}$ yields measures for local linearity, planarity, and sphericity, according to [34]:

$$C_L = (\lambda_3 - \lambda_2)/L, \; C_P = 2(\lambda_2 - \lambda_1)/L, \; C_S = 3\lambda_1/L, \tag{3}$$

with Eigenvalues of $\underset{\sim}{t}$: $\lambda_3 > \lambda_2 > \lambda_1$, $L = \lambda_1 + \lambda_2 + \lambda_3$, and associated Eigenvectors $\boldsymbol{e}_3, \boldsymbol{e}_2, \boldsymbol{e}_1$. Both the Eigenvalues and Eigenvectors will be central for the reconstruction of the curved line structures via vector field integration. It should be noted that both depend on the neighborhood radius, which controls the set of points considered for their computation. Both can vary significantly depending on how the neighborhood radius is selected. Therefore, we explored the measures further, as functions of radii.

4.2. Dependence on Neighborhood Radius

In the tensor computation above one could employ a fixed radius for the neighborhood of a certain point. However, depending on local feature scale this may or may

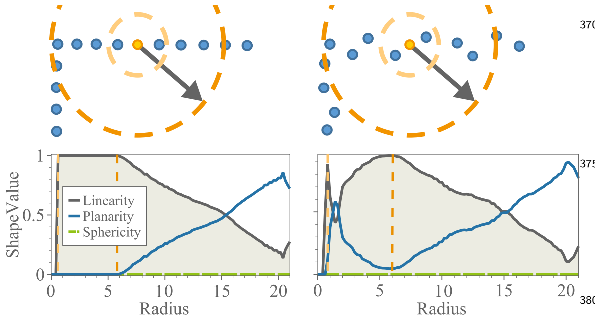


Figure 2: Geometric measures, computed at a centroid in small example 2D point clouds, with and without noise (top). The geometric measures (bottom) change with the radius; linearity is initially high and decreases when a corner is reached.

not include representative geometric information. Moreover, if 3D jitter noise is present, with an amplitude larger than the radius, the sphericity measure will be dominant, independent of other structures. Thus, it will be challenging, or even impossible, to find only a single radius that is optimal in all cases; especially, when sampling resolution changes; or linear features occur at different scales in the point cloud.

The change of the geometric measures dependent on a selected radius is visualized in Figure 2; the three measures are computed for two exemplary 2D point clouds (top), yielding graphs as functions of growing radii (bottom). Note that for this 2D case the sphericity measure yields zero; while the sum of the measures is one. The centroid (orange), for which the measures are computed, is located on a part of a point-sampled rectangle corner (blue), once without (left) and once with jitter noise (right). In the former case, as soon as the radius includes at least two neighboring points the linearity graph reaches a maximum of 1.0, and remains there until the growing radius covers the corner (stippled, larger dark-orange circle). At this point the linearity starts to decrease, while planarity increases. In the latter, noisy case, an initial peak at small scale marks the radius, at which the two closest points are covered, yielding high linearity. This measure decreases down to the jitter noise amplitude; and then increases, also until the corner is reached. Using different weighting functions in Equation (1) results in changes in the linearity graph. Therefore, we also evaluated the fitness of the weighting functions with regard to properties of the linearity graph: the maxima/minima at small radii, the plateau at large radii, and the overall smoothness of the resulting graph. Regarding the smoothness, the Fermi-Dirac (I) weighting showed a good behavior, especially, when employing the MD, WMD, and WMN centroids. Quadratic weighting also performed well, but occasionally produced discontinuity artifacts in the geometric measure graphs.

4.3. Vector Field Integration

In order to reconstruct curved line structures in point clouds, we employ vector field integration (see e.g. [7]). This denotes streamlines evolving along a certain direction in the mesh-less data, following the original geometry. Integration steps are numerically calculated using an explicit Runge-Kutta method. We tested various variants up to order five, but the influence of this was found to be small; thus, for our experiments we rely on a simple and fast third order RK32 scheme.

For the integration process, local direction vectors are required, which will be determined based on the previously computed Eigenvectors (note that the latter are bidirectional and may have to be flipped). We have analyzed different methods for computing directions; in the end two were combined into a hybrid approach.

Since we work with mesh-less data we have to either: interpolate between directions computed at existing data cloud points $\boldsymbol{p}_i$ or to directly compute a direction at an arbitrary position $\boldsymbol{x}_j$. The first approach for finding an integration direction at a location $\boldsymbol{x}_j$ computes an average of neighboring major Eigenvectors; weighted based on normalized distances:

$$\boldsymbol{d}_E(\boldsymbol{x}_j) = \frac{1}{\sum_{i=1}^N \omega(d_i)} \sum_{i=1}^N \omega(d_i)\, \boldsymbol{e}_{3,i}, \tag{4}$$

with major Eigenvectors $\boldsymbol{e}_{3,i}$ of N neighboring points, given by the selected radius; and d_i the normalized distances. The Eigenvectors are aligned according to the direction used in the previous integration step $\boldsymbol{d}_{t-1}$; i.e. they are flipped if they oppose the current streamline major direction. For the weighting function $\omega(.)$ we employ the Fermi-Dirac (II) function mentioned above, which led to slightly improved results in the interpolation. This first approach is useful for noisy data and larger scale features, providing a very smooth and robust direction selection. Since the Eigenanalysis has already been performed, no additional computation is required.

A second option for finding an integration direction is to employ the (normalized) sum of angle-weighted local difference vectors:

$$\boldsymbol{d}_A(\boldsymbol{x}_j) = \operatorname{nrm}\left(\sum_{j=1}^N \omega\left(1 - \frac{\cos(\varphi_j) + 1}{2}\right) \boldsymbol{v}_{j,i} \right), \tag{5}$$

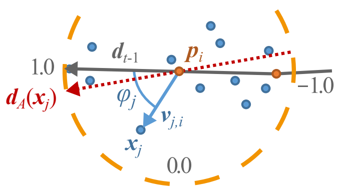


Figure 3: Angle-weighted direction vectors; the previous integration direction (gray) is used as reference. With it, cosines of angles are employed, obtained via dot products with all neighbor directions (one example shown in blue). The final direction (red) is the normalized mean of the weighted vectors in the neighborhood.

with $\boldsymbol{v}_{j,i} = \boldsymbol{x}_j - \boldsymbol{p}_i$ being difference vectors, between location $\boldsymbol{x}_j$ and N neighboring points $\boldsymbol{p}_i$, again for a given radius. The cosine term is computed using the angles φ_i between the normalized vectors $\boldsymbol{d}_{t-1}$ and $\boldsymbol{v}_{j,i}$ (see also Figure 3); finally, $\omega(.)$ again is the Fermi-Dirac (II) weighting function, this time evaluated with the angle-based term. Directions computed with the second method stay closer to the geometry, and overall reconstruction performance is better for smaller-scale features. As this method includes angle computations at the current streamline integration position it must be executed at "run-time" for each integration step. Later, we will obtain the final integration direction as a weighted average of $\boldsymbol{d}_E$ and $\boldsymbol{d}_A$ (see Section 5.2). For this, optimal weights and radii will be automatically determined based on the pre-computed geometric measures.

4.4. Piecewise Reconstruction

As an improvement of the previously outlined vector field integration, we further propose to grow multiple, smaller streamlines simultaneously, in breadth first fashion. Starting from different points, integrations are evolved in parallel, both in forward and backward direction, yielding so-called *line-lets*. In each integration step the evolving line-lets are checked for various conditions (and possibly terminated): i.e. exceeding a maximum step count, exiting the bounding box of the point cloud, passing beyond a maximum distance threshold, colliding with another line-let end, or intersecting another line-let segment. In case of a collision, line endings are merged; in case of an intersection, line endings are placed at the intersection point. In 3D, intersections of skew lines (i.e. segments) are determined by checking their closest points. Line segment endings and closest skew line segment points are merged, when their distance falls below a distance threshold: $\Delta_{\text{mrg}} h$, where h is the integration step size and Δ_{mrg} a weighting factor (set to 1.4). Furthermore, if during the integration a line direction from one step to the next changes by more than 70°, a new line-let is seeded and a branch created. Endings at intersections, collisions, as well as branch seeds are labeled as *closed*; whereas line segment endings generated due to exceeding the bounding box or the distance threshold are labeled as *open*.

In a final (pruning) step, points are deleted from all line-lets, starting from the open endings. The deletion process propagates backwards until the distance to the closest point in the point cloud is smaller than a user defined threshold. A second user controlled parameter is the maximum integration distance. Increasing the latter enables to overcome data holes and decreasing the former allows to delete long open branches. Finally, note that for efficient intersection computations, the points of the line-lets are also organized in a separate octree. In the next section, the automation of the process will addressed.

5. Process Automation

5.1. Selection of Start Points

A key element in our piecewise reconstruction is the setting of start points for multiple line-let creation. We propose to automate this according to the linearity graphs introduced above. Points for starting the vector field integration should be located on "good" linear regions in a point cloud. Thus, we analyze each linearity graph to determine such locations. As outlined above, linearity measures are determined for different radii. First, in order to accelerate the process, and to focus on features at different scales, we propose to increase the search radii r_k for this computation according to an exponential function (instead of a linear increase):

$$r_k = \Delta_{mdn}\ 1.5^k, \tag{6}$$

where k can be considered as a discrete index to the set of examined radii, proportional to $\log(r)$; the parameter Δ_{mdn} denotes the (approximate) median minimum distance between any two points in the point cloud (computed either for the whole cloud or for a random subset). Distances are computed for the N^{th}-closest neighbors (we employed $N = 6$). Based on these, the median for Δ_{mdn} is determined.

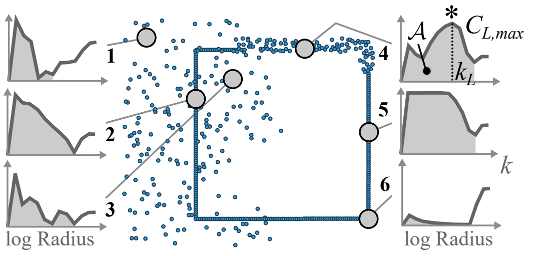


Figure 4: Six linearity graphs on a rectangle at distinct locations. The abscissas employ a log-scale for the radii. The approximated integrals $\mathcal{A}$, up to the last local minimum, as well as the maximum $C_{L,max}$ are used for start point detection.

We found that the integral area underneath a specific part of the linearity graph C_L already represents a robust feature for start point selection. To illustrate this Figure 4 shows graphs for six locations on an exemplary noisy point cloud of a rectangle (note that the abscissas in the subplots employ a log scale for r_k). Locations on a linear portion of the cloud exhibit a larger integral (see 2, 4, 5). In contrast, corner points (6) and points in noise (1, 3) typically show a smaller integral and graph maxima. Thus, we propose to estimate an approximate integral $\mathcal{A}$ of the linearity graph C_L for points in the cloud. It is computed by summing up linearity values for (exponentially growing) radii r_k, from zero to the last occurring local minimum in the graph; numerically approximating the analytic integral (shaded gray areas in plots). Note that from a certain maximal radius, linearity values will remain constant.

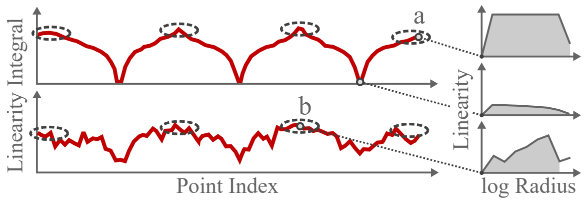


Figure 5: (Left:) Linearity integral sum values over (ordered) point indices, determined for the two point clouds shown in Figure 6. (Right:) Examples of linearity graphs for three selected points. Locations with a high sum (indicated by ellipses) are in linear regions and considered as good start points for the streamline integration.

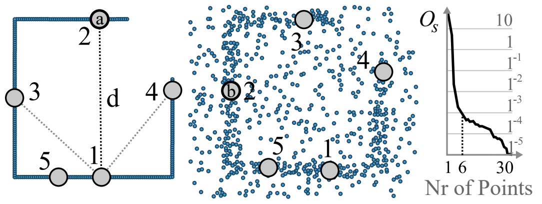


Figure 6: Start points selected for two example geometries – without and with noise. Points are chosen dependent on their score O_s; numbers indicate the order of selection. Linearity integrals at locations (a) and (b) are shown in Figure 5. On the right, the decreasing scoring function for consecutive candidates is depicted.

As a further illustration of the proposed feature, Figure 5 shows the linearity integral values computed at the centroids of the open rectangles depicted in Figure 6. This example was computed without (top) and with noise (bottom). As can be seen, the areas with large integral values (marked by ellipses) remain relatively robust. Therefore, large values of $\mathcal{A}$ may yield points in linear regions.

Based on the above, to obtain good start points we employ a score function, which can be computed either for all points or for a reduced random subset (initially with distance parameter d set to 1.0):

$$O_s = d \cdot N_1^{0.01} \left(C_{L,max} \cdot \mathcal{A}\right)^4, \tag{7}$$

where $C_{L,max}$ is the maximum linearity value in the examined interval of radii and N_1 the number of neighbors in the first non-empty neighborhood encountered (when growing the radius). Thus, orphaned points are slightly down-voted. Further, in this specific case a small computational improvement could be achieved by using the PDT for computing $C_{L,max}$, instead of using one of the other centroid computation approaches.

Employing the scoring function, we determine the highest-scoring initial point; setting $d = 1$. Next, using the latter as a seed, we then progressively identify additional start points, by evaluating the same (cached) equation, now with d being set to the shortest distance to all so far selected start points. Note that candidates can be skipped if a minimum number in N_1 is not reached (by default 2 points). In very noisy data, employing a minimum number of 6 points improved the results. The candidate in the current points with the highest score O_s is then added to the set of selected start points, and the process is repeated with the remaining ones. The process stops if a maximum number of starting points has been found.

Figure 6 illustrates identified start points on two geometries, one without and one with jitter noise. The numbers in the plot indicate the order in which candidates were selected according to the score O_s. As can be seen, the first start point is found in both cases at the center of a long rectangle edge, while corner locations and noise points are avoided. For the initial point (1), distances to start point candidates are shown by dotted lines. Larger distance is favored. Therefore, start points remain at a reasonable distance from each other. The graph on the right shows the decreasing score O_s for a progressively increasing number of selected start points. The score function is designed multiplicative, since the scale of d cannot be normalized; this still permits a relative sorting.

5.2. *Radii and Weights for Integration Direction*

As indicated above, the final streamline integration direction for the current step $\boldsymbol{d}_t$ will be determined as a weighted average of vectors $\boldsymbol{d}_E$ and $\boldsymbol{d}_A$:

$$\boldsymbol{d}_t = \mu\, \boldsymbol{d}_E + \left(1 - \mu\right) \boldsymbol{d}_A, \tag{8}$$

with μ being a blending weight. The computation of both direction vectors depends on the neighborhood radius, which we also propose to determine automatically. Similar to the automatic start point selection, the graph of the linearity measure can be utilized for this.

After a single numerical integration step, the streamline end position will usually not be located exactly at a point of the point cloud. Therefore, we first find the point $\boldsymbol{p}_i$ closest to the current integration position $\boldsymbol{x}_t$, within a search radius of $r_t = 1.25 \cdot \Delta_{mdn}$. At this point the precomputed linearity graph will be further examined. This permits finding an optimal neighborhood radius, for the current step in the streamline integration process.

Large radii would be a good selection in point cloud regions with noisy and coarse structures, while for regions with fine structures small radii should be chosen. As an example for the former, consider the maximum in the linearity graph for point 4 in Figure 4 (marked with a *); for the latter, consider the (first) maximum in the graph at location 6 in Figure 8. Thus, the analysis of the extrema in the linearity graph will be a means for automatically selecting the neighborhood radius.

Linearity graphs will generally exhibit several local minima and maxima at varying radii; typically, up to four extrema were present in our examined geometries. Accordingly, we designed a second scoring function. It is evaluated for each local maximum in the linearity graph, at the corresponding radius (with index k_j) for that specific maximum:

$$O_m = \left(1 - \frac{k_j}{K}\right) C_{L,j} + \frac{\mathcal{A}_j}{\mathcal{A}} - \frac{\delta}{2}. \tag{9}$$

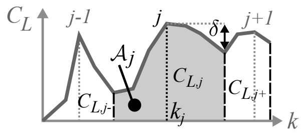


Figure 7: Elements of scoring function, based on local maximum at k_j and neighboring minima at k_{j-} and k_{j+} of the linearity graph.

Here, j denotes the radius index of the currently examined local maximum, for the radius given by k_j. $C_{L,j}$ denotes the respective linearity value, and K the total number of discrete radii indices considered for the current linearity graph. Further, $\mathcal{A}$ and $\mathcal{A}_j$ are again (numerically approximated) integrals of the graph. The former spans the same interval of radii, as explained above; however, the latter is only computed for an interval between the closest local minima, to the left and to right of j (indicated by $j-$ and $j+$, respectively). Finally, δ denotes the difference between $C_{L,j}$ and the larger of the two linearity values at the neighboring local minima, i.e. $\delta = C_{L,j} - \max(C_{L,j-}, C_{L,j+})$. For better illustration, the involved quantities are visualized in Figure 7.

The scoring function is comprised of three terms, which can take values ranging from 0.0 to 1.0; higher being better. Note that in the first term small radii are preferred (i.e. small k_j), since it is beneficial to adjust to small-scale features if they are present. Additionally, larger linearity values $C_{L,j}$ generally also indicate good radii candidates. In the second term, large local integrals $\mathcal{A}_j$ are favored, which also hints at linear structures. Finally, peaky maxima will be voted down via δ in the third term. The score O_m will be computed for all local maxima, and the radius associated with the highest score will be retained for the computation of $\boldsymbol{d}_E$ and $\boldsymbol{d}_A$. In order to illustrate this step, Figure 8 depicts obtained radii and directions $\boldsymbol{d}_E$ for two example point clouds. Line segments are shown, each with length given by the determined optimal radius and direction by vector $\boldsymbol{d}_E$. The radius, and thus length of line segments, increases in regions of large linear structures (1), in noise (2), and at low curvature (3); in contrast, it is smaller at corners (4), crossings (5), and fine details (6).

We blend the two direction vectors via μ. When the linearity and the integral at the chosen optimal local maximum j is large, then $\boldsymbol{d}_E$ would be a stable choice; otherwise $\boldsymbol{d}_A$ should be prioritized to follow close-by geometry. Thus, we propose to automatically set the blending parameter as:

$$\mu = \frac{1}{2}\,\frac{\mathcal{A}_j}{\mathcal{A}} + \Big(C_{L,j} - \frac{1}{2}\Big). \tag{10}$$

Furthermore, to ensure that both methods for direction selection still influence the final outcome, we additionally clamp this blending factor to the interval $[0.05, 0.95]$.

Finally, the step size of the numerical streamline integration is also automatically set to $h = 0.5\,\Delta_{mdn}$; note that this ensures on average a neighborhood size of about six points. In order to improve computational efficiency, we carry out in parallel the computation of geometric measures and Eigenvectors, as well as the radius selection. The direction $\boldsymbol{d}_E$ is determined at the current integration neighborhood and interpolated on the pre-computed Eigenvectors using the Fermi-Dirac (II) weighting. To compute $\boldsymbol{d}_A$ the previous integration direction and current streamline position have to be known; thus, it is obtained progressively.

5.3. User Control

While our framework exhibits several parameters, we propose to fix most of them. This is achieved either through the outlined automatic selection processes, or through extensive prior automated parameter analysis runs. In our current framework, a user has to manually control mainly three parameters:

- *MaxIterations*: The maximum number of streamline integration steps.
- *StartPointsNr*: The number of start points for linelet integration.
- *DistanceCutoff*: The maximally allowed distance from an integration point to the closest data point; used to overcome holes.

All remaining parameters are pre-defined; albeit, a user could adjust them still, if desired, for additional control. Examples are the pruning distance *DistancePrune*: the maximally allowed distance from a line-end to a closest point in the cloud, as well as *MaxRadius*: the maximum geometric analysis radius (set to $60\,\Delta_{mdn}$). We provide a user with an interactive visualization interface, for easy control and assessment of the reconstruction.

6. Analysis and Results

6.1. Point Cloud Test Geometries

For our analysis we employ six "2D" and four 3D test geometries: *Circle*, *Rectangle*, *Triangle*, *Line*, *Wave*, and *Crossing*; as well as *Elbow*, *Helix*, *Mikado*, and *Crossing3D*

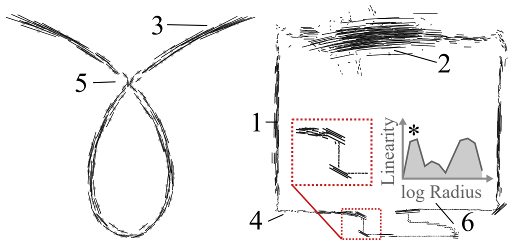


Figure 8: Optimal radii and directions $\boldsymbol{d}_E$ shown via line segments on two exemplary point cloud geometries.

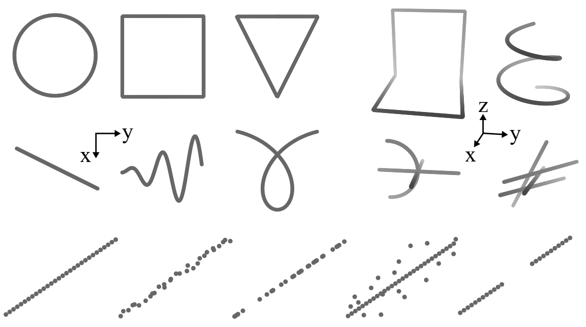


Figure 9: (Top/Left): Six test “2D” geometries: Circle, Rectangle, Triangle, Line, Wave, and Crossing. (Top/Right): Four 3D geometries: Elbow, Mikado, Helix, and Crossing3D (depth indicated by gray gradient). (Bottom): Added noise types; from left to right: undistorted reference, jitter noise, distribution noise, outlier noise, and hole(s).

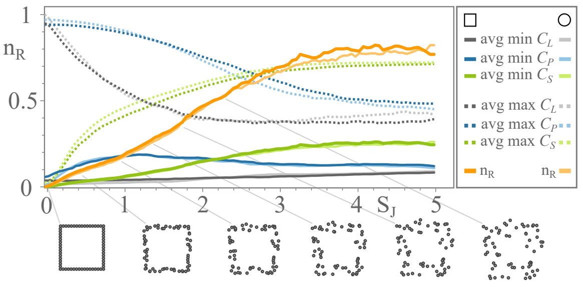


Figure 10: Noise rate n_R (orange), as well as averaged minimum and maximum geometric measures per jitter noise – for linearity (grey), planarity (blue), and sphericity (green) – for a rectangle (saturated colors) and a circle (desaturated colors) point cloud. The averaged minimum sphericity grows linearly at lower jitter values. Thus, it was chosen as a noise rate measure and scaled for normalization, see Equation (11).

(see Figure 9); these exhibit features such as varying curvatures, sharp corners, and line crossings. All are sampled as 3D point clouds (the “2D” geometries are displaced out of the plane with a cosine function). Different types of noise were added to these geometries; we employ four categories, inspired by noise that may occur during real-world data capture: unsteady trajectories and vibrations may produce *jitter* noise; shadowing or multiple scans may introduce density *distribution* noise; levitating particles or other small objects may produce random *outliers*; intensity cut-offs in sensors or occlusions may lead to *holes*. Figure 9 also illustrates the effects of these on a line segment (bottom). All effects are controllable in our framework by appropriate parameters: jitter amplitude S_j, distribution blend S_d, data hole start t_a and end t_e, and the number of additional, random outliers M.

6.2. Noise Estimate

Below, we will test the reconstruction process on various noisy example geometries. Note that only for our own, artificially generated data these noise parameters will be known exactly. In contrast, for arbitrary point clouds the latter have to be estimated. We found that the mean of all sphericity graph minima directly related to the noise in the data. Therefore, we propose a metric quantifying noise strength n_R in arbitrary data:

$$n_R = c \cdot \frac{1}{N} \sum_{i=1}^{N} \min_r C_{S,i}(r), \tag{11}$$

with $C_{S,i}(r)$ denoting sphericity of point with index i, at multi-scale radius r. Further, a constant scaling factor $c = 3.15$ was included, normalizing n_R for our test cases to interval $[0.0, 1.0]$. Figure 10 illustrates the change of this noise estimate, with respect to increasing 3D jitter noise. Plots are shown for the ”2D” test cases of Rectangle and Circle. Moreover, also the change in minimum and maximum of the three shape measures is indicated. At the bottom, a view of the rectangular point cloud with increasing noise is provided. For both geometries, n_R initially increases mostly linearly, later plateauing for higher, more extreme noise.

The noise rate n_R is independent of distribution noise and data holes. Below we will use this measure for evaluating and comparing results, dependent on noise magnitude. As indicated, also real-world datasets, for which the noise parameters are not known, could be compared in this way.

6.3. Error Metrics

In order to evaluate the automatic reconstructions in noisy point clouds, we require appropriate error metrics. An option would be to adapt existing error measures, such as in [40], to our case of curved line geometries. Nevertheless, we decided to develop metrics tailored to point-sampled linear structures, comparing differences to a ground truth in geometry, reconstruction length, as well as amount of coverage.

First, geometric differences are quantified using an adapted Hausdorff metric and an average minimal distance metric. For comparison, we employ as ground truth equidistant point samples Λ on the (known) test geometries. These are compared to our reconstructions given by the streamline integration points Ω, obtained with constant step size. For these we compute two errors – firstly, the point-based Hausdorff metric:

$$\mathcal{E}_H = \max\left\{ \max_{x\in\Lambda} \min_{y\in\Omega} d(x,y), \max_{y\in\Omega} \min_{x\in\Lambda} d(x,y) \right\}, \tag{12}$$

with $d(.,.)$ being the Euclidean distance between two points; and secondly, an average minimal distance metric:

$$\mathcal{E}_V = \frac{1}{N_\Omega + N_\Lambda} \left(\sum_{x\in\Lambda} \min_{y\in\Omega} d(x,y) + \sum_{y\in\Omega} \min_{x\in\Lambda} d(x,y) \right), \tag{13}$$

with N_Ω and N_Λ being the number of points in both compared sets. These two metrics indicate the local geometric quality of a reconstruction.

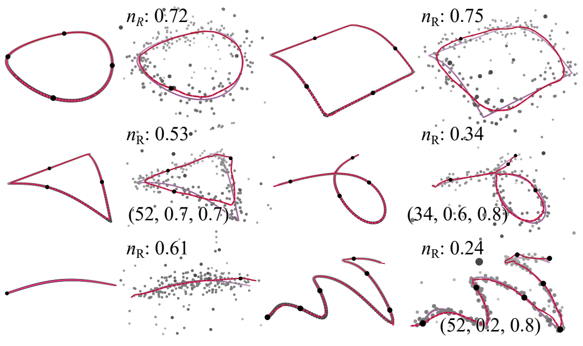


Figure 11: Automatic reconstructions (red lines) of six "2D" test geometries, without and with noise (with n_R specified). The number of points is $N = 174$. Black dots are automatically set start points. Noise cases are either generated with parameters $M = 87, S_j = 1.0, S_d = 1.0$, or according to values included inline via triplets.

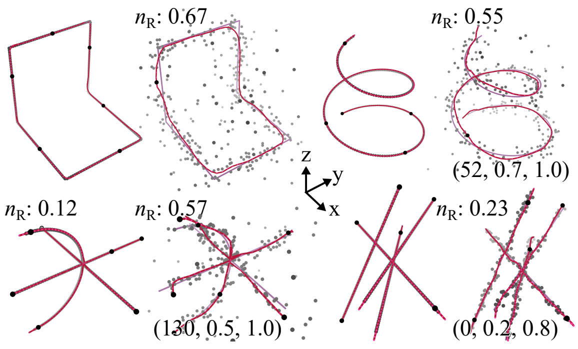


Figure 12: Automatic reconstructions (red lines) of four 3D test geometries, with and without noise (with n_R specified). Number of points is $N = 260$. Black dots mark detected start points. Noise parameters are shown as triplets; for the elbow: $M = 130, S_j = S_d = 1.0$.

As these two metrics remain stable even if only a part of a point cloud is reconstructed, we developed two additional measures, accounting for the reconstruction completeness. The first is given as the ratio between the (estimated) length of the reconstruction and the (known) length of the original line:

$$\mathcal{E}_{Len} = \text{len}(\Omega) \; / \; \text{len}(\Lambda), \quad \text{with} \qquad (14)$$
$$\text{len}(\Omega) = \sum_{i=1}^{N_\Omega - 1} |\boldsymbol{x}_{i+1} - \boldsymbol{x}_i|,$$

with $\boldsymbol{x}_i$ being consecutive points of the reconstruction. If the reconstructed line is shorter than the original one, then $\mathcal{E}_{Len} < 1.0$; which we consider as *incomplete* coverage. Moreover, if $\mathcal{E}_{Len} > 1.0$, then the reconstruction may, in some form, cover the original geometry several times (note that it may still not be fully reconstructed).

The second completeness metric determines the amount of coverage. For each line-let a corresponding arc-length interval $[s, e]_l$ is computed, with s and e being the start and end parameters (in arc-length) of Λ, and l the line-let index. The union of all intervals divided by the length of the original point cloud yields a completeness measure:

$$\mathcal{E}_{Com} = \frac{1}{\text{len}(\Lambda)} \sum\nolimits_{|\mathbf{U}|} (e_i - s_i), \quad e_i, s_i \in U_i \qquad (15)$$
$$\mathbf{U} = \bigcup_{l \in L} [s, e]_l.$$

$\mathcal{E}_{Com}$ approaches 1.0 if the union of all arc-lengths covers the full original geometry; if $\mathcal{E}_{Com} < 1.0$, then the reconstruction is again *incomplete*. Using these additional two metrics, a reconstruction is considered as unique, if $\mathcal{E}_{Len} \simeq \mathcal{E}_{Com}$.

6.4. Initial Exemplary Reconstructions

In order to illustrate the performance of our complete reconstruction pipeline we first show a smaller set of qualitative results. Examples of the previously described test geometries were reconstructed (see Figures 11 and 12), both without as well as with varying degrees of noise. If not explicitly indicated for a specific case, then jitter and distribution noise were both set to 1.0. Automatically determined start points are shown as black circles. As can be seen, the combination of start point selection and line-let integration allowed for handling of sharp corners and line crossings in these initial test examples. The adaptive selection of radius and integration direction made reconstruction of noise-free as well as noisy point clouds possible.

For these example geometries we also examined the previously mentioned error metrics. To this end, first 11 noise seeds were randomly generated for each of them. The noise parameters were not limited, possibly generating extreme cases, which could not be reconstructed. Thus, the maximum noise amplitudes were then lowered, until at least five visually successful (according to an observer) reconstructions were obtained. Then, the error metrics were computed per case, and averaged. The final results are compiled in Table 1; both for noise-free (left number) and

Table 1: Average error measures for 3D reconstructions of ten example cases, as in Figures 11 and 12. For each geometry, the numbers for the noise-free (left) as well as the noisy (right) case are indicated (separated by '/'). Only reconstructions deemed as successful were used for the error computation.

Geometry	$\mathcal{E}_{Len}$	$\mathcal{E}_H$	$\mathcal{E}_V$	Success	Time
	[%]	$[10^{-2}]$	$[10^{-2}]$	[%]	[ms]
Circle	100/99	4/68	2/27	100/91	8/17
Rectangle	98/96	14/108	2/36	100/82	8/17
Triangle	99/91	17/90	3/35	100/73	20/24
Crossing	103/105	10/95	3/26	100/73	6/14
Line	103/109	7/117	3/32	100/91	6/19
Wave	102/108	33/52	3/16	100/91	8/19
Elbow	99/100	18/109	4/35	100/91	13/27
Helix	101/106	12/110	4/26	100/82	11/26
Crossing3D	103/123	7/88	3/33	100/45	12/26
Mikado	104/108	4/39	2/10	100/45	14/13

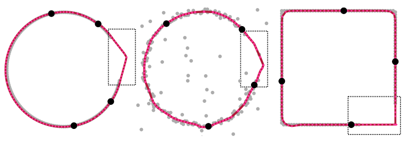


Figure 13: Reconstructions (red) of incomplete data. The stippled grey rectangles indicate the location of data holes in the original point clouds.

noisy (right number) cases. Here, success indicates the percentage of successful reconstructions, per test case, as assessed by an observer. As can be seen, when (stronger) noise is present, the success rate reduces; for these examples on average to about 60%. Geometries with corners exhibited higher errors ε_V. This is also due to smoothing effects, introduced by the weighting functions and smoothing radii. Further, Crossing, Line, and Wave were quite susceptible to outlier noise. The latter represented the most difficult case, especially if jitter noise became too high.

In a further test, we also qualitatively examined the robustness of the reconstruction to data holes. The latter have to be overcome by numerical integration across empty regions. In this case, either another line-let may be encountered or a user-specified length be exceeded. Figure 13 shows three qualitative examples on the circle and the rectangle geometry, again with and without noise. The size of the holes (indicated by stippled rectangles in the figure) is about 12% of the total line length.

6.5. Comprehensive Analysis of Reconstruction

6.5.1. Test dataset

In order to perform a more quantitative and broad performance evaluation, we obtained reconstructions with our described approach in an extensive batch processing. Overall, 4.45 million parameter runs were executed, each representing a reconstruction attempt. Runs were grouped by five centroid types (see Section 4), and 6 configuration sets. Each set with one centroid consisted of $127k$ runs. All ten test geometries were sampled with 128, 256 or 512 points, and reconstructed with varying noise. The distribution noise parameter S_d a well as the jitter parameter S_j were increased from 0.0 to 1.0; both with a step size of 0.1. Similarly M, the number of outliers, was increased from 0 to $0.4N$, with a step of $\lfloor 0.1N \rfloor$. Finally, due to the randomness in the noise generation, reconstructions were repeated seven times with different noise seeds.

6.5.2. Computational performance

The experiment was run on an Intel i7-9700 @ 3.60 GHz. Computation times as well as error metrics were recorded for each reconstruction. Over all parameter sets, the mean computation time for obtaining the geometric measures was 32.8 ms, while the integration took on average 7.8 ms. Figure 14 (left) provides box plots of the overall computation time for the reconstructions, per centroid type. Also,

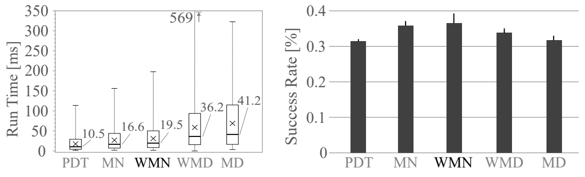


Figure 14: (Left:) Computation time of the reconstruction (multi-scale analysis and integration). (Right:) Averaged success-rates of 4.45 million parameter runs. Six configuration sets were tested with five different centroid settings.

the median times are given as numbers. The point distribution tensor is obviously the fastest, the geometric median the slowest. However, note that the PDT is anyhow used in all methods, as discussed in Section 5.1.

6.5.3. Success rate

Instead of relying on the judgement of an observer, we now chose to automatically determine reconstruction success based on the previously introduced error measures. We consider a reconstruction as *successful*, when $\mathcal{E}_{Len} \in [0.9, 1.2]$, $\mathcal{E}_{Com} \in [0.95, 1.05]$, and $\mathcal{E}_V \leq 0.25$. Figure 14 (right) illustrates the collected success rates for the different centroids. On average, the weighted mean (WMN) yields a 19% higher rate than the PDT (0.374 vs. 0.315). Note that in this study the success in general is lower, since considerable noise is added to the runs.

Next, Figure 15 indicates the success rates, per geometry. However, here only the best performing configuration set (of six) per centroid was kept. WMN performs better in almost all cases, with the exception of the Mikado geometry. It especially handles cases well that include corners; the employed quadratic inverse weighting function helps in stabilizing the reconstructions. Due to this, we recommend using WMN for the centroid computation, for generic noisy 3D point clouds. Unless otherwise specified, it has been employed in the majority of our presented cases.

Generally, Wave, Mikado, and Crossing3D are the most challenging cases for our reconstruction approach. The curved nature of the wave makes it very sensitive to jitter noise, usually failing when $S_j \geq 0.3$. At similar jitter amplitudes also the closely passing Mikado lines cannot successfully be reconstructed anymore.

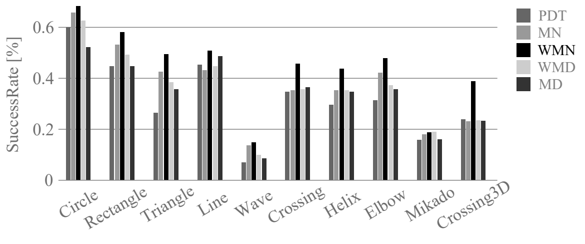


Figure 15: Success-rates per geometry, for the best performing configuration set of each centroid candidate. Overall, the weighted mean (WMN) performs best, followed by the mean (MN).

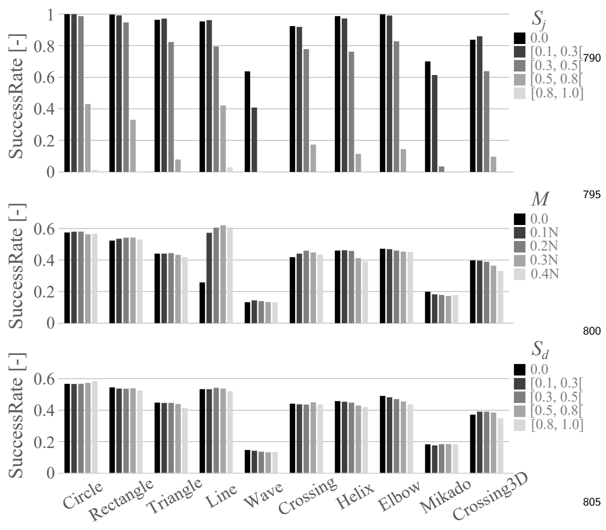


Figure 16: Effect of increasing noise on success rates, per geometry; using WMN and the best configuration set. Shown are jitter noise amplitude S_j (top), outlier noise M (center), and distribution noise S_d (bottom). Reconstruction performance is mostly independent of outlier and distribution noise. For $S_j \geq 0.8$ most reconstructions fail. Wave and Mikado are highly sensitive to jitter noise.

6.5.4. *Influence of noise*

Next, we examine the effect of noise type and parameter setting on the reconstruction results. In Figure 16 success rates are plotted, again separated by geometry. This time, rates are shown for different noise values/intervals; from top to bottom: jitter noise amplitude S_j, outlier noise M, and distribution noise S_d. Note that here, WMN was employed for the centroid computation. Further note, that a bar plotted for e.g. $S_j = 0$ includes the full ranges of the other noise parameters, i.e. in this case M and S_d. As can be seen, the success rate mainly depends on jitter noise. In contrast, changing distribution and outlier noise has a smaller effect. Also, as already discussed above, the Wave and Mikado represent the most difficult cases.

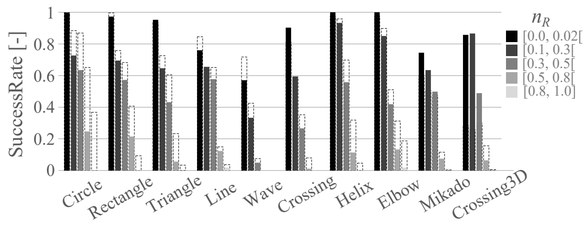


Figure 17: Success rates, plotted for varying noise rate intervals; again separated per geometry. Using a single noise rate facilitates overall comparisons of reconstruction success vs. noise, even for unknown datasets. Moreover, also effects of algorithm variations can be compactly visualized. Here, for instance, results are also shown for the variation of enforcing a minimum of 6 candidate points in the computation of O_s; here visualized with stippled bars. This yields improvements in the success rate, in presence of high noise.

6.5.5. *Comparisons using new noise rate*

Above we had introduced a new noise rate measure in Equation (11). It can be computed for arbitrary point clouds to characterize overall noise, thus also making comparisons easier. Figure 17 depicts success rates, per geometry, according to intervals of noise rate n_R. As expected, success rates drop when the noise rate increases. Again, the poor performance of the Wave example becomes apparent. In addition, also possible variations of our approach for special cases can be studied this way. As an example, for high noise cases using 6 candidate points for computing O_s could be enforced, as discussed above. As can be seen via the stippled bars in the plot, this sometimes would yield improvements.

6.6. *Comparison to FitConnect/StrechDenoise*

Next, we compared our approach to the recently published reconstruction methods *FitConnect* [19] and *StrechDenoise* [18]. Four challenging geometries introduced in their work were also tested in our framework; point clouds of a "monitor" outline, a spiral, a circle, and a "bunny". The examples include different geometric features, such as corners, curved and straight sections, sparse geometry, as well as variable jitter noise. All geometries were tested using the original geometries specified by the respective

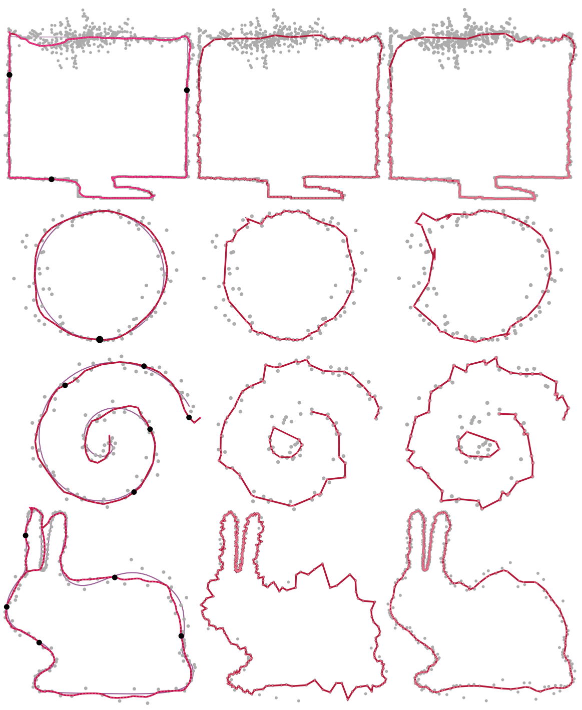


Figure 18: Comparison of the reconstruction of three geometries of [19] – using our approach (left), *FitConnect* (middle), and *StrechDenoise* (right).

authors. In addition, we also tested our geometry of a crossing with *FitConnect*.

For the experiment we used the authors' codes, which are publicly available online: [41] and [42]. Only minor modifications were made, to ensure comparable computation time measurements, vector graphics plots of results after computation, and loading of our crossing example. Figure 18 compares the reconstructions of four test cases, for our approach, *FitConnect*, and *StrechDenoise*. Our method yields smoother reconstructions; but corners may be smoothed out, unless line-lets meet. The reconstructions with *FitConnect* follow occasionally a more zigzaggy path, especially for lower noise rates. However, it can produce sharper corners. Regarding the bunny, our approach exhibited some artifacts: the small gap between the ears was not reconstructed correctly; a merged line resulted. Nevertheless, in contrast, the crossing lines example could not be reconstructed with *FitConnect* or *StrechDenoise*.

We compiled error measures and computation times for these experiments, comparing all three approaches in Table 2. Note that for this, the results from the *FitConnect* and *StrechDenoise* framework and the ground truth data were loaded into our framework to compute the error measures. We found that our method was capable of producing results of similar accuracy, while in most cases being faster.

Table 2: Error measures and timings of five reconstructions, comparing with *FitConnect (FitC.)* [19] and *StrechDenoise (StrD.)* [18]. Our method can be several times faster; with similar results. The best performing method is indicated in bold font.

Geometry	Method	$\mathcal{E}_H$ $[10^{-2}]$	$\mathcal{E}_V$ $[10^{-2}]$	$\mathcal{E}_{Com}$ $[10^{-2}]$	$\mathcal{E}_{Len}$ $[10^{-2}]$	Time [ms]
monitor	Our	02	0.3	100	**101**	**67**
	FitC.	*02*	***0.2***	*100*	*106*	*2420*
	StrD.	*02*	***0.2***	*100*	*103*	*2470*
circle	Our	**09**	04	100	103	**4**
	FitC.	*11*	***03***	*100*	***102***	*10*
	StrD.	*36*	***08***	*100*	***125***	*12*
spiral	Our	36	08	**100**	108	**5**
	FitC.	*31*	***06***	*75*	*106*	*23*
	StrD.	***20***	*07*	*80*	***99***	*34*
bunny	Our	39	13	90	**97**	7
	FitC.	*71*	*12*	***100***	*124*	***4***
	StrD.	***31***	***6***	***100***	***103***	*6*
crossing	Our	46	02	100	103	**9**
	FitC./StrD.	–	–	–	–	*>94*

A further experiment was carried out, comparing *StretchDenoise* and our method, when increasing the noise levels for a 2*D* circle. Figure 19 depicts six configurations of increasing jitter and outlier noise. The parameters, including the aggregated noise rates n_R, are indicated. *StrechDenoise* (bottom half) could handle jitter noise up to $S_j \approx 0.4$, with up to 30% additional outliers. Our approach (top half) was capable of handling $S_j \approx 0.5$, with up to 100% additional outliers. Moreover, by enforcing a 6-point neighborhood for computing O_s, this can be further improved to $S_j \approx 1.0$, with 150% outliers.

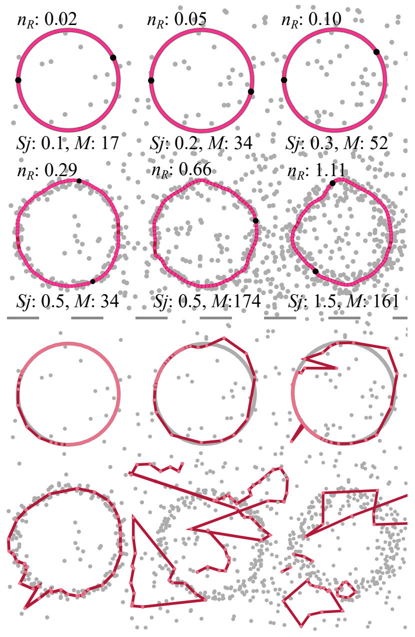


Figure 19: Comparison of a noisy circle with different strengths of noise: our approach (top half) and *StrechDenoise* (bottom half); the same noise settings are employed for both methods ($N = 174$). *StretchDenoise* exhibited difficulties with outlier noise, deteriorating when e.g. 30% outliers are present (bottom half, row 1, right). In contrast, our approach could handle jitter noise $S_j = 0.5$ and 100% additional outliers (top half, row 2, middle). Further, even strong noise with $S_j = 1.0$ and 150% can be handled, by enforcing 6 candidate points for O_s (top half, row 2, right).

6.7. *Application on Real-World LiDAR Data*

A smaller subset of 57*k* points of a massive LiDAR dataset was selected to test our method on real-world data; including houses, bushes, trees, and power lines. Figure 20 (left) illustrates the result, using the WMD and exponential radius growth. The algorithm extracts the main cable geometries of the high-resolution scan. Note also that coloring according to the median of the geometric features for each LiDAR point provides a good visual classification (Figure 20 (right)). The maximum analysis radius was chosen manually and set to 2.0, to capture all linearity scales. It was possible to extract the main linear structures from this larger, noisy real-world dataset. Nevertheless, note that an artifact appeared on the roof of the right house. Also, the integration continued onto the roof and followed some of its edges. Stopping criteria were not adjusted to this specific dataset. The WMN reconstructed the cable system successfully. Further, employing the WMD enhanced the reconstruction of the cable bundle as well as it enabled to integrate better along the roof surface edges: the gable and the border of the roof.

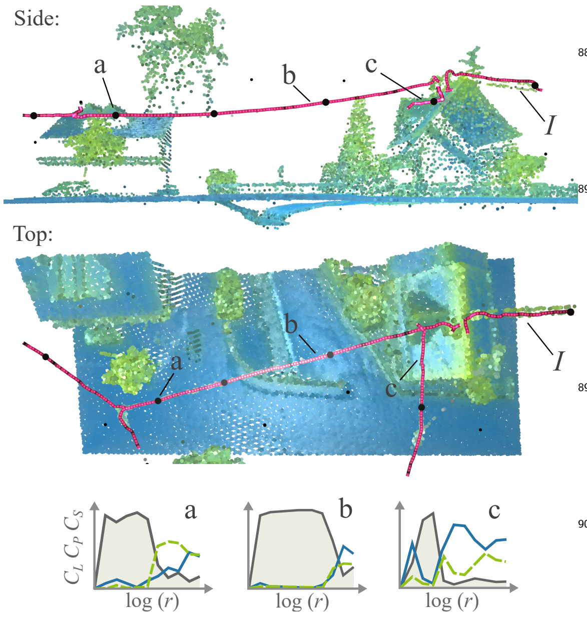


Figure 20: Power line reconstruction of an urban LiDAR scan. (Left): Two different views of the reconstruction. (Right); Computed geometric measures at selected locations: (a) close to a tree with sphericity becoming the second dominant shape value; (b) automatic start point, with dominant linearity integral on an isolated linear structure; (c) roof location where planarity becomes more dominant. The line (I) first follows a bundle of cables and then continues along a pole onto the roof's gable; i.e. a linearly directed structure of the roof-surface. Here, the WMN was employed.

6.8. Discussion of Centroid Variants

As outlined above, different methods for computing the centroids can be selected. This influences both the tensor computation and the geometric features. The latter will differ in scale and smoothness. Thus, the selection will influence the results of the score function and the vector field. In general, WMN exhibited the best performance. Still, in Table 3 we indicate the advantages and disadvantages of the different options.

Table 3: Centroid performance by different aspects.

	PDT	MN	WMN	WMD	MD
Computation time	++	+	+	–	– –
Success rate	–	+	++	+	–
Crossings	–	–	+	–	–
Wave	– –	+	+	–	–
Examples	+	+	+	+	+
Extreme noise	– –	++	+	–	–
LiDAR	–	–	+	+	–

PDT clearly outperforms the others concerning computation time.WMN performs best in the overall success rate, and it is beneficial for reconstructing crossings. Both MN and WMN show the most success for the difficult Wave case. Moreover, all approaches could handle the test cases in [41, 42] (see above). For computing the maximum linearity $C_{L,max}$ in the starting point score O_s, using the PDT has the advantage that it down-votes noisy starting points. In extreme noisy cases, as the circle above, MN and WMN performed best. For our real-world LiDAR datasets, WMN and WMD performed best in reconstructing the cables.

7. Conclusion

We have presented a framework for the reconstruction of curved line structures from noisy 3D point clouds. The method implements a piecewise streamline integration, in a neighborhood tensor field. The approach employs weighing functions at several steps, e.g. tensor computation, point cloud interpolation, and angular weighting. The Fermi-Dirac weighting function gave good results and stabilized against noise. Linearity graphs for different radii were examined to automatically set parameters and steer the algorithm. This includes setting start points, detecting optimal neighborhood radii, and finding directions for integration by using two new scoring functions based on the geometric measures. Further, multiple line-lets are grown forward and backward in parallel; they intersect to form the final global reconstruction. This permits to handle non-manifold lines and sharp corners.

A thorough analysis was carried out on different test geometries, with different types and amounts of noise. This highlighted the performance and the limits of the method for each test geometry. With the automated parameter runs, we explored different centroids for the tensor computation: e.g. mean or geometric median and weighted variants; opting for the weighted mean. Further, we compared our method to the recently introduced *FitConnect* and *StretchDenoise* algorithms, with respect to error metrics and computation time. Our reconstructions were smoother, supported higher noise rates, and could handle crossings, while usually being faster. However, in some test cases the geometry reconstruction was not fully successful. Finally, we tested our method on a real-world LiDAR scan and were able to reconstruct five power cables. However, artifacts were encountered, e.g. integration was continued onto a roof following edges of its surface. In future work, we will investigate further the application to LiDAR cases; especially, including linearity and sphericity into the stopping criteria and/or the pruning step.

The executable of the tool and the source code for geometric measure computation and line reconstruction are provided online as well as the data and evaluation tools for the parameter run benchmark [43]. A video demonstrating the method, selected test cases, and the LiDAR reconstruction can be found here: `https://marcel-ritter.com/mssfreconstruct`.

References

[1] C. Toth, G. Jóźków, Remote sensing platforms and sensors: A survey, ISPRS Journ. of Photogr. and R. Sens. 115 (2016) 22ff.

[2] S. Foix, G. Alenya, C. Torras, Lock-in time-of-flight cameras: A survey, IEEE Sensors Jour. 11 (2011) 1917ff.

[3] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, M. Teschner, SPH Fluids in Computer Graphics, in: Eurographics 2014 - State of the Art Reports, The Eurographics Assoc., 2014.

[4] R. Devore, G. Petrova, M. Hielsberg, L. Owens, B. Clack, Processing terrain point cloud data, SIAM Journal of Imaging Sciences 6 (2013) 1–31.

[5] X. Lu, W. Chen, S. Schaefer, Robust mesh denoising via vertex pre-filtering and l1-median normal filtering, Computer Aided Geometric Design, Preprint (2017).

[6] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, C. T. Silva, A benchmark for surface reconstruction, ACM Trans. Graph. 32 (2013) 20:1–20:17.

[7] J. Tao, J. Ma, C. Wang, C. Shene, A unified approach to streamline selection and viewpoint selection for 3d flow visualization, IEEE TVCG 19 (2013) 393–406.

[8] T. K. Dey, R. Wenger, Reconstructing curves with sharp corners, Computational Geometry 19 (2001) 89 – 99.

[9] Y. Zeng, T. A. Nguyen, B. Yan, S. Li, A distance-based parameter free algorithm for curve reconstruction, Computer-Aided Design 40 (2008) 210 – 222.

[10] H. Lin, W. Chen, G. Wang, Curve reconstruction based on an interval b-spline curve, The Visual Comp. 21 (2005) 418ff.

[11] O. E. Ruiz., C. Corts., M. Aristizbal., D. A. Acosta., C. A. Vanegas., Parametric curve reconstruction from point clouds using minimization techniques, in: Proc of the Intern Conf on Comp Graph Theory and App and Intern Conf on Inf Vis Theory and App - Volume 1: GRAPP, (VISIGRAPP 2013), SciTePress, 2013, pp. 35–48.

[12] S. Flöry, Fitting curves and surfaces to point clouds in the presence of obstacles, Comput. Aided Geom. Des. 26 (2009).

[13] K. Philsu, K. Hyoungseok, Point ordering with natural distance based on brownian motion, Mathematical Problems in Engineering (2010) 17.

[14] Z. Hasirci, M. Ozturk, An eigenvalue analysis based bandwidth selection method for curve reconstruction from noisy point clouds, in: 34th International Conference on Telecommunications and Signal Processing (TSP), 2011, pp. 478–482.

[15] Z. Hasirci and M. Ozturk, A novel method for thinning branching noisy point clouds, in: 36th International Conference on Telecommunications and Signal Processing (TSP), 2013, pp. 713–716.

[16] Y. Jaw, G. Sohn, Wind adaptive modeling of transmission lines using minimum description length, ISPRS Jour. of Photogr. and Remote Sensing 125 (2017) 193 – 206.

[17] S. Ohrhallinger, S. A. Mitchell, M. Wimmer, Curve reconstruction with many fewer samples, in: Proceedings of the Symposium on Geometry Processing, SGP '16, Eurographics Association, Goslar Germany, Germany, 2016, pp. 167–176.

[18] S. Ohrhallinger, M. Wimmer, Stretchdenoise: Parametric curve reconstruction with guarantees by separating connectivity from residual uncertainty of samples, in: Proc of the 26th Pacific Conf on Comp Graph and App: Short Papers, PG 18, Eurographics Association, Goslar, DEU, 2018, p. 14.

[19] S. Ohrhallinger, M. Wimmer, Fitconnect: Connecting noisy 2d samples by fitted neighborhoods, Computer Graphics Forum 38 (2019) 126ff.

[20] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, W. Stuetzle, Surface reconstruction from unorganized points, SIGGRAPH Comput. Graph. 26 (1992) 71–78.

[21] A. M. McIvor, R. J. Valkenburg, A comparison of local surface geometry estimation methods, Machine Vision and Applications 10 (1997) 17–26.

[22] Y. Ohtake, A. Belyaev, H.-P. Seidel, An integrating approach to meshing scattered point data, in: Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling, SPM '05, ACM, New York, NY, USA, 2005, pp. 61–69.

[23] H. Wendland, Scattered Data Approximation, Cambridge University Press, 2005.

[24] S. Giraudot, D. Cohen-Steiner, P. Alliez, Noise-adaptive shape reconstruction from raw point sets, in: Proc. of the 11th Eurographics/ACMSIGGRAPH Symposium on Geometry Processing, SGP '13, Eurographics Ass., Goslar, DEU, 2013, pp. 229–238.

[25] G. Taubin, Estimating the tensor of curvature of a surface from a polyhedral approximation, in: Proceedings of the Fifth International Conference on Computer Vision, ICCV '95, IEEE Computer Society, Washington, DC, USA, 1995, pp. 902–.

[26] J. Berkmann, T. Caelli, Computation of surface geometry and segmentation using covariance techniques, IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (1994) 1114–1116.

[27] M. Alexa, A. Adamson, On Normals and Projection Operators for Surfaces Defined by Point Sets, in: M. Gross, H. Pfister, M. Alexa, S. Rusinkiewicz (Eds.), SPBG'04 Symposium on Point - Based Graphics 2004, The Eurographics Association, 2004.

[28] M. Liu, F. Pomerleau, F. Colas, R. Siegwart, Normal estimation for pointcloud using GPU based sparse tensor voting, in: IEEE Intern. Conf. on Robotics and Biomimetics, 2012, pp. 91–96.

[29] P. J. Basser, S. Pajevic, C. Pierpaoli, J. Duda, A. Aldroubi, In vivo fiber tractography using dt-mri data, Magnetic Resonance in Medicine 44 (2000) 625–632.

[30] M.-C. Chou, M.-L. Wu, C.-Y. Chen, C.-Y. Wang, T.-Y. Huang, Y.-J. Liu, C.-J. Juan, H.-W. Chung, Tensor deflection (tend) tractography with adaptive subvoxel stepping, Journal of Magnetic Resonance Imaging 24 (2006) 451–458.

[31] D. Weinstein, G. Kindlmann, E. Lundberg, Tensorlines: Advection-diffusion based propagation through diffusion tensor fields, in: Proceedings of the Conference on Visualization '99: Celebrating Ten Years, VIS '99, IEEE Computer Society Press, Los Alamitos, CA, USA, 1999, pp. 249–253.

[32] D. J. Natale, M. S. Baran, R. L. Tutwiler, Point cloud processing strategies for noise filtering, structural segmentation, and meshing of ground-based 3d flash lidar images, in: 2010 IEEE 39th Applied Imagery Pattern Recogn. Worksh. (AIPR), 2010, pp. 1–8.

[33] M. Weinmann, B. Jutzi, S. Hinz, C. Mallet, Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers, ISPRS Journal of Photogrammetry and Remote Sensing 105 (2015) 286 – 304.

[34] C. Westin, S. Peled, H. Gudbjartsson, R. Kikinis, F. Jolesz, Geometrical diffusion measures for mri from tensor basis analysis, in: Proceedings of ISMRM, Canada, 1997, p. 1742.

[35] J. McDougall, E. C. Stoner, R. Whiddington, The computation of Fermi-Dirac functions, Philosophical Trans. of the Royal Society of London. Series A (1938).

[36] Y. Lipman, D. Cohen-Or, D. Levin, H. Tal-Ezer, Parameterization-free projection for geometry reconstruction, ACM Trans. Graph. 26 (2007).

[37] C.-H. Lin, J.-Y. Chen, P.-L. Su, C.-H. Chen, Eigen-feature analysis of weighted covariance matrices for lidar point cloud classification, ISPRS Jour. of Photogr. and R. Sensing 94 (2014).

[38] F. Plastria, The Weiszfeld Algorithm: Proof, Amendments, and Extensions, in: H. Eiselt, V. Marianov (Eds.), Foundations of Location Analysis, volume 155, Springer, Boston, MA, 2011.

[39] M. Ritter, W. Benger, Reconstructing power cables from lidar data using eigenvector streamlines of the point distribution tensor field, Journal of WSCG Vol20, 20-th Intern. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (2012).

[40] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, C. T. Silva, A survey of surface reconstruction from point clouds, Computer Graphics Forum 36 (2017) 301–329.

[41] FitConnect, 2018. github.com/stefango74/fitconnect.

[42] StretchDenoise, 2018. github.com/stefango74/stretchdenoise.

[43] MssfReconstruct, 2019. github.com/gileoo/MssfReconstruct.

Visual Analysis of Point Cloud Neighborhoods via Multi-Scale Geometric Measures

Abstract

Point sets are a widely used spatial data structure in computational and observational domains, e.g. in physics particle simulations, computer graphics or remote sensing. Algorithms typically operate in local neighborhoods of point sets, for computing physical states, surface reconstructions, etc. We present a visualization technique based on multi-scale geometric features of such point clouds. We explore properties of different choices on the underlying weighted co-variance neighborhood descriptor, illustrated on different point set geometries and for varying noise levels. The impact of different weighting functions and tensor centroids, as well as point set features and noise levels become visible in the rotation-invariant feature images. Finally, we show how features in real-world LiDAR data can be intuitively visualized by images created with our approach.

CCS Concepts

• ***Human-centered computing*** → ***Visual analytics;*** • ***Computing methodologies*** → ***Point-based models;***

1. Introduction

Point sets are commonly employed as a geometrical data structure, generated e.g. based on sensor or simulation data. Another use is in the context of object classification or object synthesis; there, point sets enhance labelled images and/or labelled object models [MGY*19]. In any case, local neighborhood information is an essential component in many point based algorithms; and even more so when including point set hierarchies. Related to this, we propose to employ a multi-scale view on local neighborhood features of point clouds, and use this for visualization. Our contributions in short are:

- A new concept of a multi-scale feature image, generated based on a weighted co-variance measure.
- The application of the new technique to the visual analysis of different point set properties; such as shape, noise, weighting functions, and point set centroids.
- Investigation of different weighting functions and centroids, optimized for the visualization.
- Integration into an intuitive visual analysis tool.

Related to our work, Medioni et al. [MTL00] introduced tensor voting to sample geometric properties onto a voxel grid, e.g. to reconstruct surfaces from noisy point clouds. From each tensor location information was transported onto voxels via a voting process. Nevertheless, they did not employ any weighting. Also, they followed a different multi-scale approach. In contrast to voxel sampling, we construct our multi-scale information on curves and paths. Regarding our geometric measures we follow Westin et al. [WPG*97, WMM*02], who employed tensor shape analysis for fiber tracking in the human brain. They highlighted, classified, and analysed axons, based on diffusion tensors originating from magnetic resonance imaging. Next, Natale et al. [NBT10] applied the same shape factors on point clouds, which they obtained from time-of-flight images. They introduced a multi-scale neighborhood and developed a decision network based on shape values at different scales. The latter permitted them to classify a point as being part of a planar, edge/linear structure or as noise. We also built upon this multi-scale idea and developed a visualization technique, tailored to reveal properties of the tensor computation as well as the point cloud neighborhoods, via color-mapped images. Moreover, in the tensor (co-variance) computation we employ radial distance weighting functions, which we also analyze visually. For these, several different choices are possible; for instance, SPH smoothing kernels [GM77], or compactly supported radial distance functions [Wen95]; and their recent optimizations, e.g. [CS20].

2. Geometric Shape Measures

Shape factors: Local geometric measures are computed via a weighted co-variance analysis, resulting in three shape factors, based on the tensor eigenvalues, for linearity, planarity, and sphericity [WPG*97]:

$$\underset{\sim}{t} = \frac{1}{\sum_{i=1}^{N} \omega(d_i)} \sum_{i=1}^{N} \omega(d_i)(v_i \otimes v_i), \quad \text{with} \tag{1}$$

$$v_i = p_i - c, \quad d_i = |v_i|/r, \quad \text{yielding}$$

$$C_L = (\lambda_3 - \lambda_2)/L, \;\; C_P = 2(\lambda_2 - \lambda_1)/L, \;\; C_S = 3\lambda_1/L, \tag{2}$$

with $\underset{\sim}{t}$ the second order tensor, c a point of reference (i.e. the *centroid*), p_i the neighbors, ω a distance weighting function, and r a radius of the local neighborhood of interest. The three shapefactors C_L, C_P, and C_S are computed from the eigenvalues λ_i of $\underset{\sim}{t}$; with

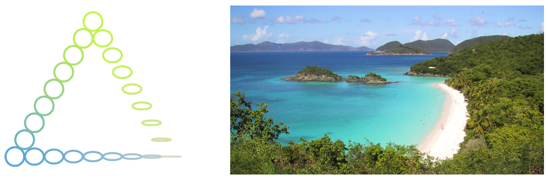


Figure 1: *Colors were chosen inspired by a real-world notion: green for spherical vegetation, blue for planar water surfaces, and light-gray for elongated shorelines (Photo: [Whi03]).*

$L = \lambda_0 + \lambda_1 + \lambda_2$. The three values can be considered as defining a barycentric coordinate system, and can thus be directly mapped to interpolate three associated base colors. Since Lin et al. [LFK*13] showed that colors are remembered better, when they are associated with concrete real-world phenomena, we chose these colors with a specific view in mind: green for *volumetric* (vegetation) blobs, blue for *planar* (ocean) surface, and light-grey for elongated *linear* (stone) shorelines. Figure 1 depicts the barycentric shape space as colored ellipses, and shows a related natural scene.

Multi-scale view: Eq. (1) initially relies on a specific fixed neighborhood radius which can be difficult to specify for an arbitrary point cloud. Instead, we propose to change this radius over multiple scales. Figure 2 illustrates how the three shape factors change, as function of the radius. The graphs were determined for a specific centroid, in a point set representing a square. As can be seen, the graph representing linearity decreases, when the radius grows to include the corner; at this point, planarity starts to increase. For a more compact representation, the three scalar graphs (and base colors) can be merged into mixed colors, which is indicated by a color-bar in the bottom of the figure. Such color-bars are computed for a specific location and represent the change of the shape factors with changing radii. Differences in these color-bars will result, dependent on point cloud location. We propose to assemble these bars next to each other for neighboring points of a point set. Figure 3 shows an example of such a visualization, for the point cloud of a square. Note that the abscissa denotes the (ordered) point index, while the ordinate represents the increasing radius. The corners in the square are dominant with respect to planarity; the edges are dominant with respect to linearity. Thus, the assembled color-bars

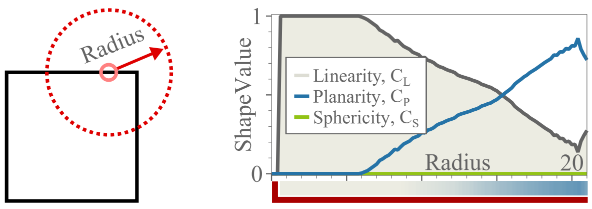


Figure 2: *Geometric measures (right) computed for a centroid location in an example point cloud of a square (left). Resulting geometric measures depend on the selected radius; linearity in the example is initially high and decreases when a corner is reached. The bottom color bar (right, bottom) indicates the barycentric mixing of the three base colors (in the 2D example only blue and light-grey).*

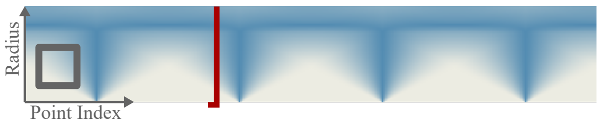


Figure 3: *The multi-scale feature image (MSFI) illustrates the geometric measures by mixing the base colors; pixel columns (vertically) denote a single measure graph at a specific location, for varying radii. The red line marks the geometric measures for the point on the square in Figure 2.*

provide insight into the dominant shape features at various scales. We denote these images as multi-scale feature images (MSFI). Note that in arbitrary point clouds, points are not necessarily ordered (as for the artificial square example). Still, an ordering could be achieved. e.g. by sorting locally along the major eigenvector of a selected point of reference. Another option could be sorting along a path provided by a user (see below).

3. Analysis of the Visualization Method

In this section we first provide an investigation of the proposed visualization approach on simple 2D and 3D geometries. We also present a visual analysis of the influence of weighting functions and chosen centroids for the tensor computation.

Test geometries: For simplicity, we visualize the multi-scale features of six 2D and three 3D sampled point clouds. Figure 4 shows the resulting, rotation-invariant MSFIs. Various geometric properties can be seen, e.g. corners (planar/spherical peaks), straight lines (bright areas), and curvature in and out of a plane (fading from light-grey). Three of the geometries will be used for further discussion and analysis below: rectangle (a), helix (b), 3D crossing (c).

Influence of tensor weighting: The weighting function ω in

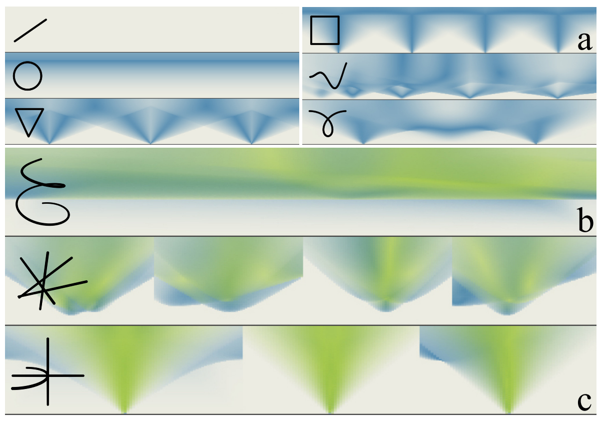


Figure 4: *Geometric measures illustrated via MSFIs, of six 2D and three 3D point clouds of curves. Respective geometries are indicated on the left of the measure image. Geometric properties, such as corners, lines passing close-by, curvature, as well as 2D & 3D distribution become visible in the images.*

Eq. (1) influences the eigenvalue computation and thus the three measure graphs. As a result, the location of extrema in the graphs may change, as well as the graph shape. During our development, we investigated 30 different options for the weighting functions, and also investigated the effect on the visualization. First, Figure 5 illustrates the effect of different choices of ω on the linearity graph (with a few prominent ones labelled).

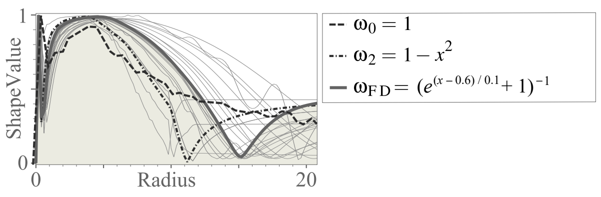


Figure 5: *Influence of weighting functions on the multi-scale features, for jitter-noise data. (Left:) Three of thirty functions are highlighted showing different minima positions and smoothness.*

Next, Figure 6 provides the MSFIs for two selected geometries (a) and (b), for three choices of ω: no weighting (ω_0), a quadratic weighting function (ω_2), and a Fermi-Dirac distribution weighting function (ω_{FD}). The visualization illustrates that for the latter two smaller radii can be used for the analysis, thus e.g. reducing computational cost for the neighborhood search. Moreover, the Fermi-Dirac function qualitatively exhibits somewhat more defined edges and regions; correspondingly, it also showed the best performance in related geometry reconstructions in noisy point clouds.

Centroid selection: Another choice in Eq. (1) is the centroid c. One could select an existing central point of the point cloud, yielding the point distribution tensor (*PDT*); or compute a mean point *mean*, thus yielding a standard co-variance matrix for principal component analysis (PCA); or select a geometric *median* instead. For the latter two, a new centroid has to be computed for each radius scale. The influence of these different choices also becomes apparent in the MSFIs, of three test cases (see Figure 7). The *mean* and *median* result in more prominent shape regions, with clearer borders.

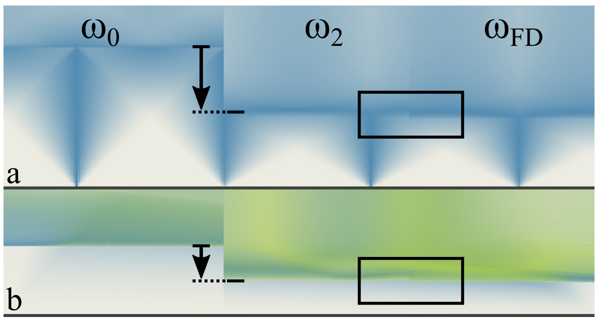


Figure 6: *Influence of different weighting functions on the geometric features: no weighting (ω_0), quadratic weighting (ω_2), Fermi-Dirac weighting (ω_{FD}). Values are shifted to smaller radii by using ω_2; and are even more enhanced by using ω_{FD}.*

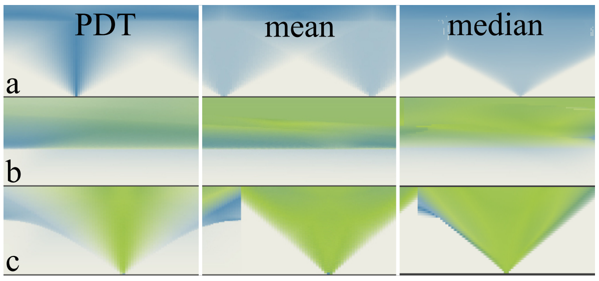


Figure 7: *Influence of different centroids for tensor computation, illustrated on the rectangle (a), helix (b), and crossing (c) example geometries. Mean and geometric median result in more homogeneous shape regions. The median provides clearer features and defined borders, indicating that it may be a better choice for further point cloud processing, such as geometry reconstruction.*

Influence of noise: Any noise present in a point cloud can also be scrutinized in the MSFIs (see Figure 8). For instance, a horizontal green band results at a radius matching the amplitude (i.e. point distortion distance) of added noise. Using the mean or geometric median as the centroid leads to better compensation of noise. Visually, the latter again results in more homogeneous regions and more defined edges.

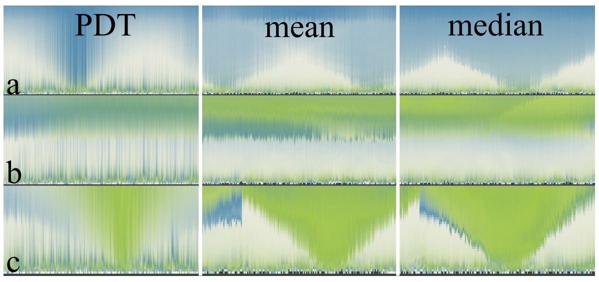


Figure 8: *Effect of different centroids on visualizing noisy point neighborhoods; the same cases as in Figure 7 are shown, but data was distorted by adding a random displacement. Mean and median centroids compensate for noise in the geometric measures.*

4. Application Example

We have integrated the proposed visualization technique into an interactive framework for analysis of real-world point cloud data, captured by laser scanning. A central interaction element is a tool to allow users to define *line probes*. These are manually placed polygonal lines, along which the geometric measures can be computed and visualized as MSFIs. Figure 9 showcases an example data set, comprising a house, a tree, and several cables. In the depictions (at the top), individual points in the cloud are colored according to the most prominent shape factor of the multi-scale measures.

Four line probes were placed by a user into the cloud, and the corresponding multi-scale measures were computed and MSFIs

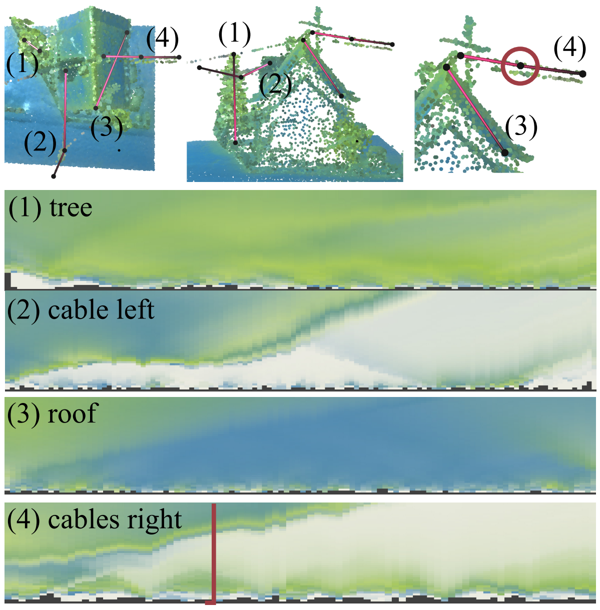


Figure 9: *(Top:) Four line probes placed in real-world data. (1) in a tree, (2) along a cable, (3) on the roof, and (4) along a bundle of cables. (Bottom:) geometric multi-scale measures along line probes; illustrating features such as: crossing cables, nearby geometry, edges, and homogeneous regions.*

generated along these. Different local features become apparent in the latter: (1) probe from top to bottom of a tree: the neighborhood is dominated by sphericity. Inner points of the trunk appear as small linear structures; (2) probe along a cable, close to the roof: linearity dominates. The progressing discoloration (right) denotes a crossing with a second cable. Close to the roof (left), the radius for dominant linearity decreases, with higher sphericity at smaller radii close to the edge; (3) probe diagonally across the roof: planarity is prominent. Sphericity increases close to the gable (left); (4) final probe located on the top one of a bundle of cables, to the right of the house: linearity is dominant at two scales; firstly, at a small scale for the single cable, and secondly, for the bundle of cables.

5. Conclusion

We have proposed a novel visualization paradigm to illustrate and analyze local geometric features in point cloud neighborhoods. Features such a geometric shape, indicating curvature or edges, and presence of noise become apparent in colored multi-scale feature images. Further, we examined the influence of radial weighting functions and the choice of a centroid on the geometric measures. In our related, other developments , the visualization supported our selection of optimal algorithm components (e.g. use of weighting ω_{FD} and of geometric median as centroid c). Finally, we presented the utilization of the visualization paradigm in an analysis tool for light detection and ranging (LiDAR) sensor data. MSFIs can be generated and inspected along manually created paths, within the point clouds. In future work we will improve the performance of the point cloud shape measures with GPGPU computing, to maintain interactivity for very large datasets. We will also explore topical classification and clustering of the multi-scale measures, to enhance the visualization. Finally, we will explore the visual reconstruction of the detected line structures, surfaces, and volumetric objects. The interactive tool as well as the source code are provided online [X.21].

6. Acknowledgment

This research was funded through the Vice Rectorate of X. within the scope of the doctoral program X.

References

[CS20] Cervenka M., Skala V.: Behavioral study of various radial basis functions for approximation and interpolation purposes. In *IEEE 18th World Symp on Appl Mach Int and Inf (SAMI)* (2020), p. 135ff. 1

[GM77] Gingold R. A., Monaghan J. J.: Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society 181*, 3 (12 1977), 375–389. 1

[LFK*13] Lin S., Fortuna J., Kulkarni C., Stone M., Heer J.: Selecting semantically-resonant colors for data visualization. In *Proc of the 15th Eurogr Conf on Visualization* (Chichester, UK, 2013), EuroVis '13, The Eurogr Association, John Wiley, Sons, Ltd., p. 401ff. 2

[MGY*19] Mo K., Guerrero P., Yi L., Su H., Wonka P., Mitra N. J., Guibas L. J.: Structurenet: Hierarchical graph networks for 3d shape generation. *ACM Trans. Graph. 38*, 6 (Nov. 2019). 1

[MTL00] Medioni G., Tang C., Lee M.: Tensor voting: Theory and applications. In *Proc. 12th Congres Francophone AFRIF-AFIA de Reconnaissance des Formes et Intell. Artif.* (2000), pp. 1–10. 1

[NBT10] Natale D. J., Baran M. S., Tutwiler R. L.: Point cloud processing strategies for noise filtering, structural segmentation, and meshing of ground-based 3d flash lidar images. In *2010 IEEE 39th Applied Imagery Pattern Recogn. Worksh. (AIPR)* (2010), pp. 1–8. 1

[Wen95] Wendland H.: Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Adv Comput Math 4* (1995), 389–396. 1

[Whi03] Whitney B.: File:StJohnTrunkBay.jpg, 2003. https://en.wikipedia.org/wiki/File:StJohnTrunkBay.jpg. 2

[WMM*02] Westin C.-F., Maier S. E., Mamata H., Nabavi A., Jolesz F. A., Kikinis R.: Processing and visualization for diffusion tensor MRI. *Med Image Anal 6*, 2 (2002 Jun 2002), 93–108. 1

[WPG*97] Westin C., Peled S., Gudbjartsson H., Kikinis R., Jolesz F.: Geometrical diffusion measures for MRI from tensor basis analysis. In *Proc. of ISMRM, 5th Meeting, Canada* (1997), p. 1742. 1

[X.21] X. X.: Multi Scale Feature Images for Point Clouds, 2021. https://github.com/xx. 4

3.3 Extensions

3.3.1 Comprehensive List of Parameters

For the reconstruction framework introduced above, we provide further information on the various involved parameters. The latter are categorized as: user, automatic, expert, and fixed. *User* parameters are expected to be provided manually; *automatic* parameters are automatically set, e.g. based on the minimal distance analysis; however, they may be adjusted manually, if desired; *expert* parameters are not meant to be changed; but they could be adjusted in exceptional cases, e.g. to accommodate extremely large point clouds, or to disable line pruning; *fixed* parameters were optimized throughout the development of the framework and have been chosen based on extended testing and evaluations; these should remain fixed.

Parameters are provided below in overview tables, sorted by category. Parameter name and type, as well as a short description are given. In Table 3.1 selected parameters are compiled – three *user* (U) and three *automatic* (A) parameters. Below, in Appendix A.2, additional *expert* (E) and *fixed* (F) parameters are gathered.

Table 3.1: Overview table of selected algorithm parameters.

Name	**Type**	**Description**
StartPointsNr	U	User defined number of start points; selects N best from the scored candidate set. If set to 0 a score threshold is applied instead.
MaxIterations	U	Maximum number of streamline integration steps (parallel, bidirectional).
DistanceCutoff	U	Integration break criterion; maximally allowed distance from integration point to closest data point; used to overcome holes, see th_p in Figure 3.7.
MSSFMaxRadius	A	Maximum radius of the multi scale computation; default is automatic; $60\Delta_{\mathrm{mdn}}$ or reaching a break criterion, comparing the last two shape factors radii.
DistancePrune	A	Maximally allowed distance from a line end to the closest point set point when shrinking lines from open ends as factor of the integration step size h; default: 1.2.
StepMergeMul	A	Search radius for line mergers and intersections defined as a factor of the integration step size h; default: 1.4.

3.3.2 Line Endings

The integration of a streamline can be stopped according to five different criteria: a) a distance check of the current integration point to the closest point of the point cloud, b) a scene bounding box check, c) limiting the maximum iteration count, d) by merging nearby line endings, and e) by intersecting lines. Line endings are labeled either *open* or *closed*. The latter two stopping criteria result in *closed* line endings. *Open* line endings are considered later in the pruning step to remove long overshooting lines. Figure 3.7 shows the five different scenarios to end the eigenvector streamline integration.

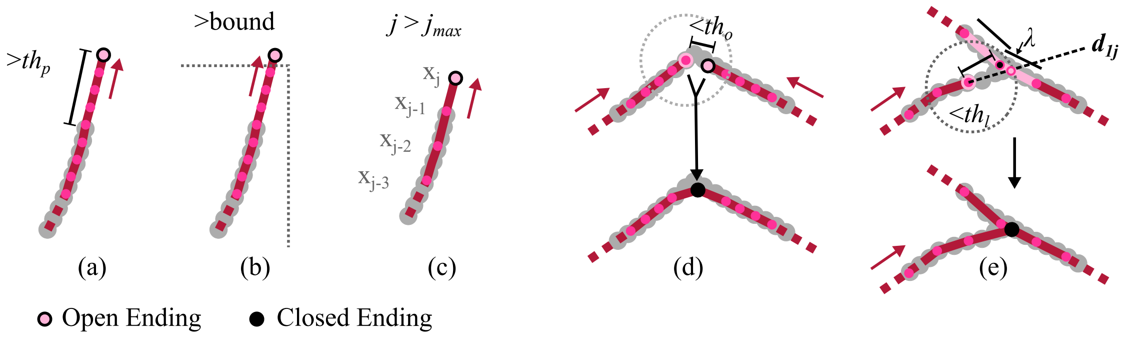


Figure 3.7: Five scenarios to end streamline integration: (a) the current integration point x_j is exceeding a distance threshold to the closest point of the point cloud, (b) x_j is located outside of the scene bounding box, (c) the maximum number of integration steps is reached $j > j_{max}$, (d) open line endings meet (x_j is close to another open line ending), and (e) a line intersects another line (x_j is close to another line). Here, the two segments connected to the closest point and the approaching line are tested by evaluating the closest point on skew lines for both segments (see Appendix A.3).

In the case of meeting open endings (d) the two end points are averaged and for both lines the point is set to be a closed ending. An intersection with a line (e) is found by checking distances to the points of the lines. Note that lines are sampled quite densely and homogeneously using the RK32 method with constant step size. Then the "intersection point" is computed by finding the minimum distance location on skew lines, as outlined in Appendix A.3. This approach is more robust than relying on actual 3D line intersections. Therefore, two lines are tested (see Figure 3.7): a line defined going through x_j in its tangential direction $d_{1j} = x_j - x_{j-1}$, as well as lines for the segments before and after the closest point on the approached line $\hat{x}_i$, with segment directions $\hat{d}_i$ and $\hat{d}_{i+1}$. The closest parameter λ on the segment lines is computed. Dependent on the two segments' λ, a point is inserted in one of the segments. Alternatively, if $\lambda \notin [0.0, 1.0]$, $\hat{x}_i$,

$\hat{x}_{i+1}$, or $\hat{x}_{i-1}$ is chosen directly. The new point is added as closed ending to the terminated line.

3.3.3 Test Geometry Creation – Curve Sampling

The base sampling of the curve geometries was done regularly in arc length on the test curves. Data defects – to simulate various effects such as sensor vibrations, dust in the air, or shadowing – can be controlled by the following parameters: jitter amplitude S_J, distribution blend S_D, data hole start t_a and end t_e, and the number of added random outliers M within an enlarged bounding box controlled by the outlier noise ratio $S_U = M/N$ with the total number of points $N = K + M$. The final point set T is created by sampling a 3D curve function $\boldsymbol{c} : \mathbb{R} \rightarrow \mathbb{R}^3$.

$$t_i = i/K,\ i\ =\ [0, ..., K],\ j\ =\ [0, ..., M] \tag{3.15}$$

$$\bar{t}_i = t_i \ \text{ adjusted to be uniform in arclength,} \tag{3.16}$$

with t_i being uniformly distributed values from 0.0 to 1.0, the initial curve parameters; $\bar{t}_i$ is uniform in arclength of the curve $\boldsymbol{c}$. Further elements of the noise generation are:

$$r_i = \big\{ r_i \sim \mathcal{U}(0.0, 1.0) \mid r_i < r_{i+1} \big\} \tag{3.17}$$

$$\boldsymbol{v}_i = \big\{ nrm(v_x, v_y, v_z) \mid v_x, v_y, v_z \sim \mathcal{U}(-1.0, 1.0) \big\} \tag{3.18}$$

$$\tau_i = (1.0 - S_D)\, \bar{t}_i + S_D\, r_i,\ S_D \in [0.0, 1.0] \tag{3.19}$$

$$P_j = \big\{ \boldsymbol{c}(\tau_i) + S_J\, \boldsymbol{v}_i \mid \text{if } \tau_i \not\in [t_a, t_e] \big\} \tag{3.20}$$

with r_i being sorted uniform random values to generate distribution noise, and $\boldsymbol{v}_i$ random direction vectors for jitter noise; τ_i are sorted values, for blending the uniform distribution and the random distribution values; $nrm()$ denotes vector normalization. The resulting points P_j now include distribution noise, jitter, and data holes. $\mathcal{U}(a, b)$ denotes a uniform random distribution within the interval $[a, b]$.

Further, outlier noise and a displacement out of plane (S_o) are added:

$$\max(A) := \big(\max_i(A_{i,x}), \max_i(A_{i,y}), \max_i(A_{i,z}))\big)^T, \quad A_i \in \mathbb{R}^3 \tag{3.21}$$

$$\min(A) := \big(\min_i(A_{i,x}), \min_i(A_{i,y}), \min_i(A_{i,z}))\big)^T, \quad A_i \in \mathbb{R}^3 \tag{3.22}$$

$$d = ||\max(P) - \min(P)|| \tag{3.23}$$

$$C = \big(\max(P) + \min(P)\big)/2 \tag{3.24}$$

$$Q_j = P_j + S_o\big(0.0,\ 0.0,\ \cos\big(\frac{\pi}{2}(P_{j,y} - C_y)/(2d_y)\big)\big)^T \tag{3.25}$$

$$U_k = \big\{(u_x, u_y, u_z)^T \mid \boldsymbol{u} \sim \boldsymbol{\mathcal{U}}(\min(P), \max(P))\big)\big\} \tag{3.26}$$

$$T = Q \cup U, \tag{3.27}$$

with d the diagonal of the bounding box, C the center of the bounding box and $\boldsymbol{\mathcal{U}}(\boldsymbol{a}, \boldsymbol{b})$ – with $a_x < b_x$, $a_y < b_y$, and, $a_z < b_z$ – a uniform random distribution of points. Note that a slightly enlarged bounding box was used in practice. Finally, T is the set of points of the final noisy geometry, additionally including outliers and out of plane displacement.

3.3.4 Multi Scale Measure Graphs

In addition to the content provided in Ritter et al. (2021b)~, Figure 3.8 also shows multi scale measure graphs of a variation of the two Fermi-Dirac parameters. The same geometry setup as in Figure 3.6 is utilized. Generally, the graphs show the effect of the weighting functions in the tensor computation.

Each graph corresponds to one of 30 different weighting functions (Left) and 56 different Fermi parameters (Right); chosen as combination of the m and T values shown in Figure 3.3. Differences can be seen in the value of the absolute maximum, the location of the minimum, and in the smoothness of the resulting linearity graph. For algorithms analyzing linearity graph $C_L(r)$, the following properties were desired: high and low extreme values, small radius at the minimum, and smoothness. The first property eases finding features, the second allows for smaller radii – speeding up neighborhood computations – and the latter improves numerical stability. Through an automated parameter variation the quadratic weighting function was identified as a good candidate. Note that this coincides with the observation in Ritter and Benger (2012), where this weighting was also selected in the case of a specific application context. It generates a superior local minimum at a rather small minimum radius. Later, it was replaced by the Fermi-Dirac I) option, which yielded a minimum at a larger radius, but gave a smoother shape (Ritter et al., 2021b)~. Especially, when employing the geometric median or

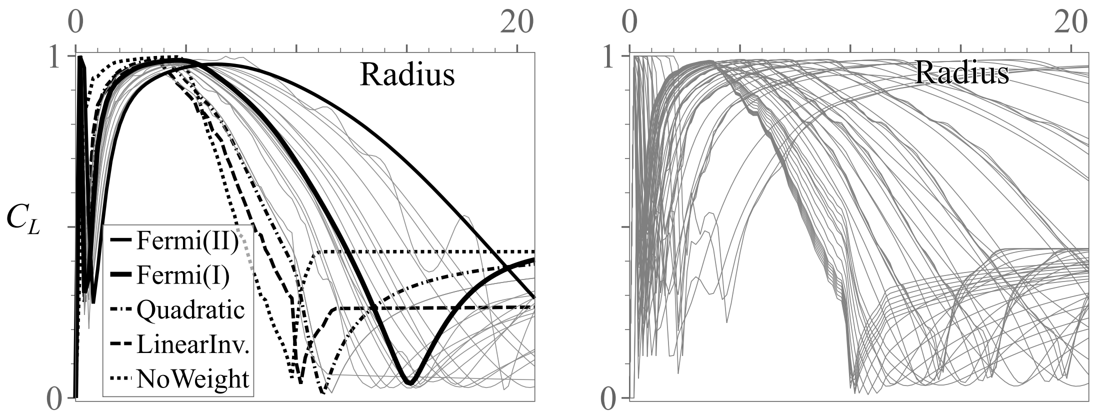


Figure 3.8: Influence of the weighting functions in jittered data on the multi scale linearity $C_L(r)$. (Left:) Five of thirty functions with different minima and smoothness. (Right:) T and m variations of the Fermi-Dirac function. A similar behavior as with the 30 functions can be achieved by adjusting the two parameters.

the weighted mean the Fermi-Dirac I) removed otherwise occurring discontinuity artifacts in the multi scale images. An explicit list and formulae of the tested weighting functions are provided in Appendix A.4.

3.3.5 Weighting Experiments

Two experiments were set up and reconstruction errors were evaluated to investigate the influence of the weighting functions. For each setup a parameter run was executed outputting the error measures: $\mathcal{E}_H$, $\mathcal{E}_V$, $\mathcal{E}_{Len}$, and $\mathcal{E}_{Com}$ (Ritter et al., 2021a)~. Parameter runs were defined by additional key value pair control files, where the key is related to a list of values. Each parameter set is mapped to a unique index. Ordered by that index the results are computed and stored in binary files; guaranteeing unique parameter and data access.

The first experiment reconstructs a circle and a rectangle. The goal was to find good candidates for weighting functions (for ω_T and ω_I). The two chosen geometries are very contrary in their shape properties: constant curvature for the circle and straight lines with sharp corners for the rectangle. The following parameter variation was chosen to investigate the reconstruction performance:

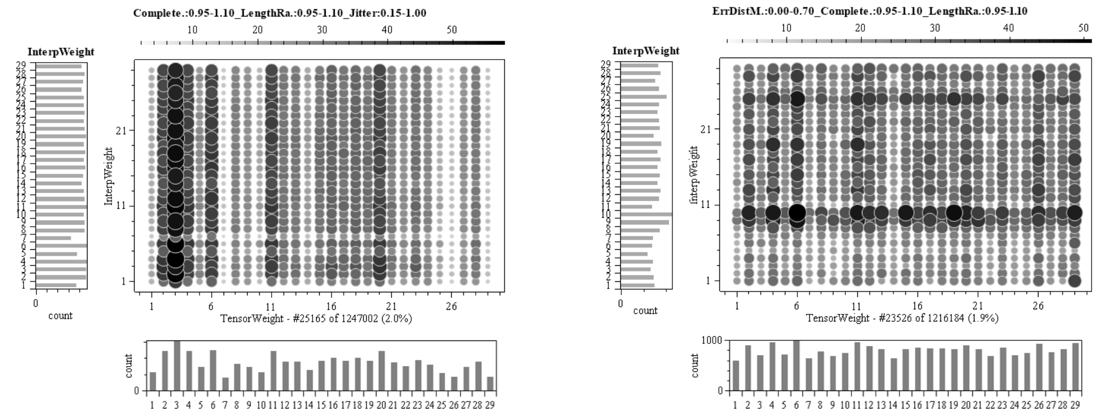
StepSize:	0.05, 0.1, 0.2, 0.4	Samples:	50, 100, 200, 400
TensorRadius:	1.0, 2.0, 4.0, 8.0	InterpRadius:	0.5, 1.0, 2.0, 4.0, 8.0
InterpWeightNr:	0, 1, 2, 3, ..., 28	TensorWeightNr:	0, 1, 2, 3, ..., 28
		S_J:	0.0, 0.1, 0.2, 0.4, 0.8

Here, S_J is the jitter amplitude. The variation study thus results in $1,345,600$

parameter sets, applied both to the circle and the rectangle. Single-threaded, on an Intel Xeon X5650 @ 2.67 GHz, the runs executed in total over 434 and 415 $mins$, respectively. Thus, a reconstruction took on average 19 ms (including binary disk IO).

As discussed above, for the streamline integration the standard RK32 integrator was chosen. Besides 29 weighting functions, tensor radii, interpolation radii, and jitter noise were varied. Figure 3.9 shows differences in the reconstruction success dependent on the involved weighting functions. Note that only 'successful' reconstructions were taken into account (Ritter et al., 2021a)~; i.e. $\mathcal{E}_{Com} > 0.95$, $\mathcal{E}_{Len} > 0.95$, and small $\mathcal{E}_H$. The numbers in the plots relate to the following weighting function options[2]:

1	:=	step at 0.5r / r	4/11	:=	quadratic/Epanechnikov
2	:=	linear	6	:=	cubic
3	:=	linear inverse	10	:=	SPH kernel order 5



(a) Circle (b) Rectangle

Figure 3.9: Scatter-plots of tensor weight (x-axis) and interpolation weight (y-axis) of successful reconstructions. (a) The influence of the interpolation weight (ω_I) is low for the circle; the 3^{rd} tensor weight is performing best. (b) The influence of the tensor weight (ω_T) is low for the rectangle, but interpolation weight 10 performs slightly better than others.

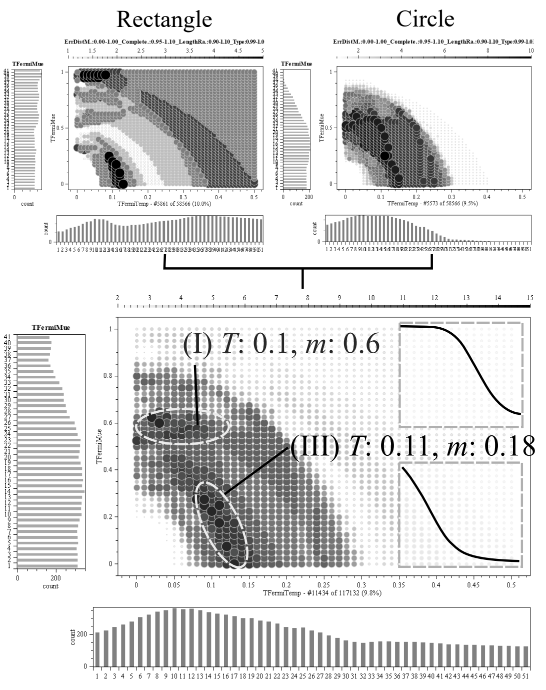
There is a relation between the chosen function and the number of successful reconstructions. As can further be seen, the results are different for the circle (a) and the rectangle (b) case. For the former, the weighting in the interpolation (y-axis) is of limited influence, in contrast to the tensor weight (x-axis); there

[2]Note that in the Appendix A.4 all functions are listed. Here, for 1 a step function was employed with the step at 0.5.

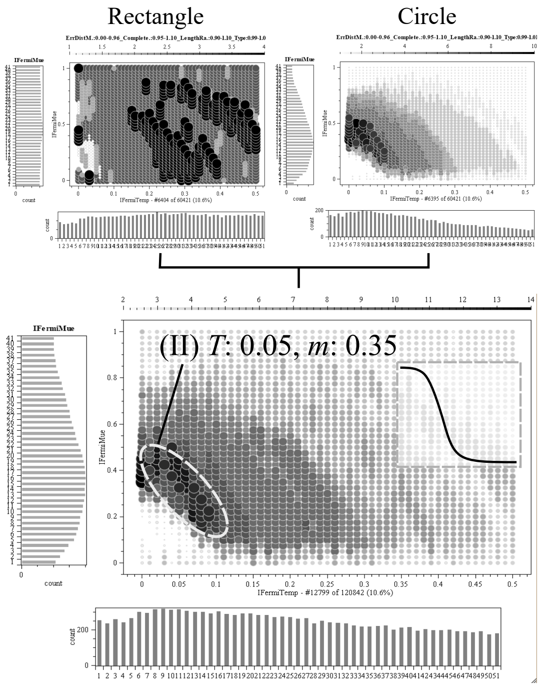
function number 3 stands out. For the rectangle (b), the tensor weight has a lower influence; as interpolation weight, function number 10 could be a good candidate.

Following this initial study, a second variation test was executed, specifically to find optimal parameters for the Fermi-Dirac (FD) function. Parameters were limited, dependent on the result of the experiment above. To find the best FD parameters for the tensor computation the interpolation weight uses the 5^{th} order SPH kernel. To find the best interpolation FD parameters, the cubic weighting was chosen. The parameter variation of the FD optimization experiment is as follows:

T:	0.5, 0.49, 0.48, ..., 0.0	Seed:	0, ..., 4	InterpRadius:	3
m:	0.0, 0.05, 0.10, ..., 1.0	Samples:	200	T/I-Weight.:	11/10
S_J:	0.2, 0.4, 0.8	StepSize:	0.05	TensorRadius:	3



(a) Parameter selection for tensor weighting.



(b) Parameter selection for interpolation weighting.

Figure 3.10: Scatter plots of the best 10% reconstructions (with respect to $\mathcal{E}_H$) varying the FD parameters T (x-axis) and m (y-axis). The upper row shows the results for the circle (left) and the rectangle (right) for each main column. The lower row illustrates the sum of the upper scatter plots. Optimal regions are marked by ellipses – two are found in case of the tensor computation and one for the interpolation.

Figure 3.10 shows the results of the experiment, again as scatter plots of the parameters; but this time limited to the 10% best cases regarding the distance error $\mathcal{E}_H$. The top row provides the plots separated by circle and rectangle. The good regions for the FD parameters are quite complementary, when comparing the top row in (a) and (b). Thus, a weighting function chosen for optimal behavior at sharp corners is not the best for curved geometry. In an attempt to find an overall good parameter set, the two cases are summed up as shown in the bottom row. For (a) two optimal regions (I) and (III) are identified. Due to the greater m, (I) takes more distant information into account than (III). Thus, (I) was chosen as the overall best candidate for ω_T. The larger footprint is beneficial for noisy scenarios. For (b) the choice is generally less sensitive. A single optimal region (II) was identified with rather low values of T and m.

3.3.6 Centroid Revisited

Weighted Mean: As outlined above, different strategies of choosing the centroids $\boldsymbol{c}$ in the tensor computation were analyzed. The final selections and some visual evaluations are presented in Ritter et al. (2021a)~ and Ritter et al. (2021b)~. Here, the locations of centroids with increasing radius is investigated further, as well as the effect of the introduced weighting function onto the geometric median $\boldsymbol{c}_{L1\omega}$.

The use of a filter was introduced, as optional pre-processing step, to improve the locations of a noisy point set before starting the multi scale analysis and reconstruction process. The filter moves each point to the location of its neighborhood centroid of a given radius and weighting function. Thus, all original points in the cloud would be replaced and the subsequent steps would apply to the replaced, filter points. Figure 3.11 demonstrates three different variants of the pre-filtering on a noisy triangle geometry and illustrates the movement of points by the applied centroid filter with increasing filter radius. The brightness of the points relates to the size of the chosen neighborhood radius: light grey $r = 0$, black/black outline $r = 4$. The side length of the triangle is 10.

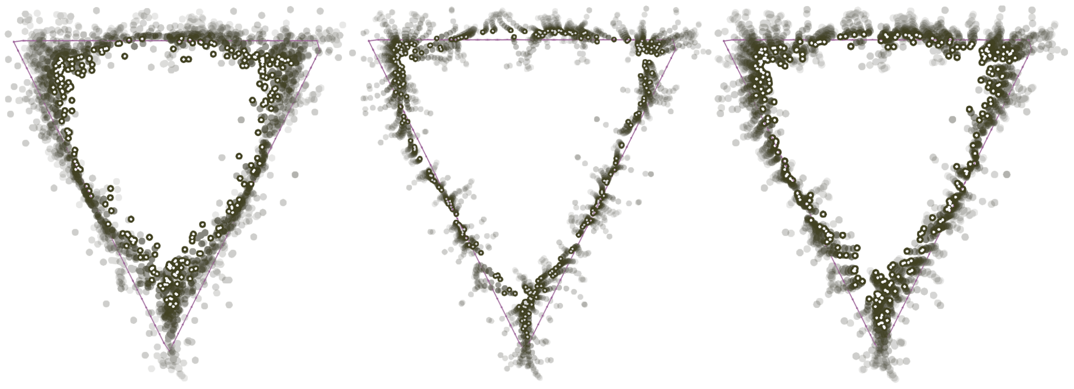

(a) Mean $\overline{c}$ (b) Median c_{L1} (c) Weighted Mean $\overline{c}_{\omega}$

Figure 3.11: An optional pre-filtering step allowed for an improvement of point locations for noisy data. A point is moved to its centroid using a specific radius r and weighting function ω_C; r is going from 0 (light points) to 4 (black/black outline points). The undistorted ground truth geometry is shown by lilac lines. Employing the c_{L1} preserves ground truth best, $\overline{c}$ worst, and $\overline{c}_{\omega}$ is located in between.

In case of the median (b) or weighted mean (c) points move on trails with increasing filter radius. No trails are noticeable for the mean; with increasing radius the points jump noisily. The final positions are marked by black outlined dots. Although the median can preserves the corners and a line structure better,

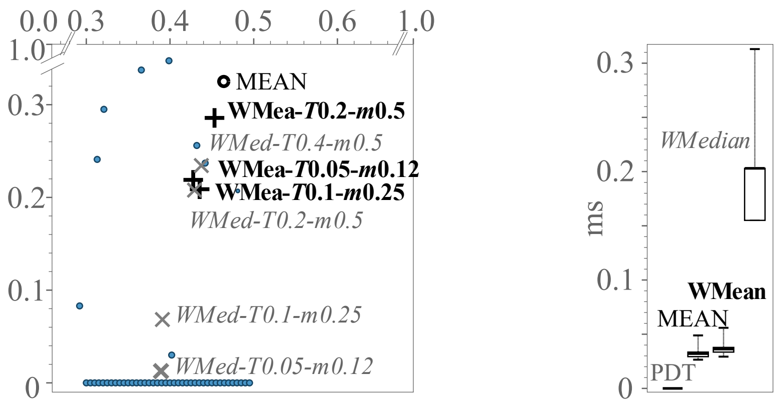


Figure 3.12: Comparison of location change and computational performance for different centroid methods. (Left:) Changing weight parameters moves weighted geometric median and weighted mean closer to denser geometry regions. Outliers are placed within $[0.0, 1.0] \times [0.0, 1.0]$; a dense line point cloud at the bottom. (Right:) Box plots of computation times per method: five geometries with 716 samples were tested twice.

it has a stronger tendency to form clusters as the weighted mean.

Besides the different geometrical behavior, the centroids exhibit varying computational complexities. Figure 3.12 demonstrates a specific test scenario for this. The influence of the weighting function ω_C as well as time measurements are provided. The test configuration consists of 2D points (blue circles) placed randomly within the domain $[0.0, 1.0] \times [0.0, 1.0]$. Additionally, a point-sampled line is inserted at $y = 0.0$, with $0.3 \leq x \leq 0.5$; in total yielding 716 points in 2D. Note that only a sub-region of the overall domain is visible in Figure 3.12 (Left); as indicated by markings on the axes.

As mentioned above, the geometric median as well as the weighted mean employ a weighting. The shape of the weighting function controls how strong the centroid is pulled towards denser point distributions. In the left subfigure, the mean $\bar{c}$ is depicted, as well as resulting locations for weighted mean $\bar{c}_\omega$ and weighted median $c_{L1\omega}$, when using different weight settings. The Fermi-Dirac function is employed to enable a continuous variation. Note that the weight parameters T and m can be used to move the geometric median (grey crosses) closer to the denser point region at the bottom (i.e. the sampled line). The same is true for the weighted mean (black plus signs); however, the position change is less pronounced.

For the time measurement (right subfigure) five random seeds for creating the 2D point test scene were employed twice; thus, the computation time box plots are based on ten measurements per method. In the examine 2D test scenario, the geometric median is on average five times slower to compute than the mean and weighted mean. The mean and its weighted variant exhibit similar performance. They scale similarly with the number of points, and are independent of the local spatial distribution in the neighborhood. In contrast, the median suffers from the addition iteration loop, while convergence also depends on the local distribution.

In the reconstruction framework, the centroid computations can be employed at several algorithmic stages, e.g. as a pre-filter on all point locations, as outlined above, replacing the original points completely; as the centroid in the tensor computation; or during the streamline integration. In the final version, the pre-filtering is not employed anymore. For the tensor neighborhood computation concerning the multi scale geometric measures, we suggest to employ the geometric median. Note that the geometric measures are pre-computed in parallel.

High Noise Rates: The effect of the centroids on the reconstruction performance in highly noisy data was also explored. A first test is indicated in Figure 3.13; a point-sampled circle with very high jitter, distribution, and outlier noise had to be reconstructed. The weighted median performed more robust, when used for starting point score, the geometric measures, and the local maximum score (Ritter et al., 2021a)~. Moreover, in this high noise case only one or two starting points were used.

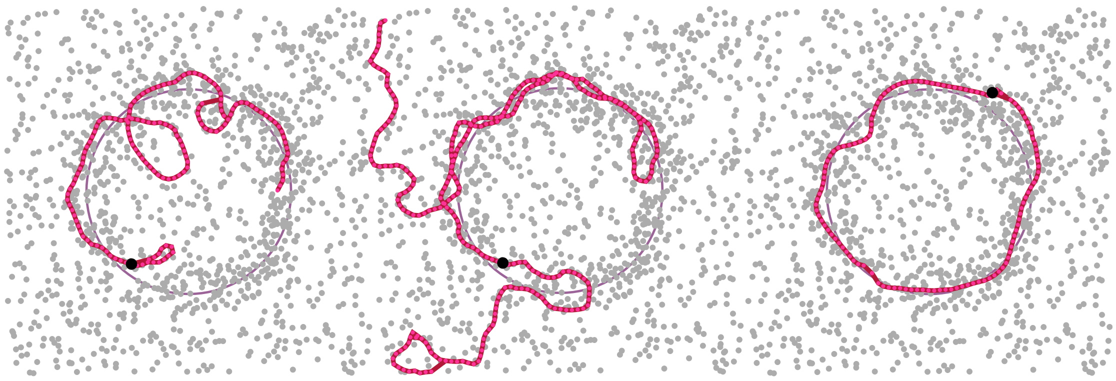


Figure 3.13: Reconstruction attempts in very noisy data ($S_D = S_J = 1.0, N = 452, M = 904$). Closed geometries, such as the circle, can be reconstructed in extremely noisy data with the proposed algorithm. The influence of different centroids is indicated. (Left to Right:) PDT, mean, weighted median.

Nevertheless, in such noisy cases the convergence of the geometric median can be slower; in this example the multi scale computation took $3300\,ms$. In contrast, using as centroid the PDT or the mean $\bar{c}$ took $2600\,ms$ and $2700\,ms$, respectively; albeit yielding unsuccessful reconstructions. In general, the computation time depends on the data set. As an example, the computation of the geometric measures for the 'monitor' case (Ritter et al., 2021a)~ requires 85 ms when using the median, 40 ms for the PDT, and 50 ms when using the $(\overline{c})$, respectively.

Further, the choice of the centroid has an influence on the starting point selection. Figure 3.14 shows the four best start point candidates for three different centroid types. While the mean provides better integration directions and radii,

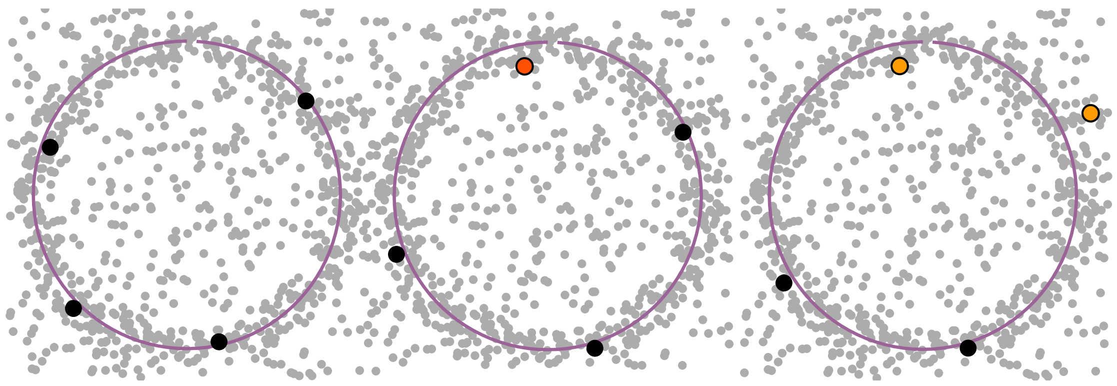


Figure 3.14: Centroid influence on start point selection (similar setup as in Figure 3.13); from left to right: existing point (PDT), mean, median. The best start points are found when using the point distribution tensor. Since the points remain fixed, candidates in the center of noise clusters are scored better.

the moved centroid results in higher starting point scores for locations at the border of or close to point clusters. In contrast, in the point distribution tensor an existing point of the cloud is used; this leads to higher scores for points centered within point clusters. These usually represent a better choice for starting the reconstruction. Thus, in the proposed final reconstruction framework, a PDT is always employed for computing the starting point score. Otherwise, the weighted mean $\overline{c}_\omega$ is chosen as centroid, with the quadratic inverse as weighting function. This was found to be the best compromise between reconstruction quality and computational performance. Finally, note that for extremely noisy cases start point candidates with less than six neighbors within a small radius were skipped (Ritter et al., 2021a)~.

Large Radii: The choice of the centroid also leads to differences in the multi scale geometric measures at large scale radii. Figure 3.15 depicts the geometric measures over a large radius scale of a jittered wave example. When using the median or mean centroid, the shape values converge to the same value over all points (yielding the same color in the top row of the visualizations). When using an existing, fixed point reference different values are reached for each point (i.e. no constant color in top row). Also, larger changes in the measure values result with growing radii.

In the reconstruction framework, a maximal analysis radius for the geometric measures can automatically be derived from point distance statistics. The absolute differences between the geometric measures of the current radius are compared to those of the previous radius. If for all points the change falls below a threshold, the computation is stopped. This automatic maximum radius detection only works for the median or mean variants, since the values converge to a similar value. In other cases, a maximum radius has to be specified by the user.

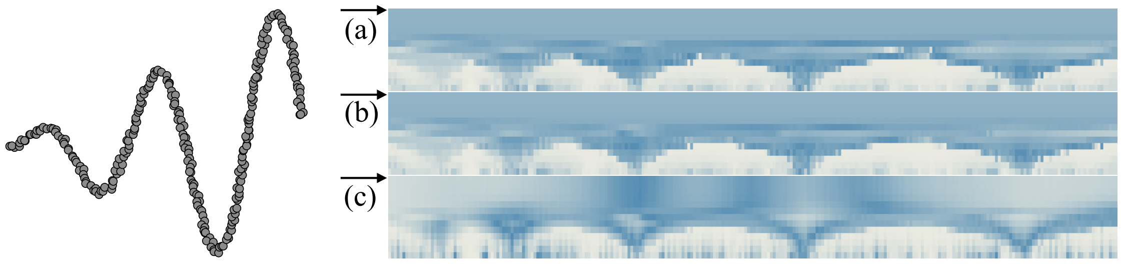


Figure 3.15: Multi scale geometric measures visualized as colormap images (Ritter et al., 2021b)~ of a wave geometry example (Left) with slight jitter noise. (Right; Top to Bottom:) resulting colormap images for median c_{L1}, mean $\bar{c}$, existing point centroid c. For large radii the median and mean converge to one single value; visualized as a row of same color at the top (a),(b). In contrast, using an existing point yields differing constant measures at large radii, but varying per point; non-constant color row (c).

3.3.7 Separated Noise Rate Performance

Data coming from sensors and real world surveying is usually distorted by noise. Four types of possible errors in the geometry relating to flaws that may happen in the data capturing were added in our experiments. Unsteady trajectories and vibrations of a moving vehicle introduce *jitter* noise. Intensity cut-offs in the light detection sensor or occlusion lead to data *holes*. Levitating dust particles or other small objects produce additional random signals as *outliers*. Varying densities of signals on the geometry may be introduced by shadowing, and multiple scan strips, leading to *distribution* noise. For instance, the latter can be included, by using uniformly distributed random numbers as parameters in the geometry creation (see Section 3.3.3). The noise rate n_R was derived to be able to create a summary success plot dependent on the noise, which is comparable for all geometries (Ritter et al., 2021a)~. As an extension, Figure 3.16 provides further error plots, employing n_R for the abscissa. The subfigures illustrate the use of the error rate, by plotting average distance error, the relative and absolute success rate, and the total number of target geometries. As can be seen, the average distance error ε_V increases with

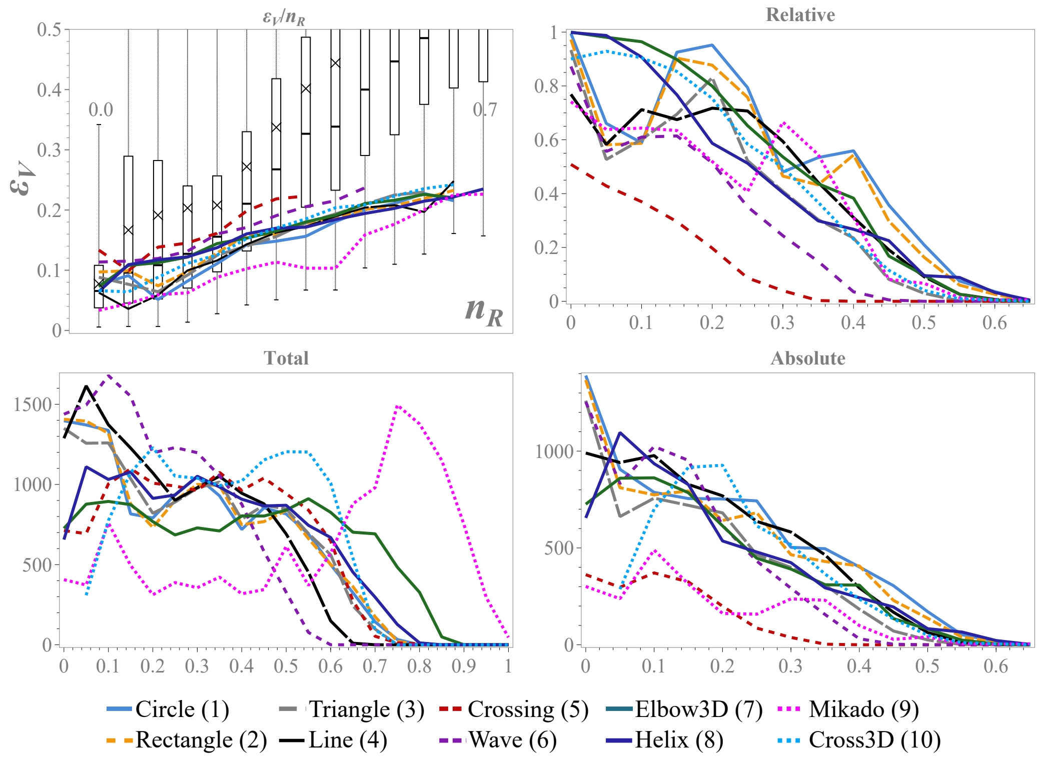


Figure 3.16: Success rates over combined noise rate n_R; as presented in Ritter et al. (2021a)~. The plots relate to the best selected WMN run shown by the bar plot in Figure 17 in the publication. The illustrated lines reveal more details of the decreasing rate over noise rate.

increasing noise rate (top, left). Related to that, the overall success performance (relative and absolute) decreases with increasing noise rate (top, right; bottom right). Note that in the plots the lines stop when no successful reconstructions could be generated. As defined in Ritter et al. (2021a)~, to be successful, the error measures, ε_H, (maximum distance to ground truth), ε_V (average distance to ground truth), ε_{Len} (length ratio), and ε_{Com} (completeness) have to fulfill certain criteria. In contrast, the box plots (top, left) indicate the distance error of all reconstructions, also including unsuccessful attempts.

As in the publication a parameter run of $127\,k$ variations was carried out as presented in Section 6.5, there. Similarly, the success performance can be read off, e.g. the *Wave* is the most challenging one with the lowest successes and the highest error. Note that the error was limited to be successful to $\varepsilon \leq 0.25$.

The bottom figures show the absolute number of all attempts ('Total') and the successful ones ('Absolute'). The total plot shows how the influence of the geometries on the noise rate. Ideally, all geometries would show the same trend.

Then, the noise rate would be fully independent of the geometry itself and capture noise properties only. This is not the case here. However, they show a rough similar trend, except for the *Mikado* where the noise rate estimate yield higher values. Figure 3.17 illustrates the dependency of n_R and the separate artificial noise rates.

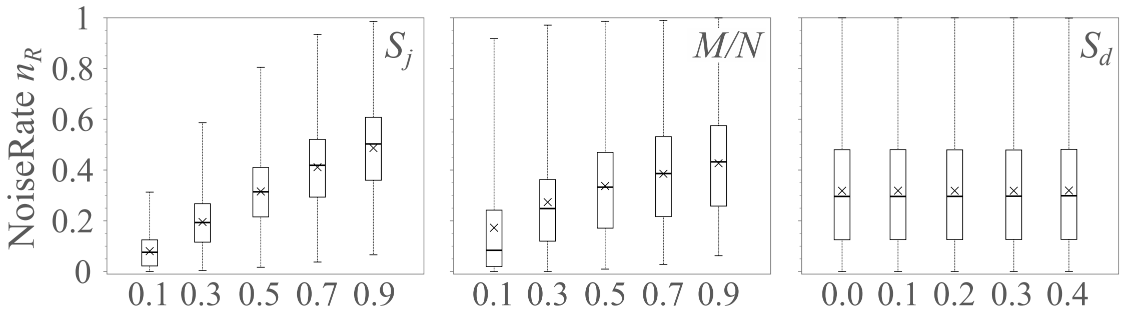


Figure 3.17: Artificial noise rates and noise rate n_R illustrated via box plots of $127\,k$ reconstruction attempts. The noise rate increases with jitter noise S_J, less with outlier noise $S_U = M/N$ and is independent of distribution noise S_D.

The noise rate n_R is positively correlated with jitter noise (left). This shows a linear trend. Further, it increases with outlier noise (mid), but is independent of distribution noise (right).

Figure 3.18 illustrates successful reconstructions dependent of each artificial noise rate. The first column of the plots shows the influence of the different noise types on the overall reconstruction average distance error; similar to Figure 3.16 (top, left). There is almost no influence on the reconstruction error due to distribution noise (mid). The average error stays constant. Also, the success rate itself varies only marginally. The jitter noise relates strongest and quite linearly with ε_V (top). The absolute success rates also drop smoothly with increasing noise (top, right). Also, geometries perform differently. While the success of *Circle* and *Rectangle* starts decreasing at $S_J > 0.4$, the other geometries decrease earlier. *Wave* and *Mikado* from the very start. The wave performs worst and is quite fragile to jitter noise, due to the high curvature regions. The uniform noise has a almost constant relation to the average reconstruction error (bottom). The overall performance depends on the geometry (right). Again, the wave shows the worst success rate. Note, the plots always include all noise types. They are just presented dependent on the different type. E.g. all the jitter and distribution noise variations are included in the leftmost box plot of the uniform average error plot (bottom, left).

The overall shapes of the error graphs demonstrate that the reconstruction process was made robust against distribution and uniform noise. The success rates

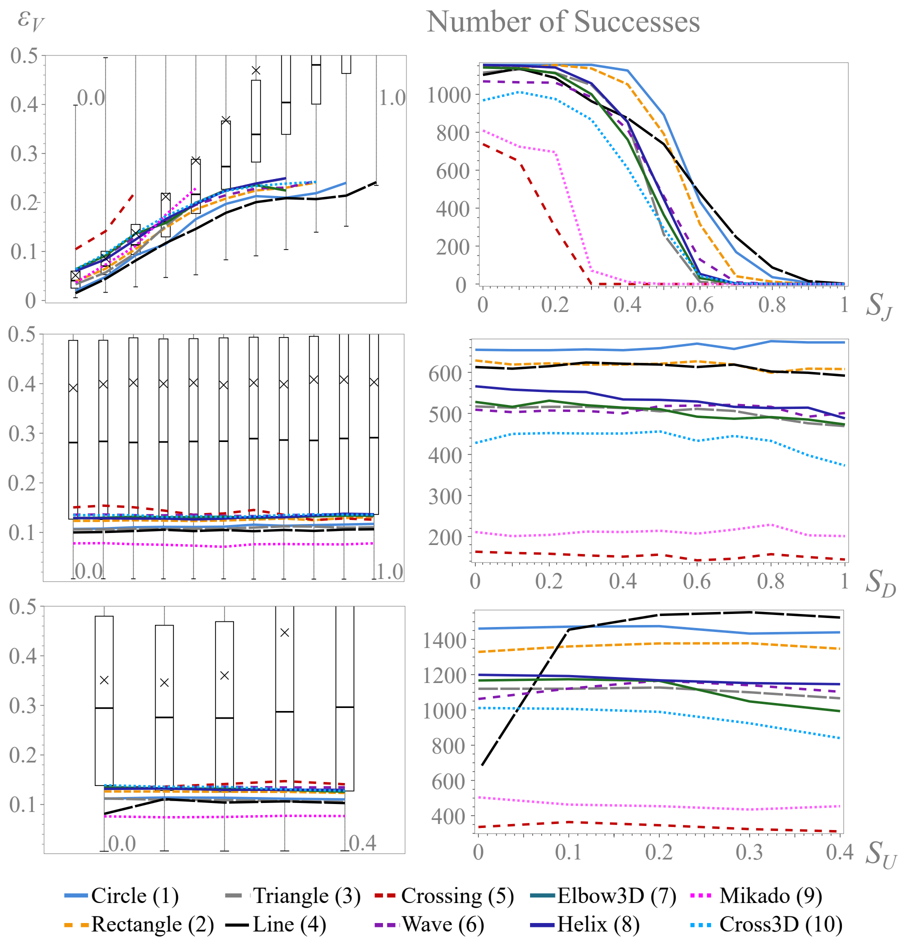


Figure 3.18: Success rates shown for each single noise measure; (Top to Bottom:) S_D, S_J, and S_U. Here, the influence of the different types of noise can be ready seen more precisely, but many more plots are required. (Left to Right:) ε_V and absolute success rate over each distinct artificial noise rate. The box plots show the error of all (also not successful) reconstruction attempts.

stay constant or decrease only slightly. Jitter noise has the strongest influence. This is an expected behavior as the noise distorts the geometry out of the tangential direction. Especially, in high curvature regions the ground truth tangential direction is destroyed. This results in high uncertainties in the reconstruction of such regions. The combined noise rate plots (Figure 3.16) is not as clearly readable as the separated analysis. However, it allows to combine all the noise types and illustrates the overall trends correctly. The geometries were explicitly chosen with very different features, such as constant curvatures, sharp corners, varying curvature, crossings, 2D/3D.

3.3.8 Fourth Order Neighborhood Tensor

A possible extension to the second order considerations is the (still preliminary) idea of carrying out an additional uplift of the second order tensor. Analogously to encoding many directions of a point neighborhood into one tensor, it may be possible to encode all second order tensors of the multi scale tensor into one 4th order tensor. Formally this is defined as:

$$\underset{\sim}{\boldsymbol{T}} = \frac{1}{\sum_{i=1}^{N} \omega_4(r_i, \underset{\sim}{\boldsymbol{t}}(r_i))} \sum_{i=1}^{N} \omega_4(r_i, \underset{\sim}{\boldsymbol{t}}(r_i)) \left(\underset{\sim}{\boldsymbol{t}}(r_i) \otimes \underset{\sim}{\boldsymbol{t}}(r_i)\right), \tag{3.28}$$

with i the index of a multi scale radius, $\underset{\sim}{\boldsymbol{t}}(r_i)$ the 3×3 neighborhood tensor at that radius, $\omega_4(r_i, \underset{\sim}{\boldsymbol{t}}(r_i))$ a weighting function dependent on the radius and the tensor at that radius (e.g. to employ a shape factor for weighting), and $\underset{\sim}{\boldsymbol{T}}$ the fourth order $3 \times 3 \times 3 \times 3$ multi scale neighborhood tensor. Note, that due to the symmetry $\underset{\sim}{\boldsymbol{t}}_{ij} = \underset{\sim}{\boldsymbol{t}}_{ji}$, also $\underset{\sim}{\boldsymbol{T}}$ is symmetric in many components: $\underset{\sim}{\boldsymbol{T}}_{ijkl} = \underset{\sim}{\boldsymbol{T}}_{jikl} = \underset{\sim}{\boldsymbol{T}}_{ijlk} = \underset{\sim}{\boldsymbol{T}}_{ikjl} = \ldots$, etc. Thus, the 81 components in effect reduce to 15 unique ones. Figure 3.19 visualizes the major eigentensor (of 2nd order) of $\underset{\sim}{\boldsymbol{T}}$, as colored ellipses at different point cloud locations.

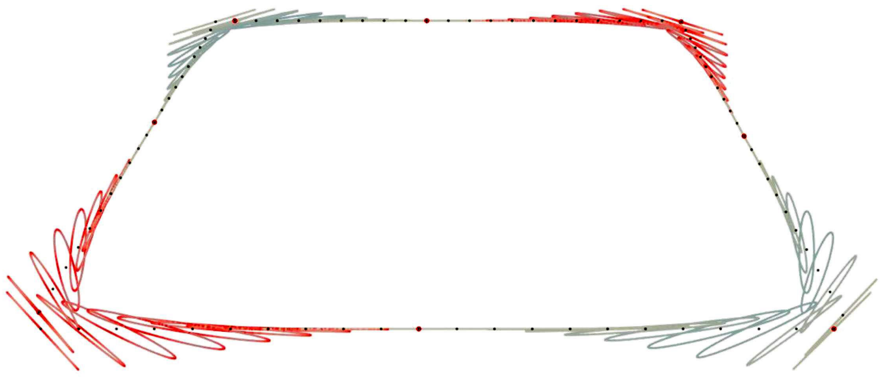


Figure 3.19: Tensor ellipses are provided to illustrate second order tensors. The respective major eigentensor of an eigenanalysis of the fourth order multi scale neighborhood tensor is shown.

For the implementation, both the most compact super-symmetric representation as well as the Mandel notation layout, utilizing 21 components, was employed. The latter is specialized for the occurring symmetries (Abbasloo et al., 2016). For further description of Voigt and Mandel layouts, consider Brannon (2018). Both implementations provide internal look-up tables for direct tensor index access. The source code is also part of Ritter (2021). Nevertheless, the use of the 4th order tensor was not explored further in the scope of this thesis.

4 Computational Performance Optimization

4.1 Preliminaries

A key component of the computation of all the neighborhood tensor or covariance variants in a point cloud is finding the involved neighboring points: i.e. finding for one point the points inside a local neighborhood, or finding a cluster neighborhood, without an explicit point of reference. For performing the related range queries several algorithms already exist and can be employed. However, algorithms can be chosen more optimally dependent on the underlying hardware or software infrastructure and on the final computational intention. Besides the pure conceptual performance of the algorithms, they also use different memory orderings of the stored data. This has an additional impact on the run time performance due to cache misses, dependent on the hardware. Within the scope of this thesis the following algorithms for neighborhood searches were implemented and analyzed: 1a) kd-tree, 1b) octree, 2a) uniform grid hashing & bitonic sorting, 2b) uniform/perspective grid accumulation, and 3) 2D sampling in screen space. The tree-based methods were implemented on the CPU, the grid-based ones on CPU and GPU. The 2D sampling is a GPU-only implementation, in screen space including depth information. Next, brief overviews of the employed techniques and results are provided.

4.1.1 Tree Neighborhood Searches

Tree-based neighborhood searches divide space into subspaces, and organize and connect those hierarchically. When looking for neighboring points, the corresponding subspace can be found quickly, and then traversed from there to other candidate subspaces. Data, such as point coordinates, can be stored directly or via an index inside the tree. Also, meta data or accumulated data can be stored inside subspace nodes.

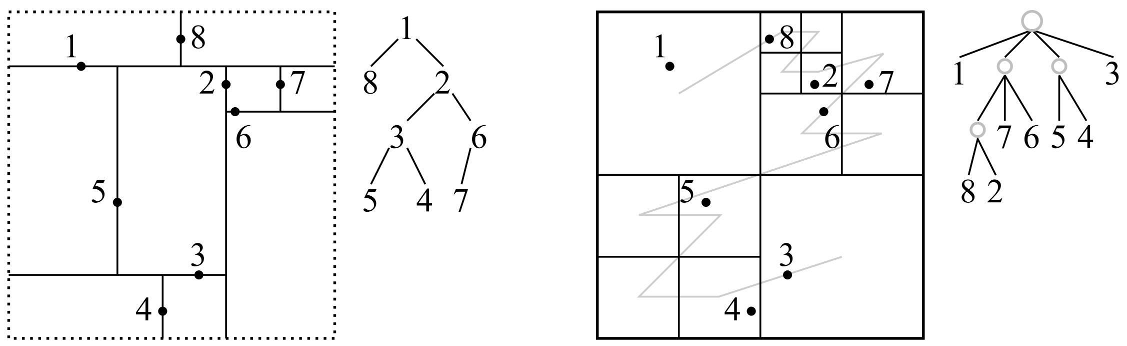


Figure 4.1: (Left:) Illustration of points organized as kd-tree. Here, the binary tree is constructed by rotating axis aligned hyper planes after each insertion. (Right:) An octree splits space by hyper cubes with half the side length of the outer cube. Each tree node may hold up to 2^{dim} children. The grey line indicates the equivalence to the Z-curve, which creates the same serialization of the points.

Kd-tree: Figure 4.1 (Left) shows the concept of a kd-tree. Here, space is split by hyperplanes (space dimension minus one) in alternating, axis-aligned orientations. This procedure allows to create a binary tree for N-dimensional spaces. The specific spatial organization depends on the order of points inserted into the tree. In the example shown, space is split with each point inserted – yielding a new node in the binary tree. Points are labeled according to their insertion order. Each consecutive split refers to the actual subspace. In this example, the two dimensional space is divided by one dimensional hyperplanes (i.e. lines). A node is inserted left in the tree if being left or above the current node's dividing line; and inserted right in the tree, otherwise. The kd-tree was the first chosen search algorithm in our work, since it allows for an N-dimensional implementation and results in a 'simple' binary tree.

Octree: As an alternative, the octree was investigated and implemented. Here, space is split into hypercuboids of equal size; in each level of split: four rectangles in 2D and eight cuboids in 3D. The example in Figure 4.1 (Right) shows the same points of the kd-tree example organized in an octree. Each node of the octree has up to four children. Empty sub-spaces are omitted in the tree structure. Octrees are the de facto standard in computer graphics, as they can be combined well with voxel and space filling curve based algorithms, such as the Z-curve. The light gray line illustrates the Z-curve ordering of the octree cells in 2D. Note that the order of the octree nodes, read from left to right, follows the same order as for the curve. Z-curves and octrees are equivalent in terms of data ordering and, thus, compatible in algorithmic designs. In our implementation a leaf node is a bucket and may contain up to N points, before it is split further. $N = 64$ was found to yield the best performance for 3D points represented in 32-bit floating point numbers. By using buckets, memory organization is beneficial with respect to cache misses. In contrast to the kd-tree, octrees require a reasonably good bounding hypercuboid[1] for initialization; before inserting the first data point. For some algorithms this information is not available.

Trees as spatial search structures have the advantage that they allow to skip large subspaces of the data; e.g. for range queries, such as selecting all points close to a certain position, within a certain radius. All subspaces farther away than the radius can be skipped. Tree structures adjust well to highly irregular data, i.e. when there are regions with a very high and with a very low (point) density within the same data set. Moreover, trees usually do not allow for a direct node access. Tree nodes have to be traversed from the root node to find the desired subspace, or, traversed from a subspace up and down in the hierarchy. Bidirectional connections are not necessarily available. Note that creating a tree already requires to iterate through the data once. Which can be inefficient, when a certain query can also be already solved by one full linear data iteration. Finally, it is not straight forward to implement trees on GPUs, as they usually build on maps or pointer based data structures. GPU memory access is organized locally to the many threads (and work groups) and memory buffers have to be prepared in a compatible structure.

Concerning the work in this thesis, the kd-tree was employed in the initial developments: Ritter et al. (2012)*, and Ritter and Benger (2012)*. The octree was later introduced via Schiffner et al. (2014)* and then used in the further developments – as it showed superior performance: Schiffner et al. (2015)*, Grasso et al. (2015)*, Ritter et al. (2021a)~, and Ritter et al. (2021b)~.

4.1.2 Grid based Methods

Grid Hashing and Bitonic Sorting: Using grid structures is another way to organize subspaces for point cloud data. Points can be inserted into cells, e.g. by storing point indices together with a cell identifier. Uniform grids are especially suited, since a point location can directly be transformed into an integer coordinate (index of a grid cell); by knowing the bounding box and resolution of the grid. Once points are organized in a grid, neighbors can be quickly found by iterating through neighbored grid cells and the associated points.

In this thesis, also a grid based method has been introduced. Its focus was on a pipeline for general purpose computation in the neighborhood of possibly large point clouds in full detail; while utilizing GPU and CPU resources. OpenCL was chosen for the implementation to support all possible, heterogeneous compute devices. The point distribution tensor was chosen as an algorithmic 'application'

[1] bounding box in $3D$

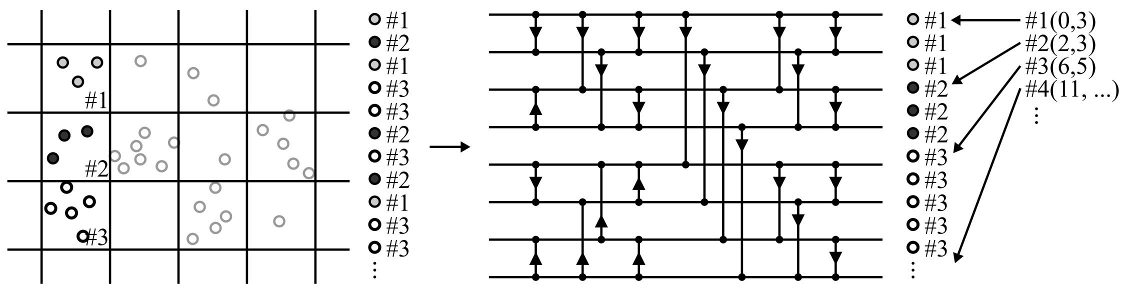


Figure 4.2: Grid hashing and sorting: points are associated with a cell identifier (left) and then sorted by that hash. A sorting network for eight numbers is illustrated (Peters et al., 2011). An additional lookup structure is set up (cell start index and cell size) to finally allow direct access of a cell in the sorted array (right).

operating in point cloud neighborhoods. To enable the support of possibly large data an out-of-core approach had to be followed. Data was first pre-processed into large data fragments – a large scale uniform grid – each fragment was then processed using the fine grain algorithms. To avoid border artifacts, data was duplicated at the borders at so called ghost zones. Ghost zones have to be as large as the neighborhood radius used in the fine grain algorithm. The overhead of data duplication was acceptable and enabled a completely independent computation per fragment. Thus, a data fragment can be sent to any compute device on any platform holding all necessary information. The pre-processing was implemented on a converter level and provided an enriched HDF5 file.

Next, each point in a fragment was labeled by an identifier of a grid cell (hash), as can be seen in Figure 4.2. Points were then sorted according to their hash value. A lookup structure was built to find the start index of the serialized cell sorted points per cell. Thus, the sorted point array can directly be accessed by a cell index, by first retrieving the cell start index in the point array (or a data array sharing the index layout). In a first pass data is sorted and in a second pass the point distribution tensor computed. Neighbors can now be efficiently found by traversing the points of a cell and of the neighboring cells. Here, the cell size was chosen, such that a maximum of $3 \times 3 \times 3$ cell neighborhood in 3D had to be traversed.

For sorting the bitonic algorithm was chosen. It is especially suited for parallelization and can also be hardwired. Figure 4.2 illustrates such a sorting 'network' (Batcher, 1968). A number can be input for each line and is compared and possibly replaced along the arrow connections. Here, a sorting of 8 numbers requires 6 steps. Each step involves a possible switch of two independent data pairs, allowing for a fine grain parallelism in each step. This approach fits well on the GPU architecture, as it is built from many small computational units. An open source bitonic sort implementation of NVIDIA was rewritten in OpenCL for our work.

Performance was evaluated by comparing to results from an earlier independent implementation. Compared to the first kd-tree based CPU parallel implementation a speed up of about 24 was achieved by switching to the grid based approach; and another speed up of again about 24 by utilizing GPU hardware (AMD Fire Pro S9000). The work further analyzes optimal data fragment scheduling on heterogeneous computing devices (see Grasso et al. (2015)* below).

Grid Cell Accumulation on the GPU A visualization representation of a point cloud does not necessarily require to draw a geometric object for each point. Especially, in laser scanning higher level geometry such as surfaces, lines, or volumes are just sampled by points. A viewer may be more interested in seeing what was scanned; i.e. the higher level geometric objects. The key idea of the here proposed method is to work with the limited resources of a GPU and create higher level geometric representatives for low level point clouds on the fly in real time. Having a limited space and computation time available, a grid structure of fixed size is defined. The size depends on the available resources and is chosen, such that the performance is optimal and resolution (visual level of detail) is feasible. For implementation the (then newly introduced) **atomic add** of GLSL 4.3 was employed[2]. This allowed to accumulate data along with vertex clusters; for each grid cell. The three key aspects of the algorithmic design were:

1. Create high level representatives for geometry in real time on the GPU.
2. Use a fixed number of representatives dependent on available GPU resources.
3. Accumulate representatives from unsorted point cloud data.

Point cloud data is uploaded raw and unsorted. A three pass method is applied: First, points are *clustered* in 3D grid cells (uniform or affinely transformed uniform) – as illustrated in Figure 4.3 – and additional properties, e.g. the mean center is accumulated during the insertion. The second pass *moves* points to a neighboring cell if the point is close to the cell's border and if the neighboring cell is less populated. The third pass *reduces* the data per cell and computes data required for a higher level representative.

In Schiffner et al. (2014)* the basic algorithm was introduced and tested on different data sets stemming from LiDAR as well as an astrophysics SPH simulation. The origin of the data sets is described in more detail in Section 5.1.1. Also, the planarity shape factor was included into the *move* operation to encourage cells locally to be more planar. Uniform grids and perspective grids were tested with different resolutions. Real time performance was reached for a few million points. Compared to a CPU parallel implementation a speed up of one order of magnitude was achieved. Visual results were good for the LiDAR data, but not so for the

[2]https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/atomicAdd.xhtml

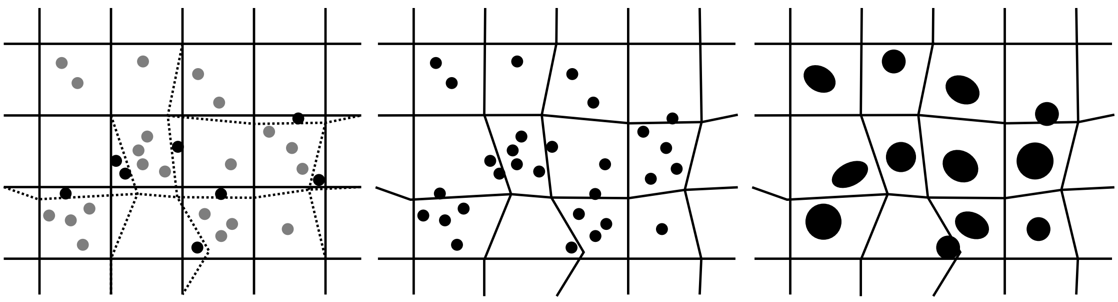


Figure 4.3: Raw points are clustered into grid cells (left), moved to neighbors when relatively densely populated and close to a cell border (center), and finally reduced (right), e.g. displayed as oriented splats. All passes are executed in real time on the GPU via compute shaders. The reduced data is then employed for final rendering. The move pass allows the grid to adjust to the underlying point set.

SPH simulation. Here, the *move* pass was not able to adjust to the wider range of scales in the data.

In Schiffner et al. (2015) the method was extended further mainly on the part of visual representatives; i.e. oriented splats, cell filled quads and boxes were created in a geometry shader. Also, a base line comparison to an octree based clustering was added. Different normal estimation methods were employed; a tensor based normal estimation showed best results, but had the highest computational complexity. A normal estimated by pointing in the direction of empty cells in combination with a larger cell neighborhood performed better with respect to visual quality and performance.

Screen Space Sampling

When the neighborhood information is required for rendering only, instead of general purpose computing or algorithms, the standard rendering pipeline can directly be utilized. Our developed method estimates a normal vector of a raw – i.e. unsorted – point set after rasterization in the fragment shader stage.

As illustrated in Figure 4.4, vertices of a point cloud are 'projected' onto the virtual camera image plane. With an enabled depth test, the resulting fragments provide depth values besides color and are organized in a 2D grid. In the simplest form, normals can be computed using the GLSL derivative functions (**dFdx** and **dFdy**). However, this turned out to be very unstable. Thus, small neighborhoods were defined, up to a sample count of 25; preferring a diagonal layout around the center fragment. Also, a multi depth buffer was employed; up to eight depth values were unprojected per fragment into world space to compute an estimated normal.

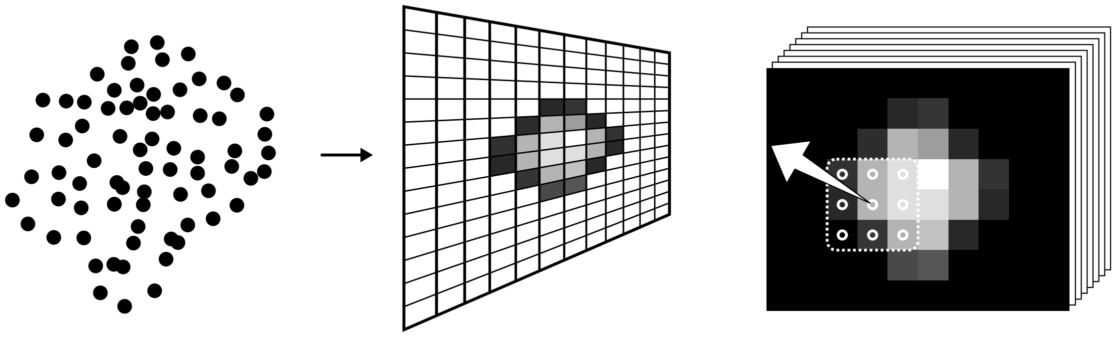


Figure 4.4: Screen space normal estimation. A point cloud (left) is projected onto the image plane (center). A depth image allows to operate on pixel neighbors to estimate a surface normal, here illustrated in a 3×3 neighborhood.

Points with too large distance to the point of reference were discarded from the computation in world space. In total, four estimation methods were tested:

1. GLSL derivative functions;
2. cross product of the unprojected samples area diagonal vectors;
3. cross product of direction vectors from the center to all neighboring unprojected samples in counter clock-wise ordering;
4. the minor eigenvector of the weighted point distribution tensor.

An optional smoothing pass was implemented operating on the estimated normals as a post processing step. It was possible to achieve feasible results in real time with a lower sample number, e.g. with the diagonal preferred sampling. The tensor estimation created the smoothest visual results. High sample counts improved the quality at cost of a low frame rate. Test cases and one LiDAR data set showed, that the method's applicability depends on the point size and point cloud density with respect to the frame buffer. Ideally, the point density provides a different point for each, or a few pixels. For the LiDAR data set this could only be achieved by overview visualizations and not by inspecting close ups. Therefore, an application could be in the quick pre-visualization of yet unprocessed raw point cloud data allowing a shaded view for users to find regions of interest for detailed processing. More information is provided below in Schiffner et al. (2013)*.

Procedia Computer Science
Volume 51, 2015, Pages 160–169
ICCS 2015 International Conference On Computational Science

Point Distribution Tensor Computation on Heterogeneous Systems

Ivan Grasso[14], Marcel Ritter[234], Biagio Cosenza[1], Werner Benger[345], Günter Hofstetter[2], and Thomas Fahringer[1]

[1] Institute for Computer Science
[2] Institute for Basic Sciences in Engineering Science
[3] Institute for Astro- and Particle Physics
University of Innsbruck, Austria
[4] AHM Software - Airborne Hydromapping
[5] Center for Computation & Technology at Louisiana State University

Abstract
Big data in observational and computational sciences impose increasing challenges on data analysis. In particular, data from light detection and ranging (LIDAR) measurements are questioning conventional methods of CPU-based algorithms due to their sheer size and complexity as needed for decent accuracy. These data describing terrains are natively given as big point clouds consisting of millions of independent coordinate locations from which meaningful geometrical information content needs to be extracted. The method of computing the point distribution tensor is a very promising approach, yielding good results to classify domains in a point cloud according to local neighborhood information. However, an existing KD-Tree parallel approach, provided by the VISH visualization framework, may very well take several days to deliver meaningful results on a real-world dataset. Here we present an optimized version based on uniform grids implemented in OpenCL that is able to deliver results of equal accuracy up to 24 times faster on the same hardware. The OpenCL version is also able to benefit from a heterogeneous environment and we analyzed and compared the performance on various CPU, GPU and accelerator hardware platforms. Finally, aware of the heterogeneous computing trend, we propose two low-complexity dynamic heuristics for the scheduling of independent dataset fragments in multi-device heterogenous systems.

Keywords:

1 Introduction

Point datasets are present in many scientific domains. Smoothed particle hydrodynamics methods in astrophysics [4], echo sounding in engineering, 3D surface reconstruction [16] and urban reconstruction [20] in graphics are typical examples where data generated by numerical simulations or observations are processed as point primitives. Today's range-sensing devices are

doi:10.1016/j.procs.2015.05.217

capable of producing highly detailed point datasets containing hundreds of millions of sample points. Light detection and ranging (LIDAR) technology, in particular, allows collecting millions of data points e.g. from airborne scanners in order to produce high-resolution digital elevation maps. However, depending on the application, large point datasets may require a prohibitively high computational power for processing.

This paper is motivated by the LIDAR surveying application of shallow waters [7] where, starting from a point dataset, a second order tensor field is computed and used as a basis for several other algorithms such as point classification and geometry reconstruction [22]. A first CPU-based parallel implementation of such a point distribution tensor was implemented in the VISH visualization shell [5]. VISH is a productive framework that provides functionalities for both efficient data processing and visualization of big data. However, as surveyed datasets grow from several thousand of points to many millions of points, the tensor computation becomes a bottleneck for data processing.

In this paper we focus on exploiting the computational power of emerging heterogeneous computing systems in order to improve the tensor computation of massive datasets of millions of points. Our study makes the following contributions: First, we implemented a new tensor computation code in OpenCL using a uniform grid space partitioning approach, and evaluated its performance against the current KD-Tree implementation available in VISH. Second, we investigated the performance of the implemented code on 8 different devices, comprising four GPUs, three CPUs and one accelerator, from desktop and server domains. Finally, we proposed two low-complexity dynamic heuristics for the scheduling of independent dataset fragments and compared them with three static scheduling heuristics in two multi-device heterogeneous systems.

2 OpenCL Programming Model

OpenCL [14] is an open industry standard for programming heterogeneous systems composed of devices with different capabilities such as CPUs, GPUs and other accelerators. The platform model consists of a host connected to one or more compute devices. Each device logically consists of one or more compute units (CUs) which are further divided into processing elements (PEs). Within a program, the computation is expressed through the use of special functions, called kernels, that are, for portability reasons, compiled at runtime by an OpenCL driver. A kernel represents a data-parallel task and describes the computation performed by a single thread, which is called *work-item* in OpenCL. During the program execution, based on an index space (N-Dimensional Range), a certain number of work-items are generated and executed in parallel. The index space can also be subdivided into workgroups, each of them consisting of many work-items. The exchange of data between the host and the compute devices is implemented through memory buffers, which are passed as arguments to the kernel before its execution. In the past few years, OpenCL has emerged as the *de facto* standard for heterogeneous computing, with the support of many vendors such as Adapteva, Altera, AMD, ARM, Intel, Imagination Technologies, NVIDIA, Qualcomm, Vivante and Xilinx.

3 Tensor Computation

For a set of N points $\{P_i : i = 1, ..., N\}$ the point distribution tensor S at the point P_i is defined as:

$$S(P_i) = \frac{1}{N} \sum_{k=1}^{N} \omega(|t_{ik}|)(t_{ik} \otimes t_{ik}^{\tau}), \qquad (1)$$

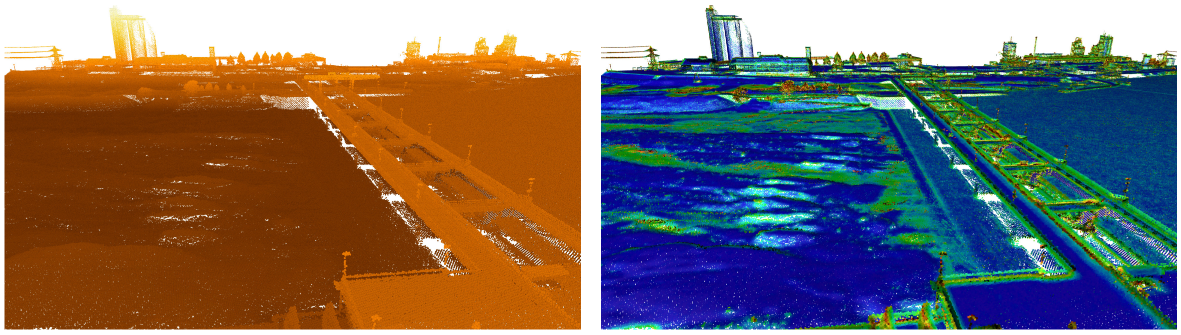


Figure 1: Input point distribution (left) and output tensor (right) of the river Rhein dataset. For the input points, the height (z-axis) drives a colormap (left) while the output tensor is used for coloring (planarity) and surface shading in VISH (right).

whereby $\omega(x) = \theta(r - x)$ is a threshold function dependent on a radius r [22], $t_{ik} = P_i - P_k$, $^\tau$ is the transpose and $\otimes$ denotes the tensor product. A graphical result of the computation is depicted in Figure 1.

The naive approach for the tensor computation is therefore to test every point with all the others, leading to a quadratic algorithmic complexity. In real models composed of millions of points this approach is not applicable due to the inherent performance problem. To mitigate this problem spatial partitioning methods have been investigated [2, 25, 26].

KD-Tree Implementation. The tensor field computation algorithm, currently implemented in VISH, makes use of a KD-Tree data structure to find the neighbors of a certain point. After the tree building phase, in which the points of the dataset are inserted into the KD-Tree, the computation of the tensor distribution is executed for each point with a series of range queries dependent on a given search radius (threshold function). The computational loop over the points was parallelized using OpenMP with dynamic scheduling and packets of 10000 loop iterations. The KD-Tree code was integrated in a computational VISH module, and implemented via C++ templates and STL containers.

Uniform Grid OpenCL Implementation. A uniform grid space partitioning approach involves a spatial partitioning of the model system into equally-sized boxes (cells) containing different numbers of points. It is important to ensure that the grid box size is not smaller than the radius size, as this would force the algorithm to check many surrounding grid boxes. On the other hand, if the grid box is larger than the radius, each box would contain numerous points and the process of locating neighbors would once again be checking many points outside the radius area. In our case, the uniform grid approach is effective because the radius is an input parameter of the program and therefore we are able to tune the grid box size accordingly.

We implemented the tensor computation application in OpenCL using a uniform grid space partitioning approach. We used a grid with a cell size of two times the radius, which implies that each point can only interact with points in the neighboring cells (27 in a 3D space). The complete program, described in Algorithm 1, is composed of three phases: initialization, computation and finalization. During the initialization phase, the OpenCL devices are initialized, the OpenCL kernels are compiled and the metadata of the dataset is loaded. The metadata contains information about the number of fragments present in the dataset, the number of points for each fragment, plus other additional information useful for the graphical visualization. The dataset consists of independent fragments of spatially ordered points to facilitate the

Algorithm 1 The OpenCL tensor computation algorithm

```
 1: $devices\_initialization()$                                  ▷ Initialization Phase
 2: $metadata \leftarrow load\_dataset\_metadata()$
 3: for all $fragments$ in $dataset$ do                         ▷ Computation Phase
 4:     $pts\_ar \leftarrow load\_points\_data(fragment)$
 5:     $write\_points\_to\_device(pts\_ar)$
 6:     $create\_uniform\_grid(pts\_ar, radius)\{$
 7:         $hash\_ar \leftarrow compute\_hash\_values(pts\_ar)$
 8:         $index\_ar \leftarrow sort\_points\_indices(hash\_ar)$
 9:         $begin\_end\_ar \leftarrow compute\_interval(hash\_ar)$
10:     $\}$
11:     $compute\_tensor(pts\_ar, index\_ar, begin\_end\_ar)$
12:     $tsr\_ar \leftarrow read\_tensor\_from\_device()$
13:     $write\_tensor\_to\_disk(tsr\_ar)$
14: end for
15: $devices\_finalization()$                                    ▷ Finalization Phase
```

data manipulation and visualization. Each fragment contains a small percentage of replicated data necessary for the computation of the tensor algorithm at points close to the border of the fragment. Once the initialization phase is completed the system is ready to schedule the fragments on the available devices and the computation phase will start.

For each fragment the point's coordinates will be loaded in main memory and transferred to the device memory where the computation will take place. On the device, the uniform grid will be created and used during the tensor computation in the search for the neighboring points. Once the computation is done, the computed tensor data is transferred back to the host's main memory and finally saved to the disk. The finalization phase releases all the devices and the used memory.

The steps necessary for the creation of the uniform grid are described in Algorithm 1 (lines 6-10). The algorithm consists of multiple OpenCL kernels. The first kernel (line 7) calculates a hash value for each point based on its cell ID and stores them in an array in device main memory (*hash_ar*). The array is then sorted based on the cell IDs while updating at the same time the order of the point IDs. Sorting is performed using a bitonic algorithm. The result of this computation is an array of point IDs sorted by cell (*index_ar*). The last kernel (line 9) is then executed to find the begin and the end position of any given cell. The kernel generates an OpenCL work-item for each point and compares the cell ID of the current point with the cell ID of the previous one in the *hash_ar* array. If the two indices are different, the current work-item ID is used as start index of the current cell and the *begin_end_ar* array is updated using a scattered write operation. During the execution of the *compute_tensor* kernel (line 11), using the *begin_end_ar* and *index_ar* arrays, we calculate the neighbor cells for each point in the fragment and for each point present in the cells we compute the difference to the current point in each dimension (x, y, z). If the length of the difference vector is less than the radius, the tensor array and the points counter are updated. Finally, in the last step, each element of the tensor array is divided by the points counter.

4 Scheduling Independent Fragments

As previously mentioned in Section 3, the tensor computation is applied on single fragments that compose the complete dataset. The fragments are completely independent of each other and can be computed in parallel using the available devices present in the system. During program execution a scheduler is responsible for the allocation of the fragments among the heterogeneous devices. The scheduling problem has been extensively investigated and numerous methods have been reported in the literature [6, 18, 17]. In our program we implement two low-complexity scheduling heuristics: *SimpleH* and *SimpleHS*. *SimpleH* analyzes the dataset metadata and sorts the list of fragments based on the number of points contained in each of them. The algorithm then proceeds by dynamically assigning the fragment with the smallest number of points to the slowest device and the fragment with the biggest number of points to the fastest device. Following this pattern, the scheduler continues to dynamically assign fragments until all of them are processed. *SimpleHS* follows a similar pattern. A fragment is assigned to the slowest device, if the predicted execution time of the fragment on that device is lower than the predicted execution time of all the remaining fragments on the fastest device. The execution time for each fragment is predicted with a quadratic regression model using the number of points of the fragment. During the program execution, information regarding number of points per fragment and execution times are stored. These information will then be used to build a more accurate model whenever the slowest device is ready to compute a new fragment. Although for simplicity the heuristic algorithms are described taking into consideration only two devices, they can be applied to heterogeneous systems composed of a single slow device (CPU) and multiple equally fast devices (e.g. GPUs). In Section 6 we evaluate and compare *SimpleH* and *SimpleHS* with three heuristics which are widely used to address the problem of scheduling independent tasks in heterogeneous computing systems: *Min-Min* [13, 6], *Max-Min* [13, 6], and *Sufferage* [19]. Because these are static heuristics, it is assumed that an accurate estimation of the expected execution time for each fragment on each device is known prior to execution and contained within an ETC (expected time to compute) matrix. The *Min-Min* heuristic proceeds by assigning a previously unassigned fragment to a device in every iteration. The assignment is decided based on a two-step procedure. In the first step, the algorithm computes the minimum completion time (MCT) of each unassigned fragment over the devices in order to find the best device which can complete the processing of that fragment at earliest time. This decision is made taking into account the current loads of the devices and the execution time of the fragment on each device. In the second step, the algorithm selects the fragment with the minimum MCT among all unassigned fragments and assigns the fragment to its best device found in the first step. The *Max-Min* heuristic differs from the *Min-Min* in the fragment selection policy adopted in the second step of the fragment-to-device assignment procedure. Unlike *Min-Min*, which selects the fragment with the minimum MCT, *Max-Min* selects the fragment with the maximum MCT and then assigns it to the best device found in the first step. *Sufferage* is also similar to *Min-Min* but adopts a different fragment selection policy. In the first step of the process, the algorithm computes the second MCT value in addition to the MCT value for each fragment. In the second step, the sufferage value, which is defined as the difference between the MCT and the second MCT values of a fragment, is taken into account. *Sufferage* selects the fragment with the largest sufferage and assigns it to the best device found in the first step.

5 Experimental Environment

In order to evaluate the performance of the KD-Tree and OpenCL implementations presented in Section 3, we use a dataset of 58 million points, generated using a combination of LIDAR and echo sounding data captured at the river Rhein in Rheinfelden [7]. The dataset is stored in the HDF5 [23] format, based on the scientific data format F5 [21, 3], to be easily manipulated with the VISH infrastructure. The dataset is composed of 65 fragments that contain between one thousand and 3.5 million points each.

Device	S9000	K20m	Phi7120	2x E5-2690v2	2x Opt.6168	Radeon5870	GTX480	i7-2600K
OpenCL vendor	AMD	NVIDIA	Intel	Intel	AMD	AMD	NVIDIA	Intel
OpenCL version	SDK v2.9	CUDA 6.5	SDK 2014	SDK 2014	SDK v2.9	SDK v2.9	CUDA 6.5	SDK 2014
Operating System	CentOS6.5	CentOS6.5	CentOS6.5	CentOS6.5	CentOS6.5	CentOS5.9	CentOS5.9	Mint16
Host Connection	PCIe 3.0	PCIe 3.0	PCIe 2.0	-	-	PCIe 2.0	PCIe 2.0	-
Type	GPU	GPU	ACL	CPU	CPU	GPU	GPU	CPU
Class	server	server	server	server	server	consumer	consumer	consumer
Compute Units	28	13	240	40	24	20	15	8
Max Workgroup	256	1024	8192	8192	1024	256	1024	8192
Clock (MHz)	900	705	1333	3000	1900	850	1401	3400
Images	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Cache	R/W	R/W	R/W	R/W	R/W	None	R/W	R/W
Cache Line	64	128	64	64	64	-	128	64
Cache Size (KB)	16	208	256	256	64	-	240	256
Global Mem (MB)	3072	4799	11634	129006	64421	1024	1536	7965
Constant (KB)	64	64	128	128	64	64	64	128
Local Type	Scratch	Scratch	Global	Global	Global	Scratch	Scratch	Global
Local (KB)	32	48	32	32	32	32	48	32

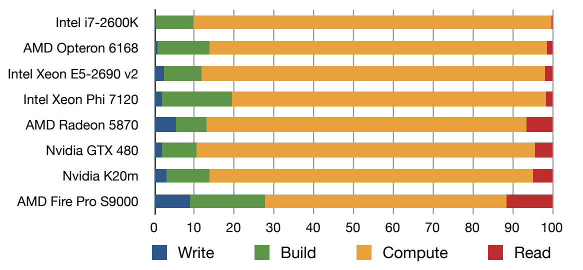
Table 1: Benchmarked OpenCL devices

To represent the broad spectrum of OpenCL-capable hardware we selected eight devices, comprising four GPUs, three CPUs, and one accelerator. Their device characteristics as reported by OpenCL are summarized in Table 1. To exploit the computational capabilities of heterogeneous machines, we evaluated different scheduling heuristics. The experiments were performed on two different heterogeneous target architectures composed of three OpenCL devices: two GPUs and one CPU. The first platform, *mc1*, consists of an Intel i7-2600K CPU and two NVIDIA GTX 480, while the second, *mc2*, holds two Intel Xeon E5-2690 v2 CPUs (reported as a single OpenCL device) and two AMD Fire Pro S9000 GPUs. For the static scheduling heuristics we utilized, as estimation time for each fragment (ETC matrix), the actual time that the fragment will take to be computed on the different devices. Differently, for the computation of the coefficients in the *SimpleHS* heuristic, we used the multi-parameter fitting present in the GNU Scientific Library.

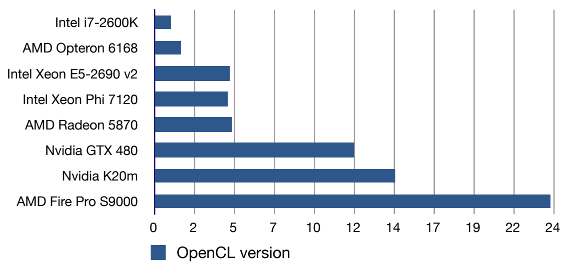
All the benchmarked programs were compiled with GCC version 4.8.1 with the -O3 optimization flag. In each different device, the OpenCL kernels were compiled by the respective vendor compilers at runtime during the program initialization. All the experiments were conducted on the previously described dataset. The measurements were collected for the computational phase of the program, excluding the initialization and finalization phases. We repeated each experiment 10 times and we computed the mean value and the standard deviation of the measured performance. In all the presented experiments, the standard deviation is negligible, thus we do not report it.

6 Performance Analysis

KD-Tree and OpenCL Implementations. To compare the performance of the KD-Tree version and our OpenCL implementation, we executed the tensor computation on the input dataset on the same multi-core CPU (Intel i7-2600K). Both implementations are parallel: the KD-Tree version uses OpenMP to parallelize the loop over all points, while the OpenCL approach is inherently parallel. The building phase of the tree in the KD-Tree implementation is sequential, however, it represents a very small part of the overall run time. The OpenCL version of the program experiences a significant speedup (24×) over the currently implemented VISH KD-Tree version, reducing the execution time from 1 hour to 150 seconds. The performance improvement comes from different reasons. First, grid data structures are more suited for range queries (all the particles around a point in a given radius) while KD-Tree structures are more suited for k-nearest neighbors queries (first N-points close to a given point). Second, vectorization is rather hard in KD-Tree codes where many data-dependent branches are present. In contrast, the uniform grid OpenCL code can be more easily autovectorized by compilers.



(a) Normalized execution time spent in the different parts of the OpenCL tensor computation algorithm



(b) Speedup of the different devices over the Intel i7-2600K

Figure 2: OpenCL Performance Analysis

Third, we applied a few code optimizations that improve the performance of the OpenCL code. However, the optimizations only partially affect the speedup over the KD-Tree version, which remains significant even in their absence (12.9×).

Heterogeneous Devices. Since OpenCL supports heterogeneous devices, we analyzed the performance of our OpenCL code on a set of heterogeneous architectures described in Table 1.

Figure 2a depicts the percentage of execution time spent in the different phases of the OpenCL tensor computation described in Algorithm 1. The blue color represents the transfer of the fragment points to the device (line 5), the green color represents the time spent building the uniform grid structure (line 6-10), the yellow color indicates the time spent in the computation (line 11), while the red color identifies the transfer of the tensors to the host device (line 12). In all the tested hardware the movement of data does not represent an important part of the execution time. *Write* and *Read* functions are always under 5% of the total time. The only exception is the AMD Fire Pro S9000 where the data transfers represent 9.0% and 11.6% of the execution time, respectively. This is mainly due to the small amount of time spent in the tensor computation thanks to the strong computational capabilities of the device.

In Figure 2b we present the performance comparison of the heterogeneous architectures. The speedup of the CPUs respects the characteristics of the hardware. The AMD Opteron, with a higher number of compute units but a lower clock rate, experiences a 1.6× speedup over the Intel i7 while the Xeon, with 40 compute units and a similar clock rate, reaches a 4.5× speedup. All the GPUs show significant improvements in performance compared to the Intel i7. The desktop GPUs AMD Radeon 5870 and NVIDIA GTX 480 reach a speedup of 4.7× and 12.0×, respectively. The server GPUs NVIDIA K20m and AMD Fire Pro S9000, designed for the HPC market, show a speedup of 14.5× and 23.8×, respectively. It is worth underlining that although the NVIDIA K20 offers higher theoretical peak performance, in our test the AMD Fire Pro S9000 is around 1.5 times faster. The only accelerator present in our test is the Intel Xeon Phi. Although its peak performance is comparable with the tested server GPUs, it reaches only a speedup of 4.4× compared to the Intel i7. The difference in performance between the GPUs and the Xeon Phi is difficult to investigate as it derives from the differences in the architecture and from the different maturity of the OpenCL toolchains.

In conclusion the results show that the problem is well-suited for massively parallel GPU architectures, reducing the processing time of the complete input dataset to 6.3 seconds in case of the AMD Fire Pro S9000.

	Device	Scheduling Heuristics				
Consumer Platform		Sufferage	Min-Min	Max-Min	SimpleH	**SimpleHS**
	mc1-CPU1	5976.34 [15]	0.00 [0]	5980.55 [7]	6124.42 [21]	4807.08 [19]
	mc1-GPU1	5971.16 [25]	5993.26 [32]	5988.02 [29]	5962.76 [23]	6014.41 [24]
	mc1-GPU2	5974.84 [25]	6502.75 [33]	5988.23 [29]	5984.76 [21]	6049.29 [22]
	Ex. time (ms)	5976.34	6502.75	5988.23	6124.42	6049.29
	Norm. to Suff.	100.00%	91.90%	99.80%	97,58%	**98.79%**
Server Platform		Sufferage	Min-Min	Max-Min	SimpleH	**SimpleHS**
	mc2-CPU1	2708.42 [21]	1341.58 [7]	2711.67 [12]	2758.30 [28]	2758.30 [28]
	mc2-GPU1	2706.41 [22]	2986.75 [29]	2710.42 [26]	2797.53 [20]	2797.53 [20]
	mc2-GPU2	2709.77 [22]	2766.57 [29]	2712.34 [27]	2838.69 [17]	2838.69 [17]
	Ex. time (ms)	2709.77	2986.75	2712.34	2838.69	2838.69
	Norm. to Suff.	100.00%	90.73%	99.91%	95.45%	**95.45%**

Table 2: Performance of the different scheduling heuristics in two heterogeneous systems

Fragments Scheduling. As previously described in Section 4, we conducted a set of experiments with scheduling heuristics in two heterogeneous machines. The objective of our scheduler is to find a fragment-to-device assignment that minimizes the total execution time (makespan). Table 2 shows, for each device in the two systems, the time spent to execute the number of assigned fragments (in square brackets) for the particular scheduling policy. The table also presents for each heuristic the makespan and the normalized result to the *Sufferage* heuristic. In both systems *Sufferage* reaches an almost perfect load balancing between the three available devices, fully utilizing the entire machines.

In both systems the static scheduling heuristics obtain similar results, with *Max-Min* that reaches almost the same performance of *Sufferage*, while *Min-Min* shows 91.90% and 90.73% of the performance, respectively. These results are justified by the structure of the dataset. Usually, datasets collected with LIDAR technology contain few fragments with a big number of points and many small fragments with fewer points. Due to the fragment selection policy, *Sufferage* and *Max-Min* perform the assignment of the large fragments in early iterations resulting in a better load balancing between the devices. Differently, *Min-Min* favors the assignment of fragments with lower cost in early iterations, not reaching the same performance in terms of makespan. It is noteworthy that the three static scheduling heuristics use a perfectly correct estimated execution time for the fragments (ETC matrix) that will not be available at scheduling time. The resulting performance of the heuristics is therefore only useful as a comparison parameter for our low-complexity heuristics *SimpleH* and *SimpleHS*, which are only based on information available at scheduling time. *SimpleH*, based on the assumption that the GPUs devices are always faster than the CPU, dynamically assigns the fragments with more points to the GPUs and the one with fewer points to the CPU. This simple mechanism facilitates the devices load balancing by avoiding assigning large fragments to slow devices. Although *SimpleH* is capable of reaching good performance, it also shows its weakness with our input dataset. The heuristic does not take into account the number of remaining fragments to assign and, when few are left, continues to distribute them to the CPU. This behavior can lead to load imbalance if the GPUs have to wait for the CPU that received one of the last fragments. This issue is solved with the *SimpleHS* heuristic, previously described in Section 4. *SimpleHS* tries to predict the approximate execution time of a new fragment based on the execution time of the previous ones. Although at the beginning the prediction error is high, it rapidly decreases during the scheduling of fragments. It is noteworthy that the overhead introduced by the prediction model is negligible and does not impact the performance of the scheduler. In our tests, *SimpleHS* is able to correctly predict when to stop the assignment of fragments to the CPU, obtaining a better load balancing between the devices. As depicted in Table 2, *SimpleHS*, scheduling less fragments to the CPU, always achieves better or equal performance compared to *SimpleH*, reaching 98.79% and 95.45% of the *Sufferage* performance in the two systems.

These results validate the success of the proposed heuristics which, using only information available at scheduling time, show comparable performance to more sophisticated methods which require an accurate estimation of the expected execution times.

7 Related Work

The study of the interaction of millions of points, present in modern datasets, requires scalable systems capable of supporting the large computational demands. In order to actually improve the scalability of such systems many spatial partitioning methods were proposed and investigated [2, 25, 26]. Some of these approaches are suitable for simulations which frequently have high density in one or several spatial locations and some perform best with uniformly distributed points. In recent years, among such methods, uniform grid data structures have received great attention from the research community. Erra et al.[8], leveraging the GPU processing power, implemented an efficient framework which permits to simulate the collective motion of high-density individual groups. Aaby et al. [1] presented the parallelization of agent-based model simulations (ABMS) with millions of agents on multiple GPUs and multi-core processors. Vigueras et al. [24] proposed different parallelization strategies for the collision check procedure that takes place in agent-based simulations. Green [11] described how to implement a simple particle system in CUDA using a uniform grid data structure. Husselmann et al. [12] presented single- and multiple-GPU solutions for grid-boxing in multi-spatial-agent simulations. While the uniform grid approach of these works is similar to ours, they are restricted by the language of choice to some specific hardware. Differently, using OpenCL, our work is not limited to a single platform and can be executed in multiple heterogeneous devices. This advantage allows us to compare different platforms and fully exploit the computational performance of heterogeneous systems as shown in other recent work [15].

8 Conclusions

This paper proposes an OpenCL implementation of the second order tensor field computation of massive point datasets. Compared with an existing KD-Tree parallel approach which uses OpenMP, our approach is 24× faster on an Intel i7-2600K. Since OpenCL supports heterogeneous devices, we investigated the performance of our implementation on a set of heterogeneous architectures, showing a remarkable reduction of the execution time. Furthermore, aware of the heterogeneous computing trend, we investigated different scheduling policies on two heterogeneous machines. The obtained results validate the success of the proposed *SimpleHS* heuristic, which shows comparable performance to more complex static heuristics, only using information available at scheduling time. In the future, we plan to extend our work to distributed environment using the libWater library [10, 9].

9 Acknowledgment

This research has been supported by the FWF Austrian Science Fund as part of project I 1523 "Energy-Aware Autotuning for Scientific Applications" and by the FWF Doctoral School CIM Computational Interdisciplinary Modelling under contract W01227.

References

[1] Brandon G. Aaby, Kalyan S. Perumalla, and Sudip K. Seal. Efficient simulation of agent-based models on multi-gpu and multi-core clusters. In *SIMUTools*, pages 29:1–29:10, 2010.

[2] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324, 1986.

[3] W. Benger. *Visualization of General Relativistic Tensor Fields via a Fiber Bundle Data Model.* PhD thesis, FU Berlin, 2004.

[4] W. Benger, M. Haider, J. Stoeckl, B. Cosenza, M. Ritter, D. Steinhauser, and H. Hoeller. *Visualization Methods for Numerical Astrophysics.* 2012.

[5] W. Benger, G. Ritter, and R. Heinzl. The concepts of vish. In 4^{th} *High-End Visualization Workshop*, pages 26–39, 2007.

[6] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. A. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *JPDC*, 61(6):810–837, 2001.

[7] W. Dobler, R. Baran, F. Steinbacher, M. Ritter, M. Niederwieser, W. Benger, and M. Aufleger. Die Zukunft der Gewässervermessung: Die Verknüpfung moderner und klassischer Ansätze: Airborne Hydromapping und Fächerecholotvermessung entlang der Rheins bei Rheinfelden. *Wasser-Wirtschaft*, 9:18–25, 2013.

[8] U. Erra, B. Frola, V. Scarano, and I. Couzin. An efficient gpu implementation for large scale individual-based simulation of collective behavior. In *HIBI*, pages 51–58, 2009.

[9] I. Grasso, S. Pellegrini, B. Cosenza, and T. Fahringer. libwater: Heterogeneous distributed computing made easy. ICS, 2013.

[10] I. Grasso, S. Pellegrini, B. Cosenza, and T. Fahringer. A uniform approach for programming distributed heterogeneous computing systems. *JPDC*, 74(12):3228–3239, 2014.

[11] S. Green. Particle simulation using cuda. *NVIDIA Whitepaper*, 2010.

[12] A. V. Husselmann and K. A. Hawick. Spatial data structures, sorting and gpu parallelism for situated-agent simulation and visualisation. In *MSV*, pages 14–20, 2012.

[13] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *J. ACM*, 24(2):280–289, 1977.

[14] Khronos OpenCL Working Group. The OpenCL 2.0 specification, 2013.

[15] K. Kofler, I. Grasso, B. Cosenza, and T. Fahringer. An automatic input-sensitive approach for heterogeneous task partitioning. In *ICS*, 2013.

[16] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk. The digital michelangelo project: 3d scanning of large statues. SIGGRAPH, pages 131–144, 2000.

[17] C. Liu and S. Baskiyar. A general distributed scalable grid scheduler for independent tasks. *JPDC*, 69(3):307–314, 2009.

[18] P. Luo, K. Lü, and Z. Shi. A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems. *JPDC*, 67(6):695–714, 2007.

[19] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *JPDC*, 59(2):107–131, 1999.

[20] P. Musialski, P. Wonka, D. G. Aliaga, M. Wimmer, L. van Gool, and W. Purgathofer. A survey of urban reconstruction. In *EUROGRAPHICS 2012 State of the Art Reports*, pages 1–28, 2012.

[21] M. Ritter. Introduction to HDF5 and F5. Technical Report CCT-TR-2009-13, Center for Computation and Technology, Louisiana State University, 2009.

[22] M. Ritter and W. Benger. Reconstructing Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field. In *Journal of WSCG*, pages 223–230, 2012.

[23] The HDF Group. HDF5 - Homepage. http://www.hdfgroup.org/HDF5, 2014.

[24] G. Vigueras, J. M. Orduña, M. Lozano, J. M. Cecilia, and J. M. García. Accelerating collision detection for large-scale crowd simulation on multi-core and many-core architectures. *Int. J. High Perform. Comput. Appl.*, 2014.

[25] M. S. Warren and J. K. Salmon. Astrophysical n-body simulations using hierarchical tree data structures. In *ACM/IEEE Conference on Supercomputing*, pages 570–576, 1992.

[26] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *ACM/IEEE Conference on Supercomputing*, pages 12–21, 1993.

SIGRAD 2014
M. Obaid, D. Sjölie, E. Sintorn and M. Fjeld (Editors)

Using Curvilinear Grids to Redistribute Cluster Cells for Large Point Clouds

D. Schiffner[1], M. Ritter [2,3], D. Steinhauser [3] and W. Benger [2]

[1]Professur für Graphische Datenverarbeitung, Goethe Universität Frankfurt, Germany
[2]AirborneHydroMapping GmbH, Technikerstr 21a, Innsbruck, Austria
[3]Universität Innsbruck, Technikerstr 13/25, Innsbruck, Austria

Abstract

Clustering data is a standard tool to reduce large data sets enabling real-time rendering. When applying a grid based clustering, one cell of a chosen grid becomes the representative for a cluster cell. Starting from a uniform grid in a projective coordinate system, we investigate a redistribution of points from and to neighboring cells. By utilizing this redistribution, the grid becomes implicitly curvilinear, adapting to the point cloud's inhomogeneous geometry. Additionally to pure point locations, we enabled data fields to influence the clustering behaviour. The algorithm was implemented as a CPU and a GPU code. The GPU implementation uses GLSL compute shaders for fast evaluation and directly manipulates the data on the graphics hardware, which reduces memory transfers. Data sets stemming from engineering and astrophysical applications were used for benchmarking. Different parameters dependent on the geometric properties were investigated and performance was measured. The method turned out to reach interactivity for medium sized point clouds and still good performance for large point clouds. The grid based approach is fast, while being able to adapt to the point cloud geometry.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Hierarchy and geometry transformations

1. Introduction

Large point cloud data sets are produced by observations and simulations. Today, laser light detection and ranging (LiDAR) applications easily generate several billions of points measurements [PMOK14, OGW*13], similar amounts of particle based data are generated by state-of-the-art astrophysical simulations, i.e. by smooth particle hydrodynamic codes [SWJ*05, Spr05]. Interactive rendering of data becomes important, i.e., when semi-automated algorithms are applied. The classification of LiDAR data, e.g., requires a quality check and 'hand-made' corrections done by users. Here, interactive response and rendering is important for an efficient work-flow. Such large amounts of geometry data do not fit into the graphic hardware's (GPU's) memory as they easily reach hundreds of giga-bytes. Thus, data has to be prepared to support out-of-core rendering, for example in spatially sorted data fragments. But, more geometry data can be loaded onto the GPU's memory than the GPU can display at interactive frame rates.

Here, our approach aims at geometry reduction on the GPU to still achieve interactive frame rates per out-of-core data fragment. We want to avoid any additional data preprocessing, but can enhance the reduction when using pregenerated information. We cluster the incoming vertices to reduce the amount of data being displayed by creating an implicit curvilinear grid originating from an affine transformed uniform grid. The cluster process consists of two steps: a grid cluster operation and a move operation. The cluster operation is simple as it operates on an initial uniform grid. The move operation uses accumulated information from the first step and processes indices only. This allows a fine grained control over individual cells and their number of contained vertices enabling to manipulate the details of the rendered data while not needing to preserve it for further processing. In the context of out-of-core rendering and big data sets, this becomes ever increasing in importance.

Our method aims to be:

fast. Utilization of the GPU and preparing the clusters via GLSL compute shaders to reduce memory transfers and unleash parallelism available in standard workstations.

simple. Data organization is linear and no hierarchical data structure is required. It is directly used for rendering.

flexible. Besides the vertices, additional data fields such as scalar, vector, or tensor fields, may be taken into account for clustering. The method can be combined with a coarse Level of Detail technique on out-of-core data fragments.

versatile. Since the grid is being adjusted to the point clouds's geometry, the method is applicable to different kinds of data. The approach makes no assumption on how the point cloud originated or if it represents a specific kind of geometry. The points may or may not describe lines, surfaces, volumes, or other geometrical distributions, available at many time steps or just at one.

We chose application motivated data sets to do benchmarks. We study different parameters of the approach and the influence of data set properties on the performance and the applicability.

After having provided some background information, we gather related and previous work in section 2. The approach is described in-depth in section 3. Data sets stemming from LiDAR and from astrophysical particle simulations, see section 4, are the basis for benchmarks in section 5. Here, the main results regarding to timing and visualization are presented and discussed. The article is concluded in section 6, and closes with thoughts on future work in section 7.

2. Related and Previous Work

The application of vertex clustering recently has grown in interest due to its fast processing capabilities. Linear methods, such as grid based clustering methods, are especially well suited for large data sets that may contain several million or even billion data points. By reducing the input set, such as presented by DeCoro [DT07] or Willmot [Wil11], the rendering of large data is possible again with a little overhead at the initial clustering phase. In the latter case, individual attributes of an input data set are stored separately to increase detail after reduction.

Promising results have also been achieved by Peng and Cao [PC12], as they are able to provide frame-to-frame coherence when applying their reduction method. Their method is based on an edge collapse algorithm, which was presented by Garland and Heckbert [GH98]. They apply the computation of the quadric error metric in parallel and then decide where to reduce and restructure the output triangles.

The selection of individual level of details is also a crucial part and often includes offline processing methods. In [SK12] we used a parallel approach to dynamically update the current representation while retaining interactivity. This could be done by computing a raw estimate of the object that is being iteratively refined.

An comparison of two clustering algorithms has been presented by [PGK02]. In this case, a hierarchical and an incremental clustering method are applied to reduce point-set-surfaces [ABCO*01], where cells were iteratively refined. Both approaches showed good results regarding reduction quality and run-time performance. Clustering especially in the context of SPH data sets has been utilized by [FAW10] with a perspective grid. They include a hierarchy (octree) in the data organization and apply texture based volume rendering in front to back order of the perspective grid for drawing.

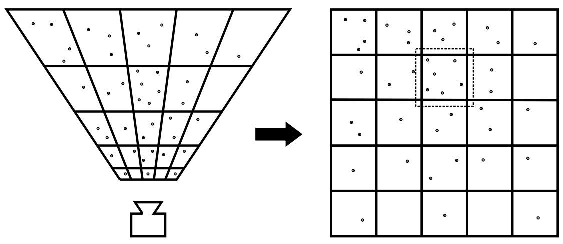


Figure 1: Points are transformed into a local coordinate system of the camera view frustum. Initial cells are defined by a uniform grid. The clustering algorithm operates in this coordinate system. The grid's geometry preserves more detail close to the camera and reduces detail far way.

[PGK02] use a covariance technique in the point neighborhood to compute a reconstructed 'surface normal' and to measure a distance from a cluster point to the original surface. A similar method based on the same dyadic product, called the point distribution tensor, was introduced in our previous work [RB12]. However, the product also includes distance weighting functions and the analysis based on the tensor's Eigenvalues is different. Three scalar fields are derived from the second order tensor called linearity, planarity, and sphericity. These describe the geometric point neighborhood and are normalized between 0 and 1. If points are distributed on a straight line, linearity is high, and if points are distributed on a plane planarity is high, respectively. We precomputed the planarity for some of the data sets used in the benchmarks and include it in the clustering process, such that variations in planarity are preserved and homogeneous planar regions are clustered.

3. Our Approach

The main idea behind our approach is to re-size individual cells based on their internal data. The less points contribute to an individual cell, the better the quality once a reduction is applied. This applies, as long as the representative is being computed using the values taken from a single cell.

The most basic scenario for shrinking a cell is that it contains more points than their neighbors. This can be achieved by reducing the cell extents. Note, that this reshaping does not alter the actual data but is only used internally to derive a new cell. More elaborate results can be achieved, by using

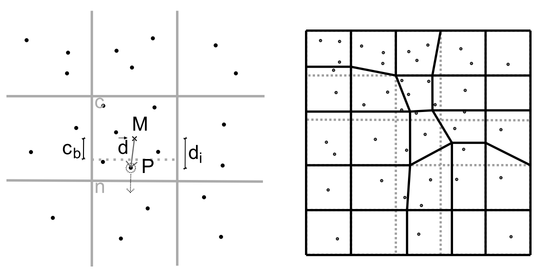


Figure 2: **Left:** Detail of the marked cell in Figure 1 illustrates how a point is "moved" from one cell to its neighbor below. A point P of the cell c is assigned to a different cell if the largest component d_i of the direction vector $\vec{d}$ from the cell center M to P is larger than a certain cell bound c_b which depends on parameters of cell c and the neighboring cell n. **Right:** Curvilinear grid after moving the points. Note that the curvilinear grid is not computed explicitly, but indirectly defined by the points assigned to each cell.

geometric properties, such as curvature or tensor data, that may have been computed in advance.

Also, this approach can be combined with classical warping techniques. In these cases, a non-uniform transformation is applied prior to the clustering, see Figure 1.

3.1. Overview

We apply a 3 step method to create the reduced input set: *cluster*, *move*, and *reduce*. The first step applies a classical clustering, but we also accumulate information needed for the second step. The incoming vertices are mapped to a grid that may can be warped via an affine transformation. The resulting position is converted to an index that is used for further computations.

The second, i.e. the *move*, identifies, whether a data point needs to be placed in a neighboring cell. It uses the accumulated information from step 1 and local information of the current data point to compute new, temporary, cell bounds. If the point is located outside the temporary cell bound it is *moved* to its neighbor. This renders the cell bounds curvilinear, as the actual shape is being altered. Figure 2 illustrates the involved geometrical objects of the method, which formally is described by:

$$\vec{d} = P - M \tag{1}$$

$$\Delta_i = \max_{j=1..3} \{|d_j|\} \tag{2}$$

$$\mathrm{w}(c,n) = \min(lb, \left(\frac{\mathrm{density}(c)}{\mathrm{density}(c)+\mathrm{density}(n)}\right)^{\gamma}) \tag{3}$$

$$c_b = w(c,n) \tag{4}$$

$$\Delta_i > c_b \begin{cases} true, & move\ P \text{ to } n \\ false, & \text{skip} \end{cases}, \tag{5}$$

with M the center point of the current cell c, P a point in c, i the index of the maximal component of vector $\vec{d}$, n the neighbor cell, lb a lower bound of the cell size of c assuring its minimal size, c_b the cell boundary in direction of the component i, and γ a non linear scaling factor.

For a point P its direction vector from the cell's center is computed. Then, the maximal absolute component of this vector is chosen and compared to the according component of

Both, step 1 and step 2 scale with the size of the input data $\mathcal{O}(N)$. Each cell, identified by the index, is processed and the according data is accumulated to compute a representative data point.

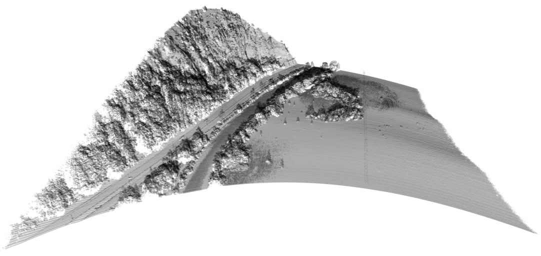
The last step simply reduces the input point set by emitting the previously averaged cell position. More sophisticated methods such as median or a quadratic error minimization could be utilized to derive the representative. As the single cells are iterated in this case, the time complexity is bound linearly with the number of cells $\mathcal{O}(C)$. The final output is a reduced point set, that can be visualized. To allow further displaying of additional data, the accumulated data of the *cluster* or *move* steps can be emitted as well.

3.2. Computation Details

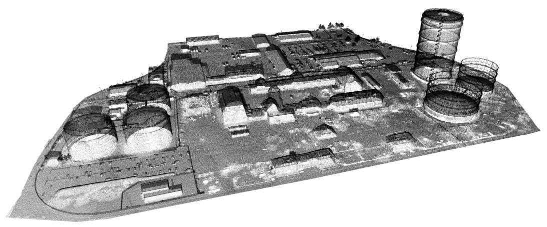
The processing flow of our method can be described as follows. On each call, the input position from the raw data set is being warped by a given projection matrix. This may be identity, if no warping should be applied. The resulting position is in normalized device coordinates and is matched to the underlying grid by multiplying it with the grid size. Finally, a grid index is derived by performing a 3D to 1D mapping. From this point, the individual shaders diverge and different operations are performed.

In the *cluster* operation, a scalar value is read from an additional buffer that is aligned with the input positions. This value represents the individual weight of an input point and is atomically added to an internal counter. We also store a maximum value to allow visualizations regarding the overall weight.

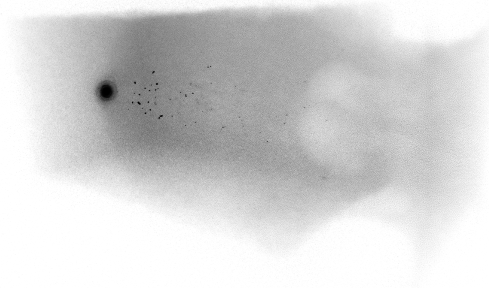
In case of the *move* operation, each cell is compared to their immediate neighbors. The previously summarized counter is used to extract the total weight of a cell. Weights that are less than the derived neighbor's weight are not taken into account and the processing is aborted. Otherwise, the new cell bounds for the current active cell are computed by applying formulas (4) and (3). We use the internal counter from the first step as the density(c) and set used a default parameter set for all tests ($lb = 0.1, \gamma = 1.0$). If the current position exceeds the new cell bound, the current point is emitted to that neighboring cell used in the computations.



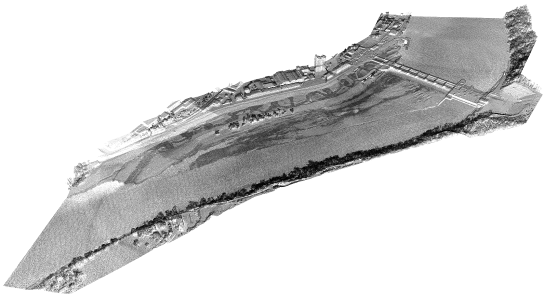
(a) SmallRiver



(b) GasTank



(c) FormingStars



(d) RiverDam

Figure 3: Four point cloud data sets were used for testing. Different sizes and different geometrical distributions are benchmarked. The points in the LiDAR data sets are mainly distributed on surfaces with small volumetric regions in vegetation and water. The point density varies relatively little over the whole data set. The SPH simulation of forming stars is fully volumetric and has small regions of much higher point densities.

Name	Nr. of Points	Scalar Field
SmallRiver	2.075.993	Planarity
GasTanks	11.133.482	Planarity
FormingStars	16.250.000	Type / Density
RiverDam	26.212.555	Planarity

Table 1: Data set sizes used for the benchmarks.

4. Data Sets

We use data sets stemming from LiDAR measurements and an astrophysical particle simulation to test our algorithm, see Figure 3. Table 1 lists the data sets, its sizes, and an available scalar field on the points.

LiDAR: For the LiDAR data three airborne scans with increasing complexity were chosen. The data was captured with a green laser system by Riegl, the VQ820g, specialized for bathymetric scanning. The laser system has an especially high pulse rate of up to 520 kHz and a wide footprint optimized for capturing shallow water regions. The *RiverDam* data set was enriched by additional sonar measurements and, thus, includes ground echos of the deeper (>3m) river sections, besides the shallow water regions of the fish ladder (<3m) [DBS*13]. Such high density bathymetric laser scans are used for hydraulic engineering, planing water related building structures, and environmental engineering. Grids for numerical hydraulic computations can be generated, e.g., for flooding simulations or morphological studies. To generate such grids from a point cloud several processing steps are required. Points are filtered and geo-referenced. Then, they are classified into, at least, the two classes: water and non-water points. Next, the water surface is extracted and non-water points are corrected to eliminate the effect of the water's refraction. Especially, the step of classification needs control and corrections by human users to support automatic algorithms. For all the LiDAR data sets the planarity was pre-computed, an attribute given per point, describing a geometrical property of the surrounding neighborhood [RB12]. It was computed via a the point distribution tensor and describes how closely points are distributed towards a fitting plane in the neighborhood. The radius of the neighborhood was set to 2 meters.

Astrophysics: The *FormingStars* data set represents one time step of a combined N-Body/Hydrodynamic simulation of a galaxy undergoing ram-pressure stripping [SHKS12]. Such simulations are performed in order to understand the evolution of galaxies in dense environments in the universe. In galaxy clusters, the largest gravitationally bound structures in the universe, galaxies move in their mutual gravitational field. Besides the galaxies and dark-matter, such clusters consist of a very hot and thin gas, the intra-cluster medium (ICM). The galaxies are encountering this gas and feel its ram pressure, nonetheless it is very thin. This induces enhanced star formation within the galaxy at first, and leads to the stripping of the inter-stellar medium (ISM), the gas within a galaxy, reservoir for forming new stars. As a consequence, star formation in the galaxy ceases, but stars can be formed from stripped gas in the wake of the galaxy. The mass distribution of different components in GADGET-2 [Spr05] (gas(type 0), dark-matter(1), old stars(2), bulge stars(3), newly formed stars(4)) is discretized and sampled using a Monte Carlo method. Except gas, all other types of matter are then modeled as a collision-less fluid, interacting only via gravity. To solve the resulting N-Body problem, a tree code is used (e.g. [BH86]). The hydrodynamic equations for the gaseous component are solved via SPH (smoothed particle hydrodynamics [Mon92]). Initially, the density estimate of each particle is calculated using a kernel interpolation technique. Consequently, the momentum and thermal energy equation can be integrated in time, the continuity equation is implicitly fulfilled.

The points of the LiDAR data sets reside mostly on surfaces, such as measured ground or building structures. Only a few points captured in vegetation and water regions represent volumes. However, in the star forming simulation the points describe a volume. We want our algorithm to perform well in all cases and want to investigate its behaviour. All data sets still fit into 1GB of GPU memory, but only the smaller ones can be displayed at interactive frame rates.

5. Results

To create test results, we have implemented our approach with OpenGL using compute shader capabilities that are available since version 4.3. We did not use an OpenCL approach, as the data is going to be rendered directly after the processing. This way, we can directly control the outcome of the cluster algorithm when altering the individual parameters.

In the core specification, no floating point atomic operations are specified but can be added by using an extension from nVidia. When using other vendors, one could emulate this feature, by converting the float value to an integer. For further details, the reader may be referred to [CCSG12].

As our approach consists of two steps, we can simply omit the second one (and the additional computations) to allow an evaluation of the overhead generated by our additional *move* operation. Thus, this algorithm applies a basic clustering to the input data set.

A CPU implementation has been realized for sake of completeness. Obviously, the CPU variant will not be able to compete with the GPU implementation.

As stated before, we want to avoid any pre-computations, e.g. computation of tensors or connectivity, on the available data sets. The algorithm is able to perform a reduction without planarity information, but can produce better results with them.

5.1. Timing

Based on our applications, several benchmarks have been conducted. They vary in terms of input size, grid size and used graphics card. In general, a test has been repeated 10 times and the mean time values are given. Timings are reported in milliseconds. Each test was run with varying input parameters, i.e. the object and the grid size. These benchmarks were executed on 3 different PC's, running on Windows 7 and Linux. The results are listed in table 2. The first machine (1) consists of an i5-3450 and a nVidia GeForce GTX 460 with 1GB RAM. The second system (2) uses an i5-670 and a nVidia GeForce 680 GTX. The last configuration (3) contains an Intel Xeon-X5650 and a nVidia Quadro 5000. (1) and (2) operate on a MS-Windows platform while (3) runs a Linux system.

Model	Sys	Our[ms]	Cluster[ms]	CPU[ms]
SmallRiver	1	68.9	49.7	700.0
	2	14.6	10.2	831.0
	3	93.9	52.0	879.0
GasTanks	1	298.1	239.5	3780.0
	2	65.4	34.8	4445.4
	3	480.0	256.5	4758.5
FormingStars	1	648.8	479.9	5751.0
	2	129.8	88.6	6858.3
	3	749.0	434.4	7146.2
RiverDam	1	950.3	671.5	8670.0
	2	206.7	146.7	10292.0
	3	1228.6	719.9	11062.7

Table 2: Benchmark results of our GPU algorithm, a basic cluster approach and a CPU implementation. All shown tests have been performed with a grid size of 75x75x15. This grid was chosen due to the planar point distribution.

The individual timings indicate an overhead due to the additional processing step of our approach. Yet, we only have an increase of roughly 50% despite the additional computations performed in the *move* operation. Note that our compute shader has not been optimized and leaves room for further improvements. A visualization of the presented timings using a different grid size can be seen in Figure 4.

The influence of the grid size is in all computation steps very small. This is due to the fact that the individual steps mostly depend on the data input size, while only the last step scales with the size of the grid. As one can see in Figure 5, the GeForce 680 outperforms the older graphics cards.

5.2. Visual Results

The visualization technique in the OpenGL demo simply draws equally sized non-transparent splats. Color is con-

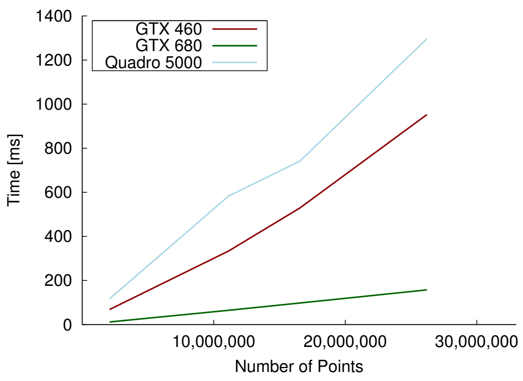


Figure 4: Timing values generated by processing each object repeatedly. The reported values are the mean of all runs. For all objects, a grid size of 200x200x100 has been used.

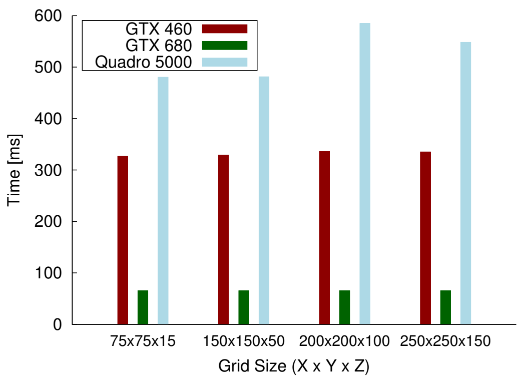


Figure 5: The influence of the grid size on the overall performance of our algorithm. The GeForce 680 GTX outperforms the other graphics cards. The Quadro, despite its larger memory, is not able to compete with the GeForce 460 GTX. We used the GasTank data set for computation.

trolled by a scalar value via a red to green color map. As presented in section 5.1, the impact of the additional *move*-step is acceptable, as the computation times are within interactive response times. The following Figures show several images that were created with both the curvilinear and a classic cluster algorithm with different grid sizes. The color map either illustrates changes based on the relative movement from the cells or the cluster cell density.

In Figure 6 some results generated with our method are shown. We used the prior mentioned data sets to apply a clustering. The colors indicate the density of the represented cell. The more red-ish the color, the more data points have been collected in this cell.

Especially with larger grid sizes, the reduction quality is increasing. In Figure 7, the cell density of each step is used for the color mapping. After application of the *move* operation, the global average is reduced, which results in the red color, as the same maximum is used for the mapping. The lower image visualizes the differences regarding the additional *move* operation. The curvilinear grid matches the underlying source more closely, as can be seen via the cluttered splats at the top right of the image.

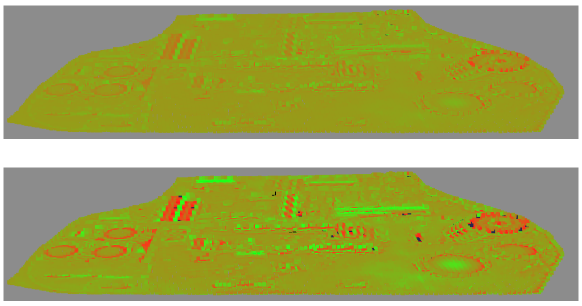


Figure 8: GasTank data set visualized clustered on a perspective grid. **Top:** *move* operation based on cell densities only. **Bottom:** *move* operation including the scalar field planarity which was computed in a pre-processing step. Smaller cells are created in regions of low planarity (e.g. edges) and, thus, preserving more detail. Dense cells are created in regions of homogeneous planar regions, were less detailed information is necessary for a good visual representation. Geometric features of the point cloud are enhanced, when taking the planarity into account.

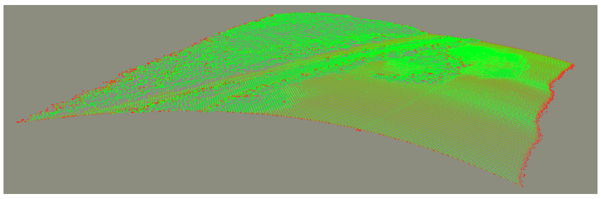
By introducing precomputed information, our algorithm can perform even better. As one can see in Figure 8, regions where edges are present are better fitted as smaller cells are used. This is indicated by the more distinct color values present in the individual cells, e.g. it the lower right of the image.

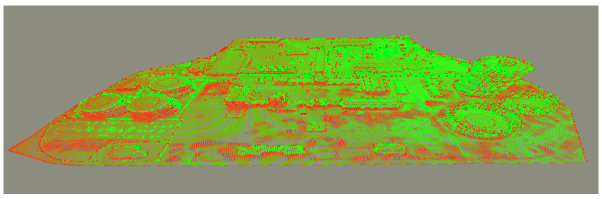
6. Conclusion

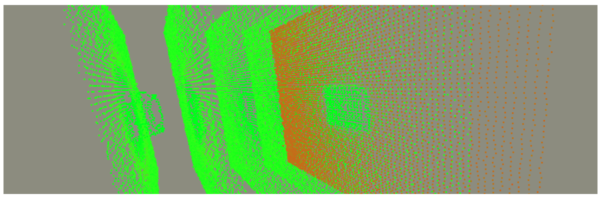
We have presented a new approach to apply a non-linear clustering to arbitrary objects. We are able to use multiple information from the current geometry and are not limited to scalar field properties. The applied reduction is made selectively, due to a restructuring of individual cells. Currently, our data sets are point based and do not incorporate connectivity information. However, an extension to triangles or polygons can easily be achieved, as shown by other researchers ([PC12, Wil11]).

The computation times of the *move* operation has been shown to be interactive for medium sized point clouds and has a good performance with large data sets. Our implementation has not been optimized and leaves room for further enhancements. For example, the calculation of cluster indices is performed in both the *cluster* and the *move* operation, which is not necessary.

We have shown the differences between classical cluster-


(a) SmallRiver


(b) GasTank


(c) FormingStars

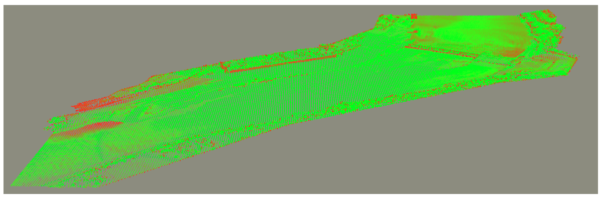

(d) RiverDam

Figure 6: Visual results of the clustering for the different data sets. Color represents the cell density. The number of points per cell is illustrated by a green to red color map going from many (red) to one (green) point. Grid size varies from $150 \times 150 \times 25$ to $300 \times 300 \times 100$ in (a), (b), and (d), which yield good results for reduced overview visualizations. In (c) the grid resolution in z-direction was reduced to 5 slices allowing to see inside the volume. When inspecting the leftmost slice one can see how the representing points are pulled toward the high point density region of the galaxy, thus emphasizing a region of interest. The simple non-transparent splat rendering prevents better insights into the volume.

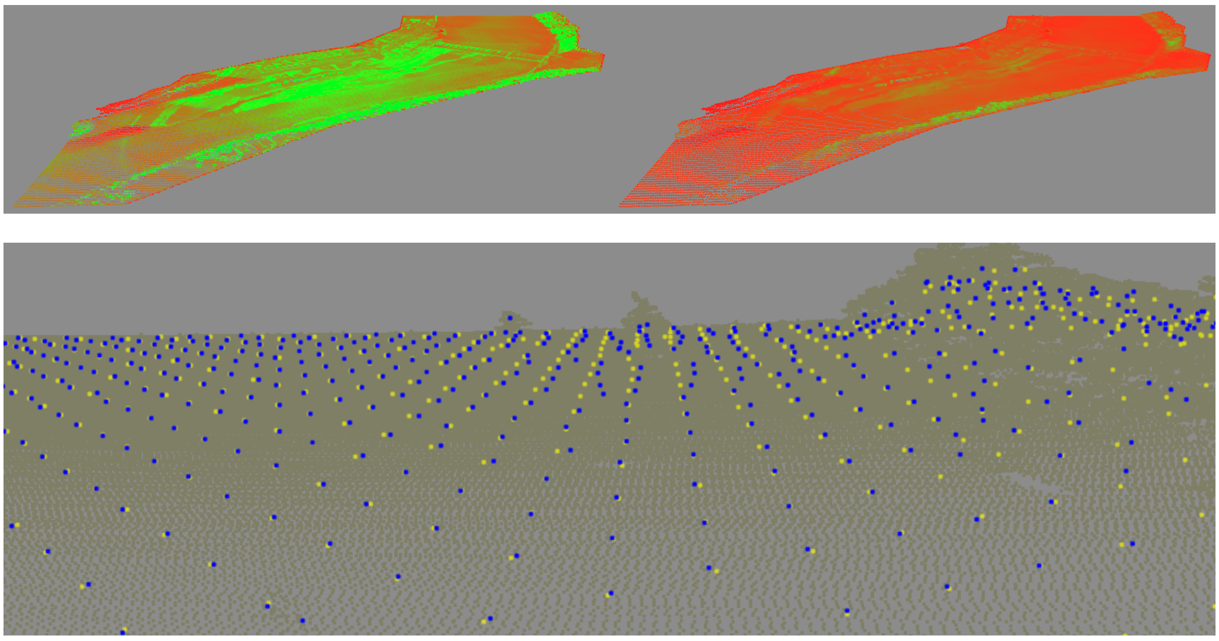


Figure 7: The differences due to the application of the proposed method. In the first picture of the top row, the green-ish regions indicate cells with high density. These are reduced by rescaling the cell sizes, which results in a more even distribution, as seen on the right top. The image below shows a detailed view, where and how the *move* operation modifies the positions of the resulting cells. The yellow cells are created by the clustering while blue ones are the result with the additional *move* operation. Note that the latter produces a splat at the tree in the top of the image.

ing and our curvilinear implementation. Due to the dynamic cells, details in an object are more likely to be preserved. This preservation of features during a rendering increases the quality and topology of the basic object, while still reducing the input data set. Thus, we have made another step towards interactive rendering of large, unprocessed data sets.

7. Future Work

The high performance of the compute shader drives us to further investigate streaming of big data. This includes a fast discard of unnecessary data, as well as selective reloading of individual fragments of a rendered object. Especially, the efficiency of the *move* allows repetitive execution (more iterations) or more complex grid modifications. We intent to use several reconstruction methods to enable the visualization of closed surfaces as well as available geometric properties, such as the point distribution tensor or the planarity. This will allow an identification of interesting regions within the large scale object. Tensor analysis may also be computed on the fly on the GPU.

The visualization can be enhanced by displaying the individual cell sizes. This way, a user could visually control, whether the implicitly generated curvilinear grid matches the expectations. Also, the information within a cluster cell could be visualized showing the influence of the available parameters to the effectively computed grid.

We also want to investigate, whether we could use the fast approximation to create a fingerprinting of these large data sets. To compare large data sets for equality, the accumulated information could be used instead of the raw data. However, it remains to be shown, if the generated data is unique enough for a clear identification.

8. Acknowledgments

This work was supported by the Austrian Ministry of Science BMWF as part of the Konjunkturpaket II of the Focal Point Scientific Computing at the University of Innsbruck and as part of the UniInfrastrukturprogramm of the Research Platform Scientific Computing at the University of Innsbruck and funded by the Austrian Science Fund (FWF) DK+ project Computational Interdisciplinary Modeling, W1227-N16. We like to thank Frank Steinbacher [ahm] to provide the LiDAR data sets.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point Set Surfaces. In *IEEE Visualization* (2001), Ertl T., Joy K. I., Varshney A., (Eds.), IEEE Computer Society.

[ahm] http://ahm.co.at.

[BH86] BARNES J., HUT P.: A hierarchical 0 (N log iV) force-calculation algorithm. *Nature* (1986).

[CCSG12] CYRIL CRASSIN, SIMON GREEN: Octree-Based Sparse Voxelization Using the GPU Hardware Rasterizer. In *OpenGL Insights*, Cozzi P., Riccio C., (Eds.). CRC Press, July 2012, pp. 303–319. `http://www.openglinsights.com/`.

[DBS*13] DOBLER W., BARAN R., STEINBACHER F., RITTER M., NIEDERWIESER M., BENGER W., AUFLEGER M.: Die Zukunft der Gewässervermessung: Die Verknüpfung moderner und klassischer Ansätze: Airborne Hydromapping und Fächerecholotvermessung entlang der Rheins bei Rheinfelden. *Wasser-Wirtschaft 9* (2013), 18–25.

[DT07] DECORO C., TATARCHUK N.: Real-time Mesh Simplification Using the GPU. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 161–166.

[FAW10] FRAEDRICH R., AUER S., WESTERMANN R.: Efficient High-Quality Volume Rendering of SPH Data. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2010) 16*, 6 (November-December 2010), to appear.

[GH98] GARLAND M., HECKBERT P. S.: Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualization* (1998), pp. 263–269.

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual review of astronomy and astrophys. 30* (1992), 543–574.

[OGW*13] OTEPKA J., GHUFFAR S., WALDHAUSER C., HOCHREITER R., PFEIFER N.: Georeferenced Point Clouds: A Survey of Features and Point Cloud Management. *ISPRS International Journal of Geo-Information 2*, 4 (2013), 1038–1065.

[PC12] PENG C., CAO Y.: A GPU-based Approach for Massive Model Rendering with Frame-to-Frame Coherence. *Comp. Graph. Forum 31*, 2pt2 (May 2012), 393–402.

[PGK02] PAULY M., GROSS M., KOBBELT L. P.: Efficient Simplification of Point-sampled Surfaces. In *Proceedings of the Conference on Visualization '02* (Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 163–170.

[PMOK14] PFEIFER N., MANDLBURGER G., OTEPKA J., KAREL W.: OPALS - A framework for Airborne Laser Scanning data analysis. *Computers, Environment and Urban Systems 45*, 0 (2014), 125 – 136.

[RB12] RITTER M., BENGER W.: Reconstructing Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field. *Journal of WSCG 20*, 3 (2012), 223–230.

[SHKS12] STEINHAUSER D., HAIDER M., KAPFERER W., SCHINDLER S.: Galaxies undergoing ram-pressure stripping: the influence of the bulge on morphology and star formation rate. *Astronomy & Astrophysics 544* (July 2012), A54.

[SK12] SCHIFFNER D., KRÖMKER D.: Parallel treecut-manipulation for interactive level of detail selection. In *20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision* (2012), vol. 20.

[Spr05] SPRINGEL V.: The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society 364*, 4 (Dec. 2005), 1105–1134.

[SWJ*05] SPRINGEL V., WHITE S. D. M., JENKINS A., FRENK C. S., YOSHIDA N., GAO L., NAVARRO J., THACKER R., CROTON D., HELLY J., PEACOCK J. A., COLE S., THOMAS P., COUCHMAN H., EVRARD A., COLBERG J., PEARCE F.: Simulating the Joint Evolution of Quasars, Galaxies and their Large-scale Distribution. *Nature* (2005).

[Wil11] WILLMOTT A.: Rapid Simplification of Multi-Attribute Meshes. In *High-Performance Graphics 2011* (August 2011).

Fast Normal Approximation of Point Clouds in Screen Space

Daniel Schiffner
Goethe Universität
Robert-Mayer-Strasse 10
D-60054 Frankfurt
dschiffner@gdv.cs.uni-frankfurt.de

Marcel Ritter
University of Innsbruck &
Airborne Hydromapping OG
Technikerstr. 13a & 21
A-6020, Innsbruck, Austria
marcel.ritter@uibk.ac.at

Werner Benger
Center for Computation and Technology,
Louisiana State University
216 Johnston Hall
LA 70803, Baton Rouge, USA
werner@cct.lsu.edu

ABSTRACT

Displaying large point clouds of mainly planar point distributions yet comes with large restrictions regarding the surface normal and surface reconstruction. Point data needs to be clustered or traversed to extract a local neighborhood which is necessary to retrieve surface information. We propose using the rendering pipeline to circumvent a pre-computation of the neighborhood in world space to perform a fast approximation of the surface in screen space. We present and compare three different methods for surface reconstruction within a post-process. These methods range from simple approximations to the definition of a tensor surface. All these methods are designed to run at interactive frame-rates. We also present a correction method to increase reconstruction quality, while preserving interactive frame-rates. Our results indicate, that the on-the-fly computation of surface normals is not a limiting factor on modern GPUs. As the surface information is generated during the post-process, only the target display size is the limiting factor. The performance is independent of the point cloud's size.

Keywords
Normal Reconstruction, Tensor Information, GPU, Point Clouds

1 INTRODUCTION

Huge data sets are nowadays generated by simulations or by observational methods. Point clouds are e.g. the result of particle based simulation codes or laser scans, such as airborne light detection and ranging (LIDAR) scanning. Surface related information, such as the surface normal, can be used to enhance the visualization of point clouds, e.g. for illumination. Traditional methods for reconstruction surface information require an expensive spatial sort operation. Therefore, these are executed during a pre-process. Our method aims at improving the exploration of LIDAR data sets, before applying more expensive approaches.

In our work, we use the large data throughput of modern GPUs to generate a fast estimation of the surface properties within screen space. Therefore, we apply three possible approaches and compare the individual results. The first approach uses the fragment shader specific `dFdx` and `dFdy` functions. The second method calculates the surface normal by computing the cross product in a local neighborhood, which is available through the pixel neighborhood. The third applies a moving-least-squares approach to acquire tensor information. The resulting co-variance matrix is then used to compute the eigenvalues and eigenvectors.

In the next section, we list similar methods to our approach. Then, we present our methods and solutions to encountered issues. These methods are compared to each other and some examples are presented. Finally, we conclude with a summary of our findings and an outlook regarding future work.

2 RELATED WORK

Generic visualization frameworks, such as openWalnut [Walnut] or the visualization shell (VISH) are utilized for data exploration and processing of a large data sets. More expensive approaches to compute visual enhancements of points distributed on surfaces and lines, and geometrical reconstructions of lines have been done in [Bou212], [Rit12b] or [Rit12a].

The calculation of a surface normal is strongly connected to any surface reconstruction method. Especially for point based representations, methods using co-variance techniques [Ber94][Bjö05] are well suited, because no exact neighborhood is available and some noise is to be expected. Alexa defined the so-called point-set surfaces and presented some projection specific calculations [Ale04]. The co-variance matrix allows to assess the quality of the point cloud data set using direct tensor field visualization methods, such as displaying tensor splats [Ben04]. To compute the eigenvalues and eigenvectors from a given co-variance matrix, the analytical approach presented by Hasan [Has01] or one of the methods presented by Kopp [Kop06] can be applied.

Yet, these methods rely on the identification of an accurate neighborhood. To acquire this information, the input data set needs to be sorted. Neighbors are either found by a brute force approach – which is not suitable at all –, by a tree search or by a Morton ordering [Con10]. A tree as well as a Morton order are highly suited for parallelization.

Instead of creating a kd-tree or a Morton order in world space, a neighborhood can also be computed in screen space. Thus, the computation is only performed on the currently visible part of the data set. This is commonly done by splatting the data points and extracting the properties from the frame buffer. Similar to the approach presented by [Sch11] or [Yan06], we use only screen space information for th selection of the neighborhood. The splats are projected using either a fixed or adapted point size, as proposed by Rusinkiewicz [Rus00]. Once the surface information is available, also high quality splatting techniques [Bot05] could be applied.

3 APPROACH

We use the information available in screen space to reconstruct a surface and its corresponding normals. We designed an approach consisting of three individual steps, as illustrated in figure 1.

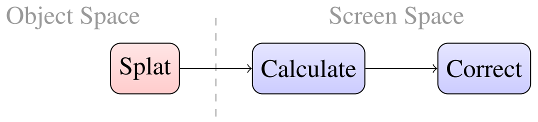


Figure 1: The outline of our screen space normal reconstruction. The first pass consists of splatting the depth values which are used in the consecutive passes. The second pass approximates the surface normal, while the optional third pass smooths the resulting values.

The first pass is a simple splatting of the input data and provides the depth information required by reconstruction. Each pixel is hereby surrounded by neighbor candidates. The second pass uses these depth values and computes surface properties. The candidates are inspected and rejected if the distance is too large, i.e. their interpolation weight is too small. The last pass is optional and allows a further enhancement of the quality of the reconstructed properties.

Splatting the Point Cloud

We draw the point cloud, which will be reconstructed, using either a fixed or an approximate point size. Our approach only requires a depth buffer for computation of the surface information. As the depth-buffer is generated, in general, by all rendering approaches, this method can be applied to all scenarios.

To increase the accuracy, we encourage using a multi-sample depth-buffer. This allows the retrieval of multiple depth values per individual sample. Using a sampling count of 8 means that we are able to capture – at most – 8 individual splat depth values at once. It is, of course, possible that the unprojected world space coordinates are identical or invalid, i.e. the depth value was not set. Still it increases the stability of the following normal calculations. Multi-sampling is only applied within the first post process.

Normal Definition

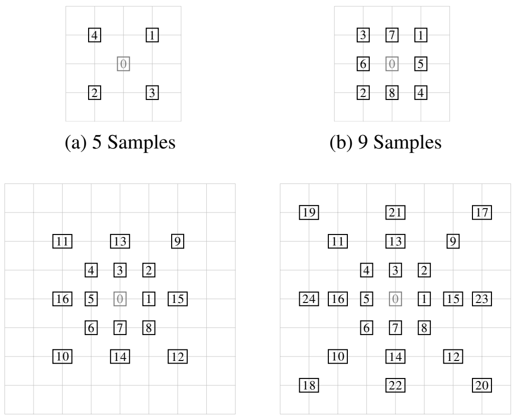
We calculate the wold space coordinates of the current pixel by un-projecting it based on the multi-sampled depth-buffer. The reconstruction of the surface normal can then be performed in three ways. The first method uses the local derivatives directly available in the fragment shader. The second and third method approximate the surface using a generic neighborhood description.

This neighborhood is defined by fixed sampling patterns. The most simple version takes 5 samples within a 3x3 neighborhood, while the most complex version selects 25 samples in a 7x7 neighborhood, see figure 4. The samples are focused on the diagonals, which increase the overall area captured during reconstruction. Note, that we use ascending indices for the opposite sample positions. This enables a simple definition of diagonals within a shader.

In our test, we did not observe any differences between the 5 and 9 sample schemes. This indicates, that the reduced representation is already able to capture the surface properties. The extended schemes, i.e. 17 and 25 samples, further increase stability of the results and are more comparable to off-line methods.

We orient all normals by inverting those, where the z-component is negative. All selected splat samples are visible and, thus, require a normal which is facing towards the camera.

To assure correct identification of possible neighbor candidates, a maximal distance is introduced. Neighboring pixels may not be true neighbors within world space due to the projection. Therefore, we reject every sample that is not within this configurable distance. This is comparable with the maximal distance in the MLS [Ale04] or tensor computations [Rit12a].



(a) 5 Samples (b) 9 Samples

(c) 17 Samples (d) 25 Samples

Figure 2: The used sampling schemes for defining the local neighborhood of a fragment. The center point 0 is optional.

Local Derivatives

Shaders support the calculation of local derivatives within the fragment shader since GLSL version 1.10. For reconstruction of the surface normal, the functions `dFdx` and `dFdy` are used. These internally extract neighbor positions from concurrent thread blocks and are only available in the fragment stage. This means that the surface is completely splatted and the individual samples may have overlapped. With c, the current position in clip-coordinates, the surface normal $\vec{n}$ is computed:

$$\vec{n}(c) = dFdy(c) \times dFdx(c)$$

This method is very sensitive to noise or irregularities in the depth buffer and in many cases produces normals not representing a good reconstructed surface. However, if the surface is continuous and the splat size is carefully chosen, this method will suffice.

Plane Approximation

Similar to the computation of mesh surface properties, we approximate face normals within this approach. The normals are accumulated and the resulting vector is normalized. Finally, we impose an orientation and align the vector.

To obtain the needed vectors, we use one of the proposed sampling schemes. Each direction vector is built up either by diagonal or counter-clock-wise (ccw) samples. The diagonals generate smoother results and do not require the center point at sample 0. This is similar to the anti-alias algorithms used in the rendering pipeline. The ccw approach accounts more for local changes and takes the center point into account. In the diagonal case, we obtain the surface normal by using the following formula:

$$\vec{n}(c) = \frac{1}{N} \sum_{i=0}^{\lfloor \frac{N}{4} \rfloor} \vec{d}_{4i} \times \vec{d}_{4i+2}$$

With $\vec{d_i} = s_i - s_{i+1}$. We optimize the sampling schemes for a diagonal pattern, since we intend to create smooth surface normals with minimal noise.

Tensor Information

Using tensor information instead of flat patches leads to a smoother reconstruction. To derive this information, the computation of eigenvalues and eigenvectors is mandatory. We compute the point distribution tensor by deriving the co-variance matrix for the current position c, as presented by [Rit12a] and similar to [Bjö05]:

$$CM(c) = \frac{1}{N} \sum_{k=1}^{N} w_{ik}(d_{ik} \otimes d_{ik}^T)$$

where $d_{ik} = c - S_k$, d_{ik}^T is the transpose, N is the number of samples around center point c, S_k the sample and w_{ik} is a weighting function. Here, we apply a weighting of $w_{ik} = \frac{1}{\|d_{ik}\|^2}$.

The tensor product $\otimes$ is built by the direction vectors pointing from the current fragment's world coordinate to its points in the neighborhood. The weighted sum of these vectors result in the final point distribution tensor.

We compute the eigenvalues with the "Cordano" method presented by [Kop06]. This approach results in more stable vectors than the method proposed by Hasan et al. [Has01]. Similar findings were made by the developers of openWalnut [Walnut]. The eigenvector related to the minor eigenvalue hereby represents the surface normal. The vector is easily oriented, since the calculation is performed in clip-coordinates and the normal vectors have to face the camera.

Smoothing Normals

In a second, optional, screen space pass we correct the computed normals. We extract and scale adjacent normals within a local neighborhood, where the center normal is being favored. The surface normal is yield by accumulation of the weighted vectors.

Different weights and neighborhood sizes can increase the accuracy of the result. However, this does not apply to all situations. Especially, when using the plane approximation method, quality decreases, when the normals contain lots of noise.

4 RESULTS

We implemented a prototype, which has been tested on a i5 670 system with 8 GB RAM and a GeForce 680 running on Windows 7.

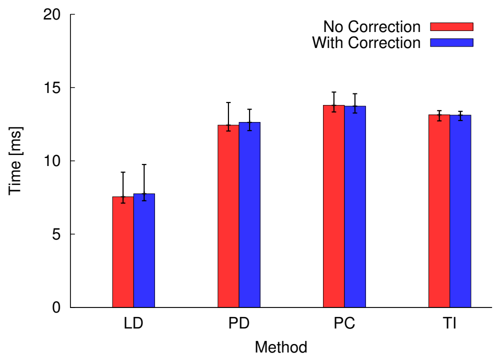


Figure 3: Timing results achieved using a screen size of 1024x768 with 8 multi-samples and the 9 samples scheme. LD denotes the local derivatives, PD the plane approximation using diagonals, PC the plane approximation using counter-clock-wise pattern, and TI the tensor information.

Timings

On all systems, we observed interactive frame rates with all methods. The fasted method is the local derivatives (LD) approximation, while the tensor information (TI) is the most expensive variant. The plane approximation with diagonals (PD) is slightly faster than the tensor variant. The ccw plane approximation (PC) is worse in terms of performance compared to the PD, due to the definition of the sampling scheme.

In figure 3, the average processing times are shown, including the generation of the depth values. We used a fixed multi-sampling count of 8 in all presented timing results. Thus, the real number of samples taken per pixel needs to be multiplied by 8. For better readability, we continue to use the introduced sampling count.

The splatting of the point cloud requires a significant amount of time. In our tests, it varied in the range of 30% to 50% mainly depend on the used screen and splat sizes.

The used sampling scheme size has a large influence on the performance and quality of the reconstruction, as seen in figure 4. The performance scales linearly with the number of used samples. However, the quality of the reconstruction is not necessarily improved when using a very high sampling count. This is due to the fact that the surface is smoothed and local information is suppressed.

We also measured the contribution of the individual steps performed by our approach. Interestingly, the splatting itself consumes a large amount of the overall processing time, while the correction requires only very little processing time. The larger the number of used samples, the higher the reconstruction times. Table 1 lists the detailed timings of the involved steps: "Splat" represents the splatting of the depth values, "Normal" the reconstruction and "Correction" the final smoothing.

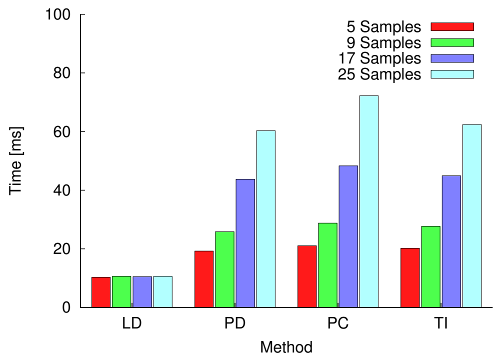


Figure 4: Influence of changing sampling scheme size for the reconstruction methods. Results taken with a screen resolution of 1600x1200. All methods use a 8 times multi-sampled depth-buffer.

9 Samples Scheme / 8 Multi-samples			
Operation	Min [ms]	Max [ms]	Avg [ms]
Splat	8.963	9.030	9.000
Normal	17.521	18.435	17.968
Correction	0.468	0.717	0.493
17 Samples Scheme / 8 Multi-samples			
Operation	Min [ms]	Max [ms]	Avg [ms]
Splat	8.801	10.654	8.980
Normal	34.278	35.711	34.890
Correction	0.466	5.740	0.702

Table 1: Distribution of the processing times among the individual operations of the proposed method. Results taken with a screen resolution of 1600x1200 using the tensor method.

Visual Results

All methods are able to reconstruct both noisy and smooth surfaces. We use several splatted object point clouds as test cases. All point clouds consist of at least 250k points to assure a high sampling density.

The results of the described reconstruction methods are shown in figure 5. These indicate that the TI method provides a stable and accurate reconstruction. The PD approach provides excellent results in smooth data sets. The LD approach always generates large noise. Despite not being suitable for a high quality surface approximation, it is the fasted approach.

To simulate noisy data, we alter the vertex positions within the splat shader. A light source is positioned below the object. The illuminated scene is shown in figure 6. The TI method generates the smoothest result, while the PD method yields more normals that differ widely from the original ones. The LD method provides the worst reconstruction. All methods generate

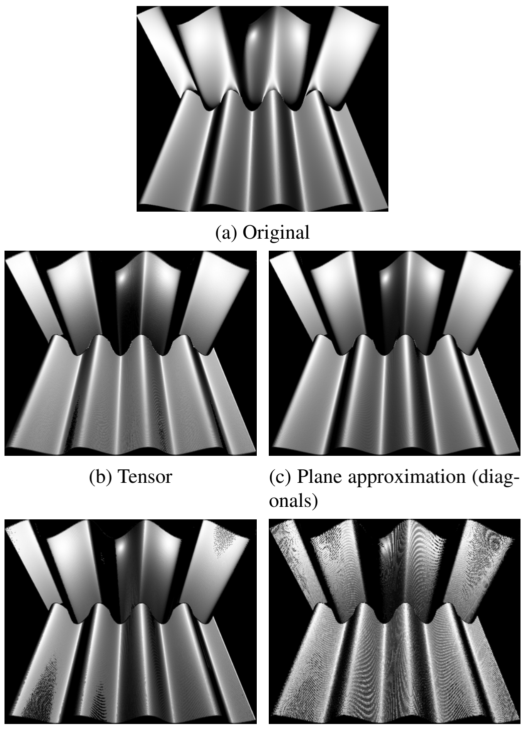

(a) Original
(b) Tensor
(c) Plane approximation (diagonals)
(d) Plane approximation (ccw)
(e) Local derivatives

Figure 5: Reconstruction of the surface normal used for illumination. (a) shows the original object with precomputed normals. (b) to (e) depict the proposed reconstruction methods.

more invalid normals in the low sampled region on the top.

Figure 7 illustrates the influence of the optional correction pass. The corrected normals are smoother and the number of correctly oriented surface normals is higher. The vectors are visualized via colors showing the x-, y-, and z-coordinates as red, green, and blue values.

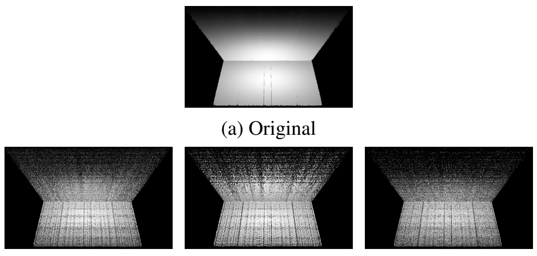


Figure 6: Reconstructed normals used for illumination in a test scenario with two planes. Noise is added to the input data. Even normals at the edge are well reconstructed, but tend to be smoothed.

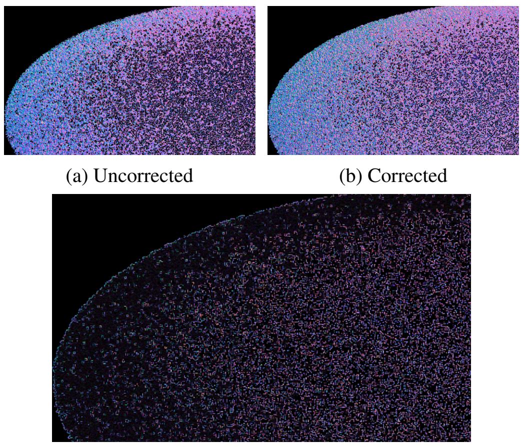


Figure 7: The influence of the correction pass applied to an ellipsoidal surface. The surface xyz-normal is illustrated as a rgb-color. The corrected version (b) contains more valid normals. The difference is visualized in (c).

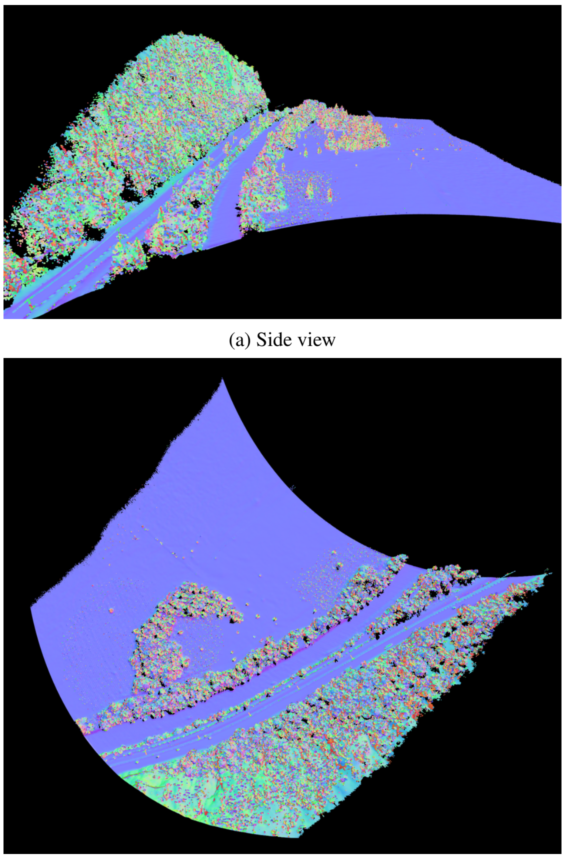
Since the correction pass is very fast and increases the stability of the reconstruction, we always enable this pass in the following tests.

Application to a LIDAR Data Set

A point could stemming from an airborne laser-scan is used for further investigation of the technique and validation of the technique by a real-world application. We chose a small section of a bathymetric scan of the river Loisach in Bavaria (Germany), acquired with the hydrographic laser scanner Riegl VQ-820G [Ste10]. The scan contains different kinds of structures: fields, trees, lower vegetation, a river, a street with cars, power cables and a steep slope partially covered with vegetation. Figure 8 shows a side and a top view of the scan.

The two million points are colored by the minor eigenvector of the point distribution tensor computed in world-space.

The point distribution tensor was computed by using a neighborhood radius of 0.5, 1.0 and 2.0 meters. Two different weighting functions have been tested: constant weight and $\frac{1}{\|d_{ik}\|^2}$ weight. Using a kd-tree for finding neighbors and 6 OpenMP parallel threads on an Intel Xeon X560@2.67GHz the according computation times are 41, 85, and 218 seconds for the three radii. This computation of the tensor is a demanding computational tasks. However, it has been shown, that the tensor can be used for feature extraction, object recognition, and to improve the segmentation of point clouds [Rit12a][Rit12b][Bjö05]. When just looking at the minor eigenvector via color, the fields, the river, the street, the slope and the vegetation can be well distinguished from each other, visually.


(a) Side view

(b) Top view

Figure 8: LIDAR laser-scan of a section of the Bavarian river Loisach in Germany. Laser echoes are illustrated as colored points. Color shows the minor eigenvector of the point distribution tensor. Vegetation can be visually distinguished from the ground and the river.

Next, we compare this expensive, fine grain computation in world space with our screen space technique. The results indicate that the approach is able to reconstruct the normals with rather high quality. The normals widely match with the normals calculated in world space, as shown in figure 9. However, differences in the forest areas of the scan are visible.

Also, where the sampling density near the camera position is not high enough to ensure high quality reconstruction in this region.

To compare the results of the different methods, we recorded a series of images from the Loisach data set. The TI method produces the most reliable results, while requiring a high sampling count. The PD method is able to create very smooth normals regardless of small surface changes, e.g. the missing power line in the upper region 10. The PC method includes it, but is more unstable. The LD method is the most efficient approach while yielding the worst quality in comparison to the other methods.

The correction pass increases the quality and the stability of the results by reducing the number of invalid surface normals. Figure 10c illustrates the enabled correction pass and figure 10d .

5 CONCLUSION

Our results show that a fast approximation of the surface normal can be achieved in real-time. Here, the surface is solely reconstructed from the depth-buffer and projection parameters. With our approach a preprocessing of surface information may be delayed until a region of interest has been selected. The results indicate that especially the tensor-based approach to determine the surface normal of a point cloud is a well-working method.

In comparison to the off-line world space method, we are able to create similar results at interactive frame rates. The loss of quality is negligible and is only visible in under-sampled regions. However, this method can only provide an approximation of the real point-cloud's surface information. The tests show that an increase of the neighborhood size decreases the performance linearly. A good quality is already achieved with small neighborhood sizes. The focus on the diagonals in the sampling schemes reduce the number of required samples.

6 FUTURE WORK

We plan to combine this technique with level of detail rendering to provide good visual representations of large airborne LIDAR scans. The surface normals provide important information to control such a level of detail algorithm.

The splatting technique could be enhanced by utilizing more information represented in the point distribution tensor. Extracting some features of the tensor will improve the readability of point clouds without expensive pre-computations.

Additionally, we plan to enhance the reconstruction method by providing more weighting functions besides the $\frac{1}{\|d_{ik}\|^2}$ weight for the computation of the co-variance matrix.

To avoid expensive re-calculations, we plan to employ a caching strategy. A re-computation of the surface normals would only be required when camera location or point coordinates are changed, further increasing the overall performance of the approach.

7 ACKNOWLEDGEMENTS

Special thanks to Frank Steinbacher for providing the LIDAR data set of the river Loisach. This work was supported by the Austrian Science Foundation FWF DK+ project Computational Interdisciplinary Modeling

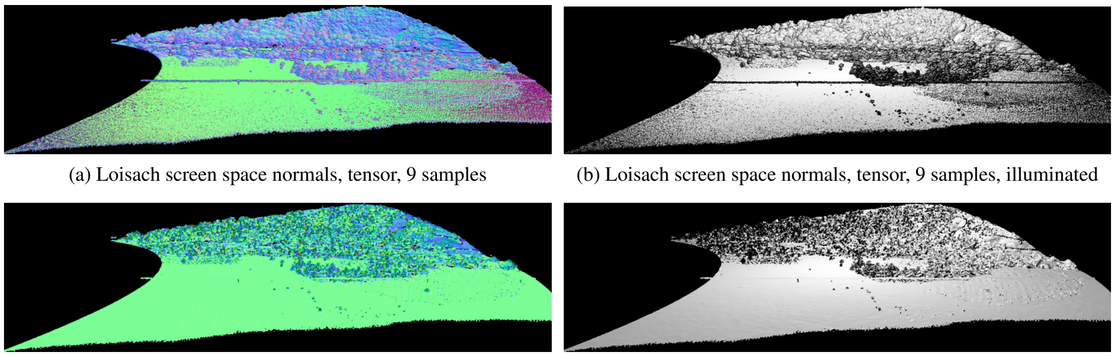

(a) Loisach screen space normals, tensor, 9 samples
(b) Loisach screen space normals, tensor, 9 samples, illuminated
(c) Loisach world space normals, 1m, squared
(d) Loisach world space normals, 1m, squared, illuminated

Figure 9: The reconstruction of the minor eigenvector using the fast screen space approach.

(W1227), and grant P19300. This research employed resources of the Center for Computation and Technology at Louisiana State University, which is supported by funding from the Louisiana legislatures Information Technology Initiative. This work was supported by the Austrian Ministry of Science BMWF as part of the Uni-Infrastrukturprogramm of the Forschungsplattform Scientific Computing at LFU Innsbruck.

8 REFERENCES

[Con10] Connor, M., and Kumar, P.: Fast Construction of k-Nearest Neighbor Graphs for Point Clouds. IEEE TVCG 16, No.4. pp.599–608, 2010.

[Yan06] Yang, R., Guinnip, D., Wang, L.: View-dependent textured splatting. The Visual Computer 22, pp.456–467, 2006.

[Has01] Hasan, K.M., Basser, P.J., Parker, D.L., Alexander, A.L.: Analytical computation of the eigenvalues and eigenvectors in DT-MRI. J. Magn. Reson. 152, pp.41–47, 2001.

[Ale04] Alexa, M., Rusinkiewicz, S., and Adamson, A.: On normals and projection operators for surfaces defined by point sets. Eurographics Symp. PBG., pp. 149–155, 2004.

[Bou212] Boulch, A., and Marlet, R.: Fast and Robust Normal Estimation for Point Clouds with Sharp Features. Comp. Graph. Forum 31, No.5, pp.1765-1774, 2012.

[Walnut] Open Walnut. `http://openwalnut.org`.

[Ben07] Benger, W., Ritter, G., Heinzl, R.: The Concepts of VISH. 4th High-End Vis. Workshop, pp.26–39, 2007.

[Ben04] Benger, W., Hege, H.-C.: Tensor splats. Conf. on Vis. and Data Analysis, Vol.5295, pp.151–162, 2004.

[Ber94] Berkmann, J., and Caelli, T.: Computation of surface geometry and segmentation using covariance techniques. IEEE TPAMI 16, No.11, pp.1114–1116, 1994.

[Rit12a] Ritter, M., Benger, W., Cosenza, B., Pullman, K., Moritsch, H., Leimer, W.: Visual Data Mining Using the Point Distribution Tensor. IARIS Workshop on Computer Vision and Computer Graphics, VisGra, 2012.

[Rit12b] Ritter, M., Benger, W.: Reconstruction Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field. WSCG, pp.223–230 ,2012.

[Bjö05] Johansson, B., and Moe, A.: Object Recognition in 3D Laser Radar Data using Plane triplets, technical report LiTH-ISY-R-2708, Dept. EE, Linköping University, 2005.

[Rus00] Rusinkiewicz, S., Levoy, M.: QSplat: A Multiresolution Point Rendering System for Large Meshes, SIGGRAPH '00, pp.343–352, 2000.

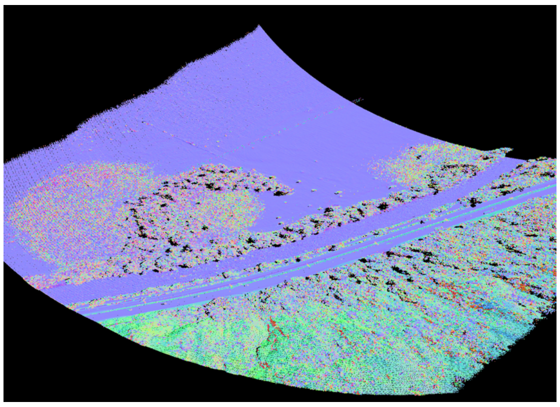
[Bot05] Botsch, M., and Hornung, A., and Zwicker, M., and Kobbelt, L.: High-quality surface splatting on today's GPUs. Eurographics VGTC Symposium on PBG, pp.17–24, 2005.

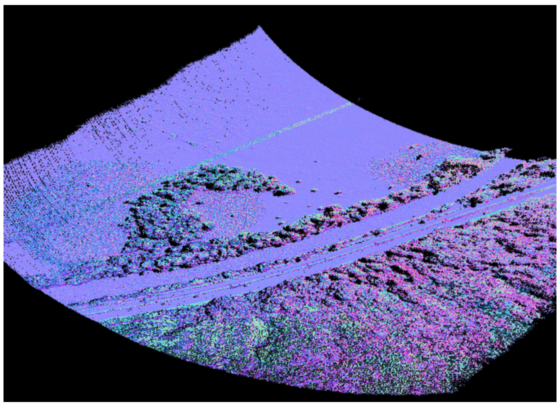
[Sch11] Schiffner, D., Krömker, D.: Three Dimensional Saliency Calculation Using Splatting, 6th ICIG, pp.835–840, 2011.

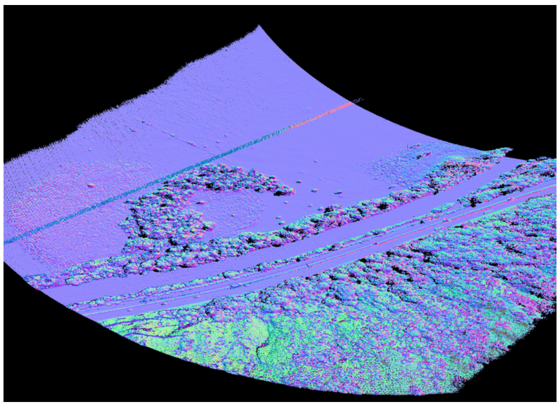
[Shi09] Shirley, P., and Marschner, S.: Fundamentals of Computer Graphics, 3rd Edition, A.K. Peters Ltd, 2009.

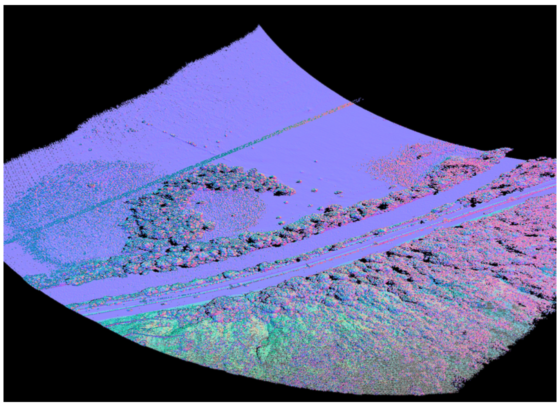
[Kop06] Kopp, J.: Efficient numerical diagonalization of hermitian 3x3 matrices, arXiv:physics/0610206v1, 2006.

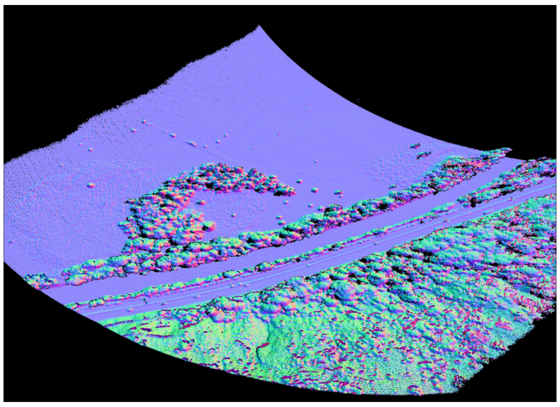
[Ste10] Steinbacher, F., Pfennigbauer, M., Ulrich, A., and Aufleger, M.: Vermessung der Gewässersohle - aus der Luft - durch das Wasser, in Wasserbau in Bewegung ... von der Statik zur Dynamik. Beiträge zum 15. Gemeinschaftssymposium der Wasserbau Institute TU München, TU Graz und ETH Zürich, 2010.


(a) World space, 1m squared


(b) Local derivatives


(c) Tensor, 25 samples


(d) Tensor, 25 samples, no correction


(e) Plane approximation, diagonals, 25 samples

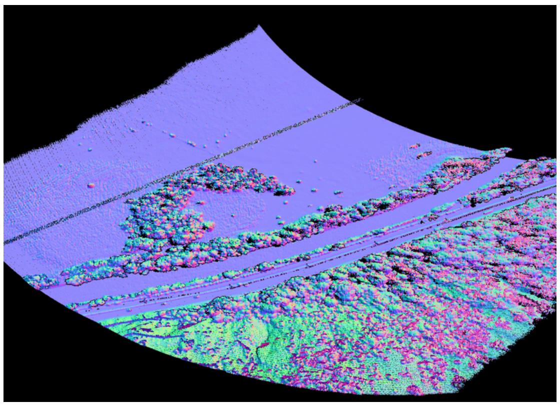

(f) Plane approximation, ccw, 25 samples

Figure 10: Comparison of the different reconstruction methods used on the Loisach LIDAR data set.

5 Application

5.1 Preliminaries

5.1.1 Enhancing Point Cloud Visualization

Tensor Splats: Since the neighborhood analysis is based on a second order tensor, a suitable visualization method was selected, extended, and employed. Tensor splats were used to illustrate the point distribution tensor. Tensor splats were introduced by Benger and Hege (2004). For visualization, a splat is drawn at each tensor's position; oriented, textured, and colored dependent on its shape factors. Barycentric blending is used to steer the visual appearance, see Figure 5.1.

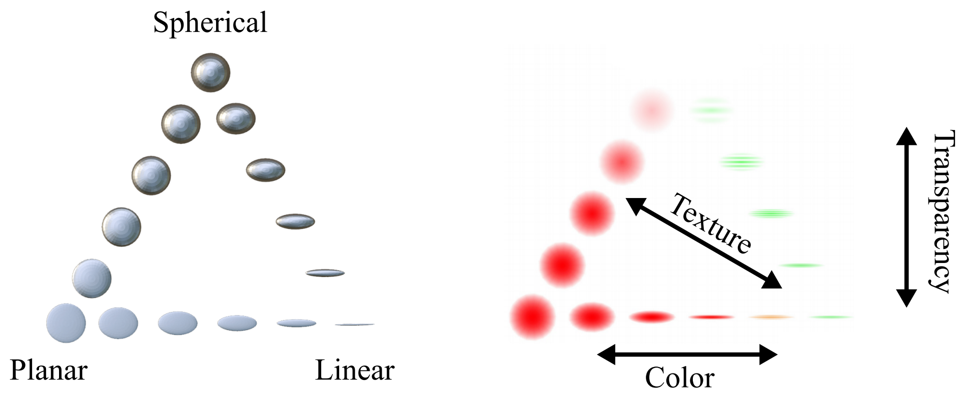


Figure 5.1: (Left:) Barycentric blending of shape factors by scaling the main axes of an ellipsoid. (Right:) Besides shape, color, texture, and transparency are adjusted for tensor splats.

In contrast to a basic ellipsoid visualization tensor splats reduce visual clutter by hiding spherical shapes (homogeneous case) and semi transparent alpha blending. A stretched Gabor function is used as a transparency texture to enhance the visual direction. The splats fade to transparent at their borders. This allows to visually blend into each other, resulting in a line like fiber structure (see figures in the publications below). In data, often the regions of high linearity and planarity are of more interest than the homogeneous regions, where the tensor splats become fully transparent and invisible. Originally, the method was developed to highlight preferred diffusion directions of water in the human brain and to illustrate space time curvature close to objects of immense density and strong gravitational fields, such as black holes or neutron stars. In both cases the homogeneous areas do not carry relevant information, thus they can be hidden in the visualization. Due to the visual properties the method is especially suited for large data. In its full complexity it allows to additionally steer brightness and saturation. However, this was not used in our applications as visualizations become hard to read when encoding too much information in one illustration.

All visualizations in the publications presented in this section were created using the vish visualization shell (Benger et al., 2004). Application data had to be first converted into the underlying data model by implementing HDF5 based data converter programs. Within vish a visualization task is organized in a network structure of modules (or components). To enable the point distribution visualization a module was developed and added – operating on vertex data – computing the second order tensor field. It was extended later to handle different particle sizes via an additional scalar data field (radii) on the vertices, to employ different radial weighting functions, and to support a discrete explicit number of neighbors. A previously implemented kd-tree was used for the neighborhood search and the tensor splats module for the visualization of the tensor field. The tensor splat module was extended to also show the inverse of the tensor.

Astrophysics: The method was first applied to astrophysical simulations. Here, two different simulations were chosen to present an application of tensor field visualization in particle based data: a cosmological evolution and a wind tunnel galaxy simulation. Both simulations investigate processes of the development of our universe. The former analyses how cosmological structure is formed. Therefore, the theory of *concordance* is employed, which identifies cold dark matter as the driving material in the formation process. Cold dark matter is modeled by collision less particles. A N-body simulation of 130 million particles is carried out. The simulation uses a hybrid approach of hierarchical multi-scale grids and particles to discretize spacetime. Initially, a Gaussian random distribution is assumed and then particles are evolved over time. Besides cold dark matter normal baryonic matter is included in the simulation. The focus of the study is on investigating secondary effects: within the formation of galaxies in dark matter halos, the chemical enrichment of the surrounding gas. Chemical transport into the inter cluster medium is of special interest. To enable the large variation in spatial and temporal scaling analytical models are included into the simulation. Semi-analytic formation models are driven by the numerical simulation. Within the scope of this thesis the visualization of the forming dark matter particles were of interest. The point distribution tensor computation was applied along with the tensor splat technique. The 130 million particles were enriched by geometrical distribution information, which enhanced the readability of the point particle visualization.

A second numerical simulation targets the process of star forming by ram pressure stripping of a galaxy. Galaxy clusters have been results from the cosmological simulation. They are the largest gravitational bound structures in the universe and consist of galaxies, dark matter, and intra cluster medium (ICM); a very hot and thin gas. Star formation happens in the interaction of the gas and the galaxies in the so called inter stellar medium (ISM); the gas in the galaxy, which is being stripped off by the ICM due to high pressures. Inside the ISM, in molecular clouds, stars are born from dust and gas (mainly hydrogen and helium) by building high density clumps. Dense clumps can create very high gravitational forces and collapse. Here, an N-body simulation of dark matter and stars is coupled with a smoothed particle hydrodynamics method for the simulation of the ICM's gas. The simulation uses heuristic recipes to model star formation and other secondary effects. A 'wind tunnel' is set up where a galaxy is moving through a cube filled with gas particles of the ICM.

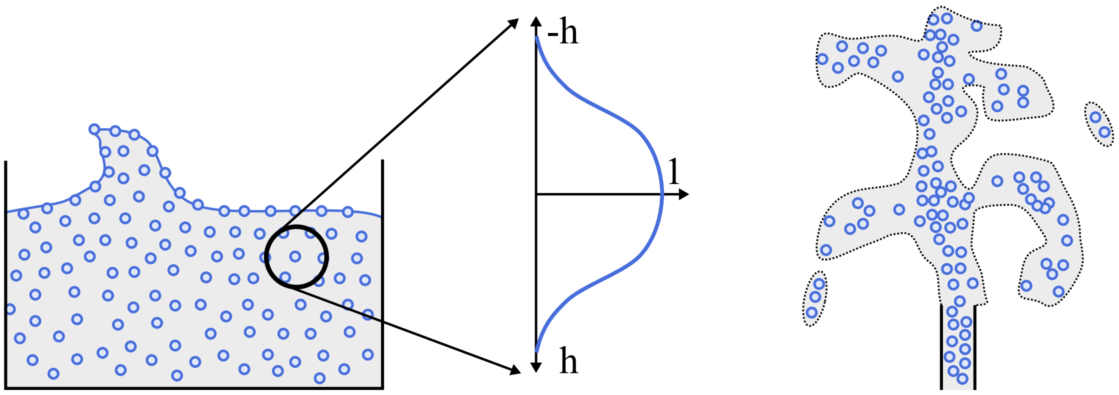


Figure 5.2: SPH simulation captures data fields in a continuum by particles. Weighting kernels are used to interpolate values; position, velocity, pressure, etc. are stored solely at the particles. In contrast to simulations of water (left), gas simulations typically result in varying densities, non homogeneous particle distances and sizes (right).

In smooth particle hydrodynamics particles follow the fluid's flow. State variables are stored for each particle. Typical field variables are: position, pressure, and velocity; more fields can be added to capture additional physical phenomena. Continuous fields are created by using kernel functions; radial distance weights. A field value in between particles is obtained by interpolation of the values at the particles using the distance weights; by 'smoothing' over the neighborhood. A field value of a field Φ in a continuum domain Ω at a location x is defined as:

$$\Phi(x) = \int_\Omega \Phi'(x)\delta(x - x')d\Omega_{x'}$$

with δ the Dirac function. In SPH smooth kernel functions $\omega(x, h)$ are employed to approximate the Dirac function. For 'small' sizes of the support radius h one obtains:

$$\begin{aligned} \Phi(x) &\approx \int_\Omega \Phi(x')\omega(x - x', h)d\Omega_{x'} && (5.1) \\ &\approx \int_\Omega \frac{\Phi(x')}{\rho(x')}\rho(x')\omega(x - x', h)d\Omega_{x'}. \\ \Phi(x_i) = \Phi_i &\approx \sum_j \frac{\Phi_j}{\rho_j}\underbrace{\rho_j V_j}_{m_j}\omega(x_i - x_j, h), && (5.2) \end{aligned}$$

with j denoting the index of the neighboring particles of particle i, V_j volume and m_j mass. The density can be estimated by the particle's spatial distribution. From densities, pressures can be computed. Note that a partial derivative of a field is obtained by combining the derivatives of the underlying kernel functions:

$$\frac{\partial\Phi}{\partial x} = \sum_j \frac{\Phi_j}{\rho_j} m_j \frac{\partial\omega}{\partial x}. \qquad (5.3)$$

Pressure is linked to density using the partial derivatives. The equations of motion for each particle can be formulated; conserving linear and angular momentum (Monaghan, 2005). Figure 5.2 illustrates a water and a gas simulation. A smoothing kernel function is indicated as well. In the wind tunnel experiment gas was simulated. Inhomogeneously distributed particles in space and time were the result. Here, the tensor visualization technique was applied to improve the readability of the gas particle distributions.

In the cosmological simulation the tensor visualization enhanced the pure transparent point visualization. It allows to visually differentiate between planar and homogeneous clusters and also revealed an hourglass structure in the central cluster, Figure 14, (Benger et al., 2012); indicating on the motion pathways of the particles. The wind tunnel simulation generally revealed, that the star formation rate was especially high in gas trails behind formed particle blobs. The tensor visualization enhanced the overall readability of the geometric distribution of the particle simulation and, also, highlighted linear regions – the trails – close to the blobs, were stars are born, see Figure 5.3 and Benger et al. (2012).

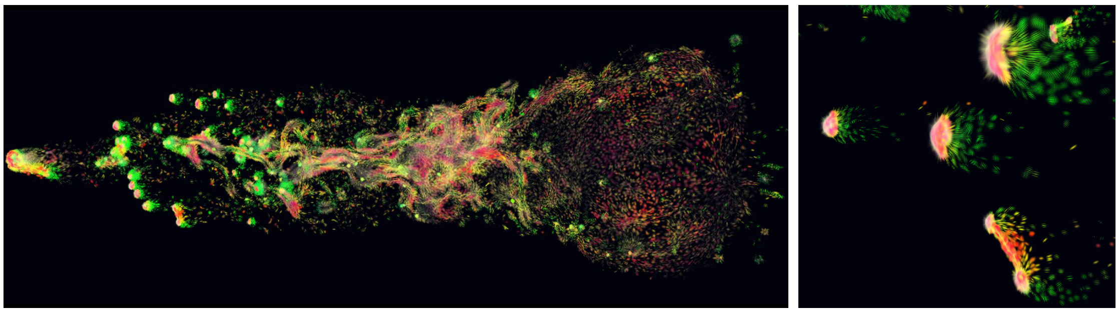


Figure 5.3: Smooth particle hydrodynamics simulation of a galaxy in the inter cluster medium. Regions in the wake of gas blobs were identified for statistically increased star formation rates (left). These trails are also enhanced by the tensor visualization, as green splats (right).

Geoscience: A second domain was chosen to experiment on point based data visualization. Two data sources stemming from geosciences were analyzed: point sampled coast lines and airborne light detection and ranging (LiDAR) laser scanning. A world wide data set of coast lines[1] was analyzed and a small, but highly dense, LiDAR scan of 4.81 million points, covering an area of $300\,m \times 50\,m$. The LiDAR dataset was captured by a measurement laser prototype on a small airborne survey mission in Lower Austria; in cooperation with the companies AHM[2] and RIEGL[3]. A helicopter was used for the data acquisition in a not yet commercial setup of the bathymetric laser scanner *Riegl-VQ820-G*, as presented in Steinbacher et al. (2012).

LiDAR data already carries internal neighborhood information on a low level. Figure 5.4 illustrates the process of airborne laser scanning and neighborhood relations. Laser light is pulsed with a very high frequency; $50\,kHz$ for the employed data set. Light pulses are emitted from the laser source. On a nanosecond scale, the light intensity distribution of one pulse has a shape similar to a Gauss distribution, see Figure 5.4 (left). The light pulse travels at the speed of light and is reflected e.g. at a solid obstacle. The light beam has a divergence. The so called footprint size was about $10\,cm$ on ground, by operating at a flight altitude of about $100\,m$, for this scan. The light is reflected and refracted in different directions at the object. A light sensor located close to the laser source starts to record light intensities when a certain intensity threshold is reached. From one pulse, several echoes return with different time stamps and intensities. The time stamped intensities can now be used to compute 3D space locations. Therefore, the angle and time

[1]https://www.ngdc.noaa.gov/mgg/shorelines
[2]http://www.ahm.co.at
[3]http://www.riegl.com

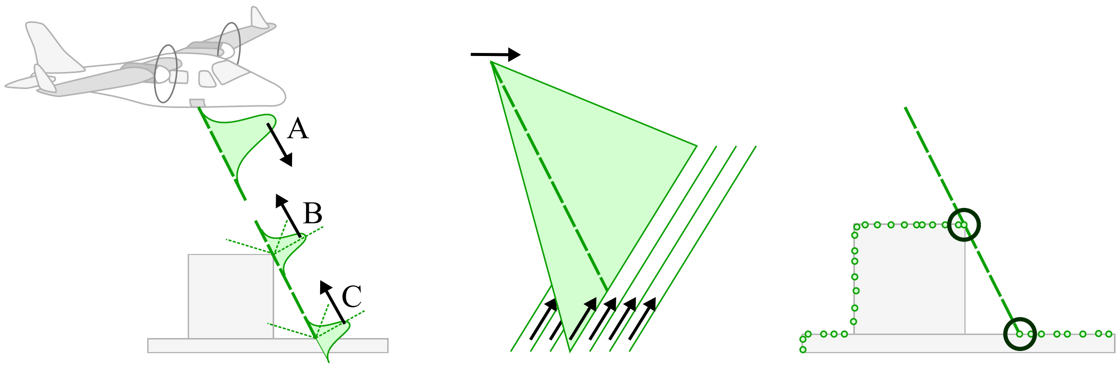


Figure 5.4: (Left:) Airborne light detection and ranging generates multiple measures per laser shot. Light intensities are sent out and recorded. (Center:) Laser shots follow a scan pattern; i.e. linear or circular. (Right:) By known time, scan angles and the position and angles of the laser scanner 3D points can be computed. Neighboring points represented in the local coordinate system of the laser scanner (time, scan line, and angle within a scan line) need not be neighbors in 3D space.

stamp of the pulse, the time stamps of the returned echoes, and the position of the laser source are required. Thus, return echoes carry a neighborhood information by time, angle, and scan line. But, neighbors in these local coordinates are not necessarily neighbors in 3D space, see Figure 5.4 (right). The two marked points share the same scan angle, laser shot, and have a similar time stamp. Still, they are far from being neighbors in space. Thus, the point distribution tensor was employed for visual analyzes.

Again, data converters were developed to enable data storage and handling via an HDF5 based format. The vish environment was then utilized for visualization, further development, and visual analyzes. The tensor algorithm was extended by more neighborhood control options, and work started on performance optimization, especially on the neighborhood search by GPU compatible grid hashing and bitonic sorting, as outlined in Section 4.1.2.

The analyzed shape files representing the coast revealed a weakness in the kd-tree data structure. Some lines were sorted in the data, such that space division by the tree happened in a very inefficient way. A few but very deep branches could be generated. A simple random shuffle on the points to be inserted into the kd-tree was implemented to circumvent this shortcoming. The visual analysis showed no strong improvement for the coastlines. However, in case of the LiDAR data a visual classification became apparent. The ground, vegetation, and the power cables were visually enhanced and became better distinguishable. The directional fibers of the tensor splats visually created linear structures of the point cloud; i.e. for the cables and rods of the power poles, see Ritter et al. (2012)* below.

5.1.2 Geometric Reconstruction

The visual exploration via tensor splats of the point distribution tensor revealed clearly linear structures, such as the power cables. Also, the directions of the eigenvectors followed the cables prominently. Inspired by methods applied on the diffusion tensor in diffusion tensor magnetic resonance imaging (DT-MRI) a next study was carried out, to reconstruct those power cables. In DT-MRI the movement of water molecules in human tissue is measured, see Figure 5.5. The resulting directions are then captured in a second order tensor field, holding the preferred diffusion directions of water. Formally, the tensor is defined via:

$$\frac{\partial \Phi(x,t)}{\partial t} = \nabla \mathcal{D}(\nabla \Phi(x,t)) \tag{5.4}$$

$$\mathcal{D}(v) = \big(D_i(0) + D_{i,j}(0)v^j + 0.5 D_{ij,k}(0)v^j v^k \ldots\big)e^i, \tag{5.5}$$

where $\mathcal{D}$, the flux, is a function of the water concentration gradient $\nabla\Phi(x,t)$ and $D_{i,j}(0)$ a second order tensor; e^i is the Euclidean basis (see e.g. Benger et al. (2006)). The tensor is the first non vanishing term in a Taylor series expansion; higher orders can usually be neglected.

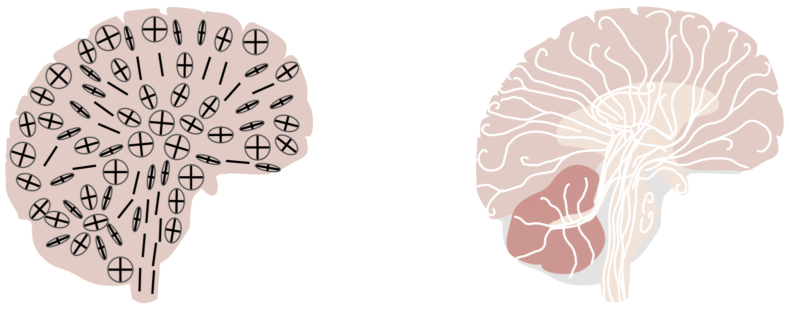


Figure 5.5: Schematic depiction of magnetic resonance imaging capturing diffusion directions of water in tissue. (Left:) The measured directions at space locations are encoded by a second order tensor; indicated by ellipses. (Right:) The eigenvectors of the tensor field can be used to trace neurons and allow to classify and segment, e.g. regions in the human brain.

When scanning a human brain, there are different types of tissue, which are separable by the directional information: grey matter and white matter. White matter contains elongated myelinated axons, thus, such tissue shows a strong dominant diffusion direction. In contrast gray matter (and also tumors) have no dominant directions. The tensor holds integrated directional information over a small spatial volume. Its precision dependents on the resolution of the scanning device. Thus, a homogeneous tensor could also be the result of crossing of axons; mixing dominant directions. Still, and with increasing available resolution, MRI data allowed to segment and classify human brain regions.

The second order diffusion tensor relates to the point distribution tensor. Instead of diffusion directions, the directions from a point of reference to its neighbors are encoded; mathematically the directional encoding is the equivalent. Fiber tracking is a technique used in DT-MRI data to trace along tensor directions and axons; to find and analyse interconnected brain regions. A similar technique was applied on the point cloud tensor data of the LiDAR scan. In contrast to axons in the human brain, power cables of a power line were traced.

A previously developed stream line integration algorithm (Ritter (2011)) was extended. The algorithmic development required to enable the support of mesh free, point based data; i.e. kernel based interpolation. In DT-MRI uniform grid structures are used. Also, the stream line integration was extended to follow eigenvector directions instead of vectors. Eigenvectors are bidirectional, thus, an integration should always follow the 'axis' of the vector, even when it is pointing contrarily with respect to the current integration direction.

Seven weighting functions were tested for data interpolation and for tensor computation. The eigenvector streamline algorithm was first extended to integrate on an original DT-MRI dataset, and then compared to the results of the same dataset but converted to a point based discretization. Two different explicit integration schemes were tested: symplectic Euler and Dormand-Prince-853 (DOP853) (Dormand and Prince, 1980). The DOP853 is an explicit Runge-Kutta scheme of order 8 with included error estimation and step size control. Internally it uses order 5 and 3 for error measure and step size optimization. It exhibits a good trade off between accuracy and run time performance, when there is a need for high accuracy (Hairer et al., 1993)[4]. The work shows in the artificial test cases of the circle and rectangle, that the high order integration improves the circle, but cannot handle abrupt changes in curvature, i.e. the sharp corners of the rectangle.

Finally, the best cable reconstruction was found by employing the DOP853 integrator, the quadratic weighting function for the tensor computation, and the SPH kernel of order 5 for the mesh less interpolation. A constant radius was employed for the tensor computation; see Ritter and Benger (2012)* below.

[4]http://www.unige.ch/~hairer/software.html

Visual Data Mining Using the Point Distribution Tensor

Marcel Ritter
Graduate School for Scientific Computing
University of Innsbruck
Innsbruck, Austria
marcel.ritter@uibk.ac.at

Werner Benger
Center for Computation & Technology
Louisiana State University
Baton Rouge, USA
werner@cct.lsu.edu
Institute for Astro- and Particle Physics
University of Innsbruck
Innsbruck, Austria
werner.benger@uibk.ac.at

Biagio Cosenza
Distributed and Parallel Systems Group
University of Innsbruck
Innsbruck, Austria
cosenza@dps.uibk.ac.at

Keera Pullman
ESRI Australia
Darwin, NT, Australia
kpullman@esriaustralia.com.au

Hans Moritsch
Distributed and Parallel Systems Group
University of Innsbruck
Innsbruck, Austria
hans@dps.uibk.ac.at

Wolfgang Leimer
Distributed and Parallel Systems Group
University of Innsbruck
Innsbruck, Austria
wolfgang.leimer@student.uibk.ac.at

***Abstract*—We explore a novel algorithm to analyze arbitrary distributions of 3D-points. Using a direct tensor field visualization technique allows to easily identify regions of linear, planar or isotropic structure. This approach is very suitable for visual data mining and exemplified upon geoscience applications. It allows to distinguish, for example, power lines and flat terrains in LIDAR scans. We furthermore present the work on the optimization of the computationally intensive algorithm using OpenCL and potentially utilizing the Insieme optimizing compiler framework.**

Keywords-metric tensor; scientific visualization; point cloud; OpenCL.

I. Introduction

Point clouds occur as primary data sources in different scientific domains, e.g., stemming from simulations in computational fluid dynamics by smooth particle simulations or from observational methods, such as light detection and ranging (LIDAR) laser scanning [1]. Classification of point clouds is still ongoing research for LIDAR laser scan data [2]. Geometric information about the local point distribution can be used for classification, for constructing surfaces, or as basis for other algorithms. An algorithm to compute Gaussian and mean curvature on polygon meshes was presented in [3], based on the tensorial product of the polygon's normal vectors. A product with additional weights was used to compute the co-variance matrix of point neighborhoods describing tangential frames for surfaces in [4]. This co-variance matrix provides us with a type of smooth transition between lines, surfaces, and volumes [5].

In this article, we utilize the direct tensor visualization technique [6] to illustrate the co-variance matrix resulting from arbitrary point clouds. Section II introduces the distribution tensor and the utilized visualization technique, presented on simple geometric point distributions. Two algorithms for the tensor computation are described: One for central processing units (CPUs) and one for graphics processing units (GPUs). Optimizations are presented and the Insieme compiler optimization framework [7] is introduced. Our visualization method is demonstrated on two geo-scientific applications in Section III: On the analysis of LIDAR laser scan data and the analysis of coastlines. The paper concludes and describes future work in Section IV.

II. Computing the Point Distribution Tensor

A. Mathematical Background

We define the "point distribution tensor" as a measure constructed from of a set of N points $\{P_i : i = 1...N\}$ similar to the co-variance matrices in [4] [8] [5]:

$$S(P_i) = \frac{1}{N} \sum_{k=1}^{N} \omega_{ik} (t_{ik} \otimes t_{ik}^{\tau}) \tag{1}$$

whereby $t_{ik} = P_i - P_k$ and an optional weighting function $\omega_{ik} := f(||P_i - P_k||, r, i)$. Here, r is an user specified distance or radius defining the neighboorhood of point P_i. The weighting function ω_{ik} is zero outside this radius. The distribution tensor is symmetric and positive definite such as the metric tensor [9] and, thus, yields three eigen-values when doing an eigen-analysis: $\lambda_3 \geq \lambda_2 \geq \lambda_1$. These are used to classify the tensor via three shape factors [10], characterizing the shape of a fitting ellipsoid of the point neighborhood in barycentric coordinates, see Figure 1:

$$\begin{aligned} c_{linear} &= (\lambda_3 - \lambda_2) & /(\lambda_1 + \lambda_2 + \lambda_3) \\ c_{planar} &= 2(\lambda_2 - \lambda_1) & /(\lambda_1 + \lambda_2 + \lambda_3) \\ c_{spherical} &= 3\lambda_1 & /(\lambda_1 + \lambda_2 + \lambda_3) \end{aligned} \quad (2)$$

with $c_{linear} + c_{planar} + c_{spherical} = 1$. A tensor field visualization method more suitable for large data than drawing tensor ellipsoids is utilized. Instead of ellipsoids textured splats are rendered with smooth transitions in color, orientation, texture and transparency, as shown in Figure 1.

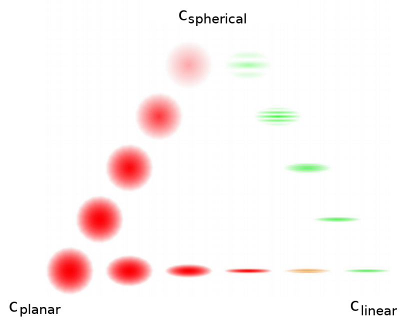
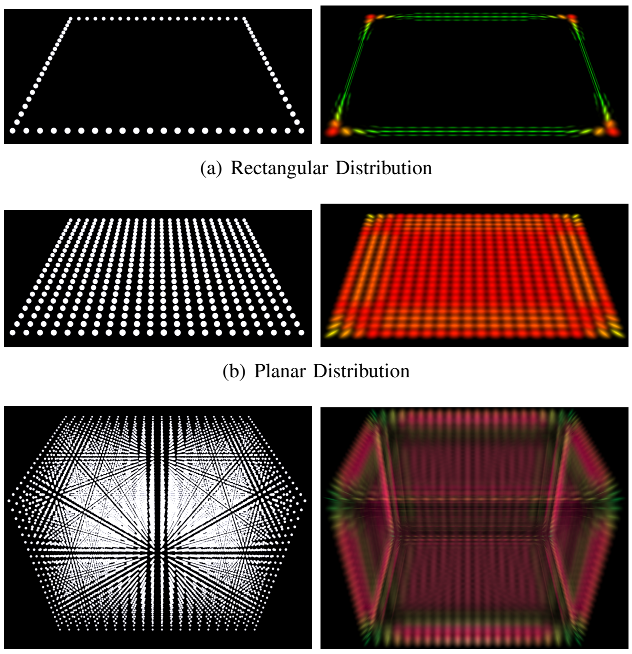


Figure 1. Tensors are visualized as textured oriented disks. The three shape factors, Equation 2, are used for smooth transitions between linear (right), planar (left), and spherical (top) shape [9]. In this context, the disks enhance the visualization of points predominantly distributed on a line, on a surface, or in a volumetric distribution.



(a) Rectangular Distribution

(b) Planar Distribution

(c) Volumetric Distribution

Figure 2. *Left:* Analytic point distributions illustrated by simple point rendering. *Right:* Corresponding distribution tensor fields. Linear 1D, planar 2D and isotropic 3D tensors are visualized using tensor splats, having a dominant linear, planar and spherical shape factor, respectively.

B. Test Cases

Simple analytic test cases were used to verify and study the properties of the distribution tensor, as illustrated in Figure 2. The point distributions have an extent of 1.0 in spatial dimensions and have been computed using a neighborhood radius of $r = 0.2$. Figure 2 (a) shows linear tensor splats textured and oriented in one direction. At the corners of the rectangle tensors become planar caused by two equally dominant directions in the neighborhood. Homogeneous distributions are fully transparent and become invisible, as demonstrated in Figure 2 (c). Here, the inner region is transparent, the border surfaces become more planar and are colored red while corner points become linear (green).

C. Algorithm

A first serial algorithm was implemented in the visualization shell VISH [11] utilizing C++ and OpenGL. Computation and visualization tasks were split in different modules. The computation module searches for neighbors in a 3D KD-Tree [12] within a user specified radius (where $\omega_{ik} > 0.0$) to limit the number of considered points. Alternatively to setting the radius, also the number of neighbors can be specified. Furthermore, a scalar field given on the points can be utilized to set the radius or number of points for each point individually. Eqn. 1 is utilized to compute the distribution tensor for each point. Different weighting functions have been implemented inside the neighborhood: $\omega_{ik} := (r - ||P_i - P_k||)/r$, $\omega_{ik} := 1/r$ and $\omega_{ik} := 1/r^2$.

Data is represented in an unified data model [9] [13] which allows support of different types of grid geometries and topologies. The computation algorithm operates on the vertices of any grid type. In the following applications point clouds and sets of lines are used for analysis.

D. GPU implementation

An alternative implementation of the algorithm was done in OpenCL [14], a framework for multicores and parallel hardware being able to execute programs also across heterogeneous platforms. The neighborhood is controlled by a fixed radius. Instead of the KD-Tree, an uniform grid was preferred as data structure to speed-up the neighborhood search. Here, the *loose grid* approach was adapted, where each particle is assigned to one cell based on its position. The grid's cell size depends on the influence radius (i.e., radius≥cell size). Therefore, each particle can affect the closest 27 cells while calculating the tensor. This method allowed to bin the particles into the cells and to sort them by their grid index. The algorithm comprises four steps:

1) for each particle a hash value is computed, i.e., the cell index where it is located;
2) particles are sorted by hash; for this step NVidia's optimized bitonic sorting [15] is utilized;

3) the sorted list is used to compute the starting cell where the particle is located, running a thread for each particle, and performing scattered memory writes;
4) tensor calculation: Each particle searches the closest 27 grid cells from its location and it computes the tensor with each of the particles in these cells.

Steps $1 - 3$ are related to the build process of the grid data structure. The sorting algorithm is highly effective because it improves the memory access coherency when calculating the tensor, and reduces thread divergence (particles in the same thread group tend to be close together in space).

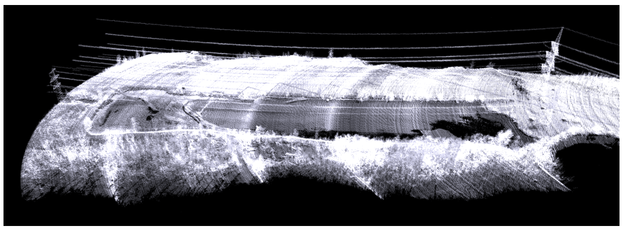
E. Optimization

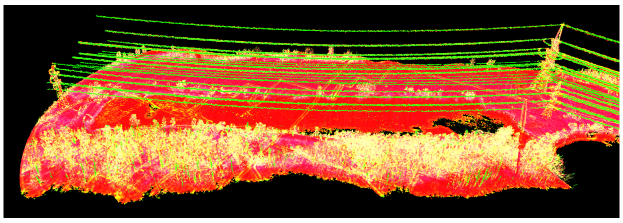
The CPU algorithm was parallelized using OpenMP [16], adding a minimal overhead in development. Furthermore, OpenMP, as also OpenCL is supported by Insieme [7].

The Insieme compiler, under development at the University of Innsbruck, is a source-to-source compiler for C/C++ aiming at the automatic optimization of parallel programs implemented with MPI, OpenMP or OpenCL. It optimizes the source code for a specific platform (e.g., NVidia Fermi architecture), and applies transformations such as loop enrolling and collapsing, thread merge and data pre-fetching. Insieme aims at supporting programmers in effectively optimizing programs across different architectures, including shifting of computations from CPU to GPU cores. Optimizations are performed at compile-time through code analysis and transformations for sequential and parallel code regions. An intermediate representation is facilitated which explicitly describes parallelism, synchronization, and communication. The program's behavior is optimized and customized to the available hardware resources at runtime by utilizing statistical machine learning techniques based on a performance analysis database. Performance measures are, e.g., execution time, energy consumptions, and computing costs. Preliminary tests will be done with this code which is now part of the Insieme test cases.

III. Application Results

The method was applied to two different geoscience applications. Figure 3 shows a scan of a water basin close to the Danube in Austria captured with the Riegl hydrographic laser scanner VQ-820G [17]. In the example, a fixed neighborhood radius of two meters and a constant weighting function $\omega_{ik} = 1$ showed good results. Other parameters for r and ω_{ik} have been tested as well. Figure 3 (a) illustrates the received and processed laser echoes as a cloud of points; (b) and (c) show the distribution tensor field. Linear structures, such as the power cables, are well identified (green). The ground is dominantly planar (red). Some regions of the ground fade to magenta indicating less planarity. Here, grass influences the planar tensor to become more isotropic. Bushes and trees are isotropic or of an interpolated intermediate shape, mostly appearing


(a) LIDAR Echos


(b) LIDAR Tensors $r = 2.0m$

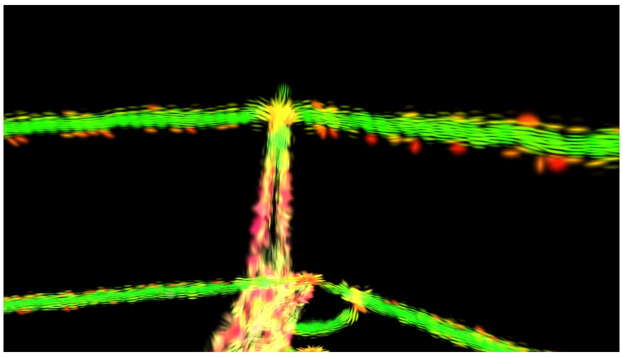

(c) LIDAR Tensors $r = 2.0m$ Detail

Figure 3. Distribution tensor field of returned laser echoes from an airborne laser scan. Linear distributions such as cables and planar distributions such as ground are emphasized. Vegetation is fading to spherical.

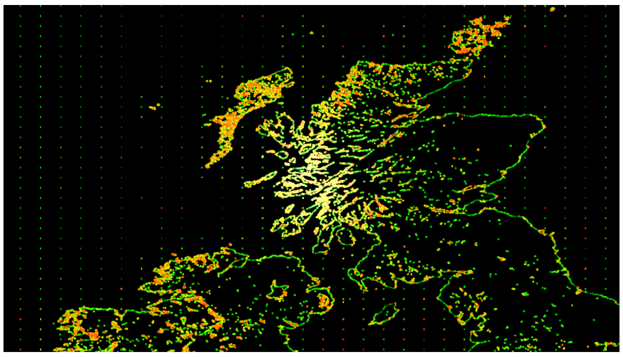


Figure 4. Distribution tensor field of an ESRI shapefile of the earth's water bodies and coastlines. The distribution tensor field with 12 fixed neighbors of the northern part of the United Kingdom is illustrated.

yellow. The computation with the OpenMP version utilizing 4 threads of the 4.81mio points with approximately 600 neighbors per point ($r = 2.0m$) took 752 seconds on a i7 M640 2.8GHz with 7.7GB RAM and NVidia Quadro FX3800M using Linux64bit, gcc 4.4.5, and Vish SVN 3854.

Another application was the analysis of coast and contour lines. Shapefiles [18] of water bodies and coastlines were investigated. Figure 4 (b) shows the distribution tensor field of the coast of the United Kingdom. Unstructured coastlines are highlighted in green whereas cliffy coast lines are shown in red, for example when looking at the northern coast of Scotland. Here, a rather small neighborhood of fixed 12 points turned out to emphasize cliffy coasts.

IV. Conclusion and Future Work

A new method of enhancing the visualization of point distributions was introduced, described, and demonstrated. Two different implementations and parallelization approaches were presented. Using an unified data model opened the possibility to apply the technique to data sets stemming from two different scientific applications: The visual extraction of power cables in LIDAR data and the visual enhancement of cliffy coastlines. We will further use the tensor analysis on LIDAR data to enhance point classification and the creation of digital terrain models. Different weighting functions and parameter studies will be investigated on more datasets. We ultimately will use the Insieme framework to optimize our parallel GPU and OpenMP codes.

Acknowledgment

Thanks to Frank Steinbacher for providing the LIDAR data sets. This work was supported by the Austrian Research Promotion Agency (FFG) *Airborne Hydromapping*, the Austrian Science Foundation FWF DK+ project *Computational Interdisciplinary Modeling* (W1227), and grant P19300. This research employed resources of the Center for Computation and Technology at Louisiana State University, which is supported by funding from the Louisiana legislatures Information Technology Initiative. This work was supported by the Austrian Ministry of Science BMWF as part of the UniInfrastrukturprogramm of the Forschungsplattform Scientic Computing at LFU Innsbruck.

References

[1] E. P. Baltsavias, “Airborne laser scanning: existing systems and firms and other resources,” *ISPRS Journal of Photogrammetry & Remote Sensing*, vol. 54, pp. 164–198, 1999.

[2] P. Dorninger, B. S. A. Zamolyi, and A. Roncat, “Automated Detection and Interpretation of Geomorphic Features in LiDAR Point Clouds,” no. 99, pp. 60–69, 2011.

[3] G. Taubin, “Estimating the tensor of curvature of a surface from a polyhedral approximation,” in *Proceedings of the Fifth International Conference on Computer Vision*, ser. ICCV ’95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 902–.

[4] M. Alexa, S. Rusinkiewicz, M. Alexa, and A. Adamson, “On normals and projection operators for surfaces defined by point sets,” in *In Eurographics Symp. on Point-Based Graphics*, 2004, pp. 149–155.

[5] J. Berkmann and T. Caelli, “Computation of surface geometry and segmentation using covariance techniques,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 16, no. 11, pp. 1114 –1116, nov 1994.

[6] W. Benger and H.-C. Hege, “Tensor splats,” in *Conference on Visualization and Data Analysis 2004*, vol. 5295. Proceedings of SPIE Vol. #5295, 2004, pp. 151–162.

[7] DPS Group at Universität Innsbruck, “The insieme compiler project.” [Online]. Available: http://www.dps.uibk.ac.at/insieme/

[8] A. Adamson, “Computing curves and surfaces from points,” Ph.D. dissertation, TU Damrstadt, 2008.

[9] W. Benger, “Visualization of general relativistic tensor fields via a fiber bundle data model,” Ph.D. dissertation, FU Berlin, 2004.

[10] C. Westin, S. Peled, H. Gudbjartsson, R. Kikinis, and F. Jolesz, “Geometrical diffusion measures for mri from tensor basis analysis,” in *Proceedings of ISMRM, Fifth Meeting, Vancouver, Canada*, Apr. 1997, p. 1742.

[11] W. Benger, G. Ritter, and R. Heinzl, “The Concepts of VISH,” in 4^{th} *High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007*. Berlin, Lehmanns Media-LOB.de, 2007, pp. 26–39.

[12] J. H. Friedman, J. L. Bentley, and R. A. Finkel, “An algorithm for finding best matches in logarithmic expected time,” *ACM Transactions on Mathematics Software*, vol. 3, no. 3, pp. 209–226, September 1977.

[13] M. Ritter, “Introduction to HDF5 and F5,” Center for Computation and Technology, Louisiana State University, Tech. Rep. CCT-TR-2009-13, 2009.

[14] KHRONOS Group, “OpenCL,” 2011. [Online]. Available: http://www.khronos.org/opencl

[15] K. E. Batcher, “Sorting networks and their applications,” in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, ser. AFIPS ’68 (Spring). New York, NY, USA: ACM, 1968, pp. 307–314.

[16] OpenMP Architecture Review Board, “OpenMP,” 2011. [Online]. Available: http://openmp.org

[17] F. Steinbacher, M. Pfennigbauer, A. Ulrich, and M. Aufleger, “Vermessung der Gewässersohle - aus der Luft - durch das Wasser,” in *Wasserbau in Bewegung ... von der Statik zur Dynamik. Beitrge zum 15. Gemeinschaftssymposium der Wasserbau Institute TU München, TU Graz und ETH Zürich*, 2010.

[18] ESRI, “ESRI Shapefile Technical Description,” Environmental Systems Research Institute, Inc, White Paper, July 1998.

Reconstructing Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field

Marcel Ritter
Institute of Basic Sciences in Civil Engineering[2]
University of Innsbruck
marcel.ritter@uibk.ac.at

Werner Benger
Center for Computation & Technology[1]
Institute for Astro- and Particle Physics[2]
werner@cct.lsu.edu
werner.benger@uibk.ac.at

ABSTRACT

Starting from the computation of a covariance matrix of neighborhoods in a point cloud, streamlines are utilized to reconstruct lines of linearly distributed points following the major Eigenvector of the matrix. This technique is similar to fiber tracking in diffusion tensor imaging (DTI), but in contrast is done mesh-free. Different weighting functions for the computation of the matrix and for the interpolation of the vector in the point cloud have been implemented and compared on artificial test cases. A dataset stemming from light detect and ranging (LIDAR) surveying served as a testbed for parameter studies where, finally, a power cable was reconstructed.

Keywords: tensor-field visualization; streamlines; mesh-free methods; particle systems; point cloud; covariance matrix; fiber tracking; LIDAR; DT-MRI

1 INTRODUCTION

Reconstructing lines from point clouds has an important application in light detection and ranging applications (LIDAR). The surveying of power lines and their geometrical analysis is of great interest for companies that transmit electrical energy. Large networks of electric facilities have to be maintained to guarantee stable electrical power supply and prevent power outages. LIDAR surveying is a suitable technique to either detect damages on the electrical facilities or detect high growing vegetation in power line corridors [19] [15]. We experiment on a new method to reconstruct linear structures, stemming from airborne LIDAR surveying. We utilize a method inspired by diffusion tensor imaging (DTI) fiber tracking developed, originally, for magnetic resonance imaging (MRI) to track neuronal structures in the human brain [5].

[1] Louisiana State University, Baton Rouge, LA-70803, USA
[2] University of Innsbruck, Technikerstraße 13/4, A-6020 Innsbruck, Austria

1.1 Related Work

Current algorithms for reconstructing power lines are usually based on data filtering followed by a segmentation of the filtered and reduced point cloud either directly on the point cloud data or on a rastered 2D image. Melzer [18] first computes a digital terrain model (DTM) by using the method by Kraus [14] to remove terrain points. The remaining points are projected onto a 2D gray-scale raster (image). A Hough-Transform (e.g. [11]) is utilized iteratively to detect straight lines. Later, Melzer [17] improved the segmentation of LIDAR data also for power cables, based on the so called mean shift clustering, originally developed for pattern recognition [9]. Liu et al. [16] introduced a methodology based on statistical analysis to first remove ground points. Then, they project points onto a 2D gray-scale raster (image) and do a Hough-Transform similar to Melzer [18], but use a different technique for the Hough-Transform [8] to detect straight lines. Jwa et al. [13] developed a four step method. First they select power-line candidates, by utilizing a voxel based Hough-Transform to recognize linear regions. After a filtering process they construct line segments based on geometric orientation rules and, finally, use a voxel-based piece-wise line detector to reconstruct the line geometries.

Weinstein et al. [23] worked on tracking linear structures in diffusion tensor data stemming from MRI. Besides following the major Eigenvector they developed some rules for overcoming areas of not linear diffusion. The flow of Eigenvectors was also used for segmentation and clustering in brain regions as, for example, shown in [6] and [20]. Jones discusses the study of connections in human brains. He states that tracking the diffusion directions is still not solved a in stable way and is an active research area [12].

Our work is based on previous work on the direct visualization of the covariance matrix describing the local geometric properties of a neighborhood distribution within in a point cloud, the so called point distribution tensor [21].

1.2 Our Approach

In our method we do not want to remove any points but operate on the entire dataset to avoid artifacts due to a complex point removal method. Instead, we first compute the point distribution tensor for each point. Eigen-analysis of the tensor yields the major Eigenvector, which indicates the dominant orientation of a point distribution. We may follow this orientation by computing streamlines along this dominant Eigenvector field in regions where one Eigenvalue dominates, so-called linear regions. In contrast, regions where the points are distributed more isotropic, are indicated by the point distribution tensor's Eigenvalues becoming more similar values. We want to avoid these regions, as they will not correspond to power cables. This approach is very similar to the fiber-tracking approach in medical visualization, but in our case the integration of the Eigenvectors needs to be done in a mesh-free way, merely on a point distribution rather than uniform grids. Thus, it can be applied to airborne LIDAR data without resampling to uniform grids (which would reduce data resolution and introduce artifacts due to the chosen resampling method).

1.3 Overview of the Paper

Section 2 presents the mathematical background and describes the implementation of the algorithm in section 2.2. Section 2.3 shows verifications by means of simple artificial point distributions. Here, the influence of different weighting functions on the tensor computation and the vector field interpolation during streamline integration is investigated. Also, two different numerical integration schemes are tested. In section 3 one set of power cables is reconstructed from a LIDAR data set stemming from actual observations. We then explore the available parameter space for weighting and integration in order to identify the best values for the given scenario.

2 ALGORITHM

2.1 Background

In [21] we defined the "point distribution tensor" of a set of N points $\{P_i : i = 1,\ldots,N\}$ as

$$S(P_i) = \frac{1}{N}\sum_{k=1}^{N}\omega_n(|t_{ik}, r|)(t_{ik}\otimes t_{ik}^{\tau}), \qquad (1)$$

whereby $\otimes$ denotes the tensor product, $^{\tau}$ the transpose and $t_{ik} = P_i - P_k$. $\omega_n(|t_{ik}|, r)$ is a weighting function dependent on the distance of a point sample to a center point P_i and a radius of a neighborhood r, which can be constant or defined by a scalar field on the points: $r(P_i)$. We did not find a generally optimal solution for the weighting function, but implemented seven choices for our first investigations:

$$\omega_1 = 1 \qquad (2)$$

$$\omega_2 = 1 - x/r \qquad (3)$$

$$\omega_3 = 1 - (x/r)^2 \qquad (4)$$

$$\omega_4 = r/x^2 \qquad (5)$$

$$\omega_5 = \begin{cases} 1 - \frac{3}{2}a^2 + \frac{3}{4}a^3 & 0 \le a < 1 \\ \frac{1}{4}(2-a)^3 & 1 \le a < 2 \\ 0 & \text{otherwise} \end{cases} \qquad (6)$$

$$\omega_6 = \begin{cases} (\frac{5}{2}-b)^4 - 5(\frac{3}{2}-b)^3 + 10(\frac{1}{2}-v)^b & [0,\frac{1}{2}) \\ (\frac{5}{2}-b)^4 - 5(\frac{3}{2}-b)^3 & [\frac{1}{2},\frac{3}{2}) \\ (\frac{5}{2}-b)^4 & [\frac{3}{2},\frac{5}{2}) \\ 0 & [\frac{5}{2},\infty) \end{cases} \qquad (7)$$

$$\omega_7 = \begin{cases} (3-c)^5 - 6(2-c)^5 + 15(1-c)^5 & [0,1) \\ (3-c)^5 - 6(2-c)^5 & [1,2) \\ (3-c)^5 & [2,3) \\ 0 & [3,\infty) \end{cases} \qquad (8)$$

with $a := \frac{2x}{r}$, $b := \frac{2.5x}{r}$ and $c := \frac{3.0x}{r}$, illustrated in Figure 1. The three functions ω_5, ω_6 and ω_7 are typical Gauss-like spline kernel functions used in smooth particle hydrodynamics (SPH) [10]. We use the same weighting functions for interpolating the vector field during Eigenvector integration. Even though, interpolation of Eigenvectors and interpolating tensors and locally computing its Eigenvectors lead to different results, we utilize the interpolation of the Eigenvector as a simpler implementation.

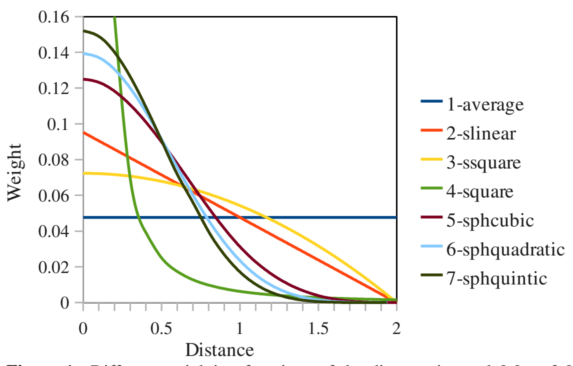


Figure 1: Different weighting functions of the distance interval 0.0 to 2.0, $r = 2.0$. Different slopes and characteristics are visualized. The *square* function (green) was clamped for axis scaling reasons and would grow further quadratically to the origin. The weights were normalized regarding to the integral of the curve in the interval. The curve numbers match the index of the weighting function: 1-average illustrates ω_1, 2-slinear illustrates ω_2, ...

We utilize tensor splats [1] for direct visualization of the tensor field. Figure 2 illustrates a point distribution along the edges of a rectangle and the corresponding tensor visualization with a neighborhood being 1/5 of the longer rectangle edge. We then use Westin's shape analysis method [24] to determine the so-called linear, planar and spherical shape factors. Points having a linearly distributed neighborhood are displayed as green oriented splats. Planar distributions are displayed as red disks. The linearity of the distribution tensor is shown in Figure 4 and Figure 5.

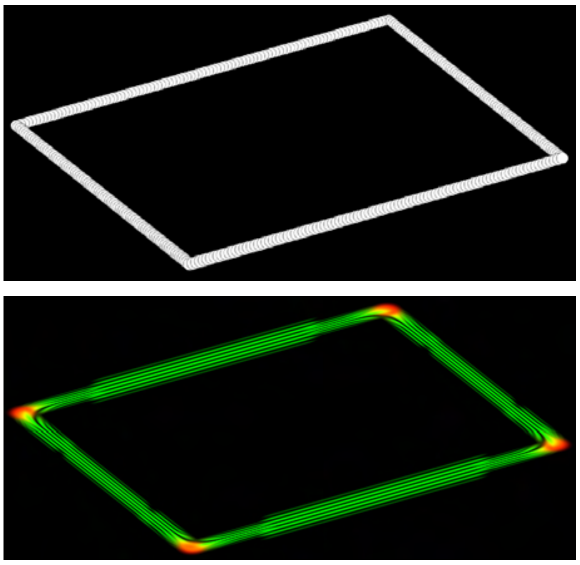


Figure 2: Distribution tensor visualization of a rectangular point distribution. *Top:* Points on a rectangle. *Bottom:* Tensor splats [1] of the point distribution tensor [21]. At each point one splat, a small textured and oriented disk, is drawn to represent the properties of the tensor's shape.

Visualizing streamlines is a common method to study vector fields. Starting from some seeding point, or initial condition, a curve $q(s)$ is computed which is always tangent to the vector field, solving the equation:

$$\dot{q}(s) = V(q(s)) \tag{9}$$

with s the curve parameter and V the vector field. Solving the differential equation at an arbitrary coordinate location Q within in the discretized data domain requires interpolation of the vector field. For mesh-free interpolation within a point cloud we use weighting functions parameterized with a specific radius of influence:

$$v(Q) = \frac{\sum_{i=1}^{N} v(P_i)\omega(|Q-P_i|, r)}{\sum_{i=1}^{N} \omega(|Q-P_i|, r)}, \tag{10}$$

with $v(P_i)$ representing the vector at point P_i.

2.2 Software Engineering Aspects

The algorithm was implemented using C++ within the VISH visualization shell [2]. The implementation extends a framework for computing integral geometries in vector fields, such as streamlines, pathlines or time surfaces. The streamline integration and visualization is separated into three different components: seeding, integration and displaying. The first component defines the initial conditions or seeding geometry. For computing streamlines within vector fields seeding points are sufficient. However, for streamlines of Eigenvector fields also an initial direction must be specified, because the Eigenvector is undirected. Integration based on an orientation continuing an user-chosen direction must be possible. Thus, requiring also a vector field on initial seeding points to disambiguate the Eigenvectors' orientations into unique directions.

Two new integration modules were developed. The first one extends the original streamline module, which was designed for vector field integration in uniform and curvilinear multi-block grids [4], to Eigenvector field integration. The second module expands this method further to allow integrating Eigenvector fields on mesh-free grids. One of the seven weighting functions (Equations 2, 3, 4, 5, 6, 7 and 8) and the radial influence weighting parameter can be specified for the interpolation of the Eigenvector inside the field domain. A range query on a KD-tree returns the points and their distances within the neighborhood of radius r. Equation 10 is utilized and Eigenvectors are aligned in orientation with respect to the Eigenvector of the closest neighbor. The Eigenvector is reversed if the dot product is negative. The integration of the streamline stops when the neighborhood becomes empty. Both integration modules support two different numeric schemes for the integration: explicit Euler and DOP853 [7]. Explicit Euler is used to get a fast yet inaccurate result. DOP853 is more expensive due to its adaptive stepsize but gives highly accurate results. When aiming at the same accuracy, DOP853 is faster than the Euler method by orders of magnitude. It is a Runge Kutta method of order eight using order five and three for error estimation and adaptive step size control, providing dense output. Accuracy measures and timing measures comparing the two integration methods were done, e.g., in [3].

The display module utilized here is reused from earlier development and implements color-coded illuminated lines utilizing OpenGL, allowing interactive navigation through the generated streamlines. Other modules, such as displaying ribbons [3] are also available.

2.3 Test Cases

We investigate the two Eigenvector integration modules on an uniform grid and on mesh-free grids. The Eigenvector field of a DTI-MRI scan [1], originally given on a uniform grid (128x128x56), was converted into a mesh-free grid, a point cloud holding the same Eigenvectors: Figure 3 (a) shows a volume rendering of the trace of the diffusion tensor along with the streamlines, revealing some brain structure and the location of the streamlines. Figure 3 (b) shows the comparison of Eigenvector streamlines computed on the uniform grid (blue) and Eigenvector streamlines computed in the point cloud (white). Both integrations were done with explicit Euler and a step size of 0.05. The size of a uniform grid cell is about 0.2, thus, utilizing about four integra-

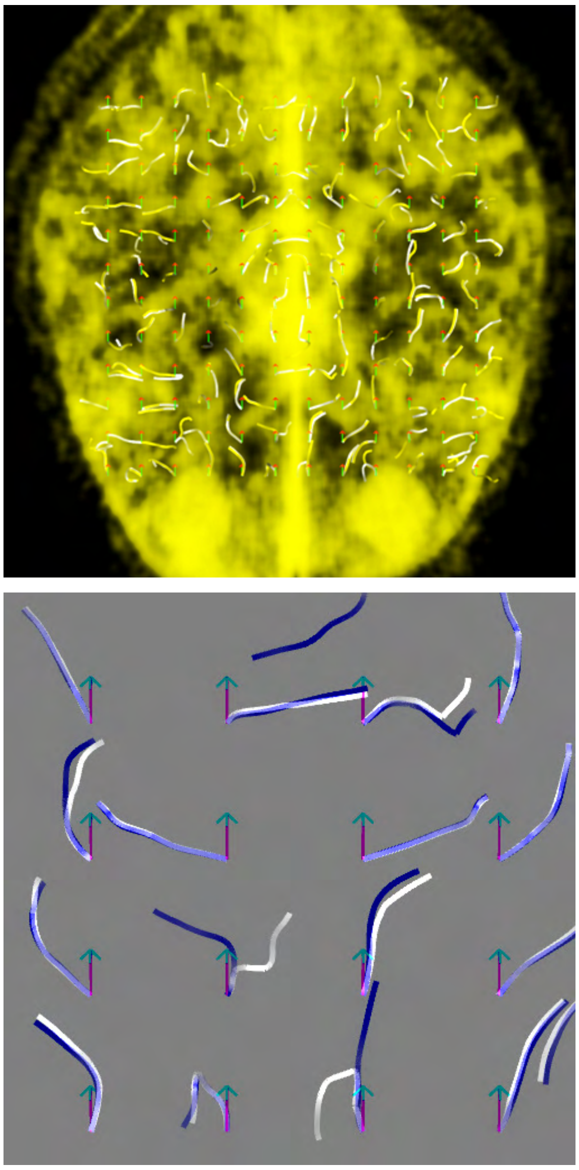
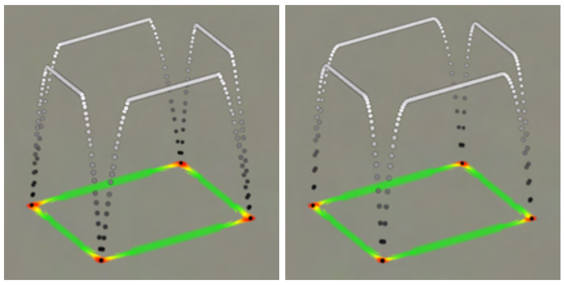
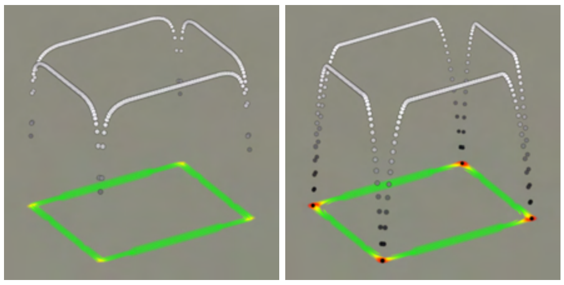


Figure 3: Comparison of the influence of integration of an Eigenvector field given on an uniform and a mesh-free grid. A mesh-free grid was generated from the uniform for testing. The arrows mark the start positions and directions of small Eigenstreamlines of a MRI diffusion tensor field. Streamlines on the uniform grid are blue. On the mesh-free grid they are white.

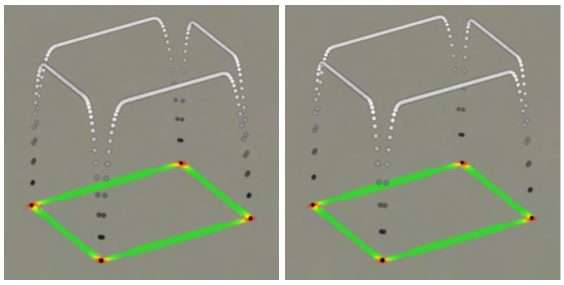
tion points per grid cell and requiring data interpolation within each cell. The length of each streamline is set to 1.0. Tri-linear interpolation was chosen for the uniform grid to compare the results with the linear weighting function ω_2 (*slinear*) for the mesh-free grid. The generated lines coincide on most cases. About 9% (13 of 144) do not coincide well. Some start in different directions. Here, the seeding vector field is almost perpendicular to the initial direction and the influence of the interpolation method results in different initial streamline directions. This issue could be cured by integrating Eigenvector streamlines in both directions starting from the initial seeding points, which would also allow avoiding the seeding vector field.



(a) average, slinear



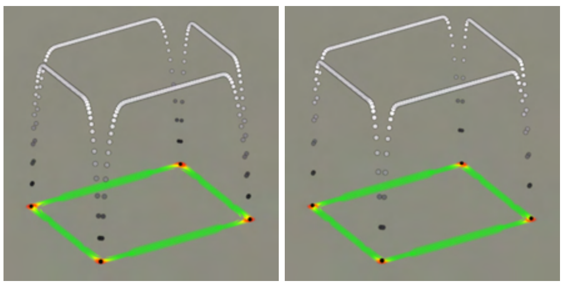
(b) square, sphcubic



(c) sphquadratic, sphquintic

Figure 4: Influence of different weighting functions on the scalar field *linearity*, compare Figure 1. The linearity is illustrated by offset and over-scaling in z-axis, and gray-scale color-map on the points. Tensor splats directly show the distribution tensor.

Next, the influence of the different weighting functions on the computation of the distribution tensor was investigated. We define an analytic distribution of points along a rectangle as test case for computing the point distribution tensor. The rectangle is set up using a width of 10 and a height of 8. The radius parameter for the neighborhood is $r = 0.2$. Figure 4 illustrates the point distribution tensor using tensor splats and its corresponding linear shape factor by offsetting, over-scaling and a gray-scale color-map. The offsetting approach for the linear shape factor clearly illustrates the influence of the weighting: The "*average*" method resulting in a very abrupt change in the slope around corners points. The "*slinear*" weighting function results in smoother changes and a more localized influence, since closer points are weighted stronger than more distant points. *Square* shows the smoothest result. The three SPH spline kernels have an increasing locality with higher order of the kernel, when comparing *sphcubic*, *sphquadratic* and *sphquintic*. This is demonstrated in Figure 5 as well: Figure 5(a) shows the result of the cubic and quadratic SPH kernel function. When the radius of the neighborhood is increased to match the kernels there is no visible difference between the *sphcubic* in Figure 5(a), *sphquadratic* and *sphquintic* in Figure 5(b) in the resulting *linearity*.


(a) *sphcubic* $r = 0.2$, *sphquadratic* $r = 0.2$

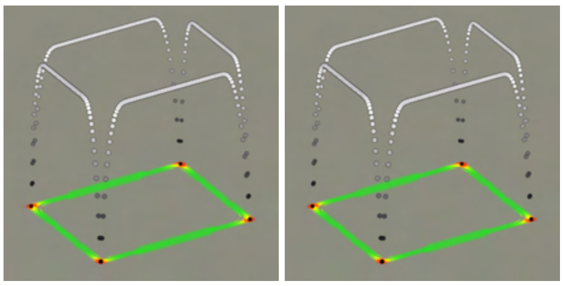

(b) *sphquadratic* $r = 0.25$, *sphquintic* $r = 0.3$

Figure 5: Different orders of the SPH kernel functions are compared, see Figure 1. *(a) sphcubic* and *shpquintic* using the same radius for the neighborhood. *(b) sphquadratic* and *shpquintic*, with adjusted neighborhood radius, have a similar result as the *sphcubic* (a)-left.

The influence of the integration scheme on the Eigenstreamline integration is demonstrated in Figure 6. The distribution tensor of a circular point distribution was computed using the *ssquare* weighting function. Tensor splats show the undirected Eigenvector, vector arrows show how the vector is directed within the internal vector representation. One Eigenstreamline is seeded downwards at the rightmost point of the circular point distribution and follows the undirected vectors. The top image shows Euler integration. Decreasing the step size would result in a more accurate integration. But, closing the gap of the integrated circle requires such a small step size, that the Runge Kutta method outperforms the Euler method. The 8^{th} order Runge Kutta method successfully closes the gap and reconstructs a circle from the circular point distribution, as shown in the bottom image. Also, a square-shaped point distribution was tested as shown in Figure 7. The length of a side is 10. Here, the influence of different weighting functions on the interpolation of the Eigenvector field was investigated. The distribution tensor was computed using the *ssquare* weighting function with $r = 2$. An Eigenstreamline is seeded downwards in the mid of the right edge. It follows the undirected vectors and flows around the corners of the rectangle. At each corner some error is introduced and the streamline is moving apart from

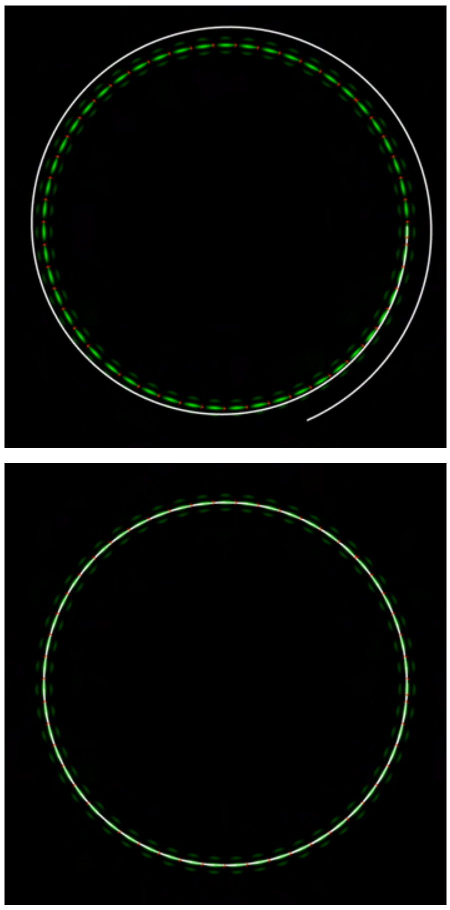


Figure 6: Comparison of different numerical integration schemes in a circular point distribution. One streamline (white) is seeded at the east-pole of the circle pointing southwards. Tensor splats and vector arrows illustrate the point distribution tensor and major Eigenvector. Note, that the Eigenvectors change orientation at north-east and south-west. *Top:* explicit Euler. *Bottom:* DOP853.

the original point distribution. Integration was done using the DOP853 method. Different weighting functions, mostly with $r = 1$, were tested for vector field interpolation. The length of the horizontal gap between the end and the start of the streamline was used as a measure for the integration error. Figure 8 shows the different errors in a bar diagram. The two best results were achieved using the *ssquare* and *average* weighting function.

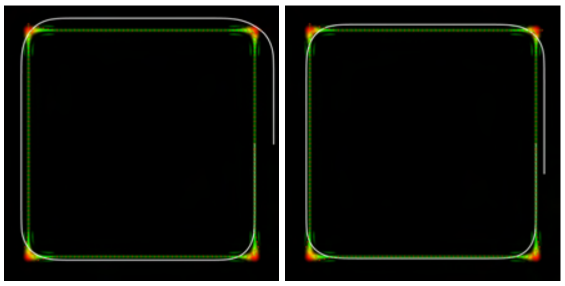


Figure 7: Comparison of Euler and DOP853 streamline integration on a square-shaped point distribution. Tensor splats and Eigenvectors are visualized besides the streamline (white) seeded downwards at the center of the right edge of the rectangle.

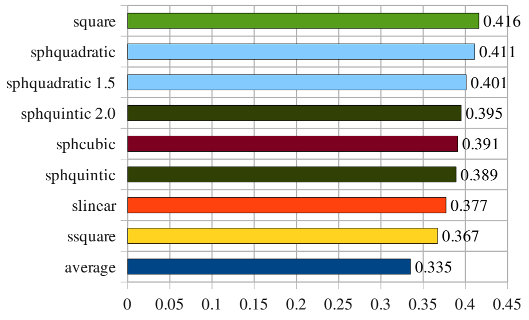


Figure 8: Comparison of errors in the square integration using different weighting functions for the vector interpolation. The weighting function for computing the tensor was *ssquare*, compare Figure 1. The values represent the horizontal distance between start and end point of the streamlines. The square's length is 10.0. The colors of the bars match the colors in Figure 1.

3 RESULTS

We used a dataset with circa eight million points covering a water basin close to the Danube in Austria. It was acquired by a Riegl's hydrographic laser scanner VQ-820G [22]. Figure 9 shows the point cloud colored by the *linearity* of a distribution tensor analysis. Here, we wanted to extract one power cable. The cable in the mid of the three lowest power cables suspended from the tall power pole. The white arrows mark the explicitly user-specified position and direction used as initial conditions of the streamline integration.

Different parameters and combinations of weighting functions for the tensor computation and the Eigenvector interpolation were investigated. The choice of a certain neighborhood radius and good weighting functions was crucial to successfully follow the 280 m long power cable. 41 parameter combinations were tested. For the tensor computation different radii $r = 0.5, r = 1.0$ and $r = 2.0$ and the weighting functions *average*, *slinear*, *ssquare* and the SPH kernels for the tensor were used. For the vector interpolation radii $r = 0.25, r = 0.5, r = 1.0, r = 2.0$ and $r = 3.0$ and all seven weighting functions were used. Figure 10 shows a view along the power cable, with a non optimal configuration. The Eigenstreamline is not following the cable to the end because it moves apart more than 1.0 m from the cable, resulting in an empty neighborhood during integration.

Best results were achieved by using the *ssquare* weighting with $r = 2.0$ for tensor computation and the *sphquintic* weighting with $r = 3.0$ for the vector interpolation. Results show that a more smooth weighting in the tensor computation and a more local interpolation weight are a good combination for reconstructing linear structures. Using the same weighting for tensor computation and vector interpolation did not work, see Figure 11 (b). The global error of the reconstruction at the end of the power cable is about 80 cm and needs to be further improved. The cable could only be followed using DOP853 integration. Explicit Euler failed to produce acceptable results. When comparing Figures $11(a)$ and $11(c)$ the global error is almost the same. The main difference is the local shape of the Eigenstreamline. A larger vector interpolation radius results in a smoother curve. Figure $11(c)$ shows the best reconstruction of the investigated technique and described parameters.

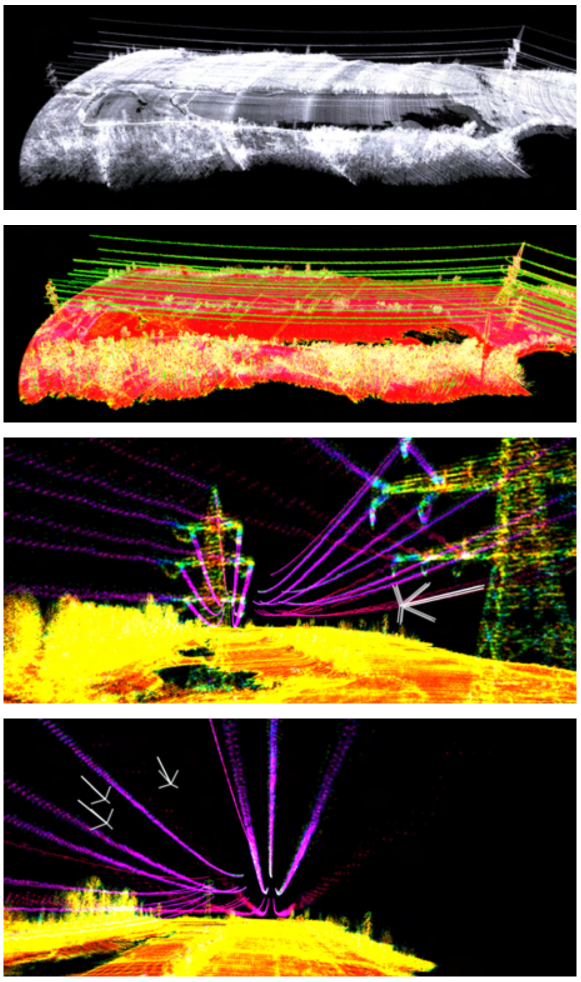


Figure 9: Overview of the LIDAR data set. The two upper images show the point cloud as points and as tensor splats (taken from [21]). In the two lower images points are colored by *linearity*. Three arrows mark the explicitly user-specified seeding points and directions of the streamline computation located at the mid lower power cable (magenta) of the larger power pole.

4 CONCLUSION

A new method of reconstructing power cables, or other linear structures in general, in point clouds was presented. The method employs the point distribution tensor as presented in previous work [21]. Different weighting functions for the tensor computation and the interpolation of the major Eigenvector field were implemented and compared. Streamline integration was verified on artificial test cases and applied to a LIDAR point cloud dataset acquired from actual observations. Finally, a power cable was reconstructed and visualized using this dataset.

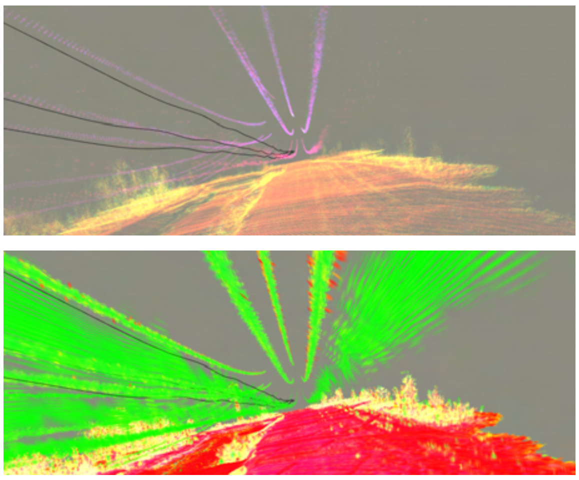
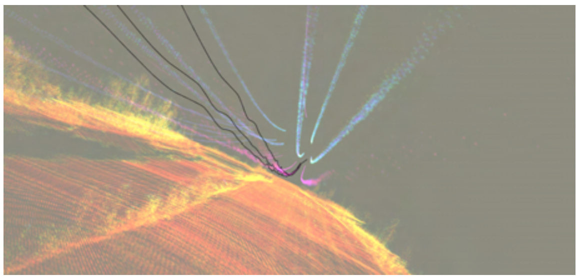
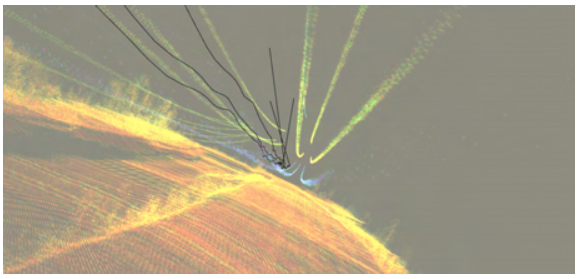


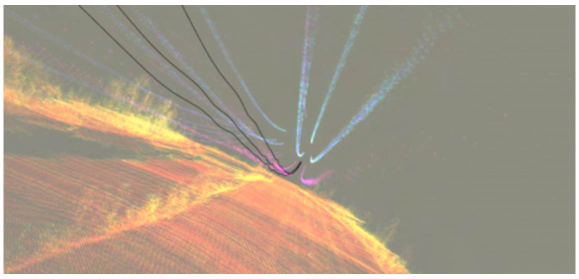
Figure 10: Power cable reconstruction via streamlines. The distribution tensor was computed using the *average* $r = 2.0$ weighting and the vector interpolation was done with the *ssquare* $r = 1.0$ weighting. *Top:* Points colored by *linearity*. *Bottom:* Tensor splats illustrate the distribution tensor. Streamlines are moving apart from the power cable and break before they can reconstruct the full 280 m of cable.



(a) Tensor: *ssquare r=2*, Vectorfield: *sphquintic r=1*



(b) Tensor: *sphquintic r=2*, Vectorfield: *sphquintic r=2*



(c) Tensor: *ssquare r=2*, Vectorfield: *sphquintic r=3*

Figure 11: Comparison of different parameters and weighting function combinations of the computation, finally resulted in a successfully reconstructed power cable. The LIDAR point cloud is colored by *linearity* of the distribution tensor. The three Eigenvector streamlines reconstruct a 280 m long cable.

5 FUTURE WORK

Other weighting functions for computing the tensor and doing the interpolation during the streamline integration need to be tested. Automatic determination of the optimal combination of weighting functions and also their parameters will be the goal of further investigations. Seeding points and directions for computing the streamlines need also to be chosen automatically, for example, by taking tensor properties into account. Following the major Eigenvector of points with high *planarity* or *sphericity* needs to be prevented during streamline integration. Finally, more datasets should be explored to stabilize the method. Furthermore, minor changes of the algorithm would enable streamline integration in datasets stemming from SPH simulations.

ACKNOWLEDGMENT

Many thanks to Frank Steinbacher for proving the LIDAR data. This work was supported by the Austrian Science Foundation FWF DK+ project *Computational Interdisciplinary Modeling* (W1227) and grant P19300. This research employed resources of the Center for Computation and Technology at Louisiana State University, which is supported by funding from the Louisiana legislature's Information Technology Initiative. This work was supported by the Austrian Ministry of Science BMWF as part of the UniInfrastrukturprogramm of the Forschungsplattform Scientific Computing at LFU Innsbruck.

REFERENCES

[1] W. Benger, H. Bartsch, H.-C. Hege, H. Kitzler, A. Shumilina, and A. Werner. Visualizing Neuronal Structures in the Human Brain via Diffusion Tensor MRI. *International Journal of Neuroscience*, 116(4):pp. 461–514, 2006.

[2] W. Benger, G. Ritter, and R. Heinzl. The Concepts of VISH. In 4^{th} *High-End Visualization Workshop, Obergurgl, Tyrol, Austria, June 18-21, 2007*, pages 26–39. Berlin, Lehmanns Media-LOB.de, 2007.

[3] W. Benger and M. Ritter. Using geometric algebra for visualizing integral curves. In E. M. S. Hitzer and V. Skala, editors, *GraVisMa 2010 - Computer Graphics, Vision and Mathematics for Scientific Computing*. Union Agency - Science Press, 2010.

[4] W. Benger, M. Ritter, S. Acharya, S. Roy, and F. Jijao. Fiberbundle-based visualization of a stir tank fluid. In 17^{th} *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 117–124, 2009.

[5] T. E. Conturo, N. F. Lori, T. S. Cull, E. Akbudak, A. Z. Snyder, J. S. Shimony, R. C. Mckinstry, H. Burton, and M. E. Raichle. Tracking neuronal

fiber pathways in the living human brain. *Proc Natl Acad Sci U S A*, 96(18):10422–10427, Aug. 1999.

[6] M. Descoteaux, L. Collins, and K. Siddiqi. A multi-scale geometric flow for segmenting vasculature in mri. In *In Medical Imaging Computing and Computer-Assisted Intervention*, pages 500–507, 2004.

[7] S. N. E. Hairer and G. Wanner. *Solving ordinary differential equations I, nonstiff problems, 2nd edition*. Springer Series in Computational Mathematics, Springer-Verlag, 1993.

[8] L. A. Fernandes and M. M. Oliveira. Real-time line detection through an improved hough transform voting scheme. *Pattern Recognition*, 41(1):299 – 314, 2008.

[9] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32 – 40, jan 1975.

[10] D. A. Fulk and D. W. Quinn. An analysis of 1D smoothed particle hydrodynamics kernels. *Journal of Computational Physics*, 126(1):165–180, 1996.

[11] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2001.

[12] D. K. Jones. Studying connections in the living human brain with diffusion mri. *Cortex*, 44(8):936 – 952, 2008.

[13] Y. Jwa, G. Sohn, and H. B. Kim. Automatic 3d powerline reconstruction using airborne lidar data. *IAPRS*, XXXVIII(2004):105–110, 2009.

[14] K. Kraus and N. Pfeifer. Determination of terrain models in wooded areas with airborne laser scanner data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 53(4):193 – 203, 1998.

[15] Z. Li, R. Walker, R. Hayward, and L. Mejias. Advances in vegetation management for power line corridor monitoring using aerial remote sensing techniques. In *Applied Robotics for the Power Industry (CARPI), 2010 1st International Conference on*, pages 1 –6, oct. 2010.

[16] Y. Liu, Z. Li, R. F. Hayward, R. A. Walker, and H. Jin. Classification of airborne lidar intensity data using statistical analysis and hough transform with application to power line corridors. In *Digital Image Computing : Techniques and Applications Conference (DICTA 2009)*, Melbourne, Victoria, December 2009. IEEE Computer Society.

[17] T. Melzer. Non-parametric segmentation of als point clouds using mean shift. *Journal of Applied Geodesy*, 1(3):159–170, 2007.

[18] T. Melzer and C. Briese. Extraction and modeling of power lines from als point clouds. In *Proceedings of 28th Workshop*, pages 47–54. Österreichische Computer Gesellschaft, 2004. talk: Austrian Association for Pattern Recognition (ÖAGM), Hagenberg; 2004-06-17 - 2004-06-18.

[19] S. Mills, M. Gerardo, Z. Li, J. Cai, R. F. Hayward, L. Mejias, and R. A. Walker. Evaluation of aerial remote sensing techniques for vegetation management in power line corridors. *IEEE Transactions on Geoscience and Remote Sensing*, October 2009.

[20] M. Persson, J. Solem, K. Markenroth, J. Svensson, and A. Heyden. Phase contrast mri segmentation using velocity and intensity. In R. Kimmel, N. Sochen, and J. Weickert, editors, *Scale Space and PDE Methods in Computer Vision*, volume 3459 of *Lecture Notes in Computer Science*, pages 119–130. Springer Berlin - Heidelberg, 2005.

[21] M. Ritter, W. Benger, B. Cosenza, K. Pullman, H. Moritsch, and W. Leimer. Visual data mining using the point distribution tensor. In *IARIS Workshop on Computer Vision and Computer Graphics - VisGra 2012*, Feb-Mar 2012.

[22] F. Steinbacher, M. Pfennigbauer, A. Ulrich, and M. Aufleger. Vermessung der Gewassersohle - aus der Luft - durch das Wasser. In *Wasserbau in Bewegung ... von der Statik zur Dynamik. Beitrage zum 15. Gemeinschaftssymposium der Wasserbau Institute TU München, TU Graz und ETH Zürich*, 2010.

[23] D. Weinstein, G. Kindlmann, and E. Lundberg. Tensorlines: advection-diffusion based propagation through diffusion tensor fields. In *Proceedings of the conference on Visualization '99: celebrating ten years*, VIS '99, pages 249–253, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[24] C. Westin, S. Peled, H. Gudbjartsson, R. Kikinis, and F. Jolesz. Geometrical diffusion measures for mri from tensor basis analysis. In *Proceedings of ISMRM, Fifth Meeting, Vancouver, Canada*, page 1742, Apr. 1997.

6 Discussion

The first part of this section is organized according to three related research directions: line reconstruction, tensor computation, and tensor visualization. The advantages and limitations of the methods developed and investigated in the scope of this thesis are discussed, organized per publication. Thereafter, the initially posed research questions are again compiled and addressed.

Line reconstruction was investigated in the scope of two publications. Initial ideas were introduced, prototyped, and tested in the scope of a concrete application scenario in Ritter and Benger (2012)*. Thereafter, the full, automated reconstruction framework was proposed in Ritter et al. (2021a)~. Core components of the method were analyzed in more detail, the robustness with respect to different types of noise was improved, and process automation was examined.

Ritter and Benger (2012)*: Here, a methodology of computing integral lines, originally from diffusion tensor magnetic resonance imaging, was transferred to the different scenario of point cloud reconstruction. Integral lines are computed based on a tensor neighborhood in meshless data. Streamlines follow the eigenvector field, successfully reconstructing curved lines in noisy point clouds. Note that the method operates fully in 3D. Besides a basic, but functional, symplectic Euler, a high order Runge-Kutta integrator improved the reconstruction errors in an artificial circle experiment. Further, weighting functions and parameters were tested and selected based on a real world example, which improved the reconstruction error. The combination of a quadratic weighting and an SPH kernel of order five yielded the smoothest, most plausible, result. A cable of a power line, of 280 m length, could be reconstructed in a real world LiDAR data set.

However, the average reconstruction error still was of about 80 cm in the LiDAR application. Generally, the integration of the eigenvector field in this setup resulted in reconstruction errors increasing with the line's length. Also, the reconstruction of the test case of a sampled rectangle introduced large errors at the corners. Generally, sharp corners are not well supported by the method. Moreover, the approach requires considerable user input. Starting points and an initial direction have to be provided, as well as global values for the tensor analysis radius and the interpolation radius have to be set. The integration cannot adjust to varying properties in the points cloud, or adjust to local dominant geometric features. Multiple continuous lines can be reconstructed, but they do not influence each other; i.e. lines do not intersect or merge. Also, no line crossings are supported in the method.

Ritter et al. (2021a)~: The mentioned line reconstruction was thoroughly extended to overcome shortcomings of the previous version. The final method is very robust at low to medium noise levels, and even gives reasonable results in the presence of very high noise. The method is especially robust to distribution noise and can overcome data holes. Two- and three-dimensional point clouds can be processed with the same approach. Generally, sharp corners, varying curvature, varying noise, crossings, and non-manifold lines are supported. Manual user input has been reduced to a few high-level control parameters. In general, three key parameters have to be specified; while three additional ones are automatically initialized, and can be fine tuned if required, see Section 3.3.1. For instance, dependent on the geometry and the noise intensity, it might become necessary to adjust a parameter that helps in overcoming holes. Furthermore, an automatic starting point detection has been proposed, which performed well in the real world scenarios of cable reconstruction between small houses. The employed LiDAR dataset was more challenging than the one used in the previous publication; sampling was sparser, non manifold lines were present, and more diverse objects were located closer to the cables, especially at the tree and the roof.

The number of (automatically detected) start points for streamline integration currently has to be selected by a user. The reconstruction result highly depends on that input; especially, when the geometry is complex or exhibits high curvature regions. Sharp corners can be reconstructed; but they are either smoothed out, if integrated over; or, alternatively they may retain their appearance, if they result from an intersection of line-lets. Line intersections are generally difficult to reconstruct in noisy data, as became evident in the Mikado test case. If independent lines pass by closely, and the jitter noise amplitude is within the range of the line distances, the streamline integration may jump over to a different line. Moreover, the reconstruction in such cases is also hampered due to other reasons; on the one hand, the angle-weighted streamline direction selection would suffer; on the other hand, the linearity in the multi scale measures decreases due to the proximity of the other sampled line geometry.

Several alternatives of choosing the integration direction, i.e. computing the streamline vector field, were analyzed; for instance, giving a stronger preference to following employed directions in previous steps. However, this led to other drawbacks, especially at corners and fine-grained geometric details. Nevertheless, note that the angular weighting term does put preference on a similar direction, since the angle of reference in the term is based on the previous integration direction.

The maximal radius to be used for the multi scale anaylsis is heuristically estimated based on a minimum distance statistic. Generally, this does not correlate well to a linear feature size and represents a rough, density dependent heuristic. Thus, it may be required to manually adjust the parameter for optimal results. In fact, to obtain the correct maximal radius for the multi scale analysis, a full multi scale analysis itself actually would have to be carried out in the first place. Analyzing the current largest multi scale features for a decision on stopping or continuing was investigated in Section 3.3.6. However, this was not robust to all parameter variations and geometries. The maximum radius has a high impact on the run time of the overall reconstruction. Thus, a heuristic estimation yielding as small maxima as possible was ultimately preferred. Nevertheless, in the LiDAR example this had to be increased manually. Regarding computation time, the multi scale analysis is the most costly part in the framework. Computation times scale non-linearly with the number of points. A large distance query with the maximal radius is carried out first, yielding (squared) distance sorted points. Then, tensors and shape factors are computed while shrinking the sorted neighborhood size. Thus, in terms of computational complexity, an additional factor of the neighborhood size has to be added. Various methods for performance optimization regarding the search were presented in Section 4.

Another concern relates to the streamline integration. Reconstructed lines can be extended for too long, before they reach the stopping distance criterion; especially, in the presence of outlier noise. In the pruning step, open ended streamlines are shrunk, until their end is close to a point in the cloud. Presence of outlier noise points nearby can thus prevent a tight pruning. Further, highly curved regions may only be reconstructed robustly if starting points are seeded on each side of the curved region, especially when high jitter noise is present. Generally, the method requires the presence of a sufficient number of samples in the point cloud. It does not work well for very sparse data; e.g. the circle example requires more than 10 point samples being present for a reconstruction. In such cases the integration step size may have to be manually reduced.

Neighborhood tensor computation and optimization was covered in three publications. Speeding up the computation via a GPU and CPU parallel approach is investigated in Grasso et al. (2015)*. In contrast to the per point view, a cluster representation is introduced and analyzed in Schiffner et al. (2014)*. Further, a technique operating in screen space is examined in Schiffner et al. (2013)*.

Grasso et al. (2015)*: By employing OpenCL, GPU and CPU parallel computation is enabled. Tensors are computed on point clouds in OpenCL kernels. Therefore, all compute devices can be utilized in a hardware system. Data buckets

– points within predefined bounding boxes – can be distributed independently on the available compute devices. Data is duplicated at the bucket's borders and enlarged by the tensor radius. Therefore, the buckets are independent and any coarse synchronization is prevented. The method scales very well for big data sets. Points are buckets sorted in the OpenCL kernels by hashing and bitonic sorting, which is matching well to the parallelism provided by GPU hardware. Finally, a speed up of more than 500 is achieved by the method, when comparing CPU to GPU. A speed up of 25 results over an earlier CPU implementation. This is remarkable, as the baseline implementation has already been using OpenMP parallelized range queries in a kd-tree structure to find point neighbors. It demonstrates the potential of optimizing algorithms with respect to memory accesses and hardware. The overhead, due to memory transfers of the points onto the GPU, as well as the sorted points and tensors from the GPU, is therefore negligible. Different hardware setups have successfully been utilized.

Still, pre-processing of the data is necessary. It requires a coarse spatial bucket sort, which is computationally cheap ($\mathcal{O}(n)$). The employed HDF5 data storage does a two pass approach for the coarse bucket sort. A first pass determines the buckets' sizes and a second pass fills the data. As stated above, data is additionally duplicated in ghost zones at the data bucket borders, which produces an overhead w.r.t. data space consumption. Data processing worked for the published test cases, but the algorithms ran into difficulties with much bigger data sets. Random crashes occurred when processing a large number of data buckets (≥ 3000), with each of them holding about 2 million points. Further, improvements on the implementation are necessary to overcome this shortcoming. Further, as with all shader codes, the computational parts have to be reorganized and written in a 'C-like' manner.

Schiffner et al. (2014)*: Vertex clustering is employed in an OpenGL compute shader to generate a reduced number of point representatives of a detailed point cloud. The grid based vertex clustering can be applied to any raw point data. The population count of cells is evened out by moving points to neighboring cells, while also taking the planarity shape factor into account. Thus, cells are optimized for higher planarity values. The tensor and its geometric measures are computed on the fly. Therefore, the grid borders adjust to the underlying geometry – becoming a curvilinear grid. GPU RAM is directly utilized to hold the raw data. The geometric representation employed for visualization is computed as late as possible. A standard data representation for a point was utilized by three floats resulting in twelve bytes. Many more data points could be loaded into memory when employing an encoding/decoding technique, by e.g. using simple Z-curve indices or more advances techniques. Recently, compression rates down to 0.2% of the original memory size was reported in Schwarz et al. (2019) and a bpp (bits per point) rate of 0.157 in Wang et al. (2020a). This could enable faster data loading and big data support. As the visual representation is separated from the underlying data, this allows to adjust to available hardware resources and to the visualization task. Generally, the clustering technique is a method at a "lower level" of visualization algorithms. It can be combined with different visual representatives, such as splats, surflets, and in-cell quads, and could be further grouped into cell clusters with higher level representatives.

The proposed redistribution of points exhibits some limitations, however. Since the move takes place between neighboring cells, several move steps might be necessary to adjust the grid well in inhomogeneously sampled regions of the point cloud. This is exemplified by the galaxy data set. In the current implementation, all data needs to be transferred to the GPU memory, before any visualization starts. If data sets are large, this might create an unacceptable overhead. Some heterogeneous systems have shared memory regions – accessible by CPU and GPU – which could counter this problem. If data becomes very large an out-of-core approach is necessary. However, as the visual representation is independent from the data the memory on GPU side can be split in two regions. One holding the representation for rendering and one for data streaming. Large data can be continuously streamed into the visual representation. Modern GPUs provide enough memory for this approach, e.g. up to $24\,GB$[1]. Further, the clustering not necessarily needs to be executed dynamically for each frame. Clusters could be cached and reused, and e.g. recomputed dependent on viewpoint changes.

Schiffner et al. (2013)*: Here, a multi layer depth buffer is utilized for tensor computation in the OpenGL rendering pipeline. Points are unprojected from screen space and the tensor's minor eigenvector utilized for shading. Employing the tensor computation allows for a more robust and smooth normal vector estimation. This is demonstrated by comparing to other screen space normal computations based on depth derivatives and cross products. As the method is executed in screen space at the end of the rendering pipeline, it is independent of any specialized pre-processing. Diagonal sampling schemes around the fragment (pixel) of interest yielded smoother shading results.

Generally, the limits of the technique make it applicable e.g. only for overview visualizations of data stemming from sensor devices. Several fragments (pixels) need to be covered for smooth results. It highly depends on the point density and point size of the rendering. Further, it turned out that the performance overhead for larger sampling counts is quite high. Twenty-five samples already take up to $60\,ms$ (including splatting). Nine samples required about $25\,ms$ and were necessary for a smooth result in the LiDAR overviews that could also capture the power

[1]e.g. NVIDIA RTX 3090, with 10496 shader units and 24 GB RAM. To compare more hardware specifications, see e.g. https://www.techpowerup.com/gpu-specs

line in the data. Still, higher sampling counts would improve the visual quality. A good trade-off has to be found for a successful application within the screen density boundaries.

Neighborhood tensor visualization concerns applications of scientific visualization, in contrast to the sections above where computational aspects were the main interest. This part is focused in the following two publications. In Ritter et al. (2012)* an existing tensor field visualization method is employed to illustrate a single scale tensor on artificial and application related point clouds. Ritter et al. (2021b)~ discusses the use of multi scale tensor images to reveal geometric features in point clouds.

Ritter et al. (2012)*: The point cloud neighborhood is encoded as a second order tensor (or weighted covariance), which is directly compatible to existing tensor field visualization techniques. The tensor splats method of Benger and Hege (2004) is chosen as it fits to the symmetric and positive definite tensor. A local geometric classification of linearity, planarity, and sphericity is successfully illustrated by the splats via coloring (green/red) and transparency. It is further exemplified on artificial test cases. Here, the coupling of transparency and sphericity reduces visual clutter, as shown in the filled cuboid example. Border surfaces and corners are highlighted, while inner splats are visually removed. The directional texture of the splats underlines the major eigenvector for linear regions, thus yielding visually perceivable lines in the point cloud. This is illustrated by the rectangle and LiDAR power cable example. The achieved visual line indication was the basis for the investigation of the geometric line reconstruction framework.

Nevertheless, the approach does not obviously improve the visualization of the contours in the coast line data set; only the cleared and better separated linear structures were highlighted. The neighborhood radius has to be set globally by a user; no initial parameters are set automatically. Thus, considerable user interaction is required to obtain good visual results. Further, the employed red to green color map may not be optimal for some viewers, e.g. in case of colorblindness. Other choices would be better distinguishable, as e.g. presented by Wong (2011). In the LiDAR data geometrically classified regions become quite homogeneous. Especially, the planar floor should provide more contrast for a better 3D readability. Some additional data, such as the LiDAR signal intensity or a shape analysis at a smaller radius could be included.

Ritter et al. (2021b)~ : This work relates to Ritter et al. (2021a)~ and was used in the reconstruction algorithm development to explore tensor properties on multiple scales. Therefore, the introduced visualization method allows to illustrate properties by showing the multi scale geometric measures. Weighting functions, centroids, and geometric features of the 3D data are such properties. The multi scale feature images (MSFIs) are dependent on these properties. Weighting functions have been explored w.r.t. smoothness and location of extrema in the MSFIs. The MSFIs also reveal noise amplitudes. On a per point basis the geometric neighborhood can be read off. The intensity as well as the radii of the features are illustrated by the colormap. Points can be visually classified as being part of a cable, roof, or tree; in the demonstrated LiDAR application.

Note that the shape factors are rotationally and scale invariant. Thus, these measures (and their visualization) could be employed for various tasks, such as automated clustering or tracking. The step size of the MSFI analysis is based on a minimum point distance statistic, and thus automatically adjusted.

However, single multi scale features of regions with large sphericity and planarity might not hold sufficient information for certain algorithms or purposes. Consider e.g. a point of a 3D crossing and a point within homogeneous 3D noise; both yield a high sphericity over multiple scales and thus are not distinguishable solely by comparing these features. Despite the multi scale view, other properties of neighbors may need to be taken into account as well, such as neighbor count or an angular density.

Reading information from the features and their visualizations requires training. The cable crossing in the LiDAR case cannot easily be recognized in the MSFI, without experience. Also note that the MSFI requires a 1D sorting of the points to become readable. Such a sorting is non trivial for larger and noisy point clouds. Locally, sorting along the major eigenvector can by carried out. Thus, no quick overall overview visualization is possible for a large dataset. In the proposed method, input is required for the sorting; a user manually specifies line probes via drawing.

The computational bottleneck is the maximum radius range query in the point cloud neighborhood, as described for Ritter et al. (2021a)~ above.

Research Questions

1. *Is an eigenvector streamline based method on top of the neighborhood tensor feasible to reconstruct lines in point cloud data?*
   In a first attempt, a method was developed that could be used to successfully integrate along the major eigenvectors of a point distribution tensor field allowing to visually follow linear structures in point clouds. A distance error and a visual inspection was used to evaluate the feasibility as a quality measure. The initial approach had limitations; the reconstruction error accumulates during integration; sharp corners are problematic; the radius for the tensor computation is constant. Using a high order integrator can help to reduce the error accumulation, but does not improve the results at sharp corners (Ritter et al., 2012)*. Nevertheless, this initial method could be extended to improve the results. Therefore, quality error measures were extended by average dis-

tance and a completeness. In the reconstruction method, the radius for the eigenvector is chosen dynamically. The neighborhood is adjusted based on multi scale shape factors of the tensor, opting for radii with dominant linear features. Secondly, also based on the multi scale feature analysis, starting points are detected and multiple bidirectional lines integrated. Lines merge and intersect during a breadth first iteration. Thirdly, an angular direction term augments the tensor's eigenvector field. This improves the behavior at small geometric features, where a small radius linearity minimum vanished due to jitter noise amplitude at similar scale, or the second maximum becomes dominant at very high curvature, such as a corner of less than about 60° where the eigenvector will point in the central axis of the corner and, thus, lead the integration outwards in corner direction. This is illustrated at the inner part of the ears of the *bunny* in Ritter et al. (2021a)~. Enforcing the small scale maximum did not solve this behavior as it led to follow outlier clusters in higher jitter noise cases. Tuning on automated test cases, on weighting functions, and on scoring parameters, resulted in a method for robust line reconstruction. In the end, the second order tensor via streamline integration for reconstruction is able to create comparable error measures with respect to a recent line reconstruction method (Ohrhallinger and Wimmer, 2019). It requires a dynamic radius (bandwidth) choice for the eigenvector in combination with a angular weighted direction.

2. *How sensitive is the line reconstruction algorithm to noise? How is it made robust?*
   The extended algorithm is quite stable against distribution and outlier noise. However, jitter noise is the most difficult type. Reconstruction of basic geometries started to fail at noise values of about 6% to 10% of the their bounding box size (see Section 6.5.2 in Ritter et al. (2021a)~). Increasing the number of starting points can remedy this in general. However, in some cases with very strong jitter noise, non-merging, parallel lines can occur. Further, using weighting functions in the tensor computation, as well as in the interpolation of eigenvectors stabilizes against noise. The introduced Fermi-Dirac I) weighting function showed improved reconstruction performance. Further, an automatically adjusted step size for the integration helped in presence of distribution noise.

3. *What is the effect of using different centroids and weighting functions in the tensor computation?*
   The shape of the multi scale shape factor graphs are influenced by both these components. Different weighting functions yield different radii for minima and maxima of the graphs. Also, the values of the extrema and the overall smoothness of the graphs are affected. The Fermi-Dirac I) weighting function creates

extrema at rather small radii, which is desirable in terms of computation time of the neighborhood searches. It also yields smooth shape factor graphs, which is beneficial in combination with a geometric median or weighted mean as centroid. The centroids have a further influence on the graphs, as they filter noise; except for the point distribution tensor. Also, the eigenvector directions of the tensor are smoothed out, as the centroid is dynamically shifted to the center of the neighborhood. This has an influence on the score functions; Equations (7) and (9) in Ritter et al. (2021a)~. Figure 3.14 illustrates the noise filtering effect of different centroids on start point selection. The PDT leads to selected points closer to the ground truth data as it prefers centers of noisy clusters. The PDT is also employed for the noise rate analysis were no noise should be compensated. In contrast, noise filtering has a positive effect on the integration direction, thus the PDT performs worst in the parameter runs, see Section 6.5 in Ritter et al. (2021a)~. For the reconstruction, the weighted mean, in combination with a quadratic inverse, showed the best performance. Other differences can be seen in the scale space at larger radii, e.g. different strategies need to be employed to detect a maximum analysis radius dependent on the centroid, see Figure 3.15. The geometric median is proposed for multi scale feature image visualization, because it creates homogeneous and smooth regions are sharper edges in the multi scale space, see Figure 7 in Ritter et al. (2021b)~.

4. *What geometric features and details can be reconstructed?*
   Ten artificial test geometries have been set up for development and evaluation of the reconstruction from noisy point cloud data, including features such as: straight and curved lines, varying curvature, 2D and 3D corners (up to 60 °), crossings, and out of plane curvature. All could be successfully reconstructed, even with added noise. High curvature regions are challenging and can lead to an outwards integration following the corner's axis direction. Further, closely parallel lines may merge to one central line and nearby skew lines may connect. The respective test cases for this behavior are the *wave*, *bunny*, and *Mikado* in Ritter et al. (2021a)~. These cases allow only small amounts of jitter noise of about 6% to 10% of the bounding box size. Further, the method requires a proper sampling rate, e.g. more than 10 samples are required for a circle.

5. *Can user dependent control parameters be removed to allow a fully automated reconstruction process?*
   While the developed framework comprises several parameters, only a handful are left for a user to adjust; with up to ten in borderline cases. The main user parameters are (sorted by relevance): number of starting points, maximum number of integration iterations (thus, length of reconstruction), data hole size to overcome, pruning distance, merging distance for approaching lines,

and maximum multi scale analysis radius. A key element of this thesis is the automatic steering of parts of the framework. Analysis of the multi scale geometric measures permits selection of optimal neighborhood radii (and direction) as well as integration starting points. A minimum distance analysis is a second component, especially, to set length related parameters; such as the integration step size or the increment of the multi scale radius. However, it was not possible to fully automate the framework. Moreover, the adjustment of the involved parameters require some experience by the user. Finally, it may not be preferable to remove all user control. An semi automated line reconstruction assistant tool would allow to better adjust to specific use cases.

6. *What is the computational bottleneck and can it be optimized or tackled by GPU computation?*
   The key bottleneck is the multi scale tensor computation. The involved neighborhood search is costly. In the proposed reconstruction implementation a CPU-based octree is employed for acceleration. The maximum radius is used as a distance query and iteratively shrunk for the centroid, tensor, and shape factor computation. This can be offloaded to the GPU, but in the multi scale case, this might not scale as well because of the larger radius. Still, the approach is directly applicable and extensible to several multi scale iterations in the tensor computation on the GPU side. A further extension could be to perform the computation not on a per point basis but on subspaces or clusters.
7. *Can a tensor visualization method stemming from spacetime curvature be employed to enhance point clouds visually by highlighting its neighborhood properties?*
   Tensor shape factors and eigenvectors could successfully be utilized for point cloud visualizations. Linear directions highlighted by textures yielded visual indications of linear structure in point clouds. Moreover, the dominant shape factor of the multi scale analysis can be selected to color point clouds. However, the benefit of this visualization over other methods has not be evaluated in user studies. The methods targeted in Schiffner et al. (2013)* and Schiffner et al. (2014)* allow for a quick inspection of raw point cloud data including the geometric measures. A proper highlighting of linear structures, as demonstrated in Ritter et al. (2012)*, enables an interactive region of interest selection by a user. In those regions, the computationally expensive automated line reconstruction could be executed to provide a semi automated line reconstruction tool for large point cloud data.

7 Conclusion

Starting from a tensor view of a point cloud neighborhood, a second order tensor was formulated, including a distance weighting. Geometric measures have been computed to describe the shape of a representative ellipsoid forming a barycentric coordinate system. First, the tensor as well as its measures have been explored on a single radius scale. It was observed that this is not sufficient in describing local geometric features in point sets, as they often are present at different scales. Thus, the tensor and the geometric measures were extended for multiple scales. Multi scale geometric measures were then analyzed further. They were employed in a reconstruction scenario and allowed to automatically identify 'good' candidate points to start the reconstruction process. Further, they enabled to find optimal radii to define a vector field – based on the tensor major eigenvector – for a stream line based reconstruction.

Moreover, the multi scale geometric measures revealed the influence of employing different radial distance weighting functions as well the influence of using different points of reference (centroids) in the tensor computation. Twenty-nine non parametric as well as one two-parametric weighting functions were evaluated for tensor computation. The two parameters of the proposed Fermi-Dirac weighting were optimized. Four types of centroids were evaluated: mean, weighted mean, geometric median, and weighted geometric median. The weighted mean employing a quadratic inverse was proposed for line reconstruction, as it performed better for line crossings, and the geometric median was proposed for multi scale feature visualization.

Computational aspects have been analyzed. The neighborhood search was identified as a major bottleneck for the tensor computation, independent of its specific variant. Search trees were developed to speed up the range queries. A first CPU parallel kd-tree implementation was replaced by an octree, which improved performance for the employed data sets. Other methods were implemented and investigated utilizing the GPU, also for different kinds of usage scenarios. A screen space based tensor computation was integrated into the OpenGL rendering pipeline to estimate planarity and a normal vector in real time when viewing raw point cloud data. An OpenGL compute shader was developed to enable real time vertex clustering based on a uniform grid. This was extended by a computationally cheap step to even out grid cell population and, thus, morphing the uniform to a 'virtual' curvi linear grid. For each grid cell then a geometric surface representative is created and rendered. Here, a surface reconstruction is computed in real time from raw point clouds for rendering. The planarity geometric measure is employed in the grid repopulation steps. Another method was introduced based on a uniform grid. Here, spatial hash keys were computed per point and then keys sorted. A range query could then directly find the involved grid cell and neighbors along with the cell registered points. Importantly, a GPU compatible sorting algorithm was chosen: bitonic sorting. The tensor computation itself was also included into the GPU codes. OpenCL was chosen as technology to enable compute kernels on heterogeneous devices – utilizing the maximum possible computational power of a shared memory and/or distributed system. The heterogeneous approach reached a speed up of more than 500 on a single workstation with one strong GPU compared to the first kd-tree based CPU parallel implementation.

Further, the single scale tensor was utilized for visual exploration of particle and point cloud data sets. A tensor visualization technique originally developed for diffusion tensor magnetic resonance imaging was now driven by the neighborhood tensor to enhance the visualization. It was compared to simple color mapped point visualizations in data sets stemming from astrophysics simulations on cosmological evolution. The tensor visualization helped to better identify the geometric distribution of points in a wind tunnel SPH simulation, where galaxy clusters move through the inter cluster medium gas. Linearly distributed point regions could be enhanced, which also correlated to a higher star birth rate. More visual exploration was carried out on geoscientific data, i.e. on airborne LiDAR scans. Here, the visualization clearly highlighted linear and planar regions; high voltage power line cables and the ground floor were visually enhanced. As an extension to the power line visualization a reconstruction algorithm was prototyped inspired by fiber tracking of the MRI diffusion tensor. The visually extracted power cable was now reconstructed as a geometric line.

The line reconstruction algorithms were revised and extended by the multi scale tensor. The new algorithm removed user input that was required before and also served as a test bed for in depth analysis, e.g., on weighting function, centroids, numerical integration schemes, interpolation methods, vector field variants, automated radius selection, and noise rate estimation. Several test geometries were set up to challenge the method by distinct geometric features of lines, such as constant and varying curvature, sharp corners, and crossing. Additionally, point disturbances were introduce to model uncertainties in data capturing by different types of noise and data holes. Error measures were developed to evaluate algorithmic performance systematically. The method identifies starting points and grows bidirectional line lets following linear structure of a points set. Line lets merge and intersect to form the finally reconstructed line. The results were compared to recent publications in line reconstruction. They are comparable and the introduced method was up to thirty times faster, depending on the point set to be reconstructed. Further, our method works in 3D.

Finally, a tool was created to enable the visual exploration of the multi scale geometric measures in arbitrary point clouds. A visualization technique of color mapped multi scale feature images is introduced. Such images can be inspected interactively on line probes as well as graph plots of the geometric measures on selected points. The images are also utilized to present the influence of the aforementioned tensor parameters visually (weighting functions, centroids, incremental choice). They provide a rotation invariant multi scale feature for any point of a point cloud or probe location.

In the discussion the advantages and the limits of the methods for each publication were highlighted and finally the research questions were addressed and shortly discussed. Ultimately, a line reconstruction method based on a second order multi scale tensor has been developed and investigated thoroughly. It is comparable to recent other developments and is suited for other use cases and has different properties, such as support for higher noise rates, crossings and 3D lines. The bottleneck of the tensor computation has been optimized by grid hashing, vertex clustering, and screen space sampling, also, utilizing GPU hardware. Finally, visualization techniques have been proposed to enhance point cloud visualization built on the neighborhood tensor's (multi scale) properties by coloring, shading, and textured splatting to either highlight regions of high linearity, enable quick overview visualization of raw data, or inspect multi scale features.

7.1 Future Work

Future work could be directed at improving and further exploring the geometric line reconstruction. Alternative stopping criteria of the integration are required in the LiDAR application, as shown in Ritter et al. (2021a)~. Instead of an eigenvector streamline a geodesic could be computed and, thus, the neighborhood tensor interpreted as a spacetime curvature. Besides solving a second order differential equation, derivatives of the tensor field need to be computed, which are required for the involved Christoffel symbols. Currently, the biggest performance loss is located in the multi scale tensor computation and especially computing the geometric median. Other algorithms to compute or estimate the median can be evaluated, e.g. using methods presented in Cohen et al. (2016). Also, as mentioned at the end of Section 3.3.6, a better estimation of a maximum stopping radius for the multi scale analysis could improve performance. It need not be chosen globally for all points and could be extended to locally different maximum radii. Further speed up could be achieved by removing the angular weighted direction computation during stream line integration. Maybe this could also be shifted into the parallelized pre-computation step; e.g. by pre-evaluating for a limited number of quantized reference directions, or by pre-evaluating with reference to the neighborhood tensor's eigenvector.

Further work can also be spent on the weighting functions. Axial symmetric and non isotrop functions could be employed instead of homogeneous radial ones to favor certain directions. This would require a two pass computation, but could e.g. further improve pre-selected directions, and allow to use a faster centroid variant. Also, instead of growing the radius of the weighting kernel, e.g. the Fermi-Dirac kernel could be grown by the parameter m. This could improve the multi scale graphs as the decreasing kernel region is not scaled. Scaling of other kernels could also be prevented by applying it to the other ring of the neighborhood.

The line reconstruction could be extended to surface reconstruction, e.g. by growing radial surflets or wave fronts from high planarity locations, or by connecting detected edge lines or cross sections. Here, also instead of employing a streamline of the eigenvector a geodesic of the tensor could be investigated.

The whole multi scale pre-computation can be offloaded onto the GPU; e.g. employing OpenGL compute shaders, OpenCL, or SYCL[1]. This will influence algorithmic choices to optimize the GPU's memory accesses. Besides faster computations time, this would enable the multi scale analysis and line reconstruction to be used as a basis for other algorithms.

It might make sense to combine the overall method of streamline integration with a sphere fitting or regression method such as employed in Pauly et al. (2003) or Ohrhallinger and Wimmer (2019). The streamlines could overcome regions with higher noise rates and be used as region and bandwidth selection for regression. The topology and curvature of the connected streamlines can help to find segments for a piece wise regression, allowing for non manifold geometries.

Broader directions to continue are on applying the multi scale tensor and geometric measures on other classes of point cloud algorithms such as classification, point clustering, segmentation, or 3D tracking. Identifying robust locations for tracking would need to be compared to existing methods. Further, the multi scale graphs would fit into machine learning frameworks as feature vectors.

The work on real time point cloud clustering and drawing cell representatives, as carried out in Schiffner et al. (2014)*, could be extended by geometric representatives for linear and volumetric shape factors, such as connecting and blending between cones (linear), planes (planar), and spheres (spherical). The blending of such different objects, may provide a shaded geometrical abstract representation

[1] www.khronos.org/sycl

of a detailed point clouds and could be enriched by color coding to encode accumulated data per cluster cell, such as density or signal intensities. Such an approach could provide an adaptive level of detail: structures viewed from far distance are simplified by e.g. cones and become separated into detailed parts when zooming in. At close ups, finally, splatted points could be blended in to inspect data at a fine grain level.

As indicated in the discussion, point cloud compression could enable support for larger point clouds, or could speed up memory transfers. A method with fast decompression would need to be identified and can then be directly included in the compute or vertex shader.

The noise rate n_R as introduced in Ritter et al. (2021a)~ could be further improved. The 3D crossing with no noise still results in a low noise rate, due to the utilized average of all sphericity minima. Maybe employing statistical quantiles or including the multi scale radius would be feasible options as well as investigating shape factor differences over neighbors. Finally, more investigation should be spent on the 4^{th} order multi scale tensor. Geometric measure dependent weighting and the correct tensor analysis could be an alternative strategy for the here employed multi scale graph based analysis. Especially, for the start point detection and radius scale selection. Also, the representation by the 15 components would be more compact than the multi scale second order tensors and measures.

8 Acknowledgements

I thank Sabine Schindler and Werner Benger for getting me interested and started at the doctoral school "computationally interdisciplinary modelling" for a PhD project.

I want to thank Günter Hofstetter for his trust, support, and integration into his research group and teaching team, although the direction of specialization drifted away from the original ideas. Special thanks to my final supervisor Matthias Harders for offering a place in his research group. Further, for his guidance, help, collaboration, precise corrections, suggestions, and for asking the right questions. Both carry a real spirit of science, always favor high quality standards, while being supportive in a productive environment.

Special thanks to Daniel Schiffner for extensive discussions, creative exchanges, fruitful collaborations, the possibility of an external research visit, corrections, and an always critical view on things. Special thanks to my brother Georg Ritter who was always an anchor for support. He stepped in to help over difficult content, such as, showing that there is no guarantee for correctness in publications.

Thanks to my fellows at Airborne Hydromapping and, especially, the CEO Frank Steinbacher. The combination of Astrophysics and airborne LiDAR scanning originally started the research and finally yielded a story-line for the thesis. Thank you for the support, collaborations, exchanges, and for providing data for analysis. Thanks to the fellows of the doctoral school. The interdisciplinary setup was widening the horizon and enabled interesting collaborations and publications with experts in different fields of science, among them: Alexander Kaiser, Ivan Grasso, Dominik Steinhauser, David Unteregger, Vincent de Groof, Antti Koselka, and Stefan Huber. Thanks to the fellows at the research group of structural mechanics. The scientific/teaching related exchanges, blackboard sessions, and relaxed working environment was a great experience; among them Benjamin Fuchs, Peter Gamnitzer, Nikolaus Fleischhacker, and Alexander Dummer. Finally, many thanks to my colleagues at the interactive graphics and simulation group. Especially, in the last very intense phase including corona lock-downs, distance teaching, and final writing up. I enjoyed the scientific exchanges with Fernando Zorilla, the discussions with Anatoli Sianov, and the collaboration with Evgeni

Zuenko.

Further, thanks to Nikolaus Rauch for his teaching support, help on content searches, many discussions, and his critical views. Last but not least, thanks to Stefan Spiss for pushing the final writing/coding/evaluation sprints to its limits. The coffee meetings with both were an anchor in the curfew times.

Biggest thanks go to my wonderful wife Sonja Herdlinger-Ritter with her loupe-view on details, corrections, her support, coffee celebrating work-sessions as well as inspiring walks, and carrying all costs of such a project.

Bibliography

[Abbasloo et al. 2016] ABBASLOO, A. ; WIENS, V. ; HERMANN, M. ; SCHULTZ, T.: Visualizing Tensor Normal Distributions at Multiple Levels of Detail. In: *IEEE Transactions on Visualization and Computer Graphics* 22 (2016), Nr. 1, p. 975–984

[Alexa et al. 2001] ALEXA, Marc ; BEHR, Johannes ; COHEN-OR, Daniel ; FLEISHMAN, Shachar ; LEVIN, David ; SILVA, Claudio T.: Point Set Surfaces. In: *Proceedings of the Conference on Visualization '01*. USA : IEEE Computer Society, 2001

[Basser 1998] BASSER, Peter: Fiber-tractography via diffusion tensor MRI. In: *Proc International Society for Magnetic Resonance in Medicine* (1998)

[Basser et al. 2000] BASSER, Peter ; PAJEVIC, Sinisa ; PIERPAOLI, Carlo ; DUDA, Jeffrey ; ALDROUBI, Akram: In vivo fiber tractography using DT-MRI data. In: *Magnetic Resonance in Medicine* 44 (2000), p. 625–632

[Batcher 1968] BATCHER, Kenneth: Sorting Networks and Their Applications. In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, Association for Computing Machinery, 1968, p. 307–314

[Beck and Sabach 2015] BECK, Amir ; SABACH, Shoham: Weiszfeld's Method: Old and New Results. In: *Journal of Optimization Theory and Applications* 164 (2015), January, Nr. 1, p. 1–40

[Bender et al. 2014] BENDER, Jan ; MÜLLER, Matthias ; OTADUY, Miguel A. ; TESCHNER, Matthias ; MACKLIN, Miles: A Survey on Position-Based Simulation Methods in Computer Graphics. In: *Comput. Graph. Forum* (2014), p. 228–251

[Benger 2004] BENGER, Werner: *Visualization of General Relativistic Tensor Fields via a Fiber Bundle Data Model*, FU Berlin, Ph.D. thesis, 2004

[Benger et al. 2006] BENGER, Werner ; BARTSCH, Hauke ; HEGE, Hans-Christian ; KITZLER, Hagen ; SHUMILINA, Anna ; WERNER, Annett: Visualizing Neural Structures in the Human Brain via Diffusion Tensor MRI. In: *Intern Journ of Neuroscience* 116 (2006), Nr. 4, p. 461–514

[Benger et al. 2012] BENGER, Werner ; HAIDER, Markus ; HÖLLER, Harald ; STEINHAUSER, Dominik ; STÖCKL, Josef ; COSENZA, Biagio ; RITTER, Marcel:

Visualization Methods for Numerical Astrophysics. In: *Astrophysics.* IntechOpen, 2012, Chap. 12

[Benger and Hege 2004] Benger, Werner ; Hege, Hans-Christian: Tensor Splats. In: *Conference on Visualization and Data Analysis 2004*, Proceedings of SPIE Vol. #5295, 2004, p. 151–162

[Benger et al. 2004] Benger, Werner ; Ritter, Georg ; Heinzl, René: The Concepts of VISH. In: *In 4th High-End Visualization Workshop*, 2004, p. 26–39

[Bergeaud and Mallat 1998] Bergeaud, François ; Mallat, Stéphane: Matching Pursuit of Images. In: *Signal and Image Representation in Combined Spaces* Volumne 7. Academic Press, 1998, p. 285 – 300

[Berger et al. 2013] Berger, Matthew ; Levine, Joshua A. ; Nonato, Luis G. ; Taubin, Gabriel ; Silva, Claudio T.: A Benchmark for Surface Reconstruction. In: *ACM Transactions on Graphics* 32 (2013), p. 20:1–20:17

[Berkmann and Caelli 1994] Berkmann, Jens ; Caelli, Terry: Computation of Surface Geometry and Segmentation Using Covariance Techniques. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16 (1994), Nr. 11, p. 1114–1116

[Brackbill et al. 1988] Brackbill, Jeremiah ; Kothe, Douglas ; Ruppel, Hans: Flip: A low-dissipation, particle-in-cell method for fluid flow. In: *Computer Phys. Comm.* 48 (1988), p. 25 – 38

[Brannon 2018] Brannon, Rebecca: Voigt and Mandel components. In: *Rotation, Reflection, and Frame Changes.* IOP Publishing, 2018, p. 26–1 to 26–20

[Burt et al. 2009] Burt, James ; Barber, Gerald ; Rigby, David: *Elementary Statistics for Geographers.* 3rd. The Guilford Press, 2009

[Cohen et al. 2016] Cohen, Michael ; Lee, Yin T. ; Miller, Gary ; Pachocki, Jakub ; Sidford, Aaron: *Geometric Median in Nearly Linear Time.* p. 9–21. In: *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, 2016

[Daropoulos et al. 2020] Daropoulos, Viktor ; Augustin, Matthias ; Weickert, Joachim: *Sparse Inpainting with Smoothed Particle Hydrodynamics.* 2020. – arXiv – 2011.11289

[Devore et al. 2013] Devore, Ronald ; Petrova, Guergana ; Hielsberg, Matthew ; Owens, Luke ; Clack, Billy: Processing Terrain Point Cloud Data. In: *SIAM Journ. of Imaging Sciences* 6 (2013), p. 1–31

[Dormand and Prince 1980] Dormand, John ; Prince, Pete: A family of embedded Runge-Kutta formulae. In: *Journal of Computational and Applied Mathematics* 6 (1980), Nr. 1, p. 19 – 26

[Foix et al. 2011] Foix, Sergi ; Alenya, Guillem ; Torras, Carme: Lock-in Time-of-Flight Cameras: A Survey. In: *IEEE Sensors Jour.* 11 (2011), Nr. 9, p. 1917ff [glm 2017] Groovounet, Christophe: *OpenGL Mathematics*. 2017. – https://glm.g-truc.net

[Grasso et al. 2015] Grasso, Ivan ; Ritter, Marcel ; Cosenza, Biagio ; Benger, Werner ; Hofstetter, Günter ; Fahringer, Thomas: Point Distribution Tensor Computation on Heterogeneous Systems. In: *Procedia Computer Science* 51 (2015), p. 160 – 169

[Guennebaud and Gross 2007] Guennebaud, Gaël ; Gross, Markus: Algebraic Point Set Surfaces. 26 (2007), Nr. 3

[Hairer et al. 1993] Hairer, Ernst ; Nørsett, Syvert ; Wanner, Gerhard: *Solving Ordinary Differential Equations*. 2nd. Springer Series in Comput. Math., 1993

[HDF5 2020 accessed 2020] HDF5: *HDF5*. accessed 2020. – https://www.hdfgroup.org

[Hoppe et al. 1992] Hoppe, Hugues ; DeRose, Tony ; Duchamp, Tom ; McDonald, John ; Stuetzle, Werner: Surface Reconstruction from Unorganized Points. In: *SIGGRAPH Comput. Graph.* 26 (1992), juli, Nr. 2, p. 71–78. – ISSN 0097-8930

[Huang et al. 2013] Huang, Hui ; Wu, Shihao ; Cohen-Or, Daniel ; Gong, Minglun ; Zhang, Hao ; Li, Guiqing ; Chen, Baoquan: L1-Medial Skeleton of Point Cloud. In: *ACM Trans. Graph.* 32 (2013), Nr. 4

[Husselmann and Hawick 2012] Husselmann, Alwyn ; Hawick, Ken: Spatial Data Structures, Sorting and GPU Parallelism for Situated-agent Simulation and Visualisation. In: *Proc. Int. Conf. on Modelling, Simulation and Visualization Methods*, 2012

[imgui 2017] Ocornut, Omar: *imgui*. 2017. – https://github.com/ocornut/imgui

[Kolb and Whisaw 2015] Kolb, Bryan ; Whisaw, Ian: *Fundamentals of Human Neuropsychology*. 7^{th}. Worth Publishers, Inc., 2015

[Kuhn and Kuenne 1962] Kuhn, Harold ; Kuenne, Robert: An Efficient Algorithm for the Numerical Solution of the Generalized Weber Problem in Spatial Economixs. In: *Journal of Regional Science* 4 (1962), Nr. 2, p. 21–33

[Lejemble et al. 2020] Lejemble, Thibault ; Mura, Claudio ; Barthe, Loïc ; Mellado, Nicolas: Persistence Analysis of Multi-scale Planar Structure Graph in Point Clouds. In: *Computer Graphics Forum* 39 (2020), p. 35–50

[Lin et al. 2014] Lin, Chao-Hung ; Chen, Jyun-Yuan ; Su, Po-Lin ; Chen, Chung-Hao: Eigen-feature analysis of weighted covariance matrices for LiDAR point cloud classification. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 94 (2014)

[Lin et al. 2005] Lin, Hongwei ; Chen, Wei ; Wang, Guojin: Curve reconstruction based on an interval B-spline curve. In: *The Visual Computer* 21 (2005),

p. 418–427

[Liu et al. 2020] Liu, Li ; Ouyang, Wanli ; Wang, Xiaogang ; Fieguth, Paul ; Chen, Jie ; Liu, Xinwang ; Pietikäinen, Matti: Deep Learning for Generic Object Detection: A Survey. In: *International Journal of Computer Vision* 128 (2020), p. 261–318

[Lu et al. 2017] Lu, Xuequan ; Chen, Wenzhi ; Schaefer, Scott: Robust mesh denoising via vertex pre-filtering and L1-median normal filtering. In: *Computer Aided Geometric Design* 54 (2017), p. 49 – 60

[Macklin and Müller 2013] Macklin, Miles ; Müller, Matthias: Position Based Fluids. In: *ACM Transactions on Graphics* 32 (2013), Nr. 4

[Mathews et al. 2010] Mathews, Edwin ; Benger, Werner ; Ritter, Marcel: Implementation of an Algorithm for Approximating the Curvature Tensor on a Triangular Surface Mesh in the Vish Environment. In: *Proc. of the* 6^th *High-End Vis. Workshop*, 2010

[McIvor and Valkenburg 1997] McIvor, Alan ; Valkenburg, Robert: A comparison of local surface geometry estimation methods. In: *Machine Vision and Applications* 10 (1997), Nr. 1, p. 17–26

[Mellado et al. 2012] Mellado, Nicolas ; Guennebaud, Gaël ; Barla, Pascal ; Reuter, Patrick ; Schlick, Christophe: Growing Least Squares for the Analysis of Manifolds in Scale-Space. In: *Computer Graphics Forum* 31 (2012), p. 1691–1701

[meson 2017] Pakkanen, Jussi: *The Meson Build System.* 2017. – http://mesonbuild.com

[Monaghan 1988] Monaghan, Joseph: An introduction to SPH. In: *Computer Physics Communications* 48 (1988), Nr. 1, p. 89 – 96

[Monaghan 2005] Monaghan, Joseph: Smoothed particle hydrodynamics. In: *Reports on Progress in Physics* 68 (2005), Nr. 8, p. 1703

[Natale et al. 2010] Natale, Donald ; Baran, Matthew ; Tutwiler, Richard: Point cloud processing strategies for noise filtering, structural segmentation, and meshing of ground-based 3D Flash LIDAR images. In: *2010 IEEE* 39^{th} *Applied Imagery Pattern Recognition Workshop (AIPR)*, Oct 2010, p. 1–8

[Ohrhallinger and Wimmer 2018] Ohrhallinger, Stefan ; Wimmer, Michael: StretchDenoise: Parametric Curve Reconstruction with Guarantees by Separating Connectivity from Residual Uncertainty of Samples. In: *Proc. of the* 26^{th} *Pacific Conf. on Computer Graphics and Applications: Short Papers*, Eurographics Association, 2018, p. 1–4

[Ohrhallinger and Wimmer 2019] Ohrhallinger, Stefan ; Wimmer, Michael: FitConnect: Connecting Noisy 2D Samples by Fitted Neighborhoods. In: *Computer Graphics Forum* 38 (2019), Nr. 1, p. 126–137

[Ohtake et al. 2005] Ohtake, Yutaka ; Belyaev, Alexander ; Seidel, Hans-

Peter: An Integrating Approach to Meshing Scattered Point Data. In: *Proc. of the 2005 ACM Symposium on Solid and Physical Modeling*, 2005, p. 61–69

[Öztürk and Hasirci 2013] Öztürk, Mehmet ; Hasirci, Zeynep: A novel method for thinning branching noisy point clouds. In: *Turkish Journal of Electrical Engineering & Computer Sciences* 21 (2013), p. 2239–2258

[Pahl and Damrath 2000] Pahl, Peter ; Damrath, Rudolf: *Mathematische Grundlagen der Ingenieurinformatik*. Springer-Verlag Berlind Heidelberg, 2000

[Pauly et al. 2002] Pauly, Mark ; Gross, Markus ; Kobbelt, Leif: Efficient Simplification of Point-Sampled Surfaces. In: *Proceedings of the Conference on Visualization '02*, 2002 (VIS '02), p. 163–170

[Pauly et al. 2003] Pauly, Mark ; Keiser, Richard ; Gross, Markus: Multi-scale Feature Extraction on Point-Sampled Surfaces. In: *Computer Graphics Forum* 22 (2003), Nr. 3, p. 281–289

[Pauly et al. 2006] Pauly, Mark ; Kobbelt, Leif ; Gross, Markus: Point-Based Multiscale Surface Representation. In: *ACM Transactions on Graphics* 25 (2006), p. 177–193

[Penner 2020] Penner, Robert: *Easing Functions*. 2020. – http://www.robertpenner.com/easing

[Peters et al. 2011] Peters, Hagen ; Schulz-Hildebrandt, Ole ; Luttenberger, Norbert: Fast in-place, comparison-based sorting with CUDA: a study with bitonic sort. In: *Concurrency and Computation: Practice and Experience* 23 (2011), Nr. 7, p. 681–693

[Philsu and Hyoungseok 2010] Philsu, Kim ; Hyoungseok, Kim: Point Ordering with Natural Distance Based on Brownian Motion. In: *Mathematical Problems in Engineering* (2010)

[Pourahmadi 2013] Pourahmadi, Mohsen: *Covariance Matrices in High-Dimensional Covariance Estimation*. p. 45–96. In: *High-Dimensional Covariance Estimation*, John Wiley & Sons, Ltd, 2013. – ISBN 9781118573617

[Qi et al. 2017] Qi, Charles ; Yi, Li ; Su, Hao ; Guibas, Leonidas: PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017, p. 5105–5114

[Reif 1965] Reif, Frederick: *Fundamentals of Statistical and Thermal Physics*. Tokyo : McGraw Hill, 1965

[Ritter 2009] Ritter, Marcel: Introduction to HDF5 and F5 / Center for Computation and Technology, Lousiana State University. 2009 (CCT-TR-2009-13). – Research Report

[Ritter 2011] Ritter, Marcel: *Geodesics in Numerical Space Times*. Verlag Dr. Müller, 2011

[Ritter 2021] Ritter, Marcel: *MssfReconstruct*. 2021. –

github.com/gileoo/MssfReconstruct

[Ritter and Benger 2010] Ritter, Marcel ; Benger, Werner: Visualizing Coordinate Acceleration and Christoffel Symbols. In: *IADIS Computer Graphics, Visualization, Computer Vision and Image Processing 2010 (CGVCVIP 2010)* (2010)

[Ritter and Benger 2012] Ritter, Marcel ; Benger, Werner: Reconstructing Power Cables From LIDAR Data Using Eigenvector Streamlines of the Point Distribution Tensor Field. In: *Journal of WSCG* 20 (2012), Nr. 3, p. 223–230

[Ritter et al. 2012] Ritter, Marcel ; Benger, Werner ; Cosenza, Biagio ; Pullman, Keera ; Moritsch, Hans ; Leimer, Wolfgang: Visual Data Mining Using the Point Distribution Tensor. In: *VisGra - ICONS 2012*, IARIA, 2012, p. 218–222

[Scharstein et al. 2001] Scharstein, Daniel ; Szeliski, Richard ; Zabih, Ramin: A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In: *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, 2001

[Schiffner et al. 2013] Schiffner, Daniel ; Ritter, Marcel ; Benger, Werner: Fast Normal Approximation of Point Clouds in Screen Space. In: *WSCG 2013 Conf. on Computer Graphics, Visualization and Computer Vision Communication Paper Proceedings*, 2013

[Schiffner et al. 2014] Schiffner, Daniel ; Ritter, Marcel ; Steinhauser, Dominik ; Benger, Werner: Using Curvilinear Grids to Redistribute Cluster Cells for Large Point Clouds. In: *Proceedings of SIGRAD 2014, Visual Computing*, 2014

[Schiffner et al. 2015] Schiffner, Daniel ; Stockhausen, Claudia ; Ritter, Marcel: Surfaces for Point Clouds using Non-Uniform Grids on the GPU. In: *WSCG 2015 Conference on Computer Graphics, Visualization and Computer Vision Communication Paper Proceedings*, 2015

[Schürmann 2014] Schürmann, Tim: Meson - a new build system. In: *Linux Magazine* 166 (2014), Sep. – http://www.linux-magazine.com/Issues/2014/166/Meson-Build-System

[Schwarz et al. 2019] Schwarz, Sebastian ; Preda, Marius ; Baroncini, Vittorio ; Budagavi, Madhukar ; Cesar, Pablo ; Chou, Philip ; Cohen, Robert ; Krivokuća, Maja ; Lasserre, Sebastien ; Li, Zhu ; Llach, Joan ; Mammou, Khaled ; Mekuria, Rufael ; Nakagami, Ohji ; Siahaan, Ernestasia ; Tabatabai, Ali ; Tourapis, Alexis ; Zakharchenko, Vladyslav: Emerging MPEG Standards for Point Cloud Compression. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9 (2019), Nr. 1, p. 133–148

[semver 2017] Preston-Werner, Tom: *Semantic Versioning 2.0.0.* 2017. – http://semver.org

[Sheshappanavar and Kambhamettu 2020] Sheshappanavar, Shivanand ; Kambhamettu, Chandra: A Novel Local Geometry Capture in Pointnet++ for 3D Classification. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, p. 1059–1068

[Skala 2012] Skala, Vaclav: Radial Basis Functions for High Dimensional Visualization. In: *VisGra - ICONS 2012*, IARIA, 2012, p. 218–222

[Skala 2017] Skala, Vaclav: RBF Interpolation with CSRBF of Large Data Sets. In: *Procedia Computer Science* 108 (2017), p. 2433 – 2437. – International Conference on Computational Science, ICCS 2017

[Solenthaler and Pajarola 2009] Solenthaler, Barbara. ; Pajarola, Renato: Predictive-Corrective Incompressible SPH. In: *ACM SIGGRAPH 2009 Papers*, 2009

[Steinbacher et al. 2012] Steinbacher, Frank ; Pfennigbauer, Martin ; Aufleger, Markus ; Ullrich, Andreas: High Resolution Airborne Shallow Water Mapping. In: *ISPRS-Intern. Arch. of the Photogr., Remote Sens. and Spatial Inform. Sciences* XXXIX-B1 (2012), p. 55–60

[Sulsky et al. 1994] Sulsky, Deborah ; Chen, Zhen ; Schreyer, Howard: A particle method for history-dependent materials. In: *Computer Methods in Applied Mechanics and Engineering* 118 (1994), Nr. 1, p. 179 – 196

[Syme et al. 2015] Syme, Don ; Granicz, Adam ; Cisternino, Antonio: *Expert F# 4.0*. 4th. New York : Apress Media, 2015. – ISBN 978-1484207413

[Taubin 1995] Taubin, Gabriel: Estimating the Tensor of Curvature of a Surface from a Polyhedral Approximation. In: *Proceedings of the Fifth International Conference on Computer Vision*, IEEE Computer Society, 1995 (ICCV '95)

[Toth and Jóźkòw 2016] Toth, Charles ; Jóźkòw, Grzegorz: Remote sensing platforms and sensors: A survey. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 115 (2016), p. 22ff

[Wang et al. 2020a] Wang, Jianqiang ; Ding, Dandan ; Li, Zhu ; Ma, Zhan: *Multiscale Point Cloud Geometry Compression*. 2020. – arXiv – 2011.03799

[Wang et al. 2020b] Wang, Xinlei ; Qiu, Yuxing ; Slattery, Stuart R. ; Fang, Yu ; Li, Minchen ; Zhu, Song-Chun ; Zhu, Yixin ; Tang, Min ; Manocha, Dinesh ; Jiang, Chenfanfu: A Massively Parallel and Scalable Multi-GPU Material Point Method. In: *ACM Transactions on Graphics* 39 (2020)

[Wei Shen et al. 2015] Wei Shen ; Xinggang Wang ; Yan Wang ; Xiang Bai ; Zhang, Z.: DeepContour: A deep convolutional feature learned by positive-sharing loss for contour detection. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, p. 3982–3991

[Weinmann et al. 2015] Weinmann, Martin ; Jutzi, Boris ; Hinz, Stefan ; Mallet, Clément: Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers. In: *ISPRS Journal of Pho-*

togrammetry and Remote Sensing 105 (2015), p. 286 – 304

[Weiszfeld 1937] WEISZFELD, Endre: Sur le point pour lequel la Somme des distances de n points donnés est minimum. In: *Tohoku Mathematical Journal, First Series* 43 (1937), p. 355–386

[Wendland 1995] WENDLAND, Holger: Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. In: *Adv Comput Math 4* (1995), p. 389–396

[Wendland 2005] WENDLAND, Holger: *Scattered Data Approximation.* Cambridge University Press, 2005

[Westin et al. 2002] WESTIN, Carl-Fredrik ; MAIER, Stephan ; MAMATA, Hatsuho ; NABAVI, Arya ; JOLESZ, Ferenc ; KIKINIS, Ron: Processing and visualization for diffusion tensor MRI. In: *Medical Image Analysis* 6 (2002), 2002 Jun, Nr. 2, p. 93–108

[Westin et al. 1997] WESTIN, Carl-Fredrik ; PELED, Sharon ; GUDBJARTSSON, Hákon ; KIKINIS, Ron ; JOLESZ, Ferenc: Geometrical diffusion measures for MRI from tensor basis analysis. In: *Proceedings of ISMRM, 5th Meeting, Canada*, Apr 1997, p. 1742

[Willmott 2011] WILLMOTT, Andrew: Rapid Simplification of Multi-Attribute Meshes. In: *Proc. of the ACM SIGGRAPH Symposium on High Performance Graphics*, 2011

[Wong 2011] WONG, Bang: Points of view: Color blindness. In: *Nature Methods* 8 (2011), Jun, Nr. 6, p. 441–441

[Zeng et al. 2008] ZENG, Yong ; NGUYEN, Thanh A. ; YAN, Baiquan ; LI, Shuren: A distance-based parameter free algorithm for curve reconstruction. In: *Computer-Aided Design* 40 (2008), Nr. 2, p. 210 – 222

[Zhu and Bridson 2005] ZHU, Yongning ; BRIDSON, Robert: Animating Sand as a Fluid. In: *ACM Transactions on Graphics* 24 (2005), Nr. 3, p. 965–972

A Appendix

A.1 Visual Analysis Tool Implementation

Implementation: A stand alone tool was implemented to analyze point cloud neighborhoods, to study their properties and to develop and test methods. Many, algorithms were prototyped in F#, due to its superior expressiveness, succinctness, debugging environments, and more agile refactoring capabilities. Refer to e.g. Syme et al. (2015) for an intermediate access to the language. Algorithms were translated into a flat C++ design using libraries for OpenGL development: the graphics framework gf[1], ImGUI (imgui (2017)), and glm (glm (2017)). The meson build system (meson (2017), Schürmann (2014)), was used allowing for package management.

The wrapper tool provided by meson was extended to simplify the configuration process and a *semver* based dependency management introduced, see semver (2017). A specified package is added automatically as a subproject into the current build project by calling the introduced argument command line tool taking two arguments. This enables an easy to use but powerful multi platform C++ source package management. Configuration and compilation stages are very efficient with minimal maintenance overhead. Up to now, no standard C++ package manager was accepted by the community. A few exist, such as Conan[2], Pacman[3], or Chocolatey[4]. They have not been standardized by the C++ community. NuGet[5] is the de-facto standard in the *dotnet* ecosystem, as it is supported by Microsoft. Package management removes the need to rebuild libraries in the own configuration, and saves compile and work time for developers. Note that the meson approach supports package management per project as well as allows to define own packages very easily. No system wide management is required. Commonly, virtual machines are set up to separate the development environment per project

[1] by Daniel Schiffner

[2] https://conan.io

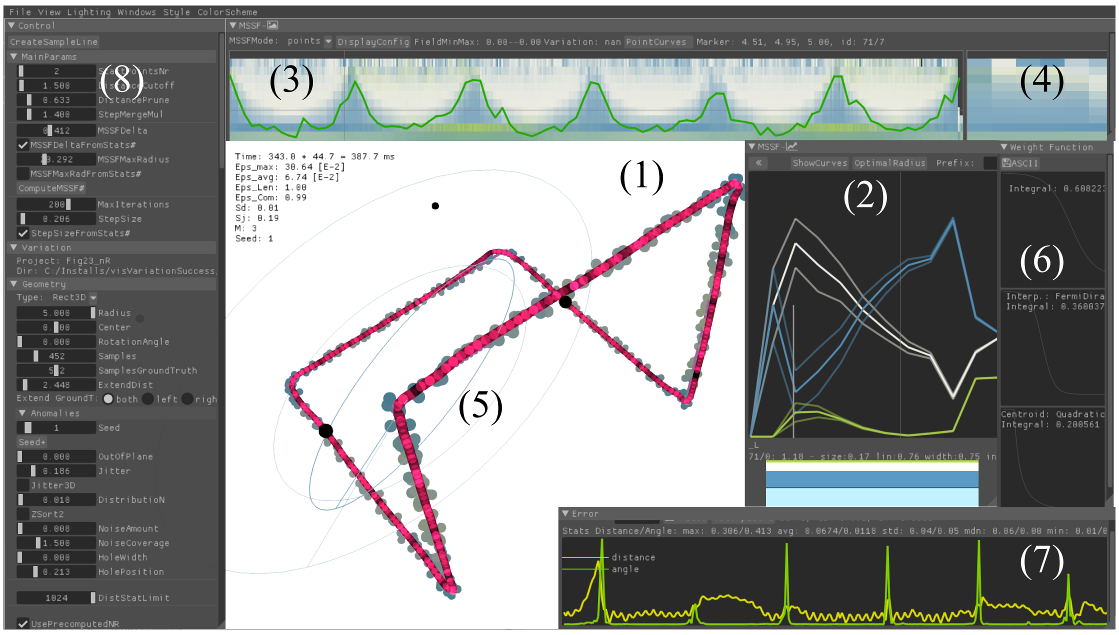
[3] https://www.archlinux.org/pacman

[4] https://docs.chocolatey.org

[5] https://www.nuget.org

if system wide package management is utilized.

Analysis Tool: The interactive tool consists of six GUI sections and a 3D display. Figure A.1 shows the overall appearance and its main components. The components are interlinked for interactive analyzes; e.g. when selecting a certain point of the reconstruction a tensor marker is moved there showing the tensor's ellipsoid and radius, the related multi scale geometric measure plots are updated in the multi scale geometric measure graph plot, and the radius marked as a vertical grey line there and in the multi scale feature image (MSFI). Either MSFIs on the reconstructed line, or on the original PC can be illustrated. The MSFI component includes a close up around the selected position either sorted by the (reconstructed) curve parameter or projection on the major eigenvector in a sole point cloud neighborhood.



```
2        StartPointsNr
1.500    DistanceCutoff
0.633    DistancePrune
1.400    StepMergeMul
0.412    MSSFDelta
✓ MSSFDeltaFromStats#
20.292   MSSFMaxRadius
MSSFMaxRadFromStats#
200      MaxIterations
0.206    StepSize
✓ StepSizeFromStats#
Project: Fig23_nR
Dir: C:/Installs/visVariationSuccess
Type: Rect3D
5.000    Radius
0.000    Center
0.000    RotationAngle
452      Samples
512      SamplesGroundTruth
2.448    ExtendDist
Extend GroundT: ● both ○ left ○ righ
1        Seed
0.000    OutOfPlane
0.186    Jitter
Jitter3D
0.010    DistributioN
ZSort2
0.008    NoiseAmount
1.500    NoiseCoverage
0.000    HoleWidth
0.213    HolePosition
1024     DistStatLimit
✓ UsePrecomputedNR
```

```
MSSFMode: points  FieldMinMax: 0.00--0.00 Variation: nan  Marker: 4.51, 4.95, 5.00, id: 71/7
```

```
Time: 343.0 + 44.7 = 387.7 ms
Eps_max: 30.64 [E-2]
Eps_avg: 6.74 [E-2]
Eps_Len: 1.00
Eps_Com: 0.99
Sd: 0.01
Sj: 0.19
M: 3
Seed: 1
```

```
_L
71/0: 1.18 - size:0.17 lin:0.76 width:0.75 in
```

```
Integral: 0.608223
Interp.: FermiDira
Integral: 0.360037
Centroid: Quadratic
Integral: 0.200561
```

```
Stats Distance/Angle: max: 0.306/0.413 avg: 0.0674/0.0118 std: 0.04/0.05 mdn: 0.06/0.00 min: 0.01/0
```

Figure A.1: Visual analysis tool developed in C++, OpenGL, and ImGui. It shows a line reconstruction setup of a noisy rectangle. The main components are labeled: (1) 3D view, (2) multi scale geometric measure graphs, (3) multi scale feature image (MSFI), (4) MSFI neighborhood, (5) tensor marker, (6) weight functions, (7) error graphs, and (8) control section.

Figure A.1 shows the tool. The main components are labeled: (1) 3D view of the geometry and the line. (2) marks the multi scale geometric measure graphs for one selected centroid: linearity (light grey), planarity (blue), and sphericity (green). Lighter lines mark the standard deviation of multiple multi scale geometric measures in a neighborhood. The thin vertical grey line marks a selected radius of the PDT for analysis and relates to the size of the tensor marker (5) and the horizontal line in (3). The thick vertical lines illustrates the local maximum score, see Equation (9) of Ritter et al. (2021a)~, and positions of local linearity maxima. (3) the multi scale feature image shows all geometric measures by color map. The vertical line marks a selected point and the horizontal line a selected radius. Scalar data on the points is drawn as overlay; including the linearity sum as green line. (4) is a MSFI neighborhood at the selected point sorted along its major eigenvector. (5) a tensor marker highlights the selected centroid by tensor ellipsoid and radius. (6) weight functions are illustrated used in the covariance, interpolation, and centroid computation. (7) Error graphs along the reconstruction line show distance (yellow) and angular (green) errors according to the error metrics to the original undistorted geometry. (8) is the main control section for reconstruction parameters. The reduced set is gathered in the *Main Params* section.

Further, the control section is grouped into: variation, geometry creation, tensor computation, and integration. Sliders, number inputs, drop-downs, checkboxes, radio-buttons, and buttons are used to map key value pairs of names and values to GUI elements. These control values are saved and loaded as ASCII files. Besides the predefined parametric geometries, custom geometries can be loaded as '.OBJ'[6] files, either from disk or a web server. The 3D view can be exported as vector graphics '.SVG'[7] files – either in an orthographic projection from the top view, or as a perspective projection. In the latter, point sizes are computed and lines are exported as outlined skeletons, such that the line width can vary dependent on depth. The tool has been split in two: one related to reconstruction and one specialized for multi scale image exploration. Here, a user can create a line probe in the scene and inspect the multi scale images. Windows binaries and the source code of the tools as well as a decoupled library for the core are available open source at Ritter (2021).

Further visualization options: the tool implements more options than presented or described in the publications. Tensor ellipsoids can be shown at each point, with various options and parameters. A normal vector is used for shading the points. Therefore, the minor eigenvector of the neighborhood tensor is employed. As the tensor is pre-computed at various scales, the scale can be selected interactively. The shading's smoothness can be adjusted by this.

[6] `http://paulbourke.net/dataformats/obj/`
[7] `https://www.w3.org/TR/SVG11`

A.2 Extended Parameters

Expert (E) and *fixed* (F) parameters of the line reconstruction method of Ritter et al. (2021a)~ (see Section 3.3.1). Note that the equation and figure labels below refer to the publication.

Name	**Type**	**Description**
StepSize	E	Define integration step size; default is automatic; the median of the radii of all neighborhoods holding six neighbors, halved: $h = \Delta_{\mathrm{mdn}} 0.5$. Several statistically motivated alternatives have been tested and all others have failed when inducing distribution noise even on the simplest geometries.
MSSFDelta	E	Increment from one multi scale radius to the next; default is automatic; the median of the radii of all neighborhoods holding six neighbors: Δ_{mdn}.
MSSFExponential	E	Toggle between exponential or linear multi scale.
PruneLines	E	Cut off orphaned stumps; independent of line to point distance.
DistStatLimit	E	Limit computation of distance statistics, e.g. six-neighborhood size, to a random subset of N points; use all when set to 0; default: 1024.
StPtsMetricLim	E	Start point score threshold; candidates are selected if greater; default 0.0.
StartP-PointsLimit	E	Limit consideration of start point candidates to a random subset of N points; use all when set to 0; default 0.
StartP-SmallNSkip	E	Ignore start points candidates if there are less that x points in a small neigborhood ($N_1 < x$); default 6.
ManualStartPts	E	Give a list of specific indices used as start points.
OptMinNeighsNr	F	Minimum number of neighbors to consider for best multi scale index (or radius) selection. A neighborhood should have at least some neighbors to yield a reasonable result; too small values decrease quality in noisy regions; fixed to 7 based on observing many noisy test cases.
AdaDirBlendType	F	Select adaptive directional blend method; 1 of 7; e.g. based on linearity integral, sum, linearity value, linearity relative index, and combinations; Equation (10).

AngularWeight	F	Select angular weighting function in Equation (5); 1 of 30; fixed to Fermi-Dirac (II).
CentroidType	F	Switch type of centroid: direct point ($\rightarrow$ point distribution tensor), mean ($\rightarrow$ PCA), weighted mean, weighted median, or geometric median; fixed to the latter.
CentroidWeight	F	Weighting function for weighted centroid variants; 1 of 30; unused for geometric median, mean, or PDT.
DirectionMode	F	Select variant of directional mixing during integration, computing $\mathbf{d_t}$; 1 of 9; interpolated eigenvector (IV), interpolated tensor, IV and angular weighted direction (AWD), ID and angular&distance weighted direction, tensor line (tend), pre-computed closest multi scale eigenvector and AWD, interpolated new computed multi-scale eigenvector and AWD, or interpolated new clamped eigenvector and AWD; fixed to the latter. The choice was made based on the test geometries, shown in Figures 11, 12 and 18 (additional tests involved more extreme noise rates).
IntegrScheme	F	Numerical scheme used for streamline integration; 1 of 6; RK32, RK38, RK4, Merson4(5), Fehlberg4(5), and DormandPrince5(4). Test were carried mainly on the circle geometry and the final error after one round of integration evaluated. The scheme order has an influence, high schemes did improve result. But it turned out, that other parameters, such as the weighing function applied in the tensor computation has a much higher influence. Also, when data becomes noisy the (minor) optimization via a higher scheme is not relevant anymore. Thus, the fastest scheme RK32 was chosen for optimized computation time.
InterpWeight	F	Weighting kernel for interpolating tensors or eigenvector values in between points of the point cloud; 1 of 30; the choice is based on extensive parameters runs (1.35 million cases) on the circle and the rectangle reconstruction. The two geometries were chosen as they capture sharp corners and curved regions, and provided a simple final error computation; sampling rates and noise levels were varied; fixed to Fermi-Dirac (II). $\rightarrow$ Same experiment used for TensorWeight selection below:

TensorWeight	F	Weighting function for the tensor computation; 1 of 30; fixed to Fermi-Dirac (I).
MedianVersion	F	Switch geometric median implementation variants; weighted median or non weighted; fixed to the latter. The geometric median performance was evaluated by investigating the properties of the multi scale linearity plots. Optimizing for clear local maxima and minima.
StartPointsIntegral	F	Select method of considering start point candidates based on the linearity graph analysis: linearity sum, graph space integral, and world space integral; basically, these are variants on computing $\mathcal{A}$ and $\mathcal{A}_j$ in Equations (9) and (10); fixed to linearity sum.
IPolRadFromStats	F	Derive interpolation radius from minimal distance statistics; fixed to enabled.
InterpRadius	F	Radius used for interpolation of tensors/eigenvectors; a reasonable number of neighbors should be covered. The factor was chosen especially based on enabling noisy test cases; fixed to $\Delta_{mdn}2.5$.
MSSFFullIntegral	F	Compute the integral $\mathcal{A}$ up to local minimum with the largest radius or up to the first minimum from the back (the last minimum) of the graph; set to the latter.
MSSFWDNR	F	Term factor used in Equation (9); unused, fixed to 0.0.
MSSFWIntegral	F	Factor of the normalized integral in Equation (9); fixed to 1.0.
MSSFWLin	F	Factor on the linearity value at the local maximum used in Equation (9); fixed to 1.0.
MSSFWLinDiff	F	Factor of the maximum difference of the maximum and the linearity value before $(k-1)$ and after $(k+1)$ the local maximum; unused, fixed to 0.0.
MSSFWMinMaxDiff	F	Difference of the minimum and the maximum of the interval used; δ in Equation. (9); fixed to -0.5.
MSSFWNeighRatio	F	Factor of a neighbor to radius relation used; unused, fixed to 0.0.
MSSFWSecond	F	Factor to up vote the second local maximum; unused, fixed to 0.0.
MSSFWSize	F	Factor for the radius or index in Equation (9). Small good maxima are preferred; thus, this is set to a negative value; fixed to -1.0.

MSSFWidth	F	Factor for interval width, replaced by MSSFWIntegral; unused, fixed to 0.0.
StPtsPowDist	F	Exponent for distance in Equation (7). Start point candidates should have large distances in between; fixed to 1.0.
StPtsPowLMaxLin	F	Exponent for the linearity value at the maximum. High is preferred; fixed to 4.0.
StPtsPowLastLin	F	Exponent for the linearity value at the largest radius; unused, fixed to 0.0.
StPtsPowLinSum	F	Exponent for the overall linearity integral ($\mathcal{A}$). Prefer high values; fixed to 4.0.
StPtsPowNRadRat	F	Exponent for the neighbors to radius relation; unused, fixed to 0.0.
StPtsPowNeighs	F	Exponent for number of neighbors in N_1. Prefer high number; fixed to 1.0.
AFermiMue/ AFermiTemp	F	Fermi-Dirac parameters μ and T to weight angular directions; fixed to (0.05, 0.35).
IFermiMue/ IFermiTemp	F	Fermi-Dirac parameters to weight the mesh free kernel based tensor/eigenvector interpolation; fixed to (0.05, 0.35).
TFermiMue/ TFermiTemp	F	Fermi-Dirac parameters to weight the tensor (or covariance) computation. ω in Equation (1); fixed to (0.1, 0.6).

A.3 Closest Point on Skew Lines

The closest point on a skew line is used instead of 3D line intersections for the line reconstruction algorithm in Ritter et al. (2021a)~. The closest point was computed as follows. Two lines are given in parameter form with A the origin, D a direction, i denoting the first, and j the second line. The closest point on j to i is determined. Note that the 'connecting' line is perpendicular to both lines:

$$\begin{aligned}
P &= A_i + \lambda_i D_i \\
Q &= A_j + \lambda_j D_j \\
P_q &= A_j - A_i + \lambda_j D_j - \lambda_i D_i \\
A &= A_j - A_i
\end{aligned}$$

$$\begin{aligned}
P_q D_i &= 0 \text{ (!) perpendicular} \\
P_q D_j &= 0 \text{ (!) perpendicular}
\end{aligned}$$

$$\begin{aligned}
0 &= AD_i + D_j D_i \lambda_j - D_i D_i \lambda_i \\
0 &= AD_j + D_j D_j \lambda_j - D_i D_j \lambda_i \\
a_i &= AD_i \\
a_p &= AD_j \\
d_{ii} &= D_i D_i \\
d_{ji} &= D_j D_i \\
d_{jj} &= D_j D_j
\end{aligned}$$

$$\begin{aligned}
0 &= a_i + d_{ji}\lambda_j - d_{ii}\lambda_i \qquad | \, /d_{ii} \\
0 &= a_j + d_{jj}\lambda_j - d_{ij}\lambda_i \qquad | \, /d_{ij}
\end{aligned}$$

$$\begin{aligned}
0 &= \frac{a_i}{d_{ii}} + \frac{d_{ji}}{d_{ii}}\lambda_j - \lambda_i \\
0 &= \frac{a_p}{d_{ij}} + \frac{d_{jj}}{d_{ij}}\lambda_j - \lambda_i \qquad |(-1)+
\end{aligned}$$

$$0 = \frac{a_i}{d_{ii}} - \frac{a_j}{d_{ij}} + \frac{d_{ji}}{d_{ii}}\lambda_j - \frac{d_{jj}}{d_{ij}}\lambda_j$$

$$\lambda_j = \frac{\frac{a_j}{d_{ij}} - \frac{a_i}{d_{ii}}}{\frac{d_{ji}}{d_{ii}} - \frac{d_{jj}}{d_{ij}}}.$$

A.4 Weighting Functions

The following table collects all weighting functions $\omega(x)$ employed for tensor computation, mesh free interpolation, angular interpolation, and centroid computation (see Section 3). Note that x was normalized to $[0.0, 1.0]$, by dividing with the neighborhood radius.

Nr	**Name**	**Equation** $\omega(x)$
1	const	1
2	linear	$1 - x$
3	linear inverse	$\min(1, \frac{0.1}{x})$
4	quadratic	$1 - x^2$
5	quadratic inv.	$\min(1, \frac{0.1^2}{x^2})$
6	cubic	$1 - x^3$
7	cubic inverse	$\min(1, \frac{0.1^3}{x^3})$
8	sph-3	$\begin{cases} 1 - \frac{3}{2}4x^2 + \frac{3}{4}8x^3, & 1 \leq 2x < 1 \\ \frac{1}{4}(2 - 2x)^3, & 2x \leq 2 \end{cases}$
9	sph-4	$\begin{cases} (\frac{5}{2} - \frac{5}{2}x)^4 - 5(\frac{3}{2} - \frac{5}{2}x)^4 + 10(\frac{1}{2} - \frac{5}{2}x)^4, & 5x < 1 \\ (\frac{5}{2} - \frac{5}{2}x)^4 - 5(\frac{3}{2} - \frac{5}{2}x)^4, & 1 \leq 5x < 3 \\ (\frac{5}{2} - \frac{5}{2}x)^4, & 3 \leq 5x < 5 \end{cases}$
10	sph-5	$\begin{cases} (3 - 3x)^5 - 6(2 - 3x)^5 + 15(1 - 3x)^5, & 3x < 1 \\ (3 - 3x)^5 - 5(2 - 3x)^5, & 1 \leq 3x < 2 \\ (3 - 3x)^5, & 2 \leq 3x < 3 \end{cases}$
11	Epanechnikov	$\frac{3}{4}(1 - x^2)$
12	IObounce	$1 - \frac{1}{2}\begin{cases} 1 - \mathit{bounce}(1 - 2x), & x < \frac{1}{2} \\ \mathit{bounce}(2x - 1) + \frac{1}{2}, & \text{else} \end{cases}$
13	IOcircular	$1 - \begin{cases} 1 - \frac{1}{2}\sqrt{1 - 4x^2}, & x < \frac{1}{2} \\ \frac{1}{2}\sqrt{(3 - 2x)(2x - 1) + 1}), & \text{else} \end{cases}$
14	IOelastic	$1 - \frac{1}{2}\begin{cases} 1 - \sin(13\pi x)2^{10(2x-1)}, & x < \frac{1}{2} \\ \sin(-6.5\pi((2x - 1) + 1)2^{-10(2x-1)+1}), & \text{else} \end{cases}$
15	IOexponential	$1 - \begin{cases} 0.5 \cdot 2^{20x-10}, & x < \frac{1}{2} \\ -0.5 \cdot 2^{10-20x}, & \text{else} \end{cases}$
16	IOsinus	$1 - (-\frac{1}{2}(\cos(\pi x) - 1))$
17	IOCubic	$1 - \begin{cases} 4x^3, & x < \frac{1}{2} \\ \frac{1}{2}(2x - 2)^3 + 1, & \text{else} \end{cases}$

18	IOQuadratic	1-$\begin{cases} 2x^2, & x < \frac{1}{2} \\ -2x^2 + 4x - 1, & \text{else} \end{cases}$
19	IOQuintic	$\begin{cases} 16x^5, & x < \frac{1}{2} \\ \frac{1}{2}(2x-2)^5 + 1, & \text{else} \end{cases}$
20	cs-rbf 1	$1 - x$
21	cs-rbf 2	$(1-x)^3(3x+1)$
22	cs-rbf 3	$(1-x)^5(8x^2+5x+1)$
23	cs-rbf 4	$(1-x)^2$
24	cs-rbf 5	$(1-x)^4(4x+1)$
25	cs-rbf 6	$(1-x)^6(35x^2+18x+3)$
26	cs-rbf 7	$(1-x)^8(32x^3+25x^2+8x+1)$
27	cs-rbf 8	$(1-x)^3$
28	cs-rbf 9	$(1-x)^3$ (5 x + 1)
29	cs-rbf 10	$(1-x)^7(16x^2+7x+1)$
30	FermiDiracI	$\left(e^{(x-0.6)/0.1} + 1.0\right)^{-1}$
31	FermiDiracII	$\left(e^{(x-0.35)/0.05} + 1.0\right)^{-1}$
32	FermiDiracIII	$\left(e^{(x-0.3)/0.1} + 1.0\right)^{-1}$

with

$$bounce(x) = \begin{cases} 7.5625x^2, & x < \frac{4}{11} \\ 9.075x^2 - 9.9x + 3.4, & \frac{4}{11} \leq x < \frac{8}{11} \\ \frac{4356}{361}x^2 - \frac{35442}{1805}x + \frac{16061}{1805}, & \frac{8}{11} \leq x < \frac{9}{10} \\ 10.8x^2 - 20.52x + 10.72\frac{9}{10} \leq x < 1 \end{cases}$$

Declaration

Hereby I declare that this work was written independently and originally by myself. No other than the specified sources were utilized. All passages, literally or with regard to the content of the specified sources are identified as such. The presented work has not yet been submitted as a PhD thesis before in the same or a similar form.