Mean shift for graph bundling

Ozan Ersoy
Institute Johann Bernoulli
University of Groningen
the Netherlands
o.ersoy@rug.nl

Christophe Hurter
DGAC-DTI R&D
ENAC/Univ. of Toulouse
France
christophe-hurter@aviation-civile.gouv.fr

Alexandru Telea
Institute Johann Bernoulli
University of Groningen
the Netherlands
a.c.telea@rug.nl

Abstract

We present a fast and simple adaption of the well-known mean shift technique for image segmentation to compute bundled layouts of general graphs. For this, we first transform a given graph drawing into a density map using kernel density estimation. Next, we apply the equivalent of mean shift segmentation on this image, *i.e.* sharpen the image my moving the drawn edges upstream in the density's gradient. We implement our method using standard graphics acceleration techniques. Our results are similar to state-of-the-art graph bundling methods but require a fraction of their cost. We demonstrate our method on several large graphs.

1 Introduction

Graphs are ubiquitous in many information analysis and information visualization applications. General graphs can be visualized by node-link diagrams [29], matrix plots [30], and graph splatting [31]. In recent years, *graph bundling* methods have gained increased attention.Given a set of node positions, edges being close in terms of graph structure, position, data attributes, or combinations thereof, are drawn as tightly bundled curves. This trades clutter for overdraw and produces images which emphasize the graph structure. Blending or shading can be used to add information or emphasize structure [14, 19, 27]. Bundling methods exist for both compound (hierarchy-and-association) [13] and general graphs [14, 4, 21, 19, 11, 9, 24]. However attractive, most bundling algorithms for general graphs are quite complex and/or have high computational costs.

In this paper, we present a new method for bundling general graphs. We work entirely image-based: Given a graph drawing, we first convolve the edges and construct an edge density map. Next, we iteratively advect edges upstream in the density's gradient while recomputing the map. This process, which is similar to the well-known mean shift image segmentation technique [3], delivers a layout with well separated and smooth bundle structures.

The structure of this paper is as follows. Section 2 reviews related work on edge bundles. Section 3 presents our method. Section 4 details implementation and shows results on real-world graphs. Section 5 discusses our method. Section 6 concludes the paper.

2 Related Work

Related work in reducing visual clutter in large graphs can be organized as follows.

Graph simplification techniques reduce clutter by simplifying the graph prior to layout *e.g.* by creating metanodes of strongly connected nodes and edges, next drawn by classical node-link layouts [1, 2]. Simplification can be sensitive to parameters which may depend on the graph type. Also, simplification events yield a set of discrete graphs rather than a continuous exploration scale [19]. Thirdly, simplification changes node positions (collapse to metanodes). This is unwanted when positions encode information.

Edge bundling techniques trade clutter for overdraw by routing related edges along similar paths. Details on clutter causes and reduction strategies are given in [7]. Bundling can be seen as *sharpening* the edge spatial density, by making it high along bundles and low elsewhere. Bundles help finding node-groups related to each other by edge-groups (bundles) which are separated by white space [11]. Dickerson *et al.* merge edges by reducing non-planar graphs to planar ones [5]. Holten bundled edges in compound graphs by routing edges along the hierarchy layout using B-splines [13]. Gansner and Koren bundle edges in a circular node layout similar to [13] by area optimization metrics [12]. Dwyer *et al.* use curved edges in force-directed layouts to minimize crossings, which implicitly creates bundle-like shapes [6]. Force-directed edge bundling (FDEB) creates bundles by attracting edge control points [14], and was adapted to separate opposite-direction bundles [24]. The MINGLE method uses multilevel clustering to accelerate the bundling process [11]. Flow maps produce a binary clustering of nodes in a directed flow graph to route curved edges [21]. Control meshes are used to route curved edges, *e.g.* [22, 33], a Delaunay-based extension called geometric-

This paper has been submitted for presentation at ICT.OPEN 2012. This is the national Dutch ICT conference and serves, for this paper, the role of training the presentation skills rather than to push scientific limits. ICT.OPEN does not claim copyright. For this reason, ICT.OPEN 2012 encourages authors to submit mostly papers that have been sent to or have recently been presented at international conferences.

based edge bundling (GBEB) [4], and 'winding roads' (WR) which use Voronoi diagrams for 2D and 3D layouts [19, 18]. Skeleton-based edge bundling (SBEB) uses the skeletons of the graph drawing's thresholded distance transform as bundling cues to create strongly ramified bundles [9].

Several bundle rendering techniques exist: edge color interpolation for edge directions [13, 4]; transparency or hue for edge density, or for edge lengths [19]. Bundles can be drawn as compact shapes whose structure is emphasized by shaded cushions [27, 23]. Graph splatting visualizes node-link diagrams as continuous scalar fields using color and/or height maps [31, 16]. Semantic lenses can be used to interactively explore overlapping bundles [15].

3 Algorithm

Given a graph drawing $G \subset \mathbf{R}^2$ and a point $\mathbf{x} \in G$, we can think of bundling as an operator $B : \mathbf{R}^2 \to \mathbf{R}^2$ which displaces $\mathbf{x}$ based on the spatial information in $G \cap \nu_\varepsilon(\mathbf{x})$ where $\nu_\varepsilon(\mathbf{x})$ is a small neighborhood centered at $\mathbf{x}$. The result $B(G)$ is a new layout whose edges are gathered in dense groups (bundles) separated by low edge-density areas (white space). Intuitively, B is an image processing function which *sharpens* the local spatial density ρ of edge points.

We model ρ using kernel density estimation (KDE) methods [26]: Given a graph drawing $G = \{e_i\}_{1<i<N}$ consisting of edges $e_i \subset \mathbf{R}^2$, we can estimate $\rho : \mathbf{R}^2 \to \mathbf{R}^+$ as

$$\rho(\mathbf{x}) = \sum_{i=1}^{N} \int_{\mathbf{y}\in e_i} K\left(\frac{\mathbf{x}-\mathbf{y}}{h}\right) \tag{1}$$

where $K : \mathbf{R}^2 \to \mathbf{R}^+$ is a density kernel of bandwidth $h > 0$. ρ can be computed by convolving G with K, or building an accumulation map of K over G.

The map ρ reflects the local edge density. A drawing with uniformly distributed edges yields a flat map. Large ρ values are zones of high edge density. More interestingly, local maxima of ρ are roughly in the *middle* of local edge agglomerations. Ersoy *et al.* have shown that these are good positions for placing edge bundles [9], and compute these points as the medial axes of the Euclidean distance transform of G thresholded at a small value $\tau > 0$. In contrast, we define bundling centers as the local maxima of a continuous density map computed with nonlinear kernels. As we shall see, this implies several differences and advantages for our method.

Given the density map ρ, we next define our kernel density estimation edge-bundling (KDEEB) operator B as the solution of the following ordinary differential equation

$$\frac{d\mathbf{x}}{dt} = \frac{h(t)\nabla\rho(t)}{\max(\|\nabla\rho(t)\|, \varepsilon)} \tag{2}$$

for all points $\mathbf{x}$ in the graph drawing, with initial conditions given by the input graph. The density gradient $\nabla\rho$ is normalized in a regularized manner – the $\varepsilon = 10^{-5}$ denominator value takes care of zero gradients. Normalizing $\nabla\rho$ constrains the movements $\|d\mathbf{x}\|$ to the kernel bandwidth $h(t)$. Since $h(t)$ decreases in time (as explained next), this stabilizes the advection process. Eqn. 2 is solved by Euler integration, *i.e.* we construct $B(G)$ by iteratively computing the density map ρ and advecting the points $\mathbf{x} \in G$ in the direction of $\nabla\rho$. Eqn. 2 sharpens the density ρ starting with the (typically straight-line, unbundled) input graph G and ending with a tightly bundled graph whose density map asymptotically reaches bundle-aligned Dirac impulses.

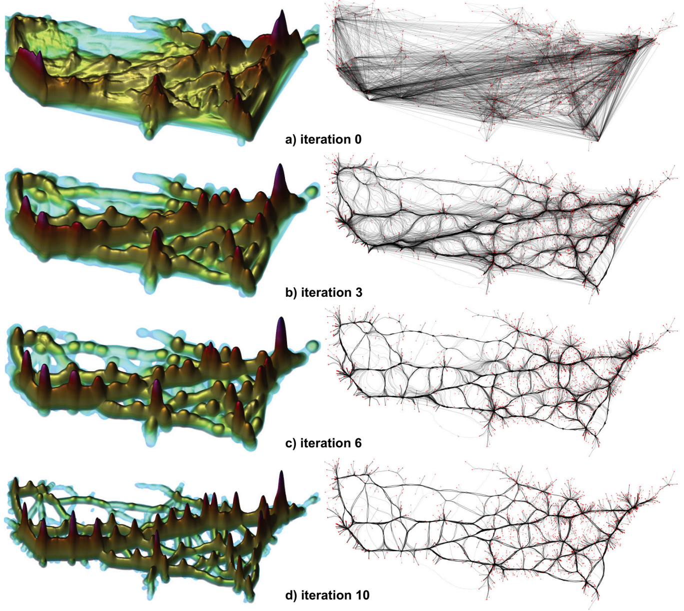

Figure 1: Evolution of density map and corresponding bundling for the US migrations graph.

Essentially, the above process (Eqns. 1 and 2) are very similar to the mean shift process [3]. For example, if we think of an edge (in our case) as being a sample point (in the mean shift case), our formulation finds bundles as the modes of the initial kernel density estimation by using the same gradient ascent as mean shift. Following the analogy, a bundle is equivalent to a cluster found by mean shift segmentation. Similarly to mean shift, we use an Epanechnikov kernel $K(\mathbf{x}) = 1 - \|\mathbf{x}\|^2$, which optimally approximates the ρ in a minimal variance sense [8, 17]. At each step i of the numerical integration, we decrease h by a geometric series $h_i = \lambda^i h_{max}$, where h_{max} is the initial kernel bandwidth, set to the average inter-edge distance in the input graph G, and λ is a kernel bandwidth reduction factor. Setting $\lambda \in [0.5, 0.9]$ yields a kernel size which follows the average edge density. The initial value h_{max} creates a smooth density ρ where any edge point is influenced by at least one other edge and also avoids density overestimations. During integration, edges get closer, so we decrease the kernel h_i to avoid density overestimation. A similar strategy was also used in several applications of mean shift. Decreasing h_i also decreases the advection speed, which stabilizes the process as the signal ρ is increasingly 'sharpened'. In other words, edges converge towards the local density maxima instead of jumping from one side to the other of such maxima. More advanced methods for estimating the kernel bandwidth, such as data-based adaptive selectors can be used, if desired [25, 17, 3]. However, we do not need an *exact* density estimation for graph bundling since we only use the density's *gradient* and recompute the density iteratively, so our simple heuristic suffices.

Figure 1 shows several iterations of the density map, drawn as a height plot (normalized in height for display purposes) and corresponding bundled layouts for the US migrations graph [14, 9]. The density map gets sharper during the iterative solving of Eqn. 2. This bundles edges along the density local maxima or distribution modes. As the density map gets sharper, the distance between local maxima increases, so bundles get tighter and separated by more white space.

4 Implementation

We implement our method using a GPU image-based approach, as follows (see also Fig. 2).

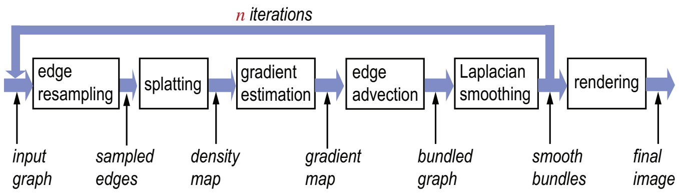

Figure 2: KDE edge bundling pipeline.

4.1 Graph representation

First, we discretize all edges e_i of the input graph into sets of points $\mathbf{x}_{ij}$, by using a small sampling step δ equal to roughly 1% of the size of the graph's bounding box, similarly to other methods [14, 13, 9, 19]. This typically yields several tens of sample points per edge on average.

4.2 Density computation and gradient estimation

To compute the density map gradient $\nabla\rho$ needed by Eqn. 2), we can splat the kernel gradient ∇K, precomputed into two OpenGL 2D luminance textures $\partial K/\partial x$ and $\partial K/\partial y$, at all edge sample points $\mathbf{x}_{ij}$, and accumulate results into two floating-point buffers by additive blending. Maximal efficiency is achieved by drawing OpenGL point sprites scaled by the bandwidth h_i (Sec. 3). The accumulation buffers' size matches the screen size.

A better approximation of the kernel density estimation (Eqn. 1) is obtained if we use edge-aligned kernels. For this, we use elliptical kernels aligned with the edge segments $(\mathbf{x}_{ij}, \mathbf{x}_{ij+1})$, *i.e.* draw rectangles textured by the radial kernel K centered at the edge sample points, aligned with the edge segments, and of size h (across the edge) and equal to the average of $\|\mathbf{x}_{ij} - \mathbf{x}_{ij+1}\|$ (along the edge). Another option is to use one-dimensional half-kernels stored as 1D textures and drawn as rectangles tangent to the edge segments. The latter method was used by Ersoy *et al.*, with a different (distance) kernel, to create distance profiles [27]. Edge-aligned kernels allow a lower edge sampling rate, since kernels are scaled separately along and across edges, thus increase splatting speed without decreasing the KDE quality.

4.3 Advection

After obtaining the gradient of our edge density map, we advect each edge by Euler integration of Eqn. 2 on the edge sample points $\mathbf{x}_{ij}$. Edge endpoints are kept fixed. Since we first compute the gradient map and then advect all edge points, integration is explicit, which parallelizes easily. After each advection step, we resample the edges (Sec. 4.1). This is needed since div $\nabla\rho \neq 0$ *and* edge endpoints are fixed, so advection stretches and/or shrinks edges, which can lead to edge self-intersections or subsampled edge fragments.

4.4 Smoothing

After each iteration, we do 5..10 Laplacian smoothing iterations of the advected edges with a kernel of fixed size, roughly 8δ, similar to [14]. This removes small-scale advection artifacts caused by the imprecise estimation of the density map ρ which is, in turn, due to errors in the kernel bandwidth estimation (Sec. 3), on the one hand, and to discretization errors in the finite edge sampling and finite kernel splat texture resolution (Sec. 4.2), on the other hand. Artifacts show up as small-scale undulations in the density map, which cause extra divergence points, *i.e.* slight rotations, of $\nabla\rho$. In turn, gradient imprecisions cause edges to become jagged during advection, thus yield slight zig-zags in the final bundles. Laplacian smoothing completely removes this problem and generates smooth bundles. Our smoothing is equivalent to anisotropically filtering the density map, prior to gradient estimation, with a kernel aligned with the map's curvature minor eigenvector, *i.e.* along its ridges [32]. However, this type of image filtering is considerably more expensive, and clearly more complicated, than our Laplacian edge smoothing.

4.5 Iterative bundling

For all tested graphs, 8..10 iterations of gradient computation, advection, and smoothing yields a stable layout. The process is monotonic: edges move in a single direction rather than back-and-forth. This is due to the structure of the density map gradient: If two edge points $\mathbf{x}, \mathbf{y} \in G$ are within each other's bandwidths at an iteration, both are equally advected towards the midpoint $(\mathbf{x}+\mathbf{y})/2$, since we use the same kernel size and shape at all points. For a more formal discussion on the stability of the original mean shift procedure, we refer to [3].

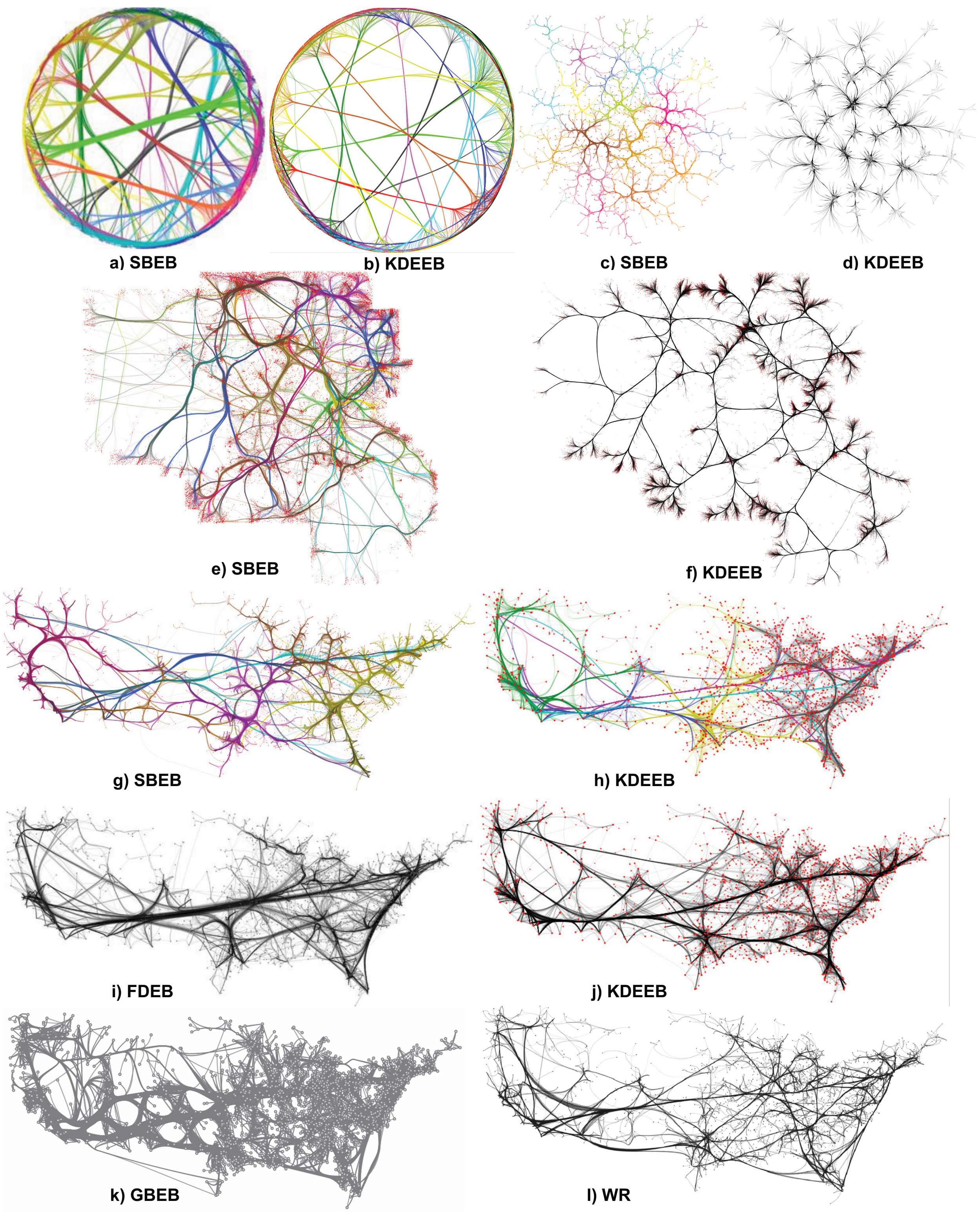

Figure 3: Bundling examples. Radial graph (a,b); Poker graph (c,d); France airlines (e,f); US migrations, clustered (g,h); US migrations, unclustered (i,j,k,l); Colors mark different edge clusters. More examples at [20]

4.6 Examples

Figure 3 compares our KDEEB with recent bundling methods: FDEB [14], GBEB [4], SBEB [9], and WR [19]. Overall, we produce tighter bundles than FDEB and GBEB, and smoother bundles than SBEB. While SBEB requires an edge pre-clustering on similar directions and positions (Fig. 3 a,c,f,g), we obtain similar or better results, *i.e.* tight, smooth, well-separated bundles, with no clustering at all. If edge clusters are provided, we can use these by bundling each cluster separately. For example, in Fig. 3 (a,b), which shows a software dependency graph with edges grouped by structural similarity, KDEEB delivers better separated bundles, than SBEB. Also, compare Fig. 3 g (US migrations graph, pre-clustered on edge similarity, bundled with SBEB) with KDEEB where we bundle each cluster separately (Fig. 3 h). Our result is more similar to bundlings which do not use clustering (*e.g.* our method, Fig. 3 j or WR, Fig. 3 l) than to SBEB. This indicates that our method could be used in cases where we want to bundle parts of a graph separately, *e.g.* interactive exploration or online graph bundling. Per-cluster bundling does not decrease the speed of our method, since its complexity is $O(EI/\delta)$ for a graph with E edges, I bundling iterations, and an edge sampling step δ.

5 Discussion

5.1 Comparison

Several differences are visible between our method *vs* existing methods (see Fig. 3 and more images at [20]): We produce smoother, less twisting, bundles than GBEB and SBEB, and tighter bundles than FDEB and MINGLE.

Equations 1 and 2 share some aspects with FDEB [14] and SBEB [9]. As FDEB, we move edge points close to each other, but we do not need any additional edge compatibility metrics ([14], Sec. 3.2). As SBEB, we move edges close to their local center. While SBEB computes this center *explicitly* as medial axes of thresholded distance functions of similar-direction edges, we move edges towards their *implicit* local center via the density map gradient. Eqn. 2 resembles solving the Eikonal equation [28], as we move edges with equal speed along a radial kernel gradient, which resembles the gradient of an Euclidean distance map. However, we recompute this gradient at each step, while [28] uses a fixed motion direction given by an explicit initial boundary.

GPU image-based techniques based on a density map computed from a graph drawing are also used by [10]. However, the aim is different: We 'concentrate' the density signal, and keep nodes fixed, to bundle edges, while [10] works in the opposite direction, spreading nodes towards less dense areas in order to declutter a given layout.

5.2 Performance and simplicity

Our entire bundling code is under 1000 lines of C#, and consists of four simple steps: density computation (Sec. 4.2), edge advection (Sec. 4.3), and edge smoothing (Sec. 4.4). Compared to other bundling methods whose implementations we could study [14, 19, 9], our pipeline is simpler, *e.g.* we do not require graph clustering, skeletons, Voronoi diagrams, or spatial search structures. We only use OpenGL 1.1 as compared to the more complex CUDA or pixel shader code in [9, 19].

Graph	Nodes	Edges	Edge samples	Bundling time (sec.)	
				8800 GTX	GeForce 580
US airlines	235	2099	86K	1.4	0.5
US migrations	1715	9780	220K	3.6	1.5
Radial	1024	4021	290K	4.5	1.5
France air	34550	17275	330K	3.8	1.8
Poker	859	2127	50K	0.8	0.4
Random	200K	100K	4.8M	43	18

Table 1: Graph statistics for datasets used in this paper.

Table 1 shows running times on two Nvidia cards, both on a 3.3 GHz Core i5 PC, for 10 iterations. The *Edge samples* column shows the number of sample points on all graph edges. Advection, resampling, and smoothing are done in C# on 4 threads, which takes about 40% of the entire time, the remainder being OpenGL-based splatting. These steps can be easily accelerated further with *e.g.* vertex shaders or CUDA. However, even without this extra boost, KDEEB is much faster than similar approaches - on average for the tested graphs, 16 times *vs* FDEB [14], 6 times *vs* GBEB [4], 5 *vs* than SBEB [9], and 4 *vs* WR [19]. The only faster bundling method we know is MINGLE [11]: 2..3 times faster than KDEEB for graphs up to 2000 edges, and about the same speed for larger graphs. The lower performance of KDEEB for small graphs is due to the relatively large amount of work done in C# on the CPU for these graphs, which gets dominated by GPU computations for larger graphs. Also, note that MINGLE arguably produces more cluttered, less bundled, layouts, as it uses only the start and endpoints of edges to bundle these, whereas we use the entire edge paths, as illustrated by the images available online at [20].

Memory-wise, we only need to store three frame buffers equal to the screen size (density map and its two gradient components). This means practically zero data overhead atop of the edge samples which store the bundled layout.

6 Conclusion

We have presented a new method for creating bundled layouts of general graphs. Our approach offers a simple, (GPU) parallelizable method which is several times faster, and arguably simpler to implement, than comparable methods. Our method produces bundled graph layouts with tight and smooth structures, robustly handles graphs of widely variable complexity and size, and requires no complex user parameter settings. We show how to constrain bundling to avoid arbitrary-shaped obstacles placed in the embedding space at user-selected positions, and also a way to globally route bundles outside the nodes' position area. Our approach, which follows an image sharpening technique, opens new ways for analyzing and refining graph bundling based on well understood image processing techniques.

Several future work directions exist. Speed-wise, our method can directly use a fully-parallel (*e.g.* CUDA) optimization. Secondly, by modifying the splat kernels, different bundling styles could be obtained *e.g.* orthogonal layouts. Following the mean shift analogy, we can use our method to perform image-based segmentation of a given graph layout. Finally, adapting our method to perform 3D graph bundling is straightforward and fast, as our recent work-in-progress shows.

References

[1] J. Abello, F. van Ham, and N. Krishnan. AskGraphView: A large graph visualisation system. *IEEE TVCG*, 12(5):669–676, 2006.

[2] D. Archambault, T. Munzner, and D. Auber. Grouse: Feature-based and steerable graph hierarchy exploration. In *Proc. EuroVis*, pages 67–74, 2007.

[3] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE TPAMI*, 24(5):603–619, 2002.

[4] W. Cui, H. Zhou, H. Qu, P. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE TVCG*, 14(6):1277–1284, 2008.

[5] M. Dickerson, D. Eppstein, M. Goodrich, and J. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. In *Proc. Graph Drawing*, pages 1–12, 2003.

[6] T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into force-directed layout. In *Proc. Graph Drawing*, pages 8–19, 2007.

[7] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE TVCG*, 13(6):1216–1223, 2007.

[8] V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability and its Applications*, 14:153–158, 1969.

[9] O. Ersoy, C. Hurter, F. Paulovich, G. Cantareira, and A. Telea. Skeleton-based edge bundles for graph visualization. *IEEE TVCG*, 17(2):2364 – 2373, 2011.

[10] Y. Frishman and A. Tal. Uncluttering graph layouts using anisotropic diffusion and mass transport. *IEEE TVCG*, 15(5):777–788, 2009.

[11] E. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Proc. PacificVis*, pages 187–194, 2011.

[12] E. Gansner and Y. Koren. Improved circular layouts. In *Proc. Graph Drawing*, pages 386–398, 2006.

[13] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE TVCG*, 12(5):741–748, 2006.

[14] D. Holten and J. J. van Wijk. Force-directed edge bundling for graph visualization. *Comp. Graph. Forum*, 28(3):670–677, 2009.

[15] C. Hurter, O. Ersoy, and A. Telea. Moleview: An attribute and structure-based semantic lens for large element-based plots. *IEEE TVCG*, 17(12):2600–2609, 2011.

[16] C. Hurter, B. Tissoires, and S. Conversy. FromDaDy: Spreading data across views to support iterative exploration of aircraft trajectories. *IEEE TVCG*, 15(6):1017–1024, 2009.

[17] M. Jones, J. Marron, and S. Sheather. A brief survey of bandwidth selection for density estimation. *J. American Stat. Assoc.*, 91(433):401–407, 1996.

[18] A. Lambert, R. Bourqui, and D. Auber. 3D edge bundling for geographical data visualization. In *Proc. Information Visualisation*, pages 329–335, 2010.

[19] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *Comp. Graph. Forum*, 29(3):432–439, 2010.

[20] O. Ersoy and C. Hurter and A. Telea. KDEEB examples, 2012. `www.cs.rug.nl/svcg/Shapes/KDEEB`.

[21] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow map layout. In *Proc. InfoVis*, pages 219–224, 2005.

[22] H. Qu, H. Zhou, and Y. Wu. Controllable and progressive edge clustering for large networks. In *Proc. Graph Drawing*, pages 399–404, 2006.

[23] R. Scheepens, N. Willems, H. van de Wetering, G. Andrienko, N. Andrienko, and J. J. van Wijk. Composite density maps for multivariate trajectories. *IEEE TVCG*, 17(12):2518–2527, 2011.

[24] D. Selassie, B. Heller, and J. Heer. Divided edge bundling for directional network data. *IEEE TVCG*, 19(12):754–763, 2011.

[25] S. Sheather and M. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *J. of the Royal Statistical Society*, B53(3):683–690, 1991.

[26] B. Silverman. Density estimation for statistics and data analysis. *Monographs on Statistics and Applied Probability*, 26, 1992.

[27] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *Comp. Graph. Forum*, 29(3):543–551, 2010.

[28] A. Telea and J. J. van Wijk. An augmented fast marching method for computing skeletons and centerlines. In *Proc. VisSym*, pages 251–259, 2002.

[29] I. Tollis, G. Di Battista, P. Eades, and R. Tamassia. *Graph drawing: Algorithms for the visualization of graphs*. Prentice Hall, 1999.

[30] F. vam Ham. Using multilevel call matrices in large software projects. In *Proc. InfoVis*, pages 227–232, 2003.

[31] R. van Liere and W. de Leeuw. GraphSplatting: Visualizing graphs as continuous fields. *IEEE TVCG*, 9(2):206–212, 2003.

[32] J. Weickert. *Anisotropic diffusion in image processing, Teuber Verlag, Stuttgart, 1998.* Teuber Verlag, 1998.

[33] H. Zhou, X. Yuan, W. Cui, H. Qu, and B. Chen. Energy-based hierarchical edge clustering of graphs. In *Proc. PacificVis*, pages 55–62, 2008.