Eurographics Conference on Visualization 2012
S. Bruckner, S. Miksch, and H. Pfister
(Guest Editors)

*Volume 31 (2012), Number 3*

# Graph Bundling by Kernel Density Estimation

C. Hurter[1], O. Ersoy[2] and A. Telea[2]

[1]ENAC/University of Toulouse, France, E.mail: christophe.hurter@aviation-civile.gouv.fr
[2]University of Groningen, the Netherlands, E.mail: o.ersoy@rug.nl, a.c.telea@rug.nl

**Abstract**

*We present a fast and simple method 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 an image sharpening technique which progressively merges local height maxima by moving the convolved graph edges into the height gradient flow. Our technique can be easily and efficiently implemented using standard graphics acceleration techniques and produces graph bundlings of similar appearance and quality to state-of-the-art methods at a fraction of the cost. Additionally, we show how to create bundled layouts constrained by obstacles and use shading to convey information on the bundling quality. We demonstrate our method on several large graphs.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

**Keywords:** Graph layouts, edge bundles, image-based information visualization.

## 1. Introduction

Graphs are present in applications such as software comprehension, geovisualization, and network analysis. Visualization metaphors for general graphs include node-link diagrams [TBET99], matrix plots [vH03], and graph splatting [vLdL03]. In recent years, *graph bundling* methods have gained increased attention. Bundling starts with a set of node positions, given as input data or computed by a layout algorithm. 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 [HvW09, LBA10b, TE10]. Bundling algorithms exist for both compound (hierarchy-and-association) [Hol06] and general graphs [HvW09, CZQ*08, PXY*05, LBA10b, GHNS11, EHP*11, SHH11]. However attractive, many bundling algorithms for general graphs are quite complex and 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 with a special kernel to construct a density map. Next, we advect edges in the gradient of this map and iterate the process for a few steps with decreasing kernels. This delivers a layout with well separated and smooth bundle structures. Separately, we modify our density map to obtain bundles which avoid user-specified obstacles of arbitrary sizes and shapes. Finally, we propose a new shading technique which conveys the bundling quality in an easy to interpret way. Our contributions are as follows:

- a bundling technique for general graphs which is robust, simple to implement, and up to one order of magnitude faster than state-of-the-art techniques;
- a technique to generate bundled layouts that smoothly avoid obstacles of arbitrary shape and position;
- a way to visually convey bundling quality via shading.

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 presents our obstacle-driven bundling and bundling quality visualization. Section 6 discusses our method. Section 7 concludes the paper.

## 2. Related Work

Related work in reducing clutter in large graph visualizations can be organized as follows.

*Graph simplification* techniques reduce clutter by simpli-

fying the graph prior to layout *e.g.* by creating metanodes of strongly connected nodes and edges, next shown by classical node-link layouts [AvHK06, AMA07]. This approach can be sensitive to simplification parameters which may depend on the graph type. It does not allow a continuous treatment of the graph: Simplification events yield a set of discrete graphs rather than an exploration scale [LBA10b]. Also, simplification changes node positions (collapse to metanodes). This is undesirable 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 [ED07]. Bundling can be seen as sharpening the edge spatial density, by making it high along bundles and low elsewhere. This improves readability for finding node-groups related to each other by edge-groups (bundles) which are separated by white space [GHNS11]. Dickerson *et al.* merge edges by reducing non-planar graphs to planar ones [DEGM03]. Holten bundled edges in compound graphs by routing edges along the hierarchy layout using B-splines [Hol06]. Gansner and Koren bundle edges in a circular node layout similar to [Hol06] by area optimization metrics [GK06]. Dwyer *et al.* use curved edges in force-directed layouts to minimize crossings, which implicitly creates bundle-like shapes [DMW07]. Force-directed edge bundling (FDEB) creates bundles by attracting edge control points [HvW09], and was adapted to separate opposite-direction bundles [SHH11]. The MINGLE method uses multilevel clustering to accelerate the bundling process [GHNS11]. Flow maps produce a binary clustering of nodes in a directed flow graph to route curved edges [PXY*05]. Control meshes are used to route curved edges, *e.g.* [QZW06, ZYC*08], a Delaunay-based extension called geometric-based edge bundling (GBEB) [CZQ*08], and 'winding roads' (WR) which use Voronoi diagrams for 2D and 3D layouts [LBA10b, LBA10a]. Skeleton-based edge bundling (SBEB) uses the skeletons of the graph drawing's thresholded distance transform as bundling cues to create strongly ramified bundles [EHP*11].

To render and explore bundled layouts, several techniques exist: edge color interpolation for edge directions [Hol06, CZQ*08]; transparency or hue for edge density, or for edge lengths [LBA10b]. Bundles can be drawn as compact shapes whose structure is emphasized by shaded cushions [TE10, SWvdW*11]. Graph splatting visualizes node-link diagrams as continuous scalar fields using color and/or height maps [vLdL03, HTC09]. To explore crowded areas where several bundles overlap, bundled layouts can be interactively deformed using semantic lenses [HET11].

## 3. Algorithm

Most general-graph bundling methods use edge-to-edge neighborhood information: 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) to minimize drawing ink. Intuitively, we can see $B$ as an image processing function which *sharpens* the local spatial density $\rho$ of edge points.

We model $\rho$ using kernel density estimation (KDE) methods [Sil92]: 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 $h > 0$. Typical kernel choices are Gaussian and Epanechnikov (quadratic) functions. $\rho$ can be computed by convolving $G$ with $K$, or building an accumulation map of $K$ over $G$.

The density map $\rho$ reflects the local edge density. A graph drawing with uniformly distributed edges yields a flat map. Large $\rho$ values are zones of high edge density. More interestingly, local maxima of $\rho$ are located roughly in the *middle* of local edge agglomerations. Ersoy *et al.* have shown that these are good positions for placing edge bundles [EHP*11], and compute these points as the medial axes, or skeletons, 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 later, this implies several differences and advantages for our method.

Given the density map $\rho$, 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 $\rho$ and advecting the points $\mathbf{x} \in G$ in the direction of $\nabla\rho$. The effect of Eqn. 2 is to sharpen the density $\rho$ 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.

The choice of the kernel $K$ and bandwidth $h$ are discussed next. We use an Epanechnikov kernel $K(\mathbf{x}) = 1 - \|\mathbf{x}\|^2$, which optimally approximates the $\rho$ in a minimal variance sense [Epa69, JMS96]. 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 $\lambda$ is

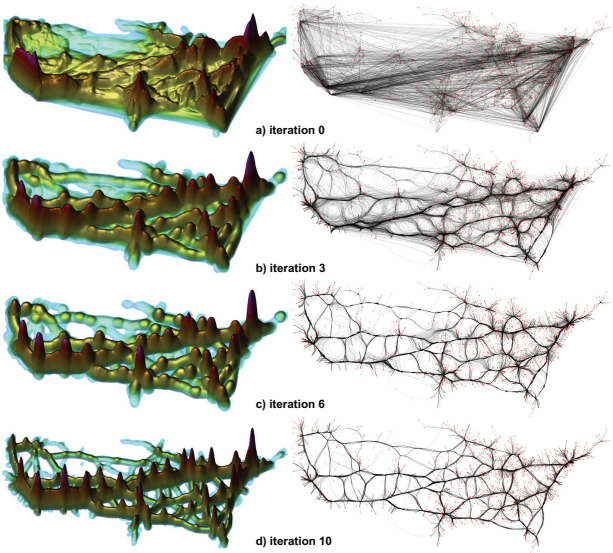


**Figure 1:** *Evolution of density map and corresponding bundling for the US migrations graph.*

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 $\rho$ 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. Decreasing $h_i$ also decreases the advection speed, which stabilizes the process as the signal $\rho$ 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 [SJ91,JMS96]. 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) and corresponding bundled layouts for the US migrations graph [HvW09,EHP*11]. The density map gets sharper during the iterative solving of Eqn. 2. This bundles edges along the density local maxima. As the density map gets sharper, the average distance between local maxima increases, so bundles get tighter and separated by more white space.

Figure 2 shows iteration 10 of bundling the same graph, this time without gradient normalization (Eqn. 2). Compared to Fig. 1, the local maxima vary more, *i.e.* edge density non-uniformities in the input graph get amplified during the bundling. Edges close to the high peak top-right in Fig. 2 get bundled strongly, while other edges converge very slowly.

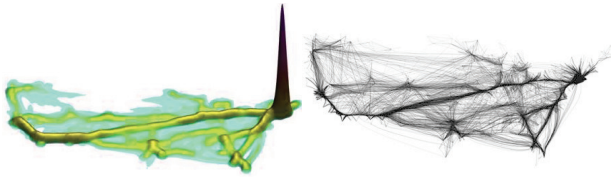

**Figure 2:** *Density map (left) and corresponding bundling for non-normalized advection (compare to Fig. 1)*

## 4. Implementation

An efficient implementation of our method uses a GPU image-based approach, as follows (see also Fig. 3).

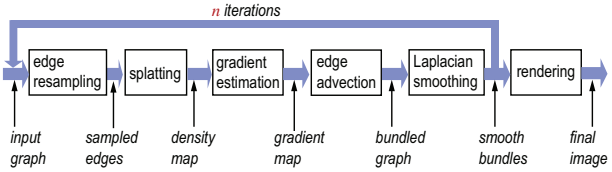


**Figure 3:** *KDE edge bundling pipeline.*

### 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 $\delta$ equal to roughly 1% of the size of the graph's bounding box, similarly to other methods [HvW09, Hol06, EHP*11, LBA10b]. This typically yields several tens of sample points per edge on average.

### 4.2. Density computation and gradient estimation

To compute the density map $\rho$ (Eqn. 1) and gradient $\nabla\rho$, we can splat the kernel $K$, precomputed into an OpenGL 2D luminance texture, at all edge sample points $\mathbf{x}_{ij}$, and accumulate results into a floating-point buffer by additive blending. Maximal efficiency is achieved by drawing OpenGL point sprites scaled by the bandwidth $h_i$ (Sec. 3). The accumulation buffer size matches the screen size. From this accumulation map, we compute $\nabla\rho$ by finite differences. A more accurate way is to precompute $\partial K/\partial x$ and $\partial K/\partial y$ as two separate luminance textures and accumulate the two components of $\nabla\rho$ by splatting the two textures separately. The two approaches are identical speed-wise: The former uses two passes (accumulate, compute gradient); the latter uses a single pass but creates two separate accumulation maps.

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 [TE10]. 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 $\operatorname{div} \nabla\rho \neq 0$ *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 $8\delta$, similar to [HvW09]. This removes small-scale advection artifacts caused by the imprecise estimation of the density map $\rho$ which is 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 [Wei98]. However, this type of image filtering is considerably more expensive, and 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.

### 4.6. Examples

Figure 4 compares our KDEEB with recent bundling methods: FDEB [HvW09], GBEB [CZQ*08], SBEB [EHP*11], and WR [LBA10b]. 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. 4 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. 4 (a,b), which shows a software dependency graph with edges grouped by structural similarity, KDEEB delivers better separated bundles, than SBEB. Also, compare Fig. 4 g (US migrations graph, pre-clustered on edge similarity, bundled with SBEB) with KDEEB where we bundle each cluster separately (Fig. 4 h). Our result is more similar to bundlings which do not use clustering (*e.g.* our method, Fig. 4 j or WR, Fig. 4 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 $\delta$. Figure 5 shows the US airlines graph bundled by FDEB, SBEB, MINGLE, and our method. Again, our results are tighter and arguably less cluttered than other methods.

## 5. Additions

We describe next two visual additions for bundled graphs that are easily added atop of our bundling method: obstacle-constrained bundles and visualizing bundling quality.

### 5.1. Obstacle-constrained bundles

Often, a layout needs to avoid some areas in the embedding space, *e.g.* labels, icons, or other zones of interest. Although many methods for laying out graphs with spatial constraints exist, this use case has not been studied, to our knowledge, for bundled layouts. We next present such an approach.

Given a set of 2D *obstacles* $\Omega_{1\leq i\leq B} \subset \mathbf{R}^2$, we want to create a bundled layout which (a) follows the general paradigm of bundling close edges into smooth and tight bundles, and (b) routes bundles around obstacles without creating sharp bends or lengthening the bundles needlessly. Obstacles are shapes of arbitrary geometry and topology, *e.g.* can have protrusions, dents, or holes, and can be placed freely. We model such shapes as binary images, with foreground pixels ($\Omega$) inside the shape and background pixels ($\overline{\Omega}$) outside.

To constrain bundles, we modify the density $\rho$ used by our method. Instead of the density $\rho$ in Eqn. 1, we use now

$$\rho_{obs} = \rho - DT\left(T\left(DT\left(\cup_{i=1}^{B}\overline{\Omega}_i\right),\tau\right)\right) \qquad (3)$$

where $DT(\Omega) : \Omega \rightarrow \mathbf{R}^+$ is the *distance transform* (DT) of the shape's boundary $\partial\Omega$ [CC00], computed on the foreground, and $T(\cdot,\tau)$ is the lower thresholding of a DT with a value $\tau$. Hence, we subtract from $\rho$ the DT of an inflated version $\Omega_{infl} = T(DT(\overline{\Omega}),\tau)$ of our obstacle $\Omega$ with a distance $\tau$. Since the gradient $\nabla DT(\Omega)$ is a vector that points

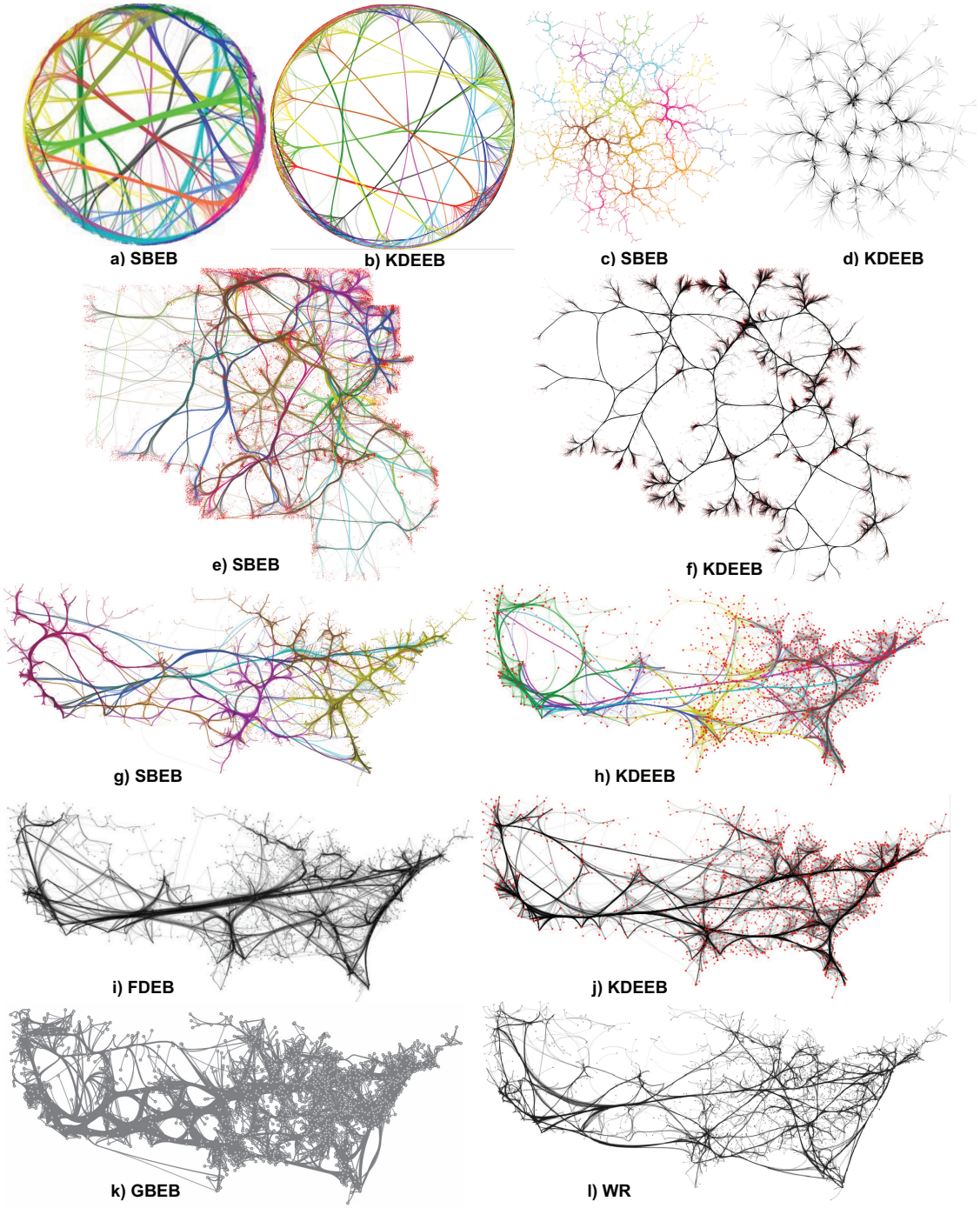


**Figure 4:** *Bundling examples. Radial graph (a,b); Poker graph (c,d); France airlines (e,f); US migrations, clustered (g,h); US migrations, unclustered (i,j,k,l); Colors mark different edge clusters. More examples at www.cs.rug.nl/svcg/Shapes/KDEEB*

from each point $\mathbf{x} \in \Omega$ to the closest point on $\partial\Omega$ to $\mathbf{x}$, by using $\rho_{obs}$ instead of $\rho$ in Eqn. 2, we force edges that cross obstacles to move in the shortest direction towards the obstacles' boundaries, *i.e.* route edges outside obstacles with minimal stretching. Once edges exit an obstacle, this repelling effect ceases, since $DT(\Omega) = 0$ outside obstacles. For shapes with sharp convex corners, $\nabla DT(\Omega)$ is not a smooth field: $\nabla DT(\Omega)$ has discontinuities along the skeleton of $\partial\Omega$, which in turn has one separate branch for each such corner [CC00, TvW02]. However, such discontinuities create no kinks or sharp bends in the advected edges, for several reasons. First, outside obstacles, edges are only influenced

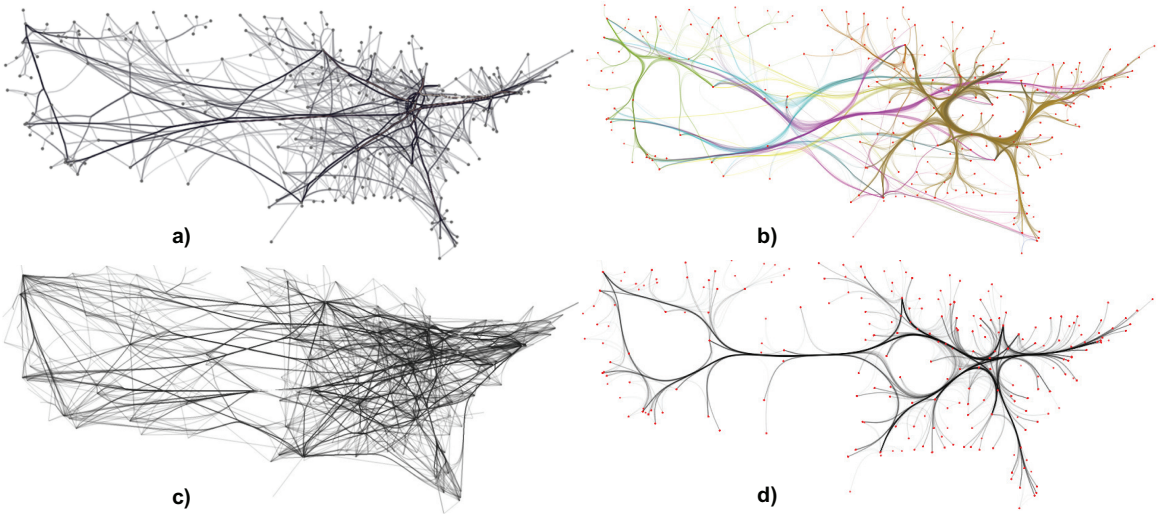


**Figure 5:** *Bundling examples. US airlines (FDEB (a), SBEB (b), MINGLE (c), KDEEB (d)).*

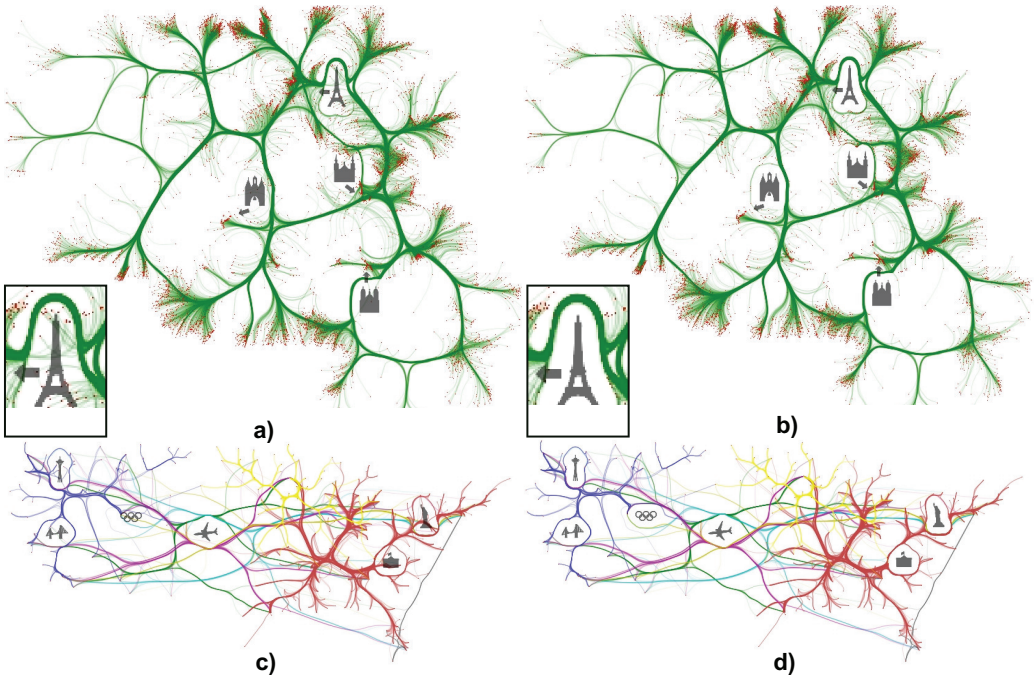


**Figure 6:** *Obstacle-constrained bundling without endpoint displacement (a,c) and with endpoint displacement (b,d).*

by the smooth $\mathscr{C}^\infty$ component $\rho$. Secondly, since we use *inflated* obstacles $\Omega_{infl}$, any corners are rounded out, so edges never get sharp bends when following the obstacles' contours. This matches our goal of smooth obstacle avoidance. The parameter $\tau$ (10..20 pixels) controls how much corners are smoothed, and also creates a thin halo-like band between the routed edges and the obstacles, which helps better separating the former from the latter.

This method has one singular case. Consider a rectangle $\Omega$ crossed by an edge which is parallel to, and far from, its short sides. The edge is parallel with $\nabla DT(\Omega)$, so it only gets shifted tangentially by $\rho_{obs}$. Laplacian smoothing (Sec. 4) eliminates tangential shifts, so the edge never exits $\Omega$.

We solve this problem as follows (see Fig. 7). For each edge $e$ that crosses an inflated obstacle $\Omega_{infl}$, we compute the intersection points $\{\mathbf{p}_i\} = e \cap \partial\Omega_{infl}$. For simplicity, we next consider that there are only two such points $\mathbf{p}_1$ and $\mathbf{p}_2$; the method works the same for more intersection points. We compute the shortest pixel path $\gamma \subset \partial\Omega_{infl}$ between $\mathbf{p}_1$ and $\mathbf{p}_2$. If there are two such paths, we take any of them. Next,

we replace the edge segment $e \cap \Omega$ inside the obstacle with $\gamma$. This pushes $e$ outside $\Omega_{infl}$ with a minimal deformation. Finally, we apply Laplacian smoothing on $e$ (Sec. 4.4), but forbid the smoothed points to re-enter $\Omega_{infl}$. This effectively rounds *concave* corners made by $e$ as it follows $\partial\Omega_{infl}$. Since *convex* corners are already rounded off by using the inflated version of $\Omega$, we obtain edges that smoothly avoid obstacles.

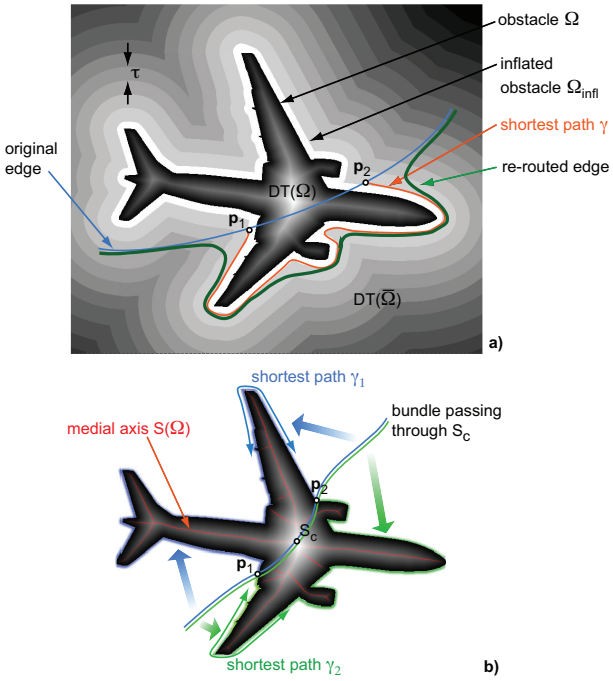


**Figure 7:** *a) Obstacle-constrained bundling refinement; b) Bundle splitting singularity. The background shows the shape's distance transform for illustration (Sec. 5.1).*

Figure 6 shows several obstacle-constrained bundles. Images (a,b) show our method on the France airlines graph (Sec. 6.2), with and without endpoint displacement. Icons show cities close to large flight endpoint agglomerations. Images (c,d) show obstacle avoidance on the US airlines graph (Sec. 6.2). Edges starting or ending inside an obstacle are routed straight to the obstacle boundary, after which they follow the bundle they are part of. If we allow node displacement, endpoints inside obstacles are moved too. The technique works both with our new bundling (Sec. 3, images (a,b)), but also on graphs bundled by other methods, *e.g.* Fig. 6 c,d whose bundling was generated with [EHP*11].

Finally, we present a different type of obstacle avoidance: global whole-area avoidance, or *outward bundling*. In this use-case, we want to create a bundled layout where bundles are routed, if possible, outside the entire area where nodes are placed. This frees up space close to and/or between nodes which can be used to show other information *e.g.* maps, annotations, or different types of (unbundled) edges. In contrast to obstacle avoidance, this is a global process, as we now want to avoid an entire, large, area rather than isolated obstacles. We achieve this by shifting the splat kernels (Sec. 4.2) slightly along the vector between the barycenter of the graph node positions and the position of the current splatting point. This effectively offsets the kernels outside the edges, and thus pulls the edges globally away from the graph center. Edges which connect nodes *radially*, *i.e.* in directions roughly leading to the barycenter, will stay unchanged. Bundles which connect nodes at relatively similar distances from the graph center will, however, be repelled further from this center. Figure 9 b shows this technique on the France airlines graph. We see that, even though the bundle constraints are large, bundles stay coherent but get routed outside the nodes' agglomerations, if possible. The inner space thus freed can be used for additional visualizations. Implementation-wise, this technique is trivial, as it requires only shifting the splatting locations in a given direction when evaluating Eqn. 1.

Obstacle avoidance is simple to implement: We compute the obstacles' distance transforms, inflations, and shortest boundary paths using the AFMM method [TvW02] on images up to $1000^2$ pixels in subsecond time. If higher speed is needed, a CUDA version hereof can be used, which takes under 10 milliseconds on modern graphics cards [EHP*11].

Obstacle avoidance can be done during, or after, bundling. In the former case, obstacles affect bunding: different edges may get bundled than when no obstacles are used. In the latter case, same-bundle edges get re-routed together on the same side of an obstacle, which keeps bundled edges together *except* in the rare case when a bundle intersects the center of the obstacle's medial axis $S(\Omega)$ (Fig. 7 b). In this case, edges which intersect $S(\Omega)$ on different sides of $S_c$ are re-routed to the two different shortest paths along the obstacle's boundary (blue and green curves $\gamma_1$ and $\gamma_2$ in Fig. 7 b). This creates a natural bundle 'flow' around the obstacle.

### 5.2. Visualizing bundling quality

Given any bundling method, how to measure its quality? One can measure the results' fitness for a given task *e.g.* by user studies. Secondly, one can measure the quality of the produced images by some given image metrics. For the latter approach, little work exists so far. We use here the second approach: We model a graph's bundling *strength* by measuring how densely packed its edges are. Areas with high edge density, separated by areas with zero density, indicate strong, clearly delimited, bundles, and minimize ink [GHNS11]. Low edge density areas indicate spurious edges which could not be bundled. These are either limitations of the bundling method or actual data *outliers*, *i.e.* edges with no other similar-direction edges in their proximity.

We address the above as follows. We compute our density map $\rho$, we compute its normal $\mathbf{n}$, and next its Phong shading, with diffuse color set to a user-chosen 'graph material' color and specular strength inversely proportional to

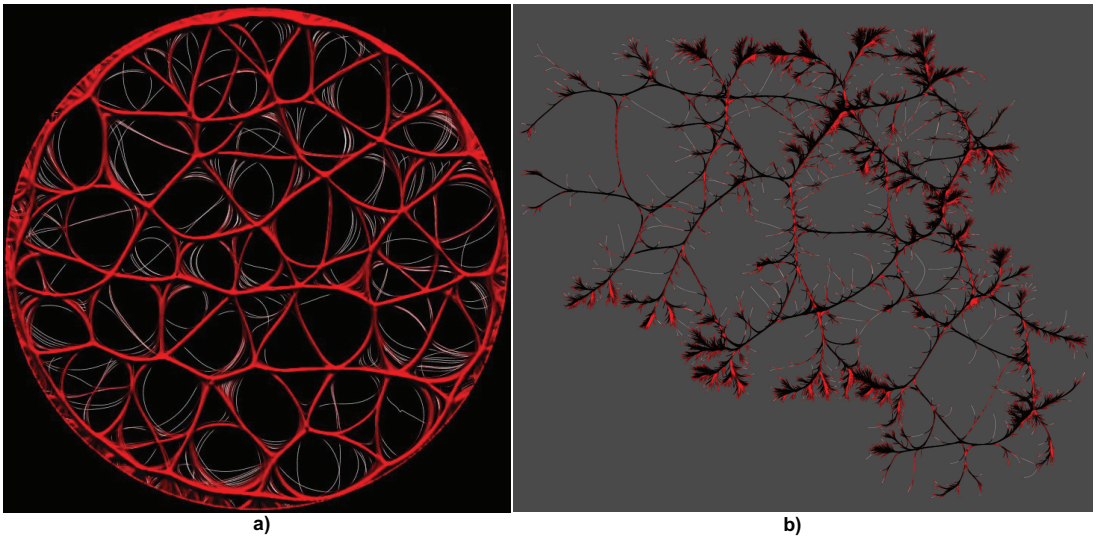


**Figure 8:** *Bundling quality visualized by shading. Shaded colorful structures indicate dense bundles. Outlier edges are white. Radial graph (a), France airlines (b)*

the density ρ. This creates two effects. First, strong bundles appear as shaded cushions in the graph's color, similar to [TE10, EHP*11]. Secondly, outlier edges appear as strongly specular (*e.g.* white). Edges are rendered as lines with classical alpha blending and shading applied at the edge sample points $\mathbf{x}_{ij}$. Technically, our method is simpler than the image-based shading in [TE10]: We only need to apply Phong lighting to the edge sample points, whereas [TE10] constructs 2D shaded bundle images by means of splatting, thresholding, and skeletonization. Thin (outlier) edges appear clearly in our shading, whereas [TE10] only shades bundles having a minimal thickness of several pixels.

Figure 8 shows two examples. The first graph (a) encodes software dependencies *i.e.* nodes are functions and edges are function calls. Shaded red structures show strong bundles indicating groups of functions *i.e.* software subsystems calling each other. These are clearly separated from outlier, unbundled, edges (white). We see that many edges are not bundled. In Fig. 8 b (France airlines graph), most edges are well bundled, as there are very few white outliers. Note that the above visualization is just an aid to reflect on the bundling strength and not a self-contained bundled graph visualization technique in itself: To be effective, it should be combined with suitable shading showing edge types, directions, and nodes.

## 6. Discussion

### 6.1. Comparison

Several differences are visible between our method *vs* existing methods (Fig. 4): We produce smoother, less twisting, bundles than GBEB and SBEB, and tighter bundles than FDEB and MINGLE. Figure 9 a shows the effect of edge-aligned kernels (Sec. 4.1): The obtained bundling (US migrations graph) resembles now more the style of GBEB (Fig. 4 k) than the smooth style of FDEB or WR (Fig. 4 i,l).

Figure 9 c shows bundling of a synthetic graph of 100K edges with nodes randomly placed in a square. The result is a set of well structured, smooth, bundles, with little clutter. There is no semantic associated to such bundles, since our graph was random. However, this shows that KDEEB can effectively declutter and bundle very dense graphs.

Our bundling (Eqns. 1 and 2) shares some aspects with FDEB [HvW09] and SBEB [EHP*11]. As FDEB, we move edge points close to each other, but we do not need any additional edge compatibility metrics ( [HvW09], 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 [TvW02], 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 [TvW02] 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 [FT09]. However, the aim is different: We 'concentrate' the density signal, and keep nodes fixed, to bundle edges, while [FT09] works in the opposite direction, spreading nodes towards less dense areas in order to declutter a given layout.

### 6.2. Performance and simplicity

Our entire bundling code is under 1000 lines of C#, and consists of four simple steps: density computation (Sec. 4.2),

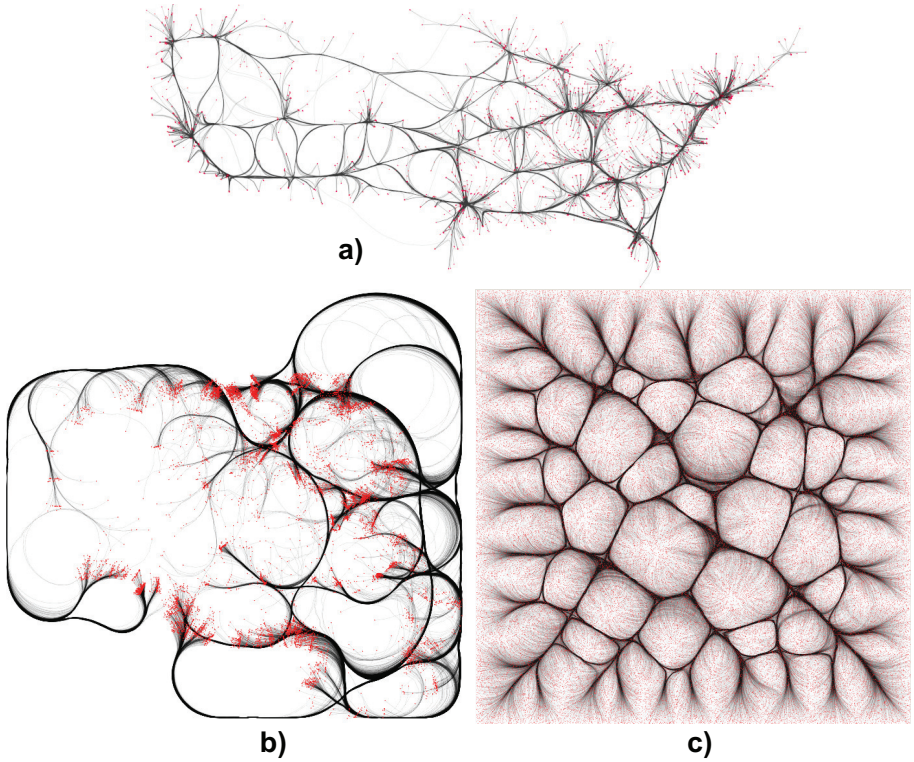


**Figure 9:** *Additional examples. GBEB-style layout (a); Outward bundling (b); Random 100K edge graph bundling (c).*

edge advection (Sec. 4.3), and edge smoothing (Sec. 4.4). Compared to other bundling methods whose implementations we could study [HvW09, LBA10b, EHP*11], 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 [EHP*11, LBA10b].

| 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 [HvW09], 6 times *vs* GBEB [CZQ*08], 5 *vs* than SBEB [EHP*11], and 4 *vs* WR [LBA10b]. The only faster bundling method we know is MINGLE [GHNS11]: 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, MINGLE arguably produces more cluttered, less bundled, layouts (Fig. 5 c vs Fig. 5 d), as it uses only the start and endpoints of edges to bundle these, whereas we use the entire edge paths.

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 describe the bundled layout.

## 7. 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 glob-

ally 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. Last but not least, our image sharpening technique may have direct applications in image processing and simplification, beyond the confines of information visualization.

## References


[AMA07] ARCHAMBAULT D., MUNZNER T., AUBER D.: Grouse: Feature-based and steerable graph hierarchy exploration. In *Proc. EuroVis* (2007), pp. 67–74. 2

[AvHK06] ABELLO J., VAN HAM F., KRISHNAN N.: AskGraphView: A large graph visualisation system. *IEEE TVCG 12*, 5 (2006), 669–676. 2

[CC00] COSTA L., CESAR R.: *Shape analysis and classification: Theory and practice*. CRC Press, 2000. 4, 5

[CZQ*08] CUI W., ZHOU H., QU H., WONG P., LI X.: Geometry-based edge clustering for graph visualization. *IEEE TVCG 14*, 6 (2008), 1277–1284. 1, 2, 4, 9

[DEGM03] DICKERSON M., EPPSTEIN D., GOODRICH M., MENG J.: Confluent drawings: Visualizing non-planar diagrams in a planar way. In *Proc. Graph Drawing* (2003), pp. 1–12. 2

[DMW07] DWYER T., MARRIOTT K., WYBROW M.: Integrating edge routing into force-directed layout. In *Proc. Graph Drawing* (2007), pp. 8–19. 2

[ED07] ELLIS G., DIX A.: A taxonomy of clutter reduction for information visualisation. *IEEE TVCG 13*, 6 (2007), 1216–1223. 2

[EHP*11] ERSOY O., HURTER C., PAULOVICH F., CANTAREIRA G., TELEA A.: Skeleton-based edge bundles for graph visualization. *IEEE TVCG 17*, 2 (2011), 2364 – 2373. 1, 2, 3, 4, 7, 8, 9

[Epa69] EPANECHNIKOV V. A.: Non-parametric estimation of a multivariate probability density. *Theory of Probability and its Applications 14* (1969), 153–158. 2

[FT09] FRISHMAN Y., TAL A.: Uncluttering graph layouts using anisotropic diffusion and mass transport. *IEEE TVCG 15*, 5 (2009), 777–788. 8

[GHNS11] GANSNER E., HU Y., NORTH S., SCHEIDEGGER C.: Multilevel agglomerative edge bundling for visualizing large graphs. In *Proc. PacificVis* (2011), pp. 187–194. 1, 2, 7, 9

[GK06] GANSNER E., KOREN Y.: Improved circular layouts. In *Proc. Graph Drawing* (2006), pp. 386–398. 2

[HET11] HURTER C., ERSOY O., TELEA A.: Moleview: An attribute and structure-based semantic lens for large element-based plots. *IEEE TVCG 17*, 12 (2011), 2600–2609. 2

[Hol06] HOLTEN D.: Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE TVCG 12*, 5 (2006), 741–748. 1, 2, 3

[HTC09] HURTER C., TISSOIRES B., CONVERSY S.: FromDaDy: Spreading data across views to support iterative exploration of aircraft trajectories. *IEEE TVCG 15*, 6 (2009), 1017–1024. 2

[HvW09] HOLTEN D., VAN WIJK J. J.: Force-directed edge bundling for graph visualization. *Comp. Graph. Forum 28*, 3 (2009), 670–677. 1, 2, 3, 4, 8, 9

[JMS96] JONES M., MARRON J., SHEATHER S.: A brief survey of bandwidth selection for density estimation. *J. American Stat. Assoc. 91*, 433 (1996), 401–407. 2, 3

[LBA10a] LAMBERT A., BOURQUI R., AUBER D.: 3D edge bundling for geographical data visualization. In *Proc. Information Visualisation* (2010), pp. 329–335. 2

[LBA10b] LAMBERT A., BOURQUI R., AUBER D.: Winding roads: Routing edges into bundles. *Comp. Graph. Forum 29*, 3 (2010), 432–439. 1, 2, 3, 4, 9

[PXY*05] PHAN D., XIAO L., YEH R., HANRAHAN P., WINOGRAD T.: Flow map layout. In *Proc. InfoVis* (2005), pp. 219–224. 1, 2

[QZW06] QU H., ZHOU H., WU Y.: Controllable and progressive edge clustering for large networks. In *Proc. Graph Drawing* (2006), pp. 399–404. 2

[SHH11] SELASSIE D., HELLER B., HEER J.: Divided edge bundling for directional network data. *IEEE TVCG 19*, 12 (2011), 754–763. 1, 2

[Sil92] SILVERMAN B.: Density estimation for statistics and data analysis. *Monographs on Statistics and Applied Probability 26* (1992). 2

[SJ91] SHEATHER S., JONES M.: A reliable data-based bandwidth selection method for kernel density estimation. *J. of the Royal Statistical Society B53*, 3 (1991), 683–690. 3

[SWvdW*11] SCHEEPENS R., WILLEMS N., VAN DE WETERING H., ANDRIENKO G., ANDRIENKO N., VAN WIJK J. J.: Composite density maps for multivariate trajectories. *IEEE TVCG 17*, 12 (2011), 2518–2527. 2

[TBET99] TOLLIS I., BATTISTA G. D., EADES P., TAMASSIA R.: *Graph drawing: Algorithms for the visualization of graphs*. Prentice Hall, 1999. 1

[TE10] TELEA A., ERSOY O.: Image-based edge bundles: Simplified visualization of large graphs. *Comp. Graph. Forum 29*, 3 (2010), 543–551. 1, 2, 4, 8

[TvW02] TELEA A., VAN WIJK J. J.: An augmented fast marching method for computing skeletons and centerlines. In *Proc. VisSym* (2002), pp. 251–259. 5, 7, 8

[vH03] VAM HAM F.: Using multilevel call matrices in large software projects. In *Proc. InfoVis* (2003), pp. 227–232. 1

[vLdL03] VAN LIERE R., DE LEEUW W.: GraphSplatting: Visualizing graphs as continuous fields. *IEEE TVCG 9*, 2 (2003), 206–212. 1, 2

[Wei98] WEICKERT J.: *Anisotropic diffusion in image processing, Teuber Verlag, Stuttgart, 1998.* Teuber Verlag, 1998. 4

[ZYC*08] ZHOU H., YUAN X., CUI W., QU H., CHEN B.: Energy-based hierarchical edge clustering of graphs. In *Proc. PacificVis* (2008), pp. 55–62. 2