FFTEB: Edge Bundling of Huge Graphs by the Fast Fourier Transform

Antoine Lhuillier* DEVI - ENAC Toulouse, France
Christophe Hurter† DEVI - ENAC Toulouse, France
Alexandru Telea‡ Institute Johann Bernoulli University of Groningen, Netherlands

ABSTRACT

Edge bundling techniques provide a visual simplification of cluttered graph drawings or trail sets. While many bundling techniques exist, only few recent ones can handle large datasets and also allow selective bundling based on edge attributes. We present a new technique that improves on both above points, in terms of increasing both the scalability and computational speed of bundling, while keeping the quality of the results on par with state-of-the-art techniques. For this, we shift the bundling process from the image space to the spectral (frequency) space, thereby increasing computational speed. We address scalability by proposing a data streaming process that allows bundling of extremely large datasets with limited GPU memory. We demonstrate our technique on several real-world datasets and by comparing it with state-of-the-art bundling methods.

Index Terms: I.3.3 [Computing Methodologies]: Computer Graphics—Picture/Image Generation; I.3.6 [Computing Methodologies]: Computer Graphics—Methodology and Techniques

1 INTRODUCTION

In recent years, edge bundling techniques have become increasingly popular in information visualization for the exploration of large graphs and trail sets. Bundling addresses the exploration of such data by simplifying their display by visually aggregating close (and related) edges. This way, clutter is traded off to overdraw, which yields images where the main connectivity patterns in a dataset are denser and separated by more whitespace, thus easier to spot.

Many bundling techniques have been proposed in the last decade. Yet, bundling truly large datasets is still challenging, due to two factors: computational speed and data volume. *Computational speed* has been addressed by parallelizing bundling on the GPU [19, 38, 30], using variants of the mean shift operator, which convolves the graph drawing with a distance kernel [4]. However, the large amount of overdraw occurring when doing the convolution strongly limits throughput. Also, the most efficient approaches work entirely on the GPU [38]. This restricts the types of data attributes that can be efficiently used to control the bundling. Also, the maximal *data volume* is limited by the available GPU video (VRAM) memory.

In this paper, we propose a technique that addresses the joint edge-bundling challenges of computational speed and data volume, as follows. First, we re-cast bundling from convolving in the image domain to the frequency domain, using the Fast Fourier Transform (FFT), which gives the name to our technique, FFTEB. FFTEB significantly reduces the convolution cost that dominates all image-based edge bundling techniques. It also allows to formulate directional and attribute-based edge bundling with a lower computational complexity than existing methods. Secondly, we present a streaming scheme that allows efficient transfer of data between the CPU and GPU. This makes edge bundling scalable to very large datasets, beyond what current methods can (efficiently) handle. Overall, FFTEB produces similar-quality results as state-of-the-art bundling methods, while allowing one to bundle faster, as well as bundle much larger datasets, than it has done ever before. We compare our results with nine well-known edge-bundling methods on a set of graphs ranging from thousands up to a million edges, and edge-sampling resolution up to a *billion* sample points.

The structure of this paper is as follows. Section 2 overviews existing bundling techniques, with a focus on recent image-based techniques that we aim to surpass in terms of scalability and speed. Section 3 presents the mathematical model, design rationale, and implementation details of FFTEB. Section 4 presents results of FFTEB on very large datasets and compares it with existing bundling methods. Section 5 compares our scalability with recent bundling techniques and also discusses our method. Section 6 concludes the paper.

2 RELATED WORK

We refine the formal model proposed in [19] to describe edge bundling as follows. Let $G = (V,E)$ be a graph with nodes $V = \{\mathbf{v}_i\}$ and edges $E = \{\mathbf{e}_i\}$. Let $D : E \rightarrow \mathbb{R}^2$ be a drawing operator. In this notation, $D(\mathbf{e})$ is the drawing of an edge $\mathbf{e}$ of G, which is a straight line (for a typical graph drawing) or a planar curve (if the input dataset is a trail set). As a shortcut, let $D(G)$ be the drawing of the entire graph G, and let $\mathscr{D} \subset \mathbb{R}^2$ be the space of all such graph drawings. Let $B : \mathscr{D} \rightarrow \mathscr{D}$ be an operator that denotes the bundling process for a whole graph; and finally let $B(D(\mathbf{e}))$ denote the curve representing the bundling of edge $\mathbf{e}$. B is a typical EB algorithm if

$$\forall(\mathbf{e_i} \in G, \mathbf{e_j} \in G) | \kappa(\mathbf{e}_i, \mathbf{e}_j) < \kappa_{max} \rightarrow \\ \delta(B(D(\mathbf{e}_i)), B(D(\mathbf{e}_j))) \ll \delta(D(\mathbf{e}_i), D(\mathbf{e}_j)). \quad (1)$$

Here, δ is a distance function between curves in $\mathbb{R}^2$, such as the Haussdorff distance. $\kappa : G \times G \rightarrow \mathbb{R}^+$ is a so-called compatibility function that captures how dissimilar edges are. κ accounts for spatial similarity in $D(G)$, *i.e.*, $\kappa(\mathbf{e}_i, \mathbf{e}_j)$ is proportional with $\delta(D(\mathbf{e}_i), D(\mathbf{e}_j))$, but can also incorporate other edge properties, such as directions or data attributes [30]. Only edges that are more similar than a threshold κ_{max} should be bundled. Essentially, Eqn.1 states that, for compatible edges, their bundled drawings are spatially much closer than their unbundled drawings.

As outlined in Sec. 1, EB reduces edge-edge clutter in a graph drawing $D(G)$ by grouping compatible edges into spatially compact and thin bundles. This eases the visual detection of coarse-scale connectivity patterns in the drawing and supports tasks such as finding node-groups which are connected to each other by edge-groups (bundles) [12, 36]. Similar simplification strategies exist, *e.g.*, map generalization in cartography [3]. Clutter causes and reduction strategies are discussed by Ellis and Dix [9] and Zhou *et al.* [41].

Edge bundling is part of a larger family of graph (drawing) simplification techniques. These can be organized in three types *vs* the edge-grouping procedure being used [25], as follows.

Data-based methods: Such methods reduce clutter by drawing a simpler graph G' obtained by either filtering away, or by aggregating, edges from G. Edge clustering methods are described by Ellis and Dix [9]. Interactive methods, such as NodeTrix, compact dense

*e-mail: antoine.lhuillier@enac.fr
†e-mail:christophe.hurter@enac.fr
‡e-mail:a.c.telea@rug.nl

subgraphs into matrix representations [15, 14]. Ploceus displays networks using different perspectives, abstraction levels, and edge semantics [27]. Filtering can be done by user queries based on multivariate criteria or by automated methods based on edge metrics such as centrality [39] or spanning trees [24]. Such methods are not in our scope, as edge bundling simplifies $D(G)$ rather than G itself.

Geometry-based methods: Early EB methods implement Eqn. 1 by using various computational geometry techniques to both assess inter-edge distances (δ in Eqn. 1) and to route the bundled edges. Dickerson *et al.* merge edges by reducing non-planar graphs to planar ones [7]. Flow maps use a binary clustering of $D(G)$ to route curved edges along [31]. Flow map control-meshes are used to cluster and route curved edges [34, 42]. Holten generalized this idea by routing edges of compound graphs along the graph's hierarchy drawing [16]. Gansner and Koren bundle edges in a circular node layout by area optimization metrics [13]. Edge routing can also use Delaunay triangulations (Geometry-Based Edge Bundling (GBEB) [6]) and Voronoi diagrams (Winding Roads (WR) [22, 21]) based on the node positions. The most popular geometric methods are force-based techniques that use a force field designed to equal the gradient of a quality function to optimize(Force-Directed Edge Bundling (FDEB) [17] and [8]), and have been adapted to separate bundles running in opposite directions [35]. A major issue of geometric techniques is computational complexity. To reduce this, graph optimization techniques [33] and multilevel clustering numerical methods have been proposed (MINGLE, [12]).

Image-based methods: Recent methods compute B (Eqn. 1) via image-processing operations, which efficiently parallelize on the GPU. Skeleton-based edge bundling (SBEB [11]) use the GPU-computed medial axes of the thresholded distance-transform of $D(G)$ as bundling cues to yield strongly ramified bundles. Recent methods use the mean-shift principle [4]: The drawing $D(G)$ is seen as an edge-density field ρ, and B amounts to shifting edges $D(\mathbf{e})$ upstream in $\nabla\rho$. Mean shift maps to simple GPU-parallelizable 2D image processing operations. Kernel density estimation edge-bundling (KDEEB [19, 20]) methods offer speed-ups of up to two orders of magnitude compared to geometric methods, *e.g.* FDEB. Recent extensions include directional bundling, based on edge-direction histograms (3DHEB [29]); and attribute-driven bundling (ADEB [30]) which defines κ using any edge attributes. CUBu implements the full mean-shift on the GPU, with careful optimizations, yielding the fastest existing EB method [38]. KDE methods have also been used to bundle dynamic graphs [23, 20] and 3D fibers [2]. Besides very fast bundling, image-based methods offer new ways to visualize $D(G)$, *e.g.* pseudo-shading the edge density ρ to highlight important bundles [22, 12, 19]; drawing bundles as compact shapes emphasized by shaded cushions [36, 11, 38]; and using animated textures to convey edge directions for static and dynamic graphs [20].

3 FFTEB Framework

We next describe our FFTEB method for bundling very large graphs using the spectral domain. Section 3.1 outlines the KDEEB technique which we are inspired from, and outlines related scalability issues. Section 3.2 introduces our method. Sections 3.3 and 3.4 cover implementation details and our proposed GPU streaming technique for handling very large datasets, respectively.

3.1 Kernel Density Bundling

Kernel density bundling (KDE) methods first compute an edge-density map $\rho : \mathbb{R}^2 \to \mathbb{R}^+$ by convolving all edges $\mathbf{e}$ in a given graph drawing $D(G)$ with a decaying radial kernel $K : \mathbb{R}^2 \to \mathbb{R}^+$ of support radius h, typically Gaussian or Epanechnikov (parabolic) [10]:

$$\rho(\mathbf{x} \in \mathbb{R}^2) = \sum_{\mathbf{y} \in D(G) | \|\mathbf{x}-\mathbf{y}\| \le h} K(\mathbf{y}-\mathbf{x}). \qquad (2)$$

Next, all points $\mathbf{x} \in D(G)$ are advected in the gradient of ρ with a small distance ε, to yield a new drawing

$$D^{new}(G) = \left\{ \mathbf{x}^{new} = \mathbf{x} + \varepsilon \frac{\nabla\rho}{\|\nabla\rho\|} \,\middle|\, \mathbf{x} \in D(G) \right\}, \qquad (3)$$

Next, a 1D Laplacian smoothing pass is applied over all edges $\mathbf{e} \in D^{new}(G)$, needed to regularize the estimation of $\nabla\rho$. The process is repeated a small number of I iterations (typically around 10) to yield the final bundled drawing $B(D(G))$. All above steps can be efficiently done on the GPU: Eqn. 2 maps to convolving $D(G)$ with a 2D texture encoding K; Eqn. 3 and the Laplacian smoothing map to simple computations done on a point-sampling of $D(G)$. The drawing $D(G)$ is resampled at each iteration to ensure a good spatial sample density, needed for a good estimation of ρ by Eqn. 2. The final bundled graph is drawn using an opacity map given by ρ, to emphasize high-density bundles. For details, we refer to [19, 38].

Performance: The complexity of KDE bundling strongly depends on the chosen computational model. Consider bundling a graph $D(G)$ having N sampling points, using I bundling iterations, a kernel K of diameter h pixels, and an image of $R \times R$ pixels. KDEEB implements Eqn. 2 by splatting (scattering) K over all sample points of $D(G)$, yielding a complexity of $O(INh^2)$ [19]. As h should be proportional to R to ensure good results, the complexity becomes $O(INR^2)$. CUBu speeds this up by about 50 times by using a gathering strategy where Eqn. 2 is evaluated at each image point based on nearby sampling points [38]. Additionally, the 2D convolution in Eqn. 2 is split into two 1D convolutions, since the kernel K is separable, *i.e.*, $K(x \in \mathbb{R}, y \in \mathbb{R})$ can be written as the product $K_x(x)K_y(y)$ of two 1D kernels K_x and K_y. This yields a complexity of $O(I(N+R^2h))$. Given the relation of h with R, this is equivalent to $O(I(N+R^3))$ worst-case. The cubic complexity in image size creates serious scalability issues when creating high-resolution bundled images. For instance, CUBu requires over 0.7 seconds to bundle a graph of about 4M sample points on a modern GPU (NVidia GTX 690) at 2048^2 screen resolution. Bundling a graph of one million edges (not huge by nowadays big data standards), each sampled with 50 points on average, would cost CUBu over eight seconds. These are too slow rates if we aim at interactive exploration of large graphs.

Attribute-driven bundling: As outlined in Sec. 2, a high-interest case for bundling concerns datasets where the edge compatibility κ is a function of both spatial position and data attributes of edges. Examples of such methods are ADEB [30] which defines edge compatibility based on directions and/or data attributes, and CUBu [38], which defines edge compatibility based on directions only. These methods essentially replace Eqn. 2 with

$$\rho(\mathbf{x} \in D(G)) = \sum_{\mathbf{y} \in D(G) | \|\mathbf{x}-\mathbf{y}\| \le h} \kappa(\mathbf{a}(\mathbf{x}), \mathbf{a}(\mathbf{y})) \cdot K(\mathbf{y}-\mathbf{x}) \qquad (4)$$

where $\mathbf{a} : E \to \mathbb{R}^m$ is a m-dimensional attribute defined on the graph edges, and $\kappa : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is a compatibility function that tells how similar two such attributes are. Setting $\mathbf{a}$ to the tangent vector of an edge and κ to the scalar product over $\mathbb{R}^2$ yields directional bundling [30, 38]. However, Eqn. 4 is *not* always separable in two 1D convolutions as Eqn. 2 was. Hence, the cost of ADEB's and CUBu's attribute-driven bundling is $O(I(N+R^2h)$ (cost of computing the undirected density map, needed for shading) plus $O(INh^2)$ (cost of Eqn. 4, identical to KDEEB), *i.e.* a total of $O(I(NR^2+R^3))$. The speed of CUBu is thus largely lost for attribute-driven bundling.

3.2 Fourier Transform Approach

To address the scalability issues outlined above, we exploit the properties of the convolution theorem: If f and g are integrable functions with Fourier transforms $\mathscr{F}[f]$ and $\mathscr{F}[g]$ respectively, the Fourier transform of the convolution $f * g$ equals the product of the Fourier

transforms of f and g, *i.e.*

$$\mathscr{F}[f * g] = \mathscr{F}[f] \cdot \mathscr{F}[g]. \tag{5}$$

If we note that the density ρ in Eqn. 2 is nothing but the convolution $K * D(G)$, we derive from Eqns. 2 and 5 that

$$\rho(\mathbf{x}) = \mathscr{F}^{-1}[\mathscr{F}[D(G)] \cdot \mathscr{F}[K]](\mathbf{x}), \tag{6}$$

where $\mathscr{F}^{-1}$ denotes the inverse Fourier transform and $\cdot$ denotes point-wise signal multiplication.

Performance: Equation 6 can be implemented on the GPU more efficiently than Eqn. 2. First, we note that Eqn. 2 needed to find all points $\mathbf{y} \in D(G)$ within a radius h from every image pixel $\mathbf{x}$. For this, CUBu computes a special map recording the number of sample points of $D(G)$ falling over each pixel $\mathbf{x}$, and this computation needs atomic write operations to parallelize ([38], Sec. 3.1). In contrast, Eqn. 6 only reads and writes every image pixel once. Secondly, approximating $\mathscr{F}$ with the Discrete Fourier Transform [5] for an input image of R^2 pixels requires a spectral map of only $((R-1)/2)^2$ pixels – which is roughly four times less than the image size needed for Eqn. 2. Overall, computing ρ with Eqn. 6 requires a direct and an inverse Fourier transform for each of the I bundling iterations, followed by a simple point-wise signal multiplication. Using the CUFFT CUDA library for FFT computations, this amounts to $O(R^2 \log R)$ operations per input image of $R \times R$ pixels [32] per iteration, thus a total FFTEB complexity of $O(I(N+R^2 \log R))$, which is lower than CUBu's $O(I(N+R^3))$.

Attribute-driven bundling: Our FFT approach becomes even more valuable for attribute-driven bundling, if we restrict κ to be bi-linear. Directional bundling, where κ is a scalar product, meets this condition. In detail, from Eqn. 4, we compute the gradient

$$\nabla\rho(\mathbf{x} \in D(G)) = \sum_{\mathbf{y} \in D(G) | \|\mathbf{x}-\mathbf{y}\| \le h} \kappa(\mathbf{a}(\mathbf{x}), \mathbf{a}(\mathbf{y})) \cdot \nabla K(\mathbf{y}-\mathbf{x}). \tag{7}$$

As κ is bi-linear, we can simplify Eqn. 7 to

$$\nabla\rho(\mathbf{x} \in D(G)) = \Bigg(\kappa\Big(\mathbf{a}(\mathbf{x}), \sum_{\mathbf{y} \in D(G) | \|\mathbf{x}-\mathbf{y}\| \le h} \mathbf{a}(\mathbf{y}) \cdot \nabla_x K(\mathbf{y}-\mathbf{x})\Big), \kappa\Big(\mathbf{a}(\mathbf{x}), \sum_{\mathbf{y} \in D(G) | \|\mathbf{x}-\mathbf{y}\| \le h} \mathbf{a}(\mathbf{y}) \cdot \nabla_y K(\mathbf{y}-\mathbf{x})\Big)\Bigg). \tag{8}$$

Using the convolution theorem (Eqn. 5), we can rewrite Eqn. 8 as

$$\nabla\rho(\mathbf{x} \in D(G)) = \Bigg(\kappa\Big(\mathbf{a}(\mathbf{x}), \mathscr{F}^{-1}\left[\mathscr{F}\left[\mathbf{a}\right] \cdot \mathscr{F}\left[\nabla_x K\right]\right](\mathbf{x})\Big), \kappa\Big(\mathbf{a}(\mathbf{x}), \mathscr{F}^{-1}\left[\mathscr{F}\left[\mathbf{a}\right] \cdot \mathscr{F}\left[\nabla_y K\right]\right](\mathbf{x})\Big)\Bigg), \tag{9}$$

where $\mathscr{F}[\mathbf{a}]$ is the Fourier transform of the attribute signal $\mathbf{a}$ defined over $D(G)$. Several differences between this model and CUBu and ADEB exist, as follows. First, Eqn. 9 has the same complexity as Eqn. 6, *i.e.*, $O(I(N+R^2 \log R))$, which is smaller than CUBu-directional and ADEB's $O(I(NR^2+R^3))$ (see Sec. 3.1). In particular, the costly NR^2 term in the latter is not present in our formulation. This is due to the efficiency of FFT-based convolution, which is higher than classical convolution for non-separable large kernels, and also due to the bi-linearity property of κ. Secondly, our choice of compatible functions is more flexible than ADEB – for example, for directional bundling, we can use the scalar product of edge tangents (like CUBu) which attracts compatible edges and repels incompatible ones. In ADEB, only the attraction part was present.

3.3 Implementation Details

FFTEB's pipeline is similar to KDEEB (Fig. 1). In detail, the m scalar components $(a_1, \ldots, a_m)$ of $\mathbf{a}$ are stored in m floating-point textures, and a DFT $\mathscr{F}[a_i]$ is computed for each. The kernel gradient DFT $\mathscr{F}[\nabla K]$ is computed once as it does not change during bundling. Next, the point-wise products of each $\mathscr{F}[a_i]$ and the two components $\mathscr{F}[\nabla_x K]$ and $\mathscr{F}[\nabla_y K]$ (with K repeated to reach the same size as ρ) are computed, yielding $2m$ textures. The inverse Fourier transform of each such texture is next computed, yielding another $2m$ textures. Next, $\nabla\rho$ is computed (Eqn. 9) by evaluating two m-dimensional compatibilities of $\kappa(\mathbf{a})$ with the two corresponding m-dimensional components of the convolutions of $\mathbf{a}$ with the kernel gradient components $\nabla_x K$ and $\nabla_y K$, respectively. Finally, we apply advection (Eqn. 3), Laplacian smoothing, and resampling of $D(G)$ as in Sec. 3.1. We implement all above in NVidia's CUDA using the CUFFT library for fast computation of direct and inverse DFT's. An implementation of FFTEB is provided at [26].

Bundled edges are rendered as curves colored by the value of one attribute interest, using the pseudo-shading based on the density map height described in [38]. We additionally emphasize important (high-density) bundles by scaling the thickness of each drawn bundled edge with the local density value $\rho(\mathbf{x})$. This way, important bundles are rendered *both* thicker and with a more prominent shading. This lets us quickly discover these even in complex visualizations, as we shall see in Sec. 4.1.

3.4 Scalability and Streaming

As outlined in Sec. 1, GPU-based bundling is challenged by large graphs. Consider the *amazon* graph [12, 38], which has over 900K edges. A dense point sampling of this graph at a resolution of 2000^2 pixels (1 sample point every few pixels) easily yields $N > 1$ billion points. Per point, we need to store at least its two floating-point coordinates (8 bytes of storage), yielding a total need for at least 8GB VRAM. Edge attributes $\mathbf{a}(\mathbf{x})$ add extra storage. Such large datasets do not fit in the VRAM of current consumer-grade graphics cards.

We address this data volume challenge by a simple but efficient streaming scheme for FFTEB bundling, as follows. We divide the sampled edges of $D(G)$, which is stored on the CPU, into C chunks c_i, each having roughly N/C sample points, so that a chunk fits in the available VRAM. We next stream each chunk to the GPU, and accumulate its density-map gradient $\nabla\rho_i$ (computed as described in Sec. 3.2), in a global gradient map $\nabla\rho$. After all chunks have added their contribution to $\nabla\rho$, we stream them again, one by one, to the GPU, to advect, smooth, and resample their edges. After a chunk is processed, we stream its new sample points back to the CPU to make space for the next chunk. This process requires $2C$ CPU-to-GPU and

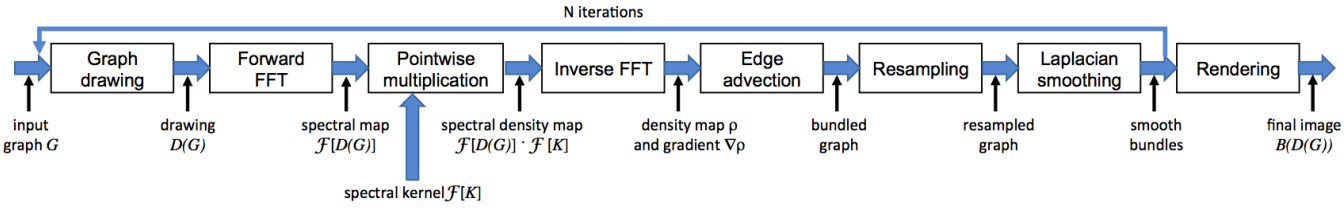

Figure 1: Flowchart of FFTEB.

C GPU-to-CPU chunk transfers, *i.e.* a total data transfer of sizes $2N$ (CPU-to-GPU) and N (GPU-to-CPU), respectively. Compared to the case where the entire sampled graph fits in VRAM, we thus pay only an extra cost of size N per iteration. Examples of using streaming for very large graphs and streaming performance considerations are given in Secs. 4.4 and 5, respectively.

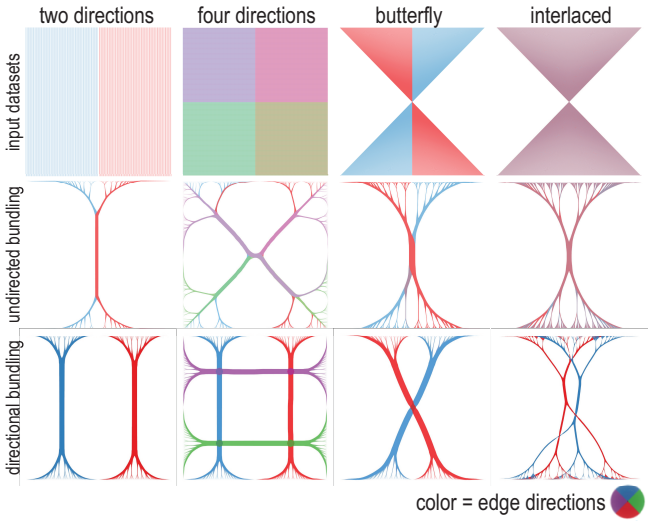

Figure 2: FFTEB undirected and directional bundling of four datasets.

3.5 Simple Bundling Example

Similarly to the tests used by ADEB [30], we illustrate FFTEB via four simple datasets bundled using the undirected, respectively, directional compatibility criteria (Fig. 2). Edges are color-coded to indicate direction. Each dataset has 2000 edges. As visible, undirected bundling (middle row) creates patterns which do not convey an intuitive simplification of the data – for instance, the undirected bundlings of the *two directions*, *butterfly*, and *interlaced* datasets look very similar, while the corresponding datasets are quite different. The bottom row shows the FFTEB directional bundling, using the scalar product on $\mathbb{R}^2$ as compatibility function. As visible, the bundled graphs now capture well the structure of the input datasets. Overall, thus, FFTEB reaches the same bundling quality as ADEB.

4 Example Applications

We next illustrate FFTEB with three studies: bundling quality as a function of resolution (Sec. 4.2); attributed-driven bundling of eye-tracking data (Sec. 4.3); and directional bundling of huge graphs that do not fit in VRAM using streaming (Sec. 4.4). These use-cases outline together the three main capabilities of FFTEB – computational speed, handling attributed graphs, and handling large data volumes.

4.1 Comparison

Figure 4 compares the undirected FFTEB bundling with eight other well-known bundling methods (FDEB, WR, KDEEB, SBEB, 3DHEB, GBEB, CUBu, and MINGLE). As datasets, we used the *US migrations* (9780 edges) and *net50* (928K edges). For the relatively small *US migrations* dataset, FFTEB produces results which are very similar in terms of positions and structure of the main bundles to FDEB, KDEEB, 3DHEB, and CUBu. The small differences are explained by different parameter values used for the kernel size, amount of Laplacian smoothing, advection speed, and shading style (which are not exactly replicable from the aforementioned papers). These similarities are expected, as FFTEB shares the same bundling principle with all the named methods. SBEB and GBEB produce, as expected, different results since they use a different model for edge routing. For the much larger *net50* dataset, FFTEB again produces results which are very similar with KDEEB and CUBu, and notably a much stronger simplification of the graph structure than KDEEB and MINGLE. This is explained by the fact that, for this example, both FFTEB and CUBu use a kernel size h (Eqn. 2) about three times larger than KDEEB. In turn, this is due the quadratic complexity of KDEEB in h (Sec. 3.1), which makes this method impracticable for strongly simplifying very large graphs like *net50*. The lower simplification of MINGLE is explained by its decision to bundle an edge with maximally $k = 10$ nearest neighbors thereof (see [12], Sec. 4). As noted there, using higher bundling factors does not decrease the saved ink to draw the bundled graph. However, this also decreases the amount of simplification one can achieve.

Figure 5 compares both the directional and undirected FFTEB bundling with the respective variants of CUBu for the *France airlines* dataset. We see that FFTEB and CUBu methods deliver similar results, which is explained by the fact that they use the same underlying algorithm (kernel density bundling, explained in Sec. 3.1). In theory the two bundling results should be the same but further investigation should be done to validate this theoretical hypothesis. The differences may come from the data transformation accuracy between spectral and image space. The less wiggling bundles in FFTEB allow a better end-to-end tracing of directed bundles, see *e.g.* the marked details in Figs. 5e,f. As before, such differences are explained by small parameter-value differences, and should not be interpreted as an inherent higher-quality of the FFTEB method. The rightmost images (Figs. 5c,f) show a strong simplification of the directional and undirected FFTEB bundlings, done by rendering the respective bundled graphs with an opacity proportional to the local edge-density and by setting edge line-widths to the same density (Sec. 3.3). This both reduces clutter caused by spurious edges or low-importance bundles and emphasizes the largest bundles in the graph.

4.2 Bundling Quality *vs* Resolution

As stated in Sec. 3.2, FFTEB's complexity is $O(I(N+R^2 \log R))$ for N samples, I iterations, and a resolution of $R \times R$ pixels. Clearly, R has the highest influence on computational speed. It seems tempting to use lower resolutions R to accelerate computations. However, for very large graphs, this can create problems in distinguishing finer-scale details, as discussed next. To study the effect of R on the results, we consider the *amazon* graph, that contains records of over 900K co-purchase relations between 520K items on *amazon.com* [12]. We bundle this graph at three different resolutions ($R^2 \in \{100^2, 400^2, 1000^2\}$ pixels). We set the kernel size h (Eqn. 2) to a value of $R/10$, following [19, 30, 38], and also adapt the sampling step of edges to be about 3 pixels for each R value, again in line with earlier studies [38].

Figure 3 shows the original unbundled graph and bundling results using MINGLE, KDEEB, CUBu, and FFTEB. As visible, both MINGLE and KDEEB do not achieve much in terms of highlighting structure in the bundled result, apart from being able to show high-density areas. As explained in Sec. 4.1 for the *net50* graph, such limitations are due to their relative low bundling strength, which is in turn justified by computational cost (KDEEB) and the desire to minimize edge bending (MINGLE), respectively. For large graphs like *amazon*, such constraints are not very effective. The density-based shading of CUBu highlights quite well the most important (densest) bundles at a resolution of 500^2 pixels (Fig. 3d). At similar resolutions, FFTEB and CUBu generate quite similar results, which is not surprising, given that they are based on the same kernel-density process (Eqns. 2, 3). The last three images, computed with FFTEB show the added-value of using a high resolution: Consider, for instance, two points A and B which appear to be part of the same bundle. Using low-to-middle resolution bundlings (Figs. 3d-f) lets one see a path c between A and B indicated by the respective red dotted lines. However, when we increase the resolution, we see that c actually

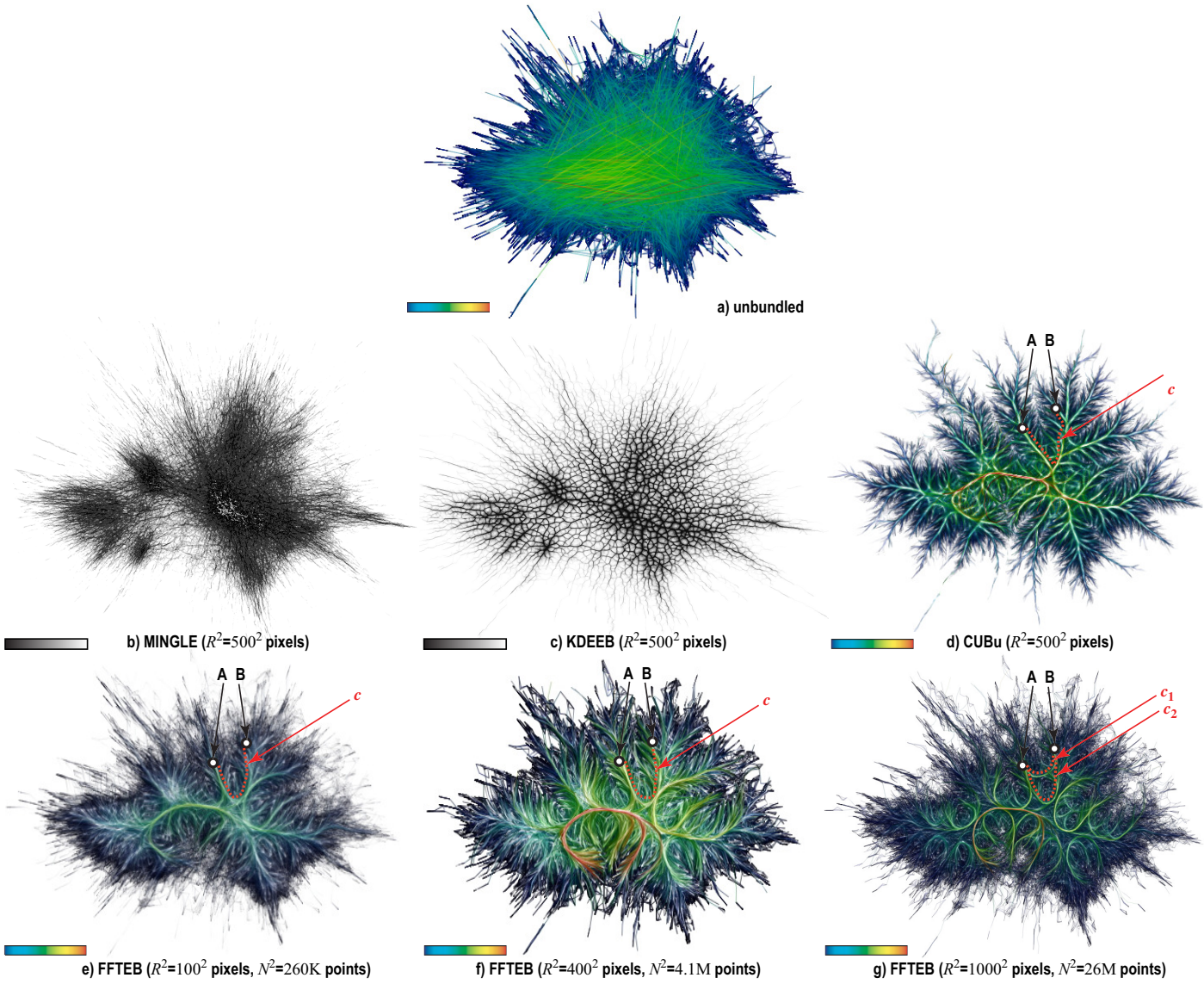

Figure 3: Bundling of *amazon* graph [12] for different screen resolutions R and total number of edge sampling points N. Colors map edge density.

splits in two paths c_1 and c_2 (Fig. 3g), where c_2 is a shorter path connecting A and B which was not visible at lower resolutions. Hence, high resolutions are needed to obtain detailed bundling insights for very large graphs. Since FFTEB scales better than all other recent fast bundling methods in terms of high resolution R and graph size (number of sample points N), this method is clearly more suitable for generating such detailed images. We further present figures to support this higher scalability of FFTEB in Sec. 5.

4.3 Attribute-Driven Bundling

Our next study considers a dataset containing eye tracking data. In the underlying use-case, described in full detail in [30, 18], an eye tracker was used to record the motion of the eyes of a pilot that looks at a plane's cockpit dashboard while performing a landing manoeuvre in a flight simulator. The recorded data can be represented as a graph where vertices are points to which the eyes where drawn (so-called fixation points), and the connecting edges indicate eye-movement between the vertices (so-called saccades). As detailed in [18], the aim of analyzing such data is to detect high-level repetitive patterns in the eye movement, which in turn let one understand how well the subject visually scanned the dashboard instruments to perform the required maneouvres. Drawing the raw saccades between fixation points generates a completely cluttered image, from which high-level connections or connection patterns between fixation points cannot be inferred (Fig. 6a). Undirected bundling can be used to reduce clutter and extract a few simple eye-motion patterns from the data (Fig. 6b). However, such an image is actually simplifying too much, as it bundles together saccades that run in opposite directions. Directional bundling corrects this by separating saccades that link the same spatial areas but have opposing directions. Figure 6c,d show the results of ADEB and CUBu for directional bundling. While these images show more detail than undirected bundling, as explained above, they still suffer from a certain amount of clutter, visible in terms of stray saccades which have not been bundled succesfully. The explanation of these is technical: Since ADEB and CUBu do not scale very well for directional bundling, as explained earlier and also detailed by timings in Sec. 5.1, the number of sample points N and the kernel size h cannot be too high if we want to obtain a high bundling speed. Hence, trails which are too far away will not be bundled, and will appear as stray curves. In contrast, FFTEB allows using large values for N and h without sacri-

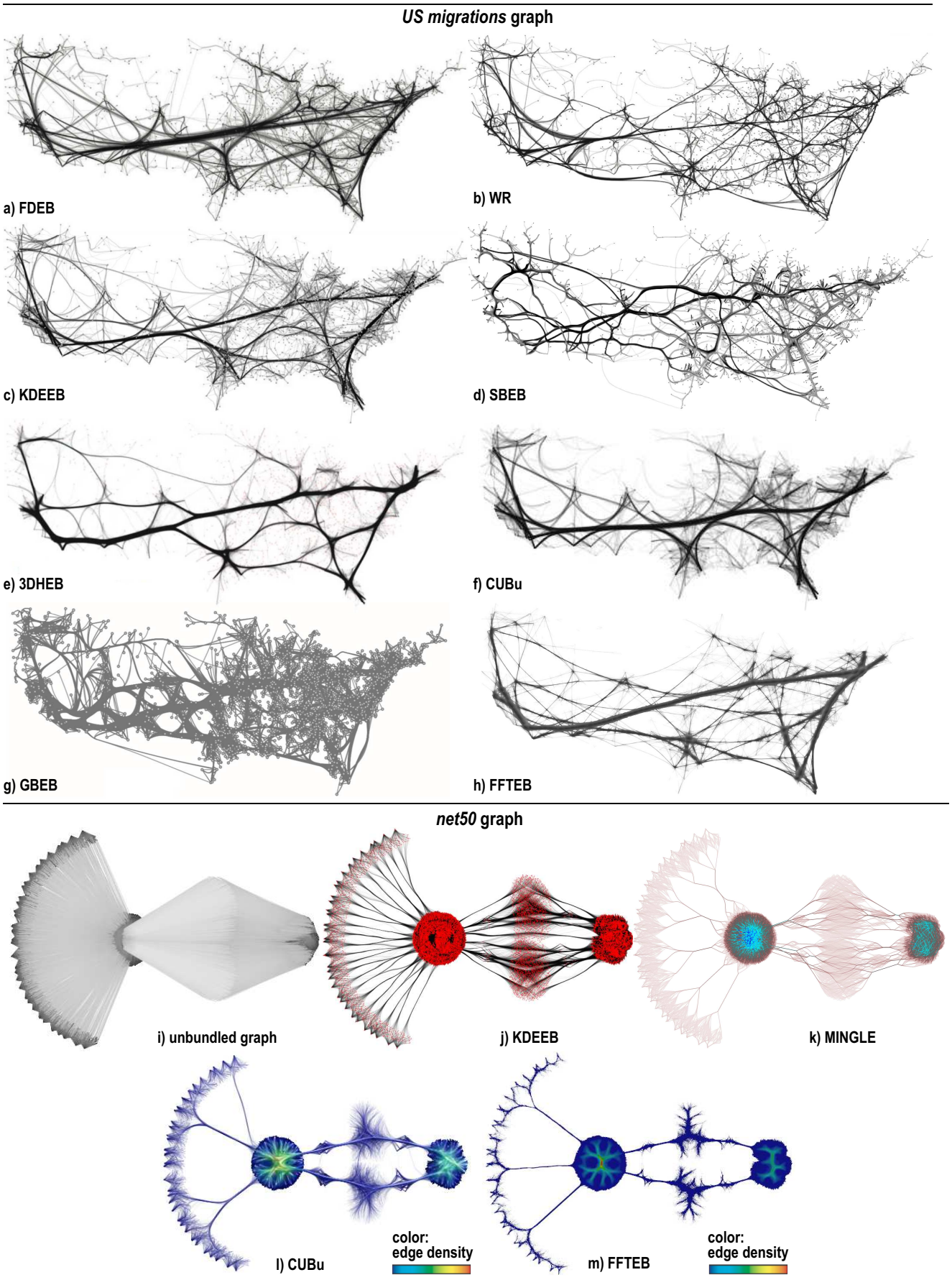

Figure 4: Comparison of undirected FFTEB with several state-of-the-art bundling methods for the *US migrations* (top) and *net50* (bottom) datasets. See Sec. 4.1.

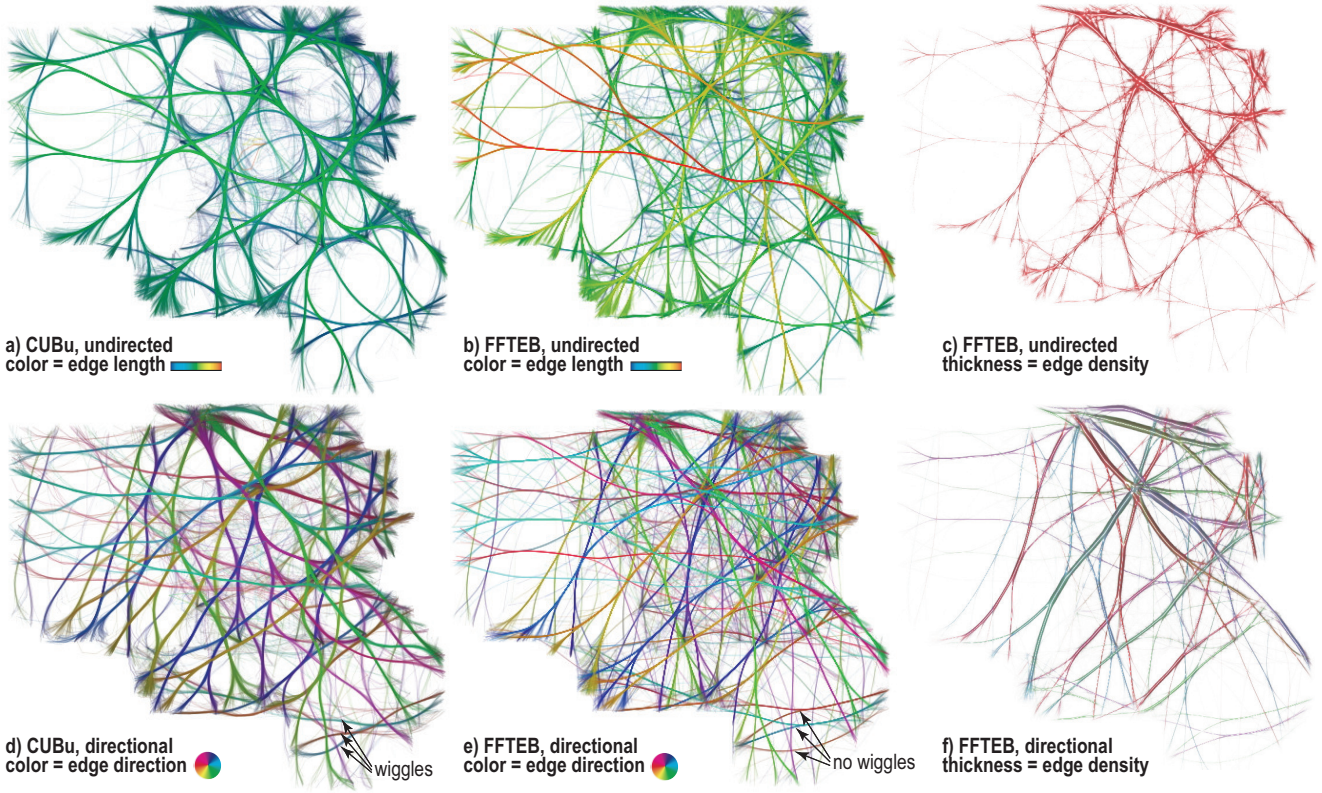

Figure 5: Comparison of directional and undirected FFTEB with the corresponding CUBu variants for the *France airlines* dataset. See Sec. 4.1.

ficing performance (see Sec. 5.1 further for performance figures). In turn, this allows a tighter bundling that yields cleaner, more simplified, images – compare Fig. 6e with Fig. 6c,d. We should stress that FFTEB's higher quality bundling is a consequence of its ability to use parameter ranges for which earlier methods become impractical, and not a result of it using a different bundling heuristic – for the same parameter values, FFTEB, ADEB, and CUBu yield very similar images, up to local differences caused by implementation details.

Figure 6f shows a further variation of attribute-based bundling done with FFTEB. Here, the edge similarity κ (Eqn. 7) combines both direction and time-information from the trails, each having the same weight. That is, trails are bundled if they run in similar directions and at similar time instants. The result is visually quite similar to the direction-only bundling. This tells us that eye saccades that run in different directions in the image, *i.e.* present in different bundles, occur at quite different moments during the recorded sequence – if not, then such trails would be bundled together due to their high time similarity, and thus Fig. 6f would look quite different from Fig. 6e.

4.4 Very Large Directional Graph Bundling

Our last study shows how FFTEB (directionally) bundles very large graphs whose point-samplings do not fit in VRAM. For this, we use a *migrations* graph of 600K edges that describes the relocation of people in the US over one year [37]. Sampling this graph at a screen resolution of 1000^2 pixels, using the sampling step constraints of KDE-based methods discussed earlier, yields over 63 *billion* points. This is three *orders of magnitude* larger than the largest sampled graph we know to have been bundled ([38], *amazon* graph, 19M sample points). While earlier bundling techniques considered subsets of this graph ([12], 9660 edges; [38, 22, 17], 9780 edges; [6], 9798 edges), this is the first time the entire dataset is bundled. Also, this is the largest example of *directional* bundling known to us so far (600K edges *vs worldflights* graph in [38], 26K edges). Our 63 billion point sampling requires over 15GB of memory, so the streaming feature of FFTEB is needed to handle this dataset on any current GPU. Performance considerations for this graph are discussed in Sec. 5.

Figure 7(top) shows the large US *migrations* graph bundled by FFTEB, with edge density encoded into bundle width and edge directions encoded by colors. The image reveals three main migratory west-to-east fluxes (Fig. 7, thick green bundles WE) and two main fluxes in the opposite direction (Fig. 7, thick purple bundles EW). On the vertical direction, we find that the eastern half of the country has much more migration than the western half. For the former, migration can be summarized by a flow going to Texas (Fig. 7, red bundle T), two flows ending in Florida (Fig. 7, red bundles F1 and F2); and one flow leaving Florida towards the Minnesota and New York areas (Fig. 7, blue branching bundle F3). As visible, FFTEB can succesfully produce an image that conveys a simplified, yet clear, view of the main migratory patterns in this very large dataset.

As a comparison, Figure 7(bottom) shows the small US *migrations* graph, directionally bundled by FFTEB. This is the same graph as used in Fig. 4a-h (9780 edges), but bundled directionally this time. As visible, the small bundled graph conveys a quite different insight from the full graph: The overall picture in terms of connection patterns is different, and the distribution of bundle thicknesses is very different. This indicates that a *subsampled* graph cannot always convey the same insight as the full graph. Hence, using an entire, non-subsampled dataset is the best option to avoid such subsampling issues. The added-value of FFTEB becomes clear here, as our method lets us do this even for very large graphs (large N values).

5 Discussion

We next discuss several aspects of FFTEB. First, we compare in Sec. 5.1 the computational performance of FFTEB with the fastest-known bundling techniques we know – KDEEB, CUBu, and ADEB. As a benchmark, we use the six graphs in Tab. 1, which appear in all recent bundling papers. In Sec. 5.2, we analyze FFTEB's performance as a function of its two free parameters – image resoluton R

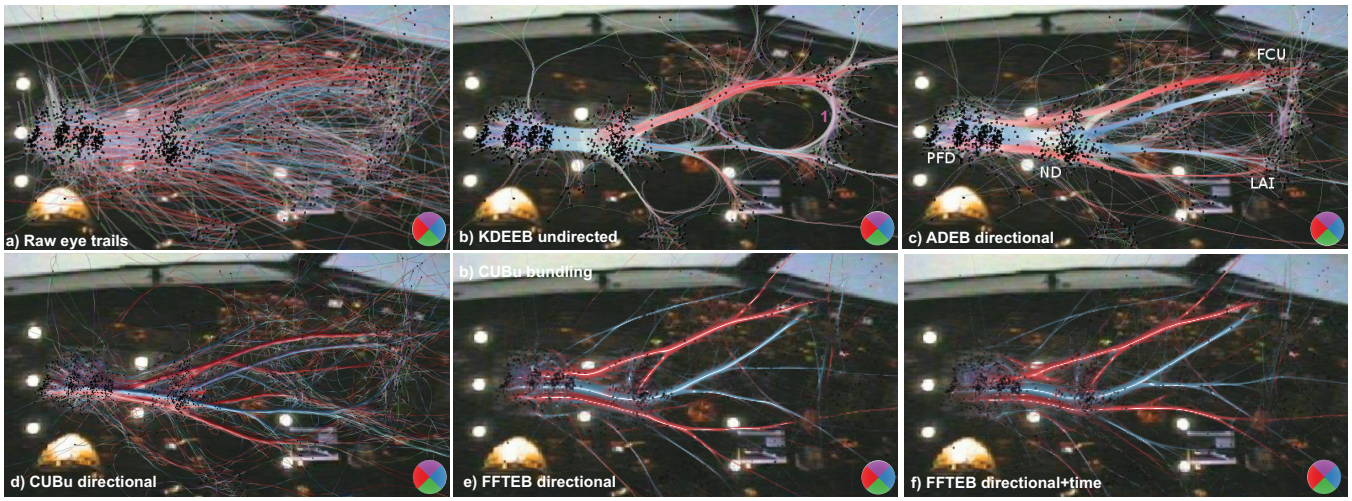

Figure 6: Bundling of eye trails for airline pilot training. See Sec. 4.3.

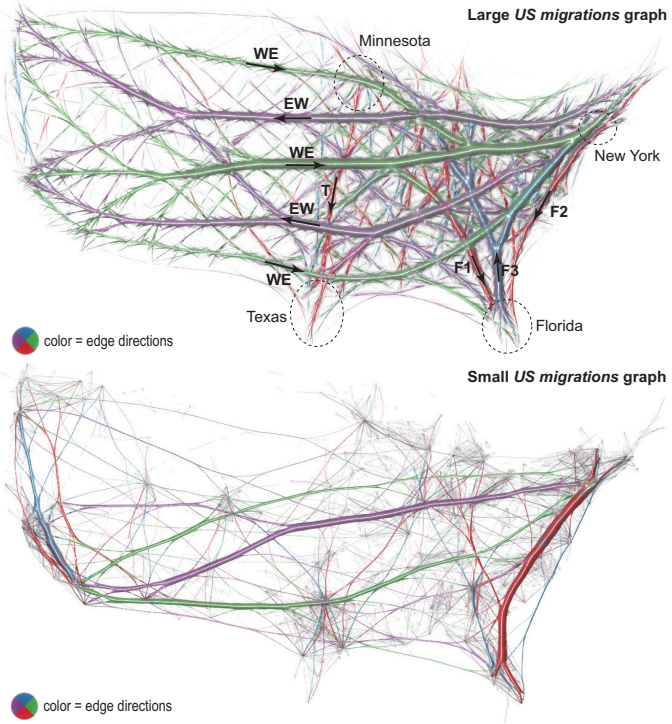

Figure 7: Top: Directional bundling, large US *migration* graph [37] (600K edges, 63 billion sampling points). Bottom: Bundling of the much smaller *migration* graph (9780 edges, 290K sampling points).

and graph sample-size N. We summarize the advantages and limitations of FFTEB in Sec. 5.3

5.1 Perfomance Comparison

For a start, we must note that the compared bundling methods treat attributed graphs differently. KDEEB cannot do attribute-based bundling. CUBu offers attribute-based bundling in terms of edge directions, apart from undirected bundling. FFTEB does attribute-based bundling by definition. For a fair comparison, we thus compare KDEEB (which does no attribute-based bundling) with FFTEB (for which we set the compatibility function κ (Eqn. 9) to identity to emulate undirected bundling) and with CUBu (undirected version); and separately ADEB with the directional version of CUBu and with FFTEB.

Undirected bundling: All tests were done using a single-GPU 4GB NVidia GTX 690 graphics card. The tests presented here use an image resolution of 1000^2 pixels and a kernel size of $h = 21$ pixels (we have done more tests, but cannot present them all due to space constraints). We adapted the edge-sampling parameters in all tested methods (KDEEB, ADEB, CUBu, FFTEB) so as to generate roughly the same number of sample points N for a graph. Exactly identical N values for the three tested methods (KDEEB, CUBu, FFTEB) could not be obtained, since N is not an explicit parameter of any of these methods, but a result of setting an edge-sampling-density parameter. Table 1 shows the obtained timings. We see that both FFTEB and CUBu are 50 to 100 times faster than KDEEB. The important message, however, is that FFTEB is up to 2 times faster than CUBu, especially for larger datasets. This makes FFTEB the fastest undirected-bundling method in existence.

Directional bundling: For these tests, we use edge directions as compatibility criterion. Parameter settings are identical to the undirected bundling evaluation presented above. Table 1 (rightmost six columns) shows the results. ADEB results for *amazon* miss, as this graph is too large for the implementation in [30]. FFTEB is 10 to 50 times faster than ADEB, with a larger speed-up for larger graphs. Also, we see that the directional version of FFTEB has basically the same cost as the undirected version. Compared to CUBu, FFTEB is roughly 3 times faster.

Given the results for directed and directional bundling presented above, we can conclude that FFTEB is the fastest undirected *and* directional method in existence.

The advantage of FFTEB extends beyond the above results. Recent work has shown that *approximate* FFTs can be efficient parallelized on CUDA, yielding speeds one order of magnitude higher than exact FFT's such as CUFFT [40]. Since we only use the FFT to compute a density map, whose gradient does not need to be very precise for the iterative advection used by the KDE model (Eqn. 3), such techniques are directly applicable to FFTEB.

5.2 Parameter Analysis

As stated in Sec. 3.2, FFTEB's complexity is $O(I(N+R^2\log R))$, lower than CUBu's complexity of $O(I(N+R^2h))$. Also, FFTEB's complexity is independent on the kernel size $h \sim R$, while CUBu's complexity is not. Apart from this, both FFTEB and CUBu are linear

Table 1: Timing comparison of KDEEB[19], CUBu[38], ADEB[30], and FFTEB.

Graph	Edges	Undirected bundling methods						Directional bundling methods					
		KDEEB		CUBu undirected		FFTEB undirected		ADEB		CUBu directional		FFTEB directional	
	$\|E\|$	Samples	Time (ms)	Samples	Time(ms)	Samples	Time(ms)	Samples	Time(ms)	Samples	Time(ms)	Samples	Time(ms)
US airlines	2099	86K	500	86K	14	105K	10	80K	150	86K	28	88K	11
Poker	2127	50K	400	50K	11	60K	8	50K	200	50K	22	50K	6
Radial	4021	290K	1500	290K	23	290K	13	300K	450	290K	46	300K	13
US migration	9780	220K	1500	221K	24	300K	15	260K	650	221K	48	290K	15
France airlines	17275	330K	1800	330K	25	330K	16	250K	400	330K	50	250K	13
Amazon	899791	19M	8053	19M	152	19M	93	n/a	n/a	19M	304	19M	98

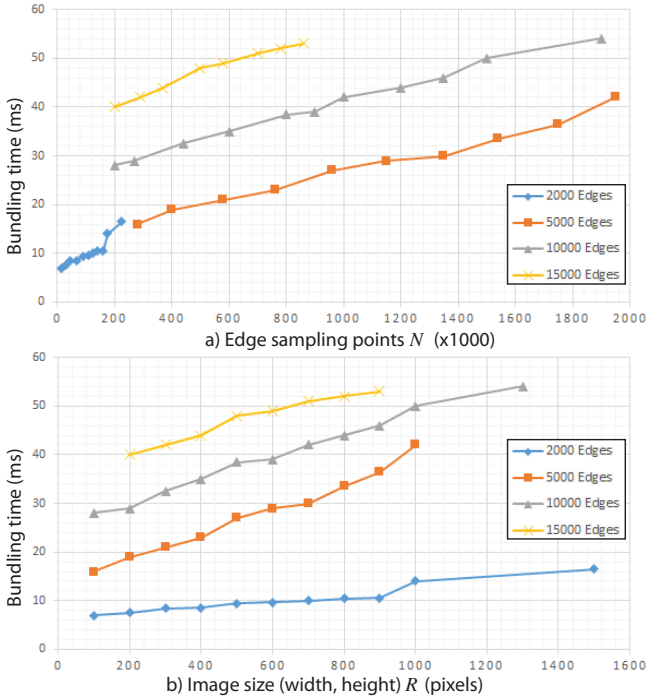

Figure 8: Scalability of FFTEB as function of number of sample points N and image size R.

in the number of bundling iterations I which, as explained, is set to a small fixed value $I = 10$.

We next study how FFTEB's performance depends on its two free parameters – number of sampling points N and image size R. Recall that N depends on the image size R, so the sampling *density*, *i.e.*, ratio N/R, is invariant. As such, we proceed as follows. We assume a fixed sampling density of a few pixels between consecutive sampling points on each edge. Next, we benchmark FFTEB for four graphs having between 2K and 15K edges, bundled at resolutions varying from 100^2 to 1500^2 pixels. For each run, we record the computational time and produced sampling-point count N.

Fig. 8 shows how FFTEB's speed depends on both R and N. As expected, the computational time is linear in N (Fig. 8a). More interestingly, however, is Fig. 8b, which shows an almost linear dependency of time on the size (width, height) R of the image, while, as we have seen, the computational complexity in R is $O(R^2 \log R)$. In contrast, CUBu's time dependency on R is clearly superlinear (see [38], Fig. 12a). This positive result shows that FFTEB scales in practice very well with the image size R.

This can be explained by the very efficient parallelization of FFTEB in CUFFT, the large number of parallel cores (3072) of the used GPU (NVidia 690 GTX) *vs* the amount of data to process, and the fact that we only evaluated performance up to a relatively low resolution ($R = 1600$). Quadratic increase of computation time with image size may actually become visible, on this GPU, only at higher image resolutions. Finally, we see that the slopes of the four graphs are quite similar. This tells that FFTEB has a throughput (number of bundled sample points per unit time) which does not strongly depend on the data size. This practically confirms the FFTEB complexity of $O(I(N + R^2 \log R))$, which is linear in the sample point count N.

5.3 Advantages and Limitations

FFTEB solves the scalability and computational speed challenges of earlier bundling methods, for all known bundling variants – undirected, directional, and general attribute-based. By itself, this is an important results, as it opens the possibility to *efficiently* handle datasets of arbitrary size and number of attributes. In particular, FFTEB strongly alleviates the main scalability problems of KDE-based methods, which, as discussed in Sec. 3.1, strongly depend on the image resolution R, kernel-size h, and a graph's sampling points count N. Putting it simply, FFTEB can generate high-resolution images (large R), with strrong bundling (large h), for big graphs (large N) faster than all known methods we are aware of.

Additionally, our proposed modulation of bundle width by local edge density emphasizes important bundles much better than shading and/or opacity can. More specifically, when looking at an image, it is easier to quantitatively compare the widths of bundles, especially when these are rendered as shaded tubes as we do, than quantitatively compare their relative opacities. This is related to the well-known power of encoding of the size (width) and opacity visual variables [1, 28]. Separately, our paper is, to our knowledge, the most extensive comparison of edge bundling methods – in total, we compare eight existing methods (apart from FFTEB) on several graphs ranging from a few thousands to a million edges.

FFTEB by itself does not solve several challenges related to the *effective* interpretation of edge-bundled drawings. Following certain edge groups end-to-end is hard, especially for large graphs – directional bundling helps here only partially. Designing compatibility functions κ that effectively capture the similarity of multivariate edges highly depends on the specific nature and meaning of edge attributes for the problem at hand. Controlling and/or showing the amount of distortion that bundling produces is an important factor that must be further considered when bundling data with spatial meaning, such as trails. Finally, quantifying the quality of a bundled layout in terms of how effective it is to solve a specific problem, or based on ground truth in terms of its ability to preserve the underlying data structure, is still an open grand challenge in the edge bundling arena. Potential ideas to address this are (1) organizing controlled experiments to test the accuracy and/or speed of performing a certain task using different bundling methods; and (2) measuring the amount of distortion and/or displacement of the bundled edges with respect to the unbundled ones, or to a relevant subset hereof, to quantify how data structure is preserved.

Given all above, we conclude that FFTEB effectively solves the bundling efficiency challenges of current methods and is a good framework to base further research in measuring and improving the effectiveness of edge-bundled drawings of large datasets.

6 Conclusion

We have presented FFTEB, a method to bundle very large attributed graphs. FFTEB's key asset is its scalability, both in size of the input

graph (number of edges or edge sampling-points) and in the resolution of the final image. FFTEB exploits the properties of the Fast Fourier Transform, and of a GPU streaming design, to bundle graphs which are much larger than what state-of-the-art methods can handle, and with less time. Additionally, FFTEB generalizes attribute-based graph bundling with no computational penalties, something that aforementioned methods cannot do. We demonstrate our approach by comparing FFTEB with nine state-of-the-art methods and using graphs that have up to *billions* of sample points – an order of magnitude that not current method can handle. Additionally, we show that the high scalability delivered by FFTEB is essential to generate images which can capture fine details of the underlying datasets, which in turn helps their better understanding.

Many possible extensions for FFTEB exist: Using recent results in computation of sparse FFT can accelerate our method one order of magnitude [40], which would make FFTEB cope with most big-data challenges envisable today. Next, FFTEB's efficient attribute-based bundling opens new ways to explore virtually any kinds of edge-compatibility criteria based on any nature and/or number of edge attributes. Finally, the overall generality and scalability of FFTEB makes its extension to interactive 3D bundling a low hanging fruit prospect, with application *e.g.* into medical imaging [2].

Acknowledgements

The authors acknowledge the support of the Federal University of Toulouse (Université Fédérale Toulouse Midi-Pyrénées – UFTMiP) under the grant "MEMOIRE" and the French National Agency for Research (Agence Nationale de la Recherche – ANR) under grant ANR-14-CE24-0006-01 project "TERANOVA".

References

[1] J. Bertin. *Semiology of graphics: Diagrams, networks, maps*. Univ. of Wisconsin Press, 1983.

[2] J. Böttger, A. Schäfer, G. Lohmann, A. Villringer, and D. Margulies. Three-dimensional mean-shift edge bundling for the visualization of functional connectivity in the brain. *IEEE T Vis Comput Gr*, 20(3):471–480, 2014.

[3] B. Buttenfield and R. McMaster. *Map Generalization: Making rules for knowledge representation*. J. Wiley & Sons, 1991.

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

[5] J. Cooley, P. Lewis, and P. Welch. Historical notes on the fast Fourier transform. *IEEE T Audio Electroacust*, 15(2):76–79, June 1967.

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

[7] M. Dickerson, D. Eppstein, M. Goodrich, and J. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. *J. Graph Alg Appl*, 9(1):31–52, 2005.

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

[9] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE T Vis Comput Gr*, 13(6):1216–1223, 2007.

[10] V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theor Probab Appl*, 14:153–158, 1969.

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

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

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

[14] S. Hadlak, H. Schulz, and H. Schumann. In situ exploration of large dynamic networks. *IEEE T Vis Comput Gr*, 17(12):2334–2343, 2011.

[15] N. Henry, J. Fekete, and M. J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE T Vis Comput Gr*, 13(6):1302–1309, 2007.

[16] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE T Vis Comput Gr*, 12(5):741–748, 2006.

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

[18] C. Hurter, O. Ersoy, S. Fabrikant, T. Klein, and A. Telea. Bundled visualization of dynamic graph and trail data. *IEEE T Vis Comput Gr*, 20(8):1141–1157, 2014.

[19] C. Hurter, O. Ersoy, and A. Telea. Graph bundling by kernel density estimation. *Comput Graph Forum*, 31(3):435–443, 2012.

[20] C. Hurter, O. Ersoy, and A. Telea. Smooth bundling of large streaming and sequence graphs. In *Proc. IEEE PacificVis*, pages 374–382, 2013.

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

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

[23] O. Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *Proc. IEEE PacificVis*, pages 232–239, 2011.

[24] B. Lee, C. S. Parr, C. Plaisant, B. B. Bederson, V. D. Veksler, W. D. Gray, and C. Kotfila. Treeplus: Interactive exploration of networks with enhanced tree layouts. *IEEE T Vis Comput Gr*, 12(6):1414–1426, 2006.

[25] A. Lhuillier and C. Hurter. Bundling, graph simplification trough visual aggregation: existing techniques and challenges. In *Proc. ACM IHM*, 2015.

[26] A. Lhuillier, C. Hurter, and A. Telea. FFTEB implementation, 2016. Visual Studio C# source code, http://recherche.enac.fr/~hurter/FFTEB/FFTEB.html.

[27] Z. Liu, S. B. Navathe, and J. T. Stasko. Network-based visual analysis of tabular data. In *Proc. VAST*, pages 41–50. IEEE, 2011.

[28] A. MacEachren. *How maps work*. Guilford Press, 1995.

[29] D. Moura. 3D density histograms for criteria-driven edge bundling, 2015. *arXiv:1504.02687v1* [cs.GR].

[30] V. Peysakhovich, C. Hurter, and A. Telea. Attribute-driven edge bundling for general graphs with applications in trail analysis. In *Proc. IEEE PacificVis*, pages 39–46, 2015.

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

[32] V. Podlozhnyuk. FFT-based 2D convolution, 2007. NVidia white papers, http://developer.download.nvidia.com/compute/cuda/2_2/sdk/website/projects/convolutionFFT2D/doc/convolutionFFT2D.pdf.

[33] S. Pupyrev, L. Nachmanson, S. Bereg, and A. E. Holroyd. Edge routing with ordered bundles. In *Graph Drawing*, pages 136–147. Springer, 2012.

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

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

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

[37] U.S. Census Bureau. County-to-county migration flow files, 2003. www.census.gov/population/www/cen2000/ctytoctyflow.html.

[38] M. van der Zwan, V. Codreanu, and A. Telea. CUBu: Universal real-time bundling for large graphs. *IEEE T Vis Comput Gr*, 2016. DOIL10.1109/TVCG.2016.2515611.

[39] F. van Ham and M. Wattenberg. Centrality based visualization of small world graphs. *Comput Graph Forum*, 27(3):975–982, 2008.

[40] C. Wang, S. Chandrasekaran, and B. Chapman. cusFFT: A high-performance sparse fast Fourier transform algorithm on GPUs. In *Proc. IEEE Symp. Parallel and Distributed Processing*, 2016. DOI:10.1109/IPDPS.2016.95.

[41] H. Zhou, P. Xu, Y. Xiaoru, and Q. Huamin. Edge bundling in information visualization. *Tsinghua Sci. Tech.*, 18(2):148–156, 2013.

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