Bundled Visualization of Dynamic Graph and Trail Data

C. Hurter [*]
ENAC/University of Toulouse, France

O. Ersoy [†]
University of Groningen, the Netherlands

S. I. Fabrikant [‡]
University of Zürich, Switzerland

T. R. Klein[§] A. C. Telea [¶]
University of Groningen, the Netherlands

Abstract— Dynamic graphs and temporal paths, or trails, are increasingly found in information systems. However, understanding how such objects change in time is hard. We present here two techniques for simplified visualization of such datasets using edge bundles. The first technique uses an efficient image-based graph bundling method to create smoothly changing bundles from streaming graphs. The second technique adds edge-correspondence data atop of any static graph bundling algorithm, and is best suited to visualize graph sequences. We show how these techniques can produce simplified visualizations of streaming and sequence graphs. Next, we show how several temporal attributes can be added atop of the visualized dynamic graphs. We illustrate our techniques with datasets from aircraft monitoring, software engineering, and eye-tracking of static and dynamic scenes.

1 Introduction

Graph visualization supports various comprehension tasks such as understanding connectivity patterns, finding frequently-taken communication paths, and assessing the overall interaction structure in relational datasets [60]. Visualizing large networks is challenging, due to inherent clutter, crossing, and overplotting problems [60, 12, 55].

Dynamic networks pose their own understanding challenges. Their sizes are far larger than for static graphs. Dynamic graphs can be further classified into *streaming graphs*, *i.e.* unstructured edge-sequences with start and end lifetime moments along a continuous time axis, and *graph sequences*, *i.e.* a discrete set of graphs between which higher-level node and edge correspondences can be inferred. For both types, one of the main visualization tasks is to help users spot *changes* in the overall network structure, while maintaining limited clutter.

Edge bundling methods have gained strong attention as a way to show the overall connectivity pattern of large graphs by trading clutter for overdraw [37, 12, 55, 25, 31, 17, 27]. For dynamic networks, however, bundling methods have been only recently investigated [41].

Recently, two scalable methods were presented for bundling of streaming graphs and graph sequences [32], based on a highly-efficient kernel-density bundling method (KDEEB) [31]. For streaming graphs, KDEEB is extended by applying its core operator, essentially the well-known mean-shift in [9], on a sliding time window. For graph sequences, each keyframe is statically bundled, and inter-keyframe edge-correspondences are used to interpolate in-between and also highlight events like the change and (dis)appearance of edge groups. Efficient and simple GPU implementations of both techniques are proposed which scale to large dynamic graphs and ensure spatial and temporal continuity, *i.e.*, preserve the user's mental map.

In this paper, we extend the dynamic graph visualizations in [32] in several directions, as follows:

- We analyze and detail the differences between stream and sequence graph bundling by applying stream bundling on graph sequences and sequence bundling on streaming graphs;
- We consider *trail sets*, which are particular streaming graphs in which appearing edges never disappear during the considered time interval. For this use case, we show how bundling can find patterns of interest embedded in eye tracking data obtained from both static and dynamic scenes;
- We present several techniques to show additional temporal attributes atop of dynamic bundled and unbundled graphs.

The structure of this paper is as follows. Section 2 presents related work on dynamic and bundled graph visualization. Section 3 details our visualization method for streaming graphs. Section 4 presents our visualization method for graph sequences. Section 5 discusses the application of streaming bundling to sequence graphs and sequence bundling to streaming graphs, and highlights the pro's and con's of these combinations. We also present here the two use-cases of bundling to analyze eye-tracking data obtained from both static and dynamic scenes. Section 6 discusses the presented dynamic bundling methods in terms of desirable features. Section 7 concludes the paper.

2 Related Work

2.1 Preliminaries

We classify our datasets of interest in three groups, as follows.
A. Streaming graphs are graphs $G = (V,E)$ with vertices V and edges

$$e \in E = \{n_{start}(e) \in V, n_{end}(e) \in V, t_{start} \in \mathbb{R}, t_{end} \in \mathbb{R}\} \qquad (1)$$

defined by start and end nodes n_{start} and n_{end}, and lifetime $[t_{start}, t_{end} > t_{start}]$. Eqn. 1 can be also used to model streaming graphs where only an ordering of the t_{start} and t_{end} values is given rather than absolute values. Streaming graphs occur when an entire graph is not known in advance, *e.g.*, events collected from live, online data sources [1].
B. Graph sequences are ordered sets of graphs $G^i = (V^i, E^i)$ which typically capture snapshots of a system's structure at N moments $1 \leq i \leq N$ in time. We call a graph G^i in such a sequence a *keyframe*. In contrast to streams, edges are explicitly grouped in keyframes, and additional semantics can be associated with each such keyframe. Following this, sequences may contain so-called correspondences

$$c : E^i \rightarrow \{\{e_{corr} \in E^{i+1}\}, \varnothing\}. \qquad (2)$$

Here, $c(e \in E^i)$ yields an edge $e_{corr} \in E^{i+1}$ which logically corresponds to e (if such an edge exists), or the empty set (if no such edge exists). Hence, c models edge-pairs in consecutive keyframes that are related application-wise, *e.g.*, caller-callee relations between the *same* function definitions in consecutive revisions of a software system.
C. Trails are sequences

$$T = \{\mathbf{p}_i = ((x,y) \in \mathbb{R}^2, t \in \mathbb{R}^+)_i\} \qquad (3)$$

with increasing values t_i, *e.g.*, the path of a vehicle in time, where points $\mathbf{p}_i$ are the recorded samples of the vehicle's position. Formally,

[*]e-mail:christophe.hurter@enac.fr
[†]e-mail:o.ersoy@rug.nl
[‡]e-mail:sara.fabrikant@geo.uzh.ch
[§]e-mail:t.r.klein@student.rug.nl
[¶]e-mail:a.c.telea@rug.nl

Manuscript received 31 March 2007; accepted 1 August 2007; posted online 2 November 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

a trail-set $\{T\}$ is a streaming graph with $\mathbf{p}_i$ as nodes and pairs $(\mathbf{p}_i, \mathbf{p}_{i+1})$ as edges. As we show in Sec. 5.3, trails can be more effectively visually analyzed with specific techniques than for streaming graphs.

2.2 Dynamic graph visualization

Visualizing changing graphs has a long history. Methods can be divided into two classes, as follows.

Unfolding the time dimension along a spatial one, *e.g.*, using the 'small multiples' approach [5, 56], has led to many dynamic graph visualizations. In graph drawing, specific solutions are known for planar straight-line graphs [4]. In software visualization, TimelineTrees [6], TimeRadarTrees [7], and TimeArcTrees [54], and CodeFlows [54] lay out a graph along a 1D space, *e.g.*, circle or line, and juxtapose several such instances on an orthogonal axis to show the graph evolution. Although reducing clutter by not using a node-link drawing metaphor, such methods are visually not highly scalable, nor are they very intuitive, especially for long time series with complex event dynamics.

Animation is a second way to show dynamic graphs, and can specifically help finding change relationships in complex spatio-temporally coordinated events [49]. Ware and Bobrow have empirically shown how motion can provide cognitively and perceptually supported efficient and effective access to large graphs [61]. Several techniques create incremental node-link graph drawings by optimizing a cost function that includes static-graph-drawing aesthetic criteria and layout stability for unchanging graph parts [23, 18, 22, 29]. Animation can be preferable to small-multiples in conveying dynamic patterns, especially for long repetitive time series [57]. Such methods, however, may suffer from visual clutter, due to the underlying node-link metaphor.

2.3 Bundled edge graph visualization

Edge bundling mitigates clutter by routing related edges along similar paths. Clutter causes and reduction strategies are discussed in [16, 66]. Such strategies are similar to long-standing map generalization in cartography [8], concerned with legibly depicting a complex world in static, small-scale, 2D views. Bundling can be seen as sharpening the edge spatial density, by making it high on bundles and low elsewhere [31]. This creates images where individual edges are emphasized less, but high-level graph structures are easier to follow, *e.g.*, we can find node-groups related to each other by edge-groups (bundles) separated by white space [25, 55].

Dickerson *et al.* merge edges by reducing non-planar graphs to planar ones [13]. Hierarchically edge bundles (HEBs) bundle compound graphs by routing edges along the hierarchy layout as B-splines [27]. Gansner and Koren bundle edges in a circular node layout similar to [27] by area optimization metrics [26]. Dwyer *et al.* use curved edges in force-directed layouts to minimize crossings, which implicitly creates bundle [14]. Force-directed edge bundling (FDEB) creates bundles by attracting edge control points [28], and was adapted to separate opposite-direction bundles [48]. MINGLE uses multilevel clustering to accelerate the bundling process [25]. Flow maps produce a binary clustering of nodes in a directed flow graph to route curved edges [44]. Control meshes are used to route curved edges [45, 67], a Delaunay-based extension called geometric-based edge bundling (GBEB) [12], and 'winding roads' (WR) which use Voronoi diagrams for 2D and 3D layouts [37, 36]. Skeleton-based edge bundling (SBEB) uses the skeleton of the graph drawing's thresholded distance transform as bundling cues to create strongly ramified bundles [17].

To render bundles, edge-direction color interpolation [27, 12] and transparency or hue for edge density or edge lengths [37, 17] are used, following [3]. Bundles can be drawn as compact shapes whose structure is emphasized by shaded cushions [55, 47]. Graph splatting visualizes node-link diagrams as smooth scalar fields using color and/or height maps [59, 33]. To explore crowded areas (overlapping bundles), semantic lenses can be used [30]. Ambiguity-free bundling improves the local detailed view of a bundled graph by combining a semantic lens technique with a bundle refinement step that reroutes and/or selectively bundles edges so that bundles avoid unrelated nodes [38].

2.4 The challenge of bundling dynamic graphs

Given the above, it seems suitable to use edge bundling to visualize the (simplified) structure of dynamic graphs. This has been pioneered by Nguyen *et al.*, who cut a streaming graph into a set of graphs using a sliding time-window, and draw each such graph with existing edge-bundling methods [28, 27]. Edge similarity, or compatibility, is used to take into account temporal coherence. We improve this idea in several directions: scalability (number of edges handled), ensuring a high spatio-temporal continuity of the produced animations where large-scale and long-life structures are stable over time and display space, and using the correspondences present in graph sequences. We next present two bundling methods for streaming and sequence graphs which incorporate these improvements.

3 VISUALIZING STREAMING GRAPHS

Given a graph G with node positions, we model bundling as an operator $B : G \to \mathbb{R}^2$ which creates a drawing $B(G)$ which maps edges that are close in G to close spatial positions (bundles) [31]. Different bundling algorithms give different ways to model edge closeness in G: tree-distance of edge end-nodes in a hierarchy [27], closeness of edges in a straight-line drawing of G [17, 28, 31], or the more general mix of graph-theoretic and image-space distances [41].

Consider now a streaming graph (Eqn. 1), the 'instantaneous' graph $G(t) = \{e \in G | t \in [t_{start}(e), t_{end}(e)]\}$, and its bundling $B(t) = B(G(t))$ by a bundling operator B. Ideally, we want that $B(t)$ (a) varies continuously in time, and also (b) keeps the spatial properties of the underlying operator B, *i.e.*, puts close edges in tight bundles.

Property (b) is satisfied by using a 'good' bundling algorithm B that guarantees that any input graph is (strongly) bundled, such as [25, 37, 12, 17, 31], or to a lesser extent [27, 28], as we shall see. Property (a) means that, when $G(t)$ changes slightly, then $B(G(t))$ should also change slightly, so graph structures stable in time are also stable in the animation. Conversely, if the graph changes strongly, there should be a visible change in the animation. However, even when such large changes occur, discontinuous bundle *jumps* in the animation should be avoided, since visually tracking such jumps is hard [21].

A partial answer to (a) is to reduce the dynamics of $G(t)$, *e.g.*, by applying a low-pass filter to $G(t)$. In other words, the bundling result shown at moment t is $B(\tilde{G}(t))$ where $\tilde{G}$ is the filtered graph. This is the solution proposed by StreamEB, who pioneered bundled layouts for streaming graphs [41]. They use a sliding window technique (finite-support box filter) to compute $\tilde{G}$ as all edges alive in $[t, t + \Delta t]$.

However, this approach has two limitations. First, the smoothness of the final animation depends strongly on the variation rate of $\tilde{G}$. If graphs for two consecutive time moments $\tilde{G}(t)$ and $\tilde{G}(t + \Delta t)$ differ too much, *e.g.*, there are too many edges added or deleted per time unit, or the filtering time-window is too small, then there is no guarantee that the corresponding bundlings $B(\tilde{G}(t))$ and $B(\tilde{G}(t + \Delta t))$ are spatially close. If this is not the case, users notice a disruptive visual jump from t to $t + \Delta t$. Secondly, the computational efficiency of the approach in [41] strongly depends on the scalability of the underlying static bundling operator B. Algorithms which ensure good spatial stability [28, 12] are also quite expensive, roughly $O(|\tilde{E}|^2)$ for $|\tilde{E}|$ edges in $\tilde{G}(t)$. Faster bundling algorithms [25, 17, 37] cannot ensure continuity, *i.e.*, a small change in the input graph may generate a large change in the bundled image, so are less suitable for stream bundling.

3.1 Algorithm

We address the above challenges by exploiting the properties of a recent bundling method for large graphs: kernel-density estimation edge bundling (KDEEB) [31]. Given a graph drawing $G = \{e_i\}_{1 \le i \le N}$, KDEEB estimates the spatial edge density $\rho : \mathbb{R}^2 \to \mathbb{R}^+$

$$\rho(\mathbf{x}) = \sum_{i=1}^{N} \int_{\mathbf{y} \in e_i} K\left(\frac{\mathbf{x} - \mathbf{y}}{h}\right) \tag{4}$$

where $K : \mathbb{R}^2 \rightarrow \mathbb{R}^+$ is an Epanechnikov kernel of bandwidth h [24]. KDEEB iteratively moves all edge points $\mathbf{x}$ upstream in $\nabla\rho$ following

$$\frac{d\mathbf{x}(t)}{dt} = \frac{h(t)\nabla\rho(t)}{\max(\|\nabla\rho(t)\|, \varepsilon)} \tag{5}$$

where ε is a small regularization constant. After a few Euler iterations for solving Eqn. 5, during which we decrease h and recompute ρ, edges converge into bundles. A final 1D Laplacian edge-smoothing pass is done to remove small wiggles (for full details, see [31]). Upon a closer analysis, not reported by [31], we see that this process is nothing else but applying the well-known mean-shift algorithm [9] on the drawn edges. Thus, the bundled graph is a *clustering* of the graph drawing based on edge similarity. This observation is important, since smoothness, noise robustness, and stability results proven for mean shift [9] can be readily extrapolated to KDEEB.

Key to KDEEB is the fast computation of the density map ρ. This is done by splatting the kernel K, stored as an OpenGL texture, into an accumulation map. This allows bundling graphs of tens of thousands of edges in a few seconds on a modern GPU.

To bundle streaming graphs, we now iterate KDEEB in sync with the stream time t. The principle is simple (see Alg. 1): We move a sliding window $[t, t+\Delta t]$ over the time range of the streaming graph, compute $\rho(t)$ from $\tilde{G}(t)$, and advect edges by Eqn. 5. There are two key advantages to this approach. First, $\rho(t)$ can be very efficiently computed by the underlying KDEEB algorithm, which is $O(|\tilde{E}|)$, *i.e.* linear in the edge count of the current graph $\tilde{G}$. Secondly, and most importantly, KDEEB requires $I = 5..10$ iterations for a single static graph to be bundled. We remove this iterative process by letting $\tilde{G}$ bundle *while* advancing t. This makes sense since, if $\tilde{G}$ changes very slowly, advancing t is nearly equivalent to performing iterations for a fixed t, so we obtain a strongly bundled $\tilde{G}$, which is what we want to see. If $\tilde{G}$ changes rapidly, then our process has less time to bundle, and thus we see looser bundles, which shows precisely the dynamics of $\tilde{G}$. Details on performance and parameter settings are given in Sec. 6.1.

```
1  $t \leftarrow 0$
2  while stream not ready do
3      $\rho \leftarrow 0$
4      $E_{live} \leftarrow \{e \in E | [t_{start}(e), t_{end}(e) \cap [t, t+\Delta t] \neq \varnothing\}$
5      foreach $e \in E_{live}$ do
6          splat $e$ into $\rho$ ;                        //Splat live edges (Eqn. 4)
7      end
8      foreach $e \in E | t_{end}(e) \in [t - \delta t, t]$ do
9          relax $e$ towards its original position ;   //Vanishing edges
10     end
11     foreach $e \in E_{live}$ do
12         advect $e$ one step ;                           //See Eqn. 5
13         apply 1D Laplacian smoothing on $e$ ;
14         draw $e$ in the visualization ;
15     end
16     $t \leftarrow t + \delta t$ ;                          //Advance sliding window
17 end
```

Algorithm 1: Bundling streaming graphs with KDEEB

Our dynamic bundling can be seen as a process where edges continuously track the local density maxima of a dynamically-changing graph. Since an advection step moves edges with a bounded amount h (line 12, Alg. 1), and since advection is done while advancing the stream time t, the maximal amount an edge-point can move at any time is h (Eqn. 5). Hence, the bundles move smoothly on the screen.

We additionally interpolate disappearing edges from their current (bundled) position towards their original (unbundled) position in the input stream (line 9, Alg. 1). This makes the animation symmetric: New edges progressively bundle as times goes by, while disappearing edges relax, or unbundle, towards their original positions after exiting the sliding time-window. To further emphasize this effect, we modulate the edges' transparencies in a similar fashion. We note that this effect is optional. If left out, disappearing edges will exit silently, without relaxation. The choice of using relaxation or not depends on whether users want to see edge-vanishing events or not.

An important goal of animation is to help users find change over time or deviations from regular patterns [57, 60]. We support this by shading bundles to convey their change speed, using a simple and fast image-based method: We compute the density moving-average $\tilde{\rho}(t)$ over $[t, t+\Delta t]$, and color bundles by the normalized difference $|\rho(t) - \tilde{\rho}(t)|/\tilde{\rho}(t)$ using a white-to-purple colormap. Results are shown next.

3.2 Applications

Figure 1 shows six frames from a streaming visualization of US flights [52] (6 days, 41K flights). The streaming graph contains flights with start and end date-and-time and geographical locations. The resulting flight-trail bundles are smooth, visually salient, and clutter-free, and show a continuous variation in time[1]. In Fig. 1, we see that same time-of-day flight patterns are quite similar for several days. However, they vary strongly over a day: During the evening, the East coast has the most intense traffic. During the afternoon, the entire US is uniformly covered with flights. During the night, flights linking the two coasts dominate.

Figure 2 shows a similar visualization for flights over France (7 days, 54K flight trails). For each trail (Eqn. 3), at each recorded time-moment t_i, we know the airplane position $\mathbf{p}_i$, the plane height $h_i \in \mathbb{R}^+$, and flight number $ID_i \in \mathbb{N}$. For bundling, we only use $\mathbf{p}_i$ and t_i. Visualizing additional attributes atop of the bundling is discussed in Sec. 5.2.

As for the US dataset, bundles are smooth and clutter-free in both space and time. Colors show the bundles' speed of change (white=stable, purple=rapid changes, see Sec. 3.1). Red dots show the first and last positions when a plane was monitored. Dots inside France are actual airports. Dots outside the French territory show international flights which enter/exit the French airspace. We see that, during the day, the main 'backbone' flight pattern is quite stable over different days, and contains mainly north-south routes, with Paris as a key hub (Fig. 2 top row). A different pattern, also quite stable, appears at night (Fig. 2 bottom row): The vertical bundles are Southern flights bound to Paris. We also see more purple, which shows that night-time flight paths are much less stable than during-the-day flight paths.

For the US dataset, a qualitative comparison of our results with StreamEB [40] shows that KDEEB produces stronger bundles and an overall smoother animation. This is first due to the fact that KDEEB can produce bundles with many inflexion points, while FDEB has a smoothing factor built in its edge compatibility metric that disfavors such shapes. Secondly, this is due to the built-in smoothness of our method which bundles edges as they arrive in the input stream.

4 Visualizing graph sequences

Graph sequences G^i (Sec. 2.1) exhibit different properties from streaming graphs. First, streams allow defining an infinity of "instantaneous" graphs $G(t) = (V, \{e \in E | t_{start} < t < t_{end}\}), \forall t \in \mathbb{R}$ (see Sec. 3.1). Some of these graphs may not have a direct meaning or usefulness. In contrast, graph sequences contain a finite set of graphs which have been explicitly computed in specific ways, *e.g.*, for particular time moments, *e.g.*, (major) revisions of a software system. Secondly, keyframe correspondences add higher-level, edge-centric, information, *e.g.*, the fact that two files f_1, f_2 share a common piece of text in version 1, and next f_1 shares the same text with a file f_3 in version 2. In contrast, streaming graphs (Eqn. 1) only specify how edges appear and disappear in time, but do not necessarily encode logical connections between edges at different time moments. Thirdly, graph sequences do not necessarily come with birth and death moments for individual edges. Finally, keyframes in graph sequences must be wholly available before processing, whereas edges in a streaming graph can be, in most cases, analyzed "online" as they appear. All in all, the above make a case for treating graph sequences differently from graph streams.

[1]For this and the other examples next, see the submitted videos.

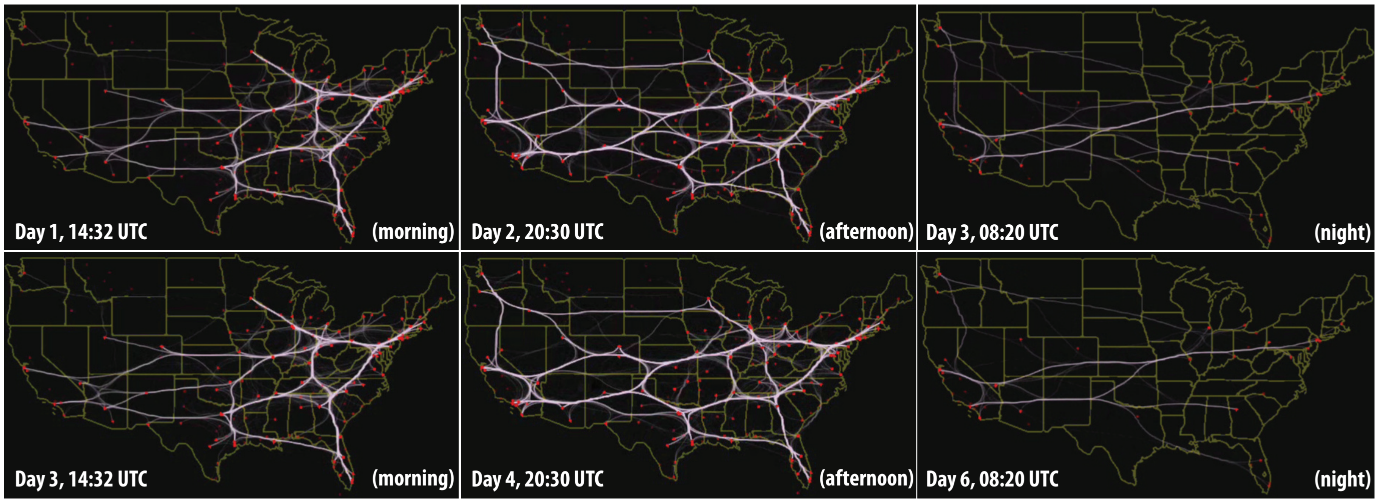

Fig. 1. Streaming visualization for 6-days US airline flight dataset (Sec. 3.2)

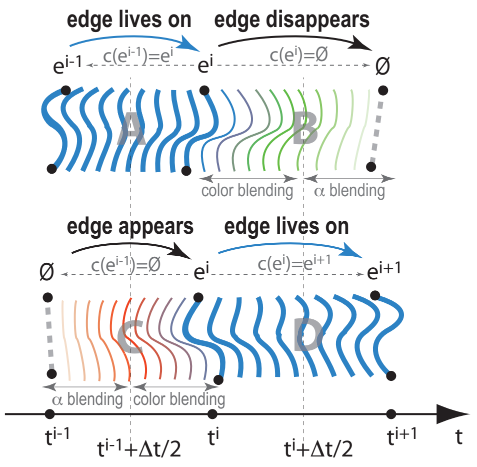

Fig. 3. Interpolation for graph sequence visualization

4.1 Algorithm

For graph sequences, we propose the following bundling method: For each keyframe G^i, we compute its bundled layout $B^i = B(G^i)$, using a given bundling algorithm B. Next, we interpolate these layouts between a keyframe i and the previous and next keyframes $i-1$ and $i+1$ respectively using the correspondence data (see Fig. 3). Consider a time axis t along which we place keyframes *e.g.*, at moments $t_i = i\Delta t$ (any other definition of t_i can be easily used, if available). For each edge $e \in G^i$, if $c(e) = e^{i+1} \in E^{i+1}$, we linearly interpolate $B^i(e)$ to $B^{i+1}(e^{i+1})$ over the interval $[t_i, t_{i+1}]$ (Fig. 3D). If $c(e)$ is the empty set, *i.e.* e has no correspondence in E^{i+1}, we interpolate $B^i(e)$ to the line segment $L(e) = (n_{start}(e), n_{end}(e))$ over the same time interval (Fig. 3B). Symmetrically, if $c^{-1}(e) = e^{i-1}$, we interpolate from $B^{i-1}(e^{i-1})$ to $B^i(e)$ over $[t_{i-1}, t_i]$ (Fig. 3A), else we interpolate from $L(e)$ to $B^i(e)$ over the same time interval (Fig. 3C).

We emphasize (dis)appearing edges by shading: Edges with correspondences between two keyframes i and $i+1$ are blue and thick. Edges that disappear from i to $i+1$ get a linearly interpolated color from blue (at t_i) to green (at $t_{mid} = \frac{t_i+t_{i+1}}{2}$), and next get an alpha value decreasing from opaque (at t_{mid}) to fully transparent (at t_{i+1}). Edges that appear from $i-1$ to i get an alpha increasing from fully transparent (at t_{i-1}) to opaque (at t_{mid}), followed by a color interpolated from red (at t_{mid}) to blue (at t_i). Hence, edges appear by fading in to red (highlights their incoming), then smoothly merge in a blue bundle, and disappear by unbundling, becoming green (highlights their vanishing), and fading out. Examples next explain the color choice.

4.2 Applications

We illustrate our sequence visualization with two datasets from software engineering. The first dataset contains 22 revisions of Mozilla Firefox [39]. For each revision, we extracted the code hierarchy (folders and files), and also the *clones*, or code duplicates, using the freely available clone detector SolidSDD [46, 51]. So, for each revision, we obtain a compound hierarchy-and-associations graph where two files are linked by an edge if they share a code clone. If a code fragment is cloned in several files, all these files are pair-wise linked by edges.

Figure 4 shows snapshots from SolidSDD's HEB visualization for such graphs. Node colors show cloning amount (red=high, green=low). Seeing clone patterns helps assessing how much, and where, did adaptive maintenance (*i.e.*, adding new features) introduce new clones, and how much, and where, did perfective maintenance succeed to remove clones [42]. This helps planning clone removal with minimal impact on system architecture in perfective maintenance. For this, we need to easily compare clone evolution patterns. This is hard to do using such small-multiple displays.

To support this task, we proceed as follows. First, we create a so-called union hierarchy containing all graph nodes in the analyzed releases [2]. This contains 13856 file and folder nodes. Next, we build correspondences between clones in consecutive releases: Two clone relations e^i and e^{i+1} correspond if they link the same files, *i.e.* files having the same fully qualified names, in G^i and G^{i+1}. Other ways to find correspondences, *e.g.*, using the actual text content of the clones [42], can be readily used too. The above steps deliver a graph sequence G^i (Sec. 2.1) which contains 5687 unique edges (counting corresponding edges as one) and 48591 edges in total.

We now use our sequence visualization on this data. Figure 5 shows several frames from this animation (see submitted videos). The bottom row shows results produced using KDEEB as underlying bundling method. Disappearing edges are green (removing clones is good); appearing edges are red (introducing new clones is bad). Additionally, we color hierarchy nodes as follows: Nodes which contain a changing clone count are colored by the clone count change, using red for positive values and green for negative values. Nodes where the clone count stays constant are colored blue. In all cases, we use saturation to indicate absolute values (saturated=high, desaturated=low values).

We note several events of interest. First, we see a stable 'core clone-structure that lives for a long time (blue bundles). These can be hard to remove clones, or clones that maintainers did not know of, *e.g.*, if no clone detector was actively used during perfective maintenance. We also see several moments when major clone-pattern changes occur, *e.g.*, from revision 2.0.0.10 to 3.0, many green edges appear, so many clones are removed (Fig. 5 d). Node coloring helps finding high-clone-density subsystems. For instance, from revision 3.6.10 on, we see two such dark-blue groups (dotted circles, Fig. 5 bottom row, e-h). Since

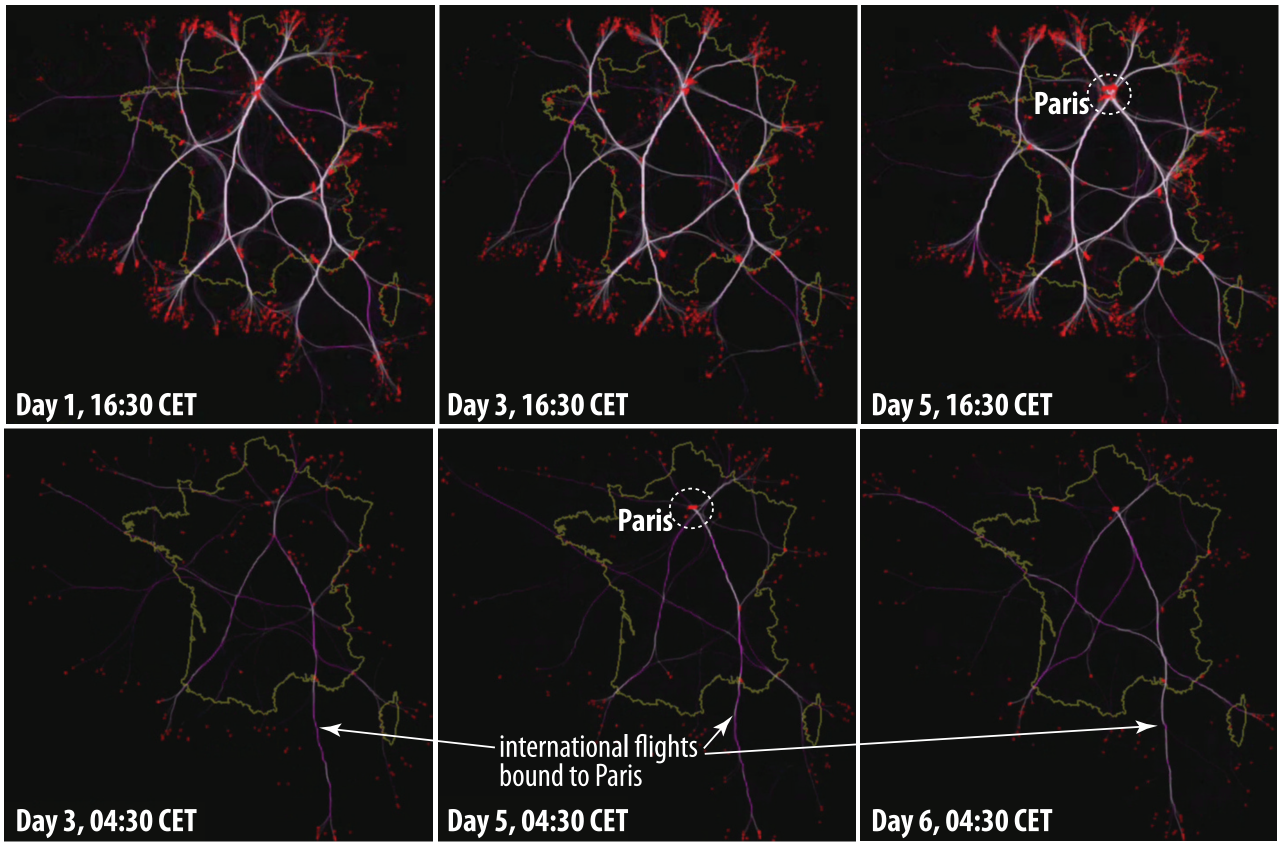

Fig. 2. Streaming visualization for 7-days France airline flight dataset (Sec. 3.2)

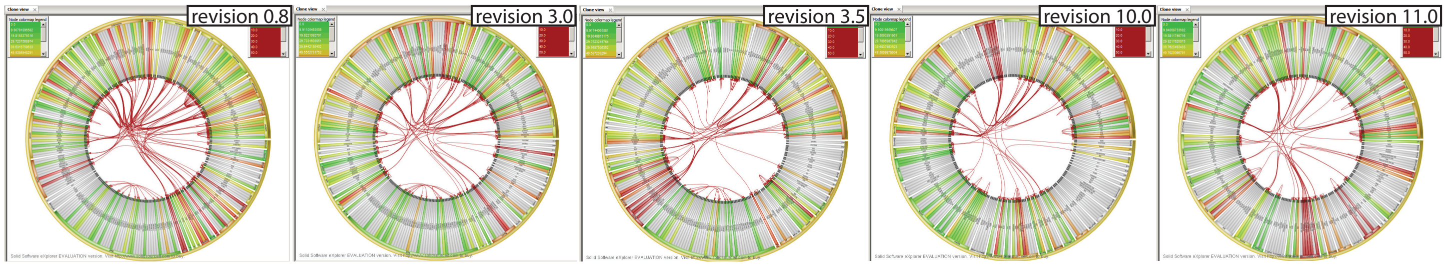

Fig. 4. Small multiples visualization for clones in Mozilla Firefox for five selected revisions (Sec. 4.2)

these groups stay visible in several revision, they indicate "stubborn" clones which, for several reasons, could not be removed for a long time. Although this information is encoded in the bundles too, finding such patterns on nodes is easier than visually following bundles. In other words, node colors help finding *aggregated* patterns, *e.g.*, high-clone-density systems during the evolution, while bundle changes help seeing which particular *subsystems* share such clones. We also see a red spot in revision 2.0.0.10 (Fig. 5 c): This is a subsystem where many *intra-system* clones have been added. Seeing such clones without node coloring would be hard, since their (bundled) edges are very short.

Between revisions 7.0 and 8.0 we see several interesting events: First, several 'stubborn' clones are removed (green edges shown after passing revision 7.0) Next, clones between the *same* files are added back again (red edges seen when approaching revision 8.0). This typically happens when one changes related code in two subsystems, *e.g.*, by independently applying twice the same given design pattern. However, developers were likely not aware of the clones, otherwise we would expect the clone to be removed during such a perfective refactoring. Finally, comparing the first and last frame shows that the core clone pattern did not change significantly. Also, the bundle pattern shows that clones connect *unrelated* subsystems, *i.e.* nodes in the radial icicle plot that are not close to each other, hence not in the same parent system. This is a negative sign for code quality, since removing such clones requires system-wide understanding and refactoring.

As a second example, we extracted a compound digraph with folders, files, and functions (hierarchy) and function calls (associations) from 14 revisions of the Wicket open-source software [63]. We next build the same union hierarchy as in our first example (8799 nodes), and compute correspondences using the fully qualified signatures of caller-callee pairs. We get 11953 unique edges and 92810 total edges. Figure 6 shows several frames from this sequence visualization. To better depict the animated transitions, we focus here on a short period (3 revisions). This visualization helps reasoning about the system's (change of) modularity, a challenging task in program comprehension [2]. The interpretation is as follows: The stable pattern (blue bundles) shows the stable control-flow system logic, *i.e.* calls that do not change much across versions. We see that this pattern is quite complex, *i.e.* connects many subsystems in different hierarchy parts, so the overall modularity of this system is *and* stays relatively low. In detail, we see that in version 1.4.18, a significant coupling is added between systems A and B (thick red bundle A-B, Fig. 6 c). Interestingly, in the *same* revision, many calls are removed between the same systems (large green bundle A-B, Fig. 6 f). This shows a refactoring of the A-B system interaction – note the similarity with the clone insertion-

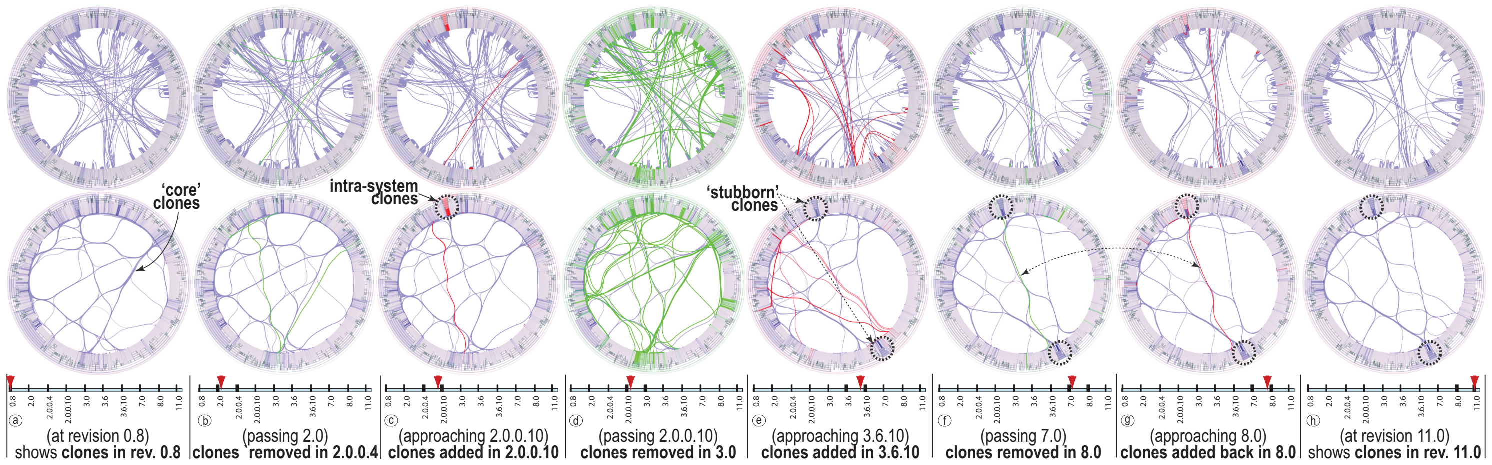
'core' clones
intra-system clones
'stubborn' clones
(a) (at revision 0.8) shows clones in rev. 0.8
(b) (passing 2.0) clones removed in 2.0.0.4
(c) (approaching 2.0.0.10) clones added in 2.0.0.10
(d) (passing 2.0.0.10) clones removed in 3.0
(e) (approaching 3.6.10) clones added in 3.6.10
(f) (passing 7.0) clones removed in 8.0
(g) (approaching 8.0) clones added back in 8.0
(h) (at revision 11.0) shows clones in rev. 11.0

Fig. 5. Sequence-based visualization for clones in Firefox (8 frames). Top row: HEB bundling. Bottom row: KDEEB bundling (Sec. 4.2)

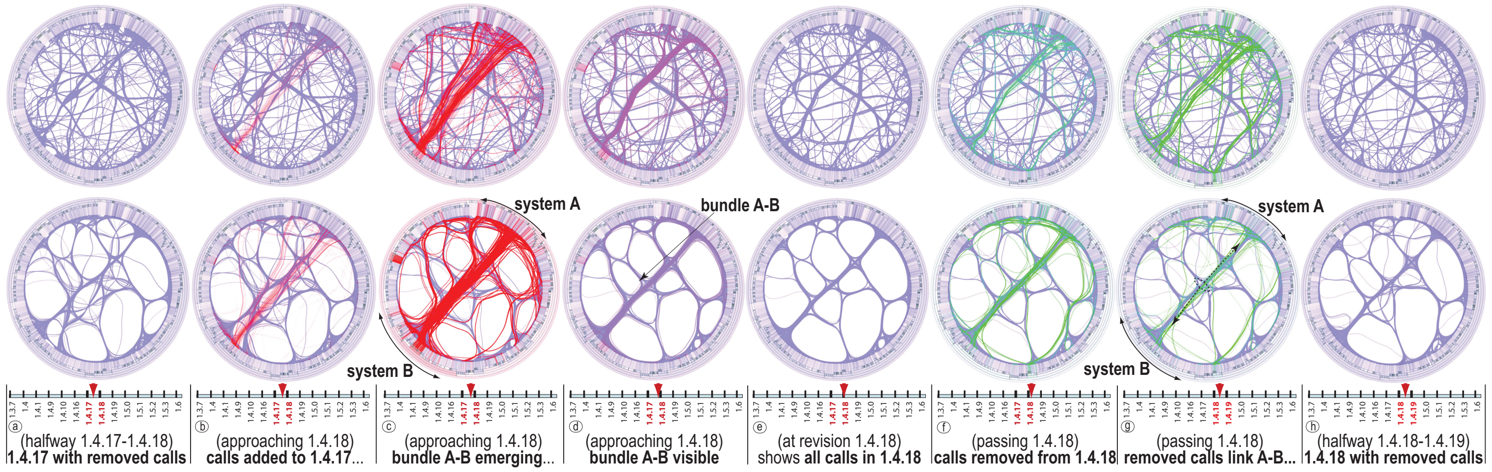
system A
bundle A-B
system B
system A
system B
(a) (halfway 1.4.17-1.4.18) 1.4.17 with removed calls
(b) (approaching 1.4.18) calls added to 1.4.17...
(c) (approaching 1.4.18) bundle A-B emerging...
(d) (approaching 1.4.18) bundle A-B visible
(e) (at revision 1.4.18) shows all calls in 1.4.18
(f) (passing 1.4.18) calls removed from 1.4.18
(g) (passing 1.4.18) removed calls link A-B...
(h) (halfway 1.4.18-1.4.19) 1.4.18 with removed calls

Fig. 6. Sequence animation – Wicket call graphs (8 frames around release 1.4.18). Top: SBEB bundling. Bottom: KDEEB bundling (Sec. 4.2)

deletion pattern and its interpretation discussed for the Firefox dataset.

5 Additional Applications

5.1 Streaming *vs* sequence graphs

In Sections 3 and 4 we have presented two techniques for visualizing streaming and sequence graphs. An open question is: Can we use the streaming algorithm for a sequence graph, and/or conversely? Why do we need two techniques? Below we analyze this aspect.

5.1.1 Streams with sequence-based visualization

For the first experiment, we convert our France air-traffic streaming graph (Sec. 3.2) to a sequence graph of 7 keyframes $G^i, 1 \leq i \leq 7$. For this, we divide the 7-days stream into 7 one-day periods. Edges are assigned to keyframes based on start time. Next, we add correspondences between edges in consecutive keyframes (days) whose geographic start and end locations are very similar and flight IDs are identical.

Figure 7 shows several frames from the resulting sequence-based animation. For the keyframe pairs $(1,2)$ and $(4,5)$ we show two intermediate frames at around the first third, respectively second third, of each day (see legends in Fig. 7). Color encoding follows Figs. 5 and 6 – corresponding edges between two consecutive keyframes are blue; appearing edges are red; and disappearing edges are green.

Comparing Fig. 7 with the streaming bundling of the same dataset (Fig. 2), we see that the sequence method yields much thicker bundles than the streaming method. There are two reasons for this:

Time window: In the streaming method, each bundled graph $\tilde{G}(t)$ contains edges which are alive in a time-window Δt. As detailed next in Sec. 6.3, we set Δt to match a small (5%) change in the number of edges in $\tilde{G}$. So, if the stream changes rapidly, Δt is quite small. In contrast, the sequence visualization of the stream (Fig. 7) corresponds to a very *coarse* regular time-sampling, where Δt is one-seventh of the entire stream duration. The keyframes G^i in this sequence are much larger than the instantaneous graphs $\tilde{G}(t)$ in the corresponding stream-based visualization, hence they yield thicker bundles.

Resolution: In the streaming method, the time-step δt for sliding the time-window controls the bundle tightness. As explained in Sec. 6.3, we set δt to $1/I$ times the average edge lifetime in the stream, where $I \simeq 10$ is the number of bundling iterations. For our flight data, this average is a few hours (an edge is a flight over France), so δt is a few tens of minutes. Hence, the 7-day streaming animation has hundreds up to a thousand frames, each being a slightly different graph bundling. In contrast, the sequence visualization of the same stream has only *seven* different graph bundlings. In-between frames are created by linear interpolation between keyframes. The keyframes are quite different, since flight patterns for consecutive days are different. Hence, linear interpolation has a strong tendency to relax bundles.

We see a large amount of red (appearing) and green (disappearing) flight edges in frames located between the seven keyframes (Fig. 7). This is due to the way we compute edge correspondences between keyframes: As stated at the start of Sec. 5.1.1, we match consecutive-day flights with close start and end points and same flight IDs. From

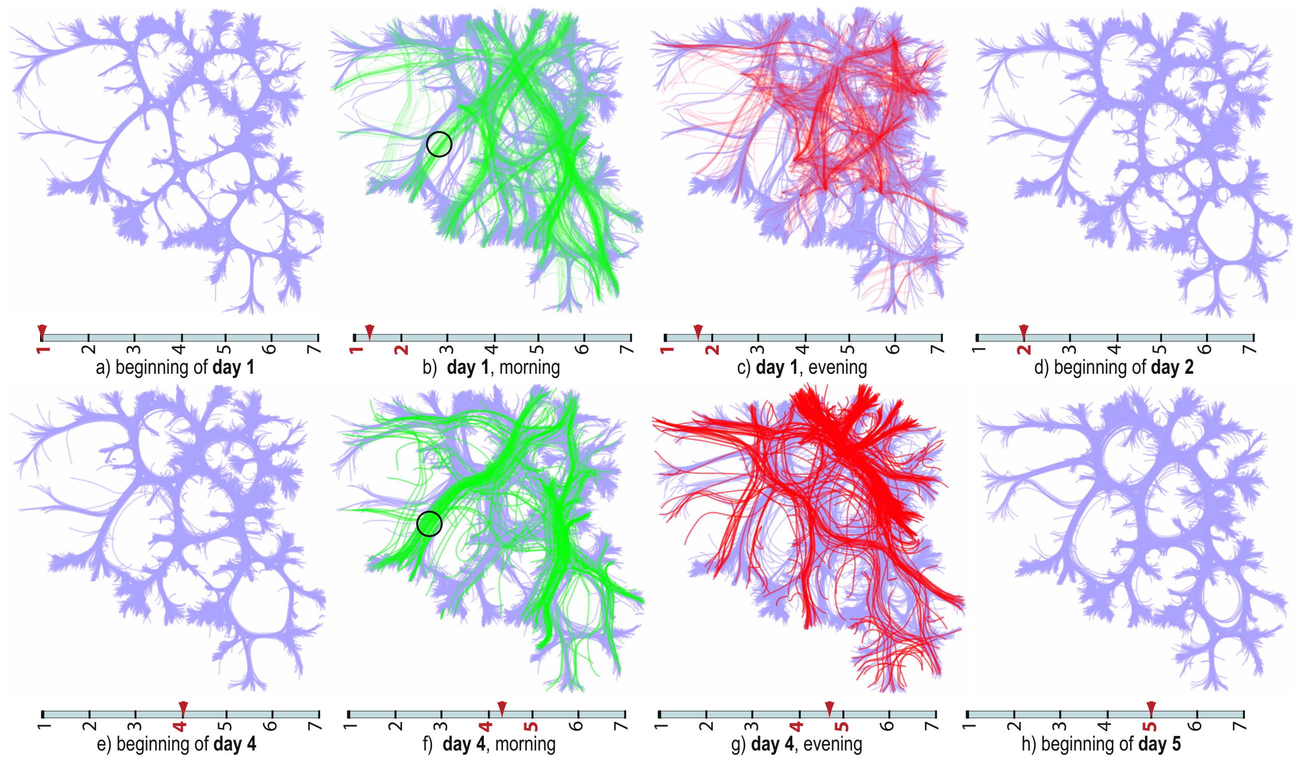
a) beginning of **day 1**
b) **day 1**, morning
c) **day 1**, evening
d) beginning of **day 2**
e) beginning of **day 4**
f) **day 4**, morning
g) **day 4**, evening
h) beginning of **day 5**
1 2 3 4 5 6 7

Fig. 7. Sequence visualization of streaming graph (France airline dataset). The sequence has 7 frames, each with the flights during one day. We show animation keyframes at the start and end of days 1 (a,d) and 4 (e,h), and intermediate in-between frames (b,c,f,g) (see Sec. 5.1.1).

54K edges, we obtain 16567 unique edges (counting corresponding edges as one). Hence, about 30% of all stream's edges have no correspondences, *i.e.* they appear red and green in the animation. This helps us find further insights. For example, in Fig. 7 (f), we see two large southwest-northeast green bundles. These are morning flights in day 4 which do not have similar morning flights in day 5. In Fig. 7 (g), we see two thick northwest-southeast red bundles appearing in the center and to the right. These are evening flights in day 5 which do not have similar evening flights in day 4. If we compare Figs. 7 (b,f), we see that one of the green bundles (marked in black) is similar. This means that, along that route, morning flights from day 1 were not present in day 2 *just as* morning flights in day 4 were not present in day 5. However, if we compare Figs. 7 (c,g), we see that the red bundles are very different. Since there are much larger red bundles in image (g), it means that there were much more evening flights appearing in day 5 *vs* day 4 than in day 2 *vs* day 1. These types of insight are not directly obtainable using the streaming visualization, since that visualization does not require, and thus does not depict, edge correspondence information.

We also tried a less restrictive correspondence criterion, *i.e.* matching flights in consecutive days which are spatially close (and thus may have different flight IDs). This yields only 8811 unique edges, *i.e.* about 16% of the stream edges will not have correspondences. Although this produces a smoother dynamic visualization, as there are more inter-keyframe correspondences, bundles will have weaker semantics. Specifically, we will be able to visually track the evolution of geographically similar flight groups, but we will not be able to say anything about these flights having the same ID.

Visualizing streams as graph sequences involves delicate data modifications, *e.g.*, cutting the stream at possibly irrelevant moments into disjunct chunks, and adding edge-correspondences that may not be meaningful. When such a transformation is not evident, and when fine-grained time data is important for comprehension, one should not visualize graph streams as graph sequences.

5.1.2 Sequences with stream-based visualization

For the second experiment, we convert our Wicket graph sequence (Sec. 4.2) to a streaming graph, by inserting 100 uniformly-spaced time moments between each two consecutive keyframes. This delivers a total of 700 frames, which is of the same order of magnitude as the frame count in a typical streaming graph visualization (see Sec. 5.1.1). We next visualize the resulting streaming graph using the streaming visualization method.

Figure 8 shows three frames from the resulting animation, taken between revisions 1.5.0 and 1.5.1. The sequence method (top row) shows a stable core indicating unchanging call patterns (blue bundles), and also outlines the removed calls (green) and added calls (red). As expected, these results are quite similar with the ones shown in Fig. 6, which were computed with the same method and for the same dataset. The bottom row in Fig. 8 shows the equivalent frames from applying the streaming method to the sequence graph. Although doing a good job in creating a smooth and stable bundling, this method cannot emphasize edge additions and removals, since it has no correspondence data to separate the treatment of stable and (dis)appearing edges.

5.2 Visualized attributed dynamic graphs with bundling

Bundling of streaming or sequence graphs highlights graph *structural* changes, *e.g.*, (dis)appearance or persistence of edges. However, many such graphs also have attributes. For instance, our French flight dataset has, at each recorded time-sample t_i, height data (see Sec. 3.2). From these, we can also compute derived attributes such as flight directions and flight speed. Correlating such attributes with the (bundled) flight paths provides additional insight. We next show how to add the following attributes to streaming bundled graph visualizations (none of which is depicted by the streaming graph visualization presented in Sec. 3.1):

A1: instantaneous positions of in-flight airplanes;
A2: height along flight trails;
A3: flight directions along trails;
A4: airplane flight speed along their flight trails.

As explained in Sec. 3.1, our streaming method considers all graph edges, or trails, in a window $w(t) = [t, t + \Delta]$. Rather than drawing entire trails T, we now consider *trail segments* $T_\Delta(t)$ which contain all sample points of an *unbundled* trail T falling in w. As background, we

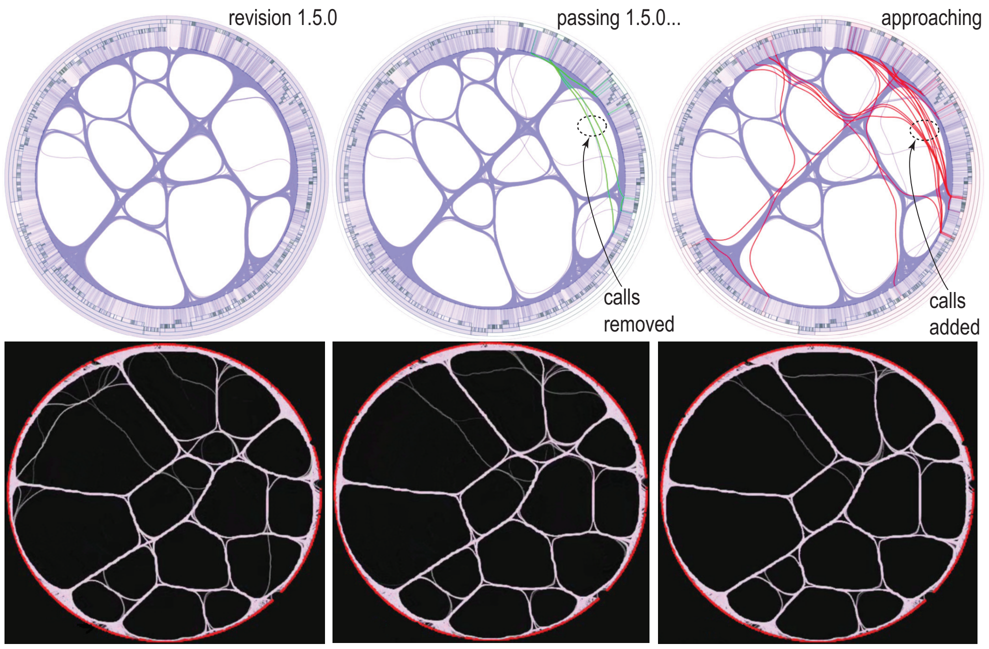

Fig. 8. Streaming visualization of graph sequence (Wicket dataset) *vs* equivalent frames in a sequence visualization (compare with Fig. 6).

draw the density map ρ (Eqn. 4) of all trails in w, luminance-coded. This creates the spatial *context* in which we can *focus* on plane motions along flight routes. We next texture the trail segments with an an alpha texture Φ_{Delta} built by placing Gaussian half-pulses ϕ_i at the sample point positions $\mathbf{p}_i$ under a Gaussian envelope over w (Fig. 10). We color-code trail segments by flight height (blue=low, red=high). Texturing serves two purposes. First, setting Δ to very low values creates images where the arrow-like (high to low alpha) shapes created by ϕ_i, and their motion, shows the instantaneous plane positions at a given time moment (**A1**), and their motion along trails (Fig. 9 a). Secondly, setting Δ to larger values creates 'trains' of arrow-like shapes that slide along trails (Fig. 9 b shows a snapshot from such an animation). Here, short pulses indicate slow-motion planes, while longer pulses show fast planes. For instance, in Fig. 9 b (inset), we see a fine-grained blue trail segment indicating a slow, low height, outlier flight in an area with fast (long pulse) and higher (green) flights (**A4**).

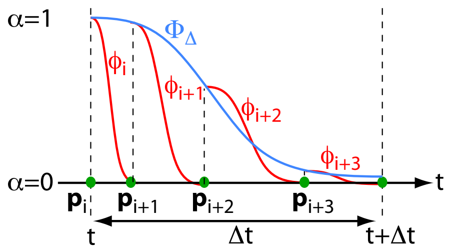

Fig. 10. Construction of directional pulses for animation (see Sec. 5.2).

Increasing Δ allows us to smoothly navigate from instantaneous views on the data to more global views. Figure 9) c shows this for Δ set to roughly 8 hours for our 7-days flight stream. Colors map flight heights (**A2**). Blue spots indicate regions densely populated by landing zones (airports). Warm lines show in-flight routes. By looking at the latter, we can see that most studied flights have the same altitude. This observation correlates with flight rules for civil aircraft for the studied territory (France). Figure 9) d shows a similar map, with trails colored now using a directional hue colormap (see colorwheel), thus addressing **A3** over the entire studied time period.. The direction color coding lets us discover several close-and-parallel, opposite-direction, flight paths, *e.g.* A_1,A_2; B_1,B_2, C_1,C_2 and D_1,D_2 (going southwest-northeast and conversely); and E_1,E_2 (going roughly northwest to southeast and conversely). Similar patterns (not shown here for conciseness) exist for the almost all other similar-size time intervals in the studied 7-day period. From such images, we can conclude that flights linking pairs of airports follow parallel paths but are structurally not overlapping in space.

In Figs. 9 (e,f), we next use the same color-coding as in In Fig. 9 d, but now the layout is given by two frames of the *bundled* streaming flight graph, which correspond to two moments in two different days in our 7-day sequence. Since trails are bundled, geographical (spatial) information is lost: The bundles indicate now just *connections* between airports, rather than actual flight paths. Still, directional color-coding is useful to show temporal insights. First, wee see that the connection pattern is roughly identical for the two studied moments. Flights in bundles A and B keep their directions over time, respectively northwest (green) and southeast (green). Flights in the big central white bundle structure C go equally in both directions at both studied moments, since white is the result of additively blending opposite colors in our colormap. In contrast, flights in bundle D go southwest (yellow D_1 in Fig. 9 e) and then return northeast at moment 2 (blue D_2 in Fig. 9 f).

5.3 Visual analysis of eye tracking trails

In the applications discussed so far, our temporal data was an explicit graph: For streaming graphs, nodes are airports and edges are flight paths between airports; for graph sequences, nodes are software artifacts and edges are clone relations. In both cases, bundling is an effective instrument to find coarse-scale connection patterns between groups of related nodes, and see how these patterns change in time.

In this section, we show two different usages of bundling for finding spatio-temporal patterns from non-graph datasets. We consider *trails* created by high-resolution eye trackers which record the instantaneous position of the gaze of a subject watching a given scene. A trail $S = \{\mathbf{p}_i\}$ (see Sec. 2.1) is thus the temporal trajectory of the subject's gaze. The so-called fixation points (FPs) $\mathbf{p}_i$ are points where a subject's eyes are relatively static, focusing on and attending to an object of interest in the watched scene. Fixation points are connected by continuous, ongoing eye movements called *saccades* [53, 20, 11].

Eye tracking analysis has a long history in experimental psychology [65, 53]. Figure 11 (a,b) shows some of the earliest recorded eye tracking datasets [65]. One key task involving eye-tracking trails is to extract the so-called *fixation areas* (FAs), or compact spatial zones where several FPs are clustered, *i.e.* areas around which the subject fo-

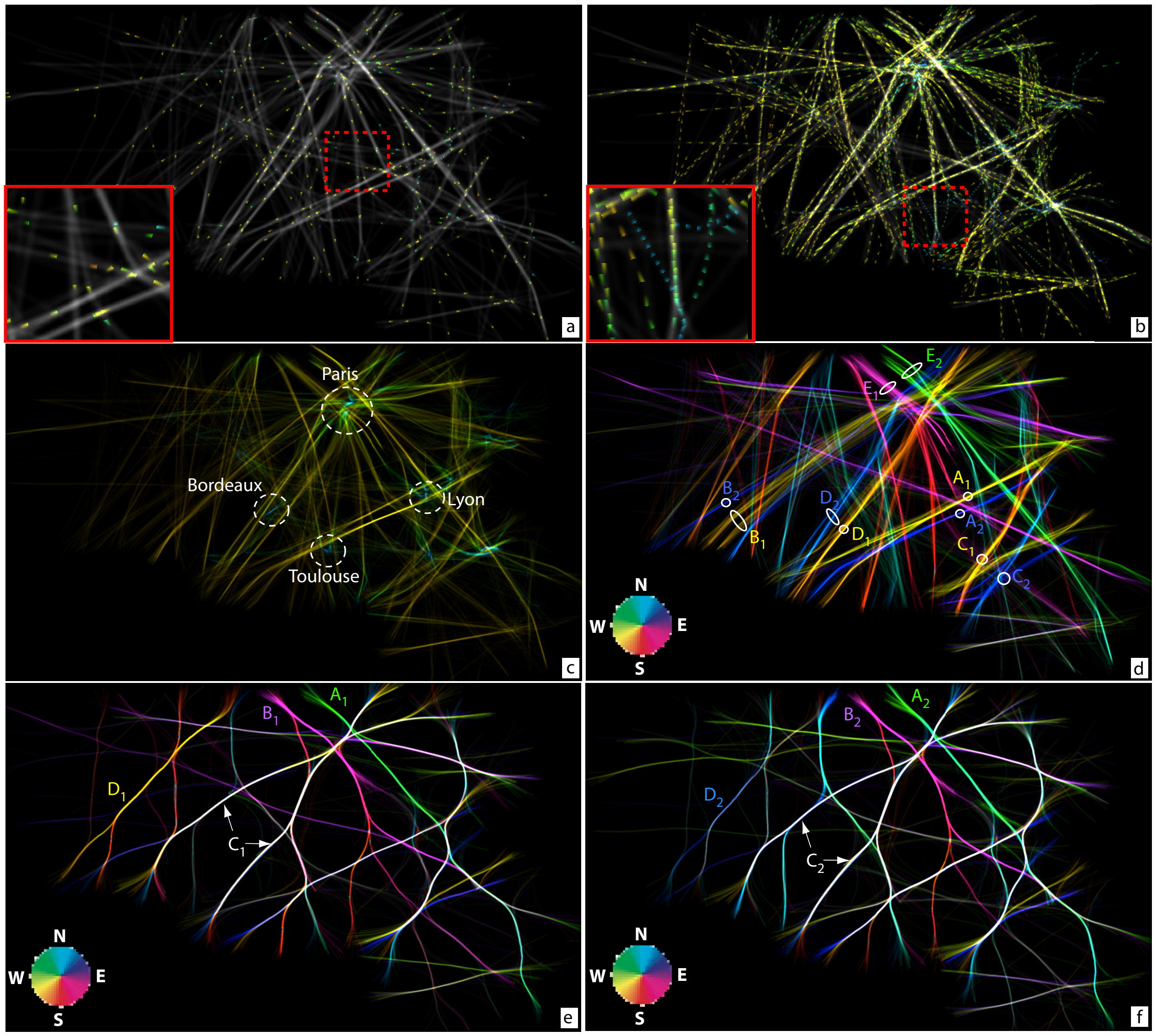

Fig. 9. Attribute visualization in dynamic graphs, French airline dataset. (a) Instantaneous plane positions, with color-coded height. (b) Trail segments over short time periods, with color-coded height. Trails over entire studied period with color coding height (c) and direction (d). (e,f) Bundled trails, color-coded by flight direction, at two different time moments during the 7-day studied period (see Sec. 5.2).

cused for a considerable period of time, or repeatedly, while scanning the scene. For example, Fig. 11 (c,d) shows the eye tracking of a subject driving a car [11]. In image (c), FAs are shown as high-value (red) isolines of the FP density map. This shows how the driver focused on the steering wheel, gear shift, and various dashboard instruments. In image (d), mean shift clustering [9] was used to find the *fixation area centers* (FACs), thereby producing a simplified view of the trails.

FP density maps and their clustered versions (Figs. 11 c,d) help finding the coarse-scale regions (and region centers) of subject attention in a scene. However, they do not address the following additional tasks:

T1: Find which FPs belong to which FA and FAC;

T2: Find the main scanning patterns connecting FAs to each other.

We next show how graph bundling can effectively address these tasks. We illustrate this with eye-tracking data obtained from watching both static scenes (Sec. 5.3.1) and dynamic scenes (Sec. 5.3.2).

5.3.1 Eye-tracking for static scenes: Viewing infographics

Small multiple displays (SMDs) are a well-known technique used In information visualization to compare a small set of similar objects by representing them side-by-side using similar visual encodings. They are effective in helping users to compare such objects and find pairwise differences or the scope of alternatives in a given set of options [56].

Although SMDs are typically static, several researchers argue for a continuum between static visualizations and dynamic visualizations, such as our dynamic graphs presented earlier. Krygier *et al.* characterize this continuum using *interactivity intensity* [35]: At one end, static graphics afford mental (internal) interactivity. For example, a SMD is mentally interactive in the sense that users can proactively control with their eyes the viewing order of this sequence; they can, *e.g.*, choose to study the sequence at their own pace and in any order they wish. In contrast, an animation featuring start, stop, and rewind buttons is slightly more (externally) interactive, but is less (internally) interactive, since it must be passively viewed in a predefined order.

In this context, a key challenge is to design objective success measures to evaluate static *vs* dynamic visualizations [19]. For this, Fabrikant *et al.* proposed the concept of *inference affordance* [20, 10] that integrates informational equivalence (amount and quality of content) and computational equivalence (inference quality and efficiency based on design) between two such displays [50]. To do this, they asked subjects to study a static SMD showing ice cream consumption over 12 months over the counties of a fictitious country, and next answer

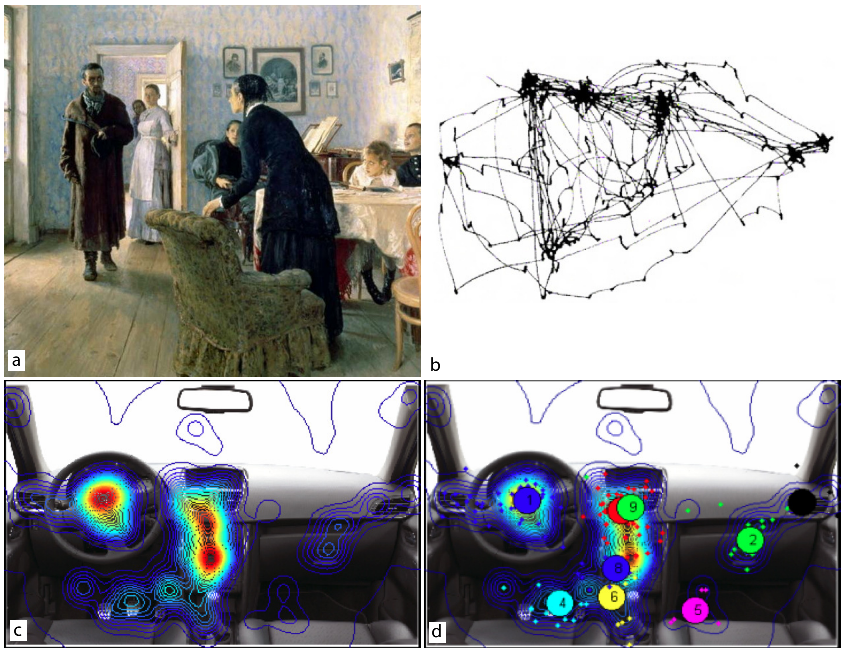

Fig. 11. Early studies of eye tracking: Static image focused by subjects (a). Eye tracking trails obtained during the image's free examination (b), data from [53]. Eye tracking for vehicle driving video: One frame with fixation area density isolines (c). Clustered fixation points (d), data from [11].

several questions on the data [20]. The subjects' eye movements were recorded and displayed atop of the SMD (Fig. 12 a).

However, raw eye movements are too unstructured to address task **T2**. To handle this, Fabrikant *et al.* proposed a combination of data *categorization*, *i.e.* the classification of temporal (sub)sequences into different classes, by extracting relevant sequence features; and data *summarization*, *i.e.* the simplified depiction of the gaze's dynamics, by agglomerative clustering of similar saccades. Figure 12 c shows a summarization of the trails in Fig. 12 a. Although summarization reduces clutter in the raw trails, and shows a simplified view, it also removes relevant data points. Also, depending on its parameters, clustering may produce too coarse summarizations which do not convey finer insights. Finally, clustering alters FPs as it creates and displays averages of the real FPs – so we cannot use such summarizations to find where the subjects precisely looked in the input scene (task **T1**).

To address the above, we apply our KDEEB bundling to the entire eye-tracking trail. Figure 12 b shows the bundling of the trails in Fig. 12 a. After bundling, several data patterns become clearly visible:

Fixation points: These points (shown red) are the same as in the raw data, as bundling does not move segment endpoints. This insight is lost by clustering (Fig. 12 c). Bundling also de-clutters the view by pulling away the eye trails from FPs, making the latter easier visible.

Scanning: The bundles in Fig. 12 b show that the subject scanned the image chiefly in vertical and horizontal patterns (task **T2**). This correlates well with the spatial layout of the SMD plot. A similar insight was found in [20], based on a more complex visual analysis of the raw trails. Our bundled plot makes seeing this pattern much easier (compare Figs. 3,4,5,9 in [20] with Fig. 12 b).

Fixation areas: Several star-like patterns formed by short edges linking red FPs with a FAC are visible (*e.g.*, the one located atop of the top-left plot). These indicate which FPs belong to a FA (task **T1**). By comparing these patterns across all multiples, we can see whether the users scanned similar areas in the maps, *e.g.*, if they viewed all counties in the twelve maps. From Fig. 12 b, this is not the case – both the red FACs and the star bundles are quite different for the twelve maps.

Semantic zones: The bundled graph (Fig. 12 b) has three different zones: At the top, a few bundles (blue markers) 'link' the plot title with the central trails. Half-way, we see the grid-like pattern that encodes how the subject has visually compared the maps. At the bottom, a long horizontal bundle covers the footnote text which shows the questions that users should answer using the plot. This bundle summarizes the eye motions involved in reading this text. This bundle is connected by four branches to the central grid-like pattern (green markers). This is an interesting finding, which showing that the subject switched *several* times back-and-forth between reading the task description and actually performing the task by looking at the SMD plot (task **T2**).

5.3.2 Eye-tracking for dynamic scenes: Pilot training

Aircraft pilots spend much of their training time using flight simulators. These devices include a realistic flight cabin, with steering controls, dashboard instrumentation, and computer-generated imagery for the window views. Just as for car drivers, flight training emphasizes several scanning patterns – the pilot has to scan the dashboard with a given frequency and in a certain order [64, 62]. Critical to both pilot training and proficiency assessment is finding how (and whether) the pilot scans the dashboard sufficiently and visually 'connects' the instruments as instructed [58, 34].

Figure 13 a shows a frame from a video recorded in a flight simulator engine. The simulation was performed at the EAB [15], the French authority responsible for safety investigations into incidents and accidents in civil aviation. The background shows the simulator cabin. Bright spots show the lit dashboard instruments, such as altimeter, artificial horizon, heading indicator, wet compass, turn coordinator, and airspeed meter. The jagged curve in the foreground is the entire recorded trail of the pilot's gaze, recorded with the Pertech system [43] (similar to Figs. 11 b and 12 a). To estimate the FAs, we compute the trail's 2D spatial density ρ, using the same technique as for graph bundling (Sec. 3.1, Eqn. 4). Coloring the trail by ρ using a heat colormap (yellow=low, red=high) shows several red spots, which match the FAs. Toggling the trail visualization on and off, we can check if the FAs correspond to the dashboard instrument locations that were required to be scanned during the training sequence. However, just as for the example in Fig. 11 c, this image does not show how the pilot's gaze navigated between instruments, *i.e.*, it does not show the *structure* of the scanning sequence (task **T2**).

To recover this structure, we first apply the KDEEB bundling on the entire trail data. Figure 13 b shows the bundling result color-coded by the bundled trails' density. The result is insightful. First, bundling removes much of the clutter of the original trails, so we now can better

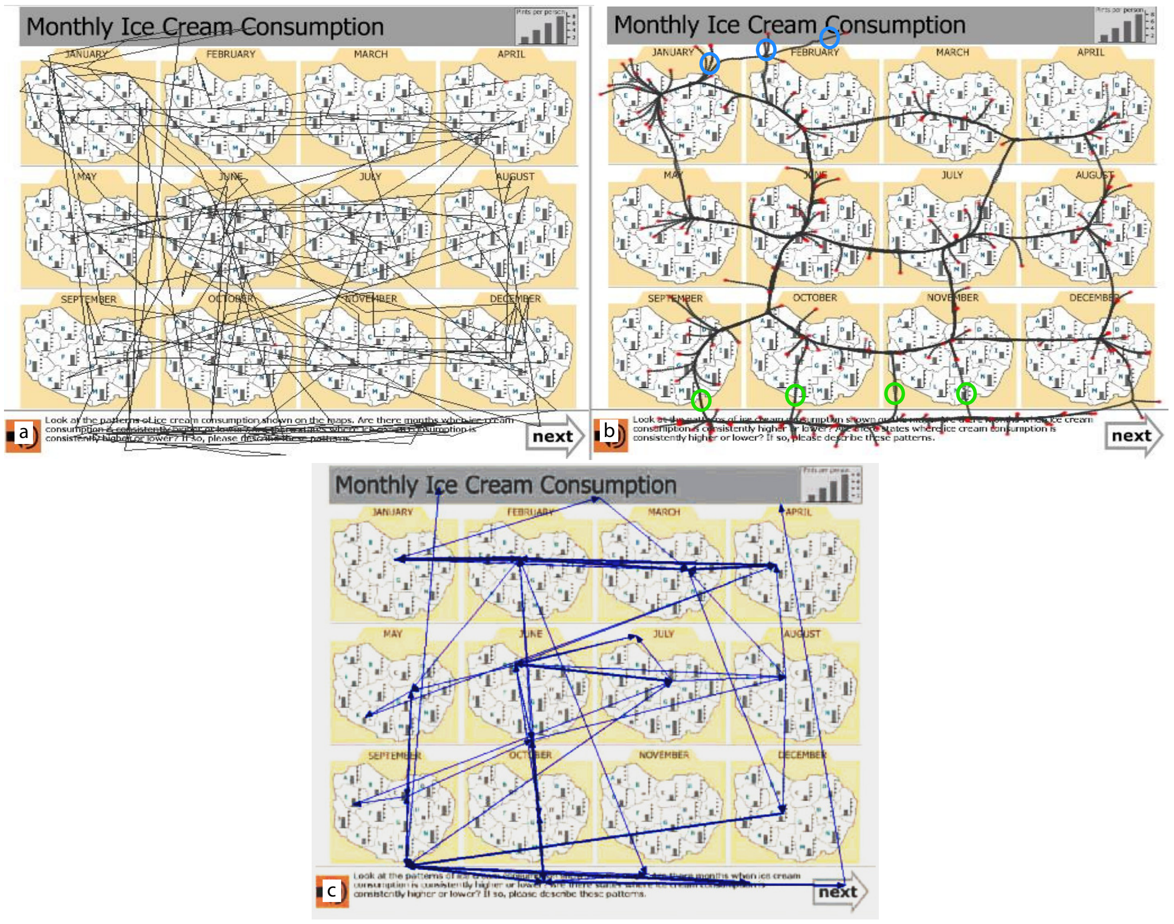

Fig. 12. Eye tracking data of a subject viewing a small multiples display. (a) Raw trails. (b) Bundled trails. (c) Summarized trails (Sec. 5.3.1).

see how FAs match the dashboard instruments. Secondly, and more importantly, we see that the gaze trail consists of two main patterns:

A. Star-like patterns occur around the fixated instruments and show the distribution (spread) of FPs in the FAs. These patterns are quite small and also regularly spread in the x, y directions, which means that the subject focused in a statistically uniform way over the respective instruments. The star 'centers' are the fixation area centers (FACs). Since KDEEB uses the same mean shift technique as in [11], it means that these centers are equivalent to the ones computed by [11] (see *e.g.* Fig. 11 d). The star 'branches' show which FPs belong to a FAC, an insight not shown in Fig. 11 d. This addresses task **T1**.

B. Bundles connecting FACs show how the subject's gaze moved from instrument to instrument during the training sequence. Essentially, we reduced the trail to an implicit graph whose nodes are FPs and FACs; short FP-FAC bundles show which FPs belong to a FAC; and long FAC-FAC bundles show how FACs are related to each other in the scanning sequence. This addresses task **T2**. Analyzing this graph lets us quickly see whether the subject's gaze 'connected' the scanned instruments in the desired pattern or not. For instance, we see a cross-like pattern in the central area of the image. Flight experts from ENAC confirmed that this cross corresponds to a known, desired, dashboard-scanning pattern that pilots have to obey during their flight (scan several instruments left-to-right and back, scan other instruments top-to-bottom and back, then revert the gaze to the dashboard center). Comparing Fig. 13 b (bundled) with Fig. 13 a (unbundled), we see that this kind of insight is not visible in the unbundled (raw) data, but is easily obtainable in the bundled image.

However, the analysis in Fig. 13 removes the time dimension, as we bundled *all* eye saccades in a single image. This delivers a *coarse* summarization of the eye movement, *i.e.* tells us which FAs the subject visually connected during the *entire* experiment. However, we also want to analyze how the gaze dynamics changed across various time intervals. For this, we use the stream bundling (Sec. 3.1) with a window Δ set to a few seconds. Figure 14 shows the results for six consecutive moments where interesting patterns were detected. In image (a), we see how the pilot's attention mainly focuses on the central instruments (FA area A_1), while also doing a quick scan of peripheral instruments (loop L_1). Next, the pilot keeps on focusing on the central instruments (FA area A_2, image (b)), while the peripheral scanning pattern is similar, but breaks the earlier loop structure. Next, in image (c), the pilot focuses more on the peripheral instruments, as denoted by the two scanning loops L_3 and L'_3, but also keeps an eye on the central instruments, as these loops are connected to the central screen area. In image (d), we see a repetition of the initial pattern (a), with the pilot focusing on the central instruments (A_4) and quickly scanning the periphery (L_4). The same pattern persists in image (d), compare A_4 *vs* A_5 and L_4 *vs* L_5. At the end of the training sequence (image (f)), the pilot focuses most on two instruments of the dashboard center, A_6 and A'_6.

To conclude, both static bundling (Fig. 13) and dynamic bundling (Fig. 14) of temporal trails are useful, but for different aims, as follows:

- Bundling entire sequences is useful when want to see connection patterns between fixation areas, regardless of *when* and in which *order* these patterns occurred in time. Hence, static bundling shows the structure of an entire gaze trail;

- Dynamic bundling is useful when we want to see what the subject did around a certain *moment*, and how different moments resemble and/or follow each other. Hence, dynamic bundling emphasizes short-term spatio-temporal patterns in the gaze trails.

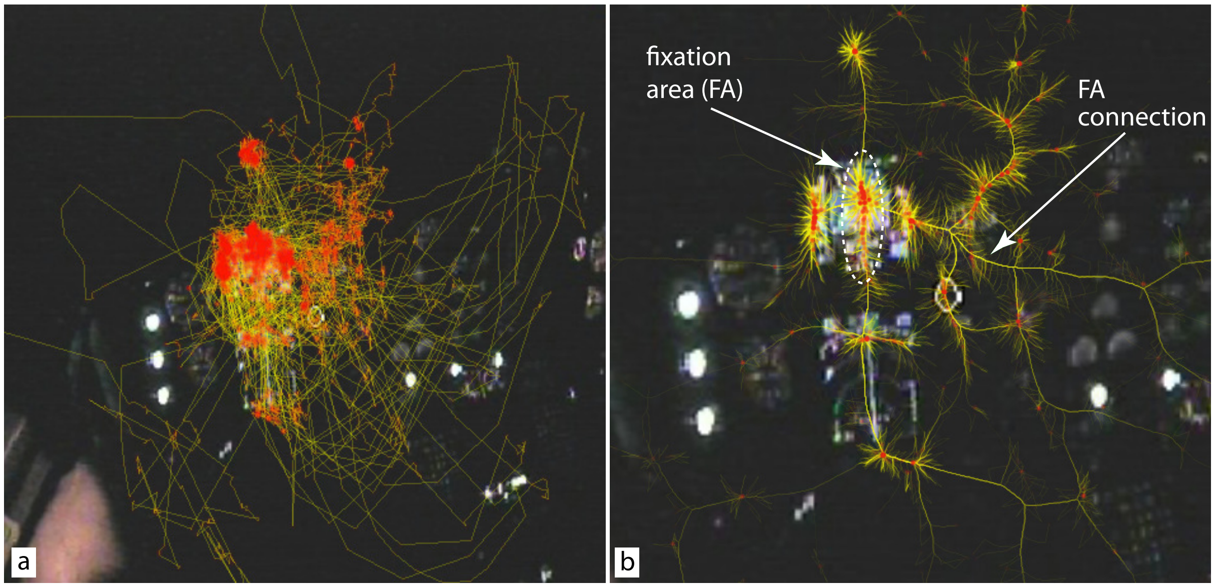

Fig. 13. Visual analysis of pilot eye-tracking data, with static bundling. Raw trails (a). Bundled trails (b). Color shows trail density.

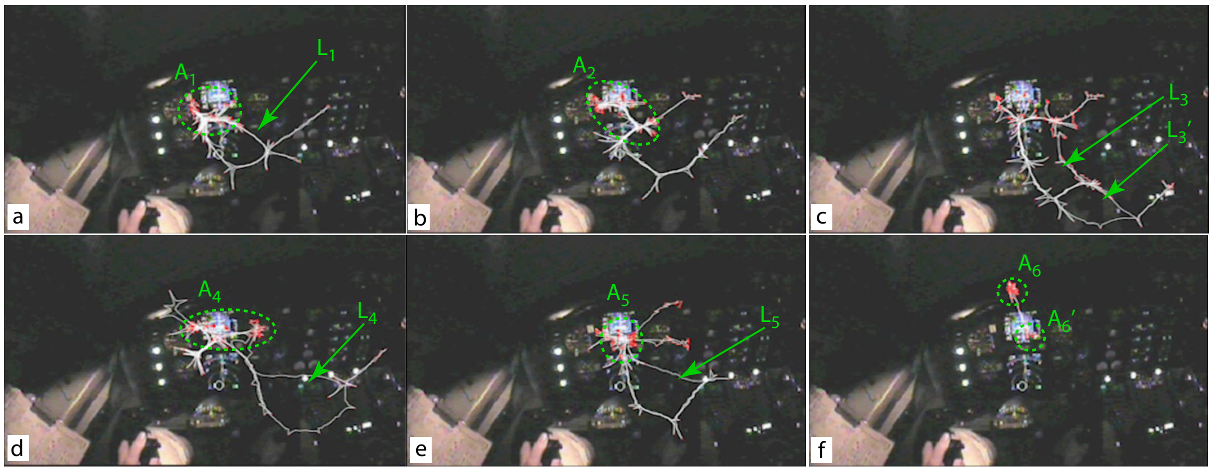

Fig. 14. Visual analysis of pilot eye-tracking data, with dynamic bundling. Six salient eye-movement patterns are shown (Sec.5.3.2).

6 Discussion

6.1 Scalability

Streaming graphs: The streaming graph method (Sec. 3.1) has a complexity of $O(|\tilde{E}|)$ per animation frame, where $|\tilde{E}|$ is the average number of edges in any time-window of size Δt at any moment t in the stream. This is so since we run the bundling process in sync with the stream time, as explained in Sec. 3.1. In other words, there is a single density-splat and advection step for each edge present in a frame. In contrast, StreamEB [41] is $O(|\tilde{E}|^2)$ per frame. We implemented our graph streaming and sequence visualizations in C# using the KDEEB algorithm which is itself written in C# using OpenGL 1.1, and ran them on a 2.3 GHz PC with 8 GB RAM and an NVidia GT 480 card. On this platform, producing one streaming-animation frame took 0.05 seconds for the US dataset ($|\tilde{E}| = 2$K edges on average) and 0.17 seconds/frame for the France dataset ($|\tilde{E}| = 15$K edges on average). Per frame, we are roughly 10 times faster than the original KDEEB ([31], Tab. 1), which is expected, as we do only one iteration per frame (see Sec. 3.1). Using FDEB as core bundling algorithm requires, for the US dataset, 19 seconds/frame on comparable hardware ([28], Sec. 4.2). Using StreamEB for a graph of $|\tilde{E}| = 900$ edges on a 1.7 GHz PC requires 6 seconds/frame ([41], Fig. 12).

Graph sequences: The sequence graph method (Sec. 4.1) isf $O(BN)$ for a sequence of N graphs and an underlying bundling algorithm of complexity B. This is the same cost as StreamEB, modulo the fact that our bundling algorithm B is faster, as already explained. Also, our produced visualization is different, since we (a) emphasize (dis)appearing edges and (b) smoothly interpolate consecutive bundled layouts by using edge correspondences.

Online graphs: In graph visualization, we distinguish between *online* methods, which can treat graphs as they become available, and *offline* methods, where the entire streaming graph or graph sequence must be known in advance [23, 41]. Both our streaming and sequence visualizations are online methods. For streaming graphs, we only need to know the edges in a time-window of size Δ around the current moment. For graph sequences, we only need to know the previous keyframe G^{i-1} and next keyframe G^{i+1} around the current keyframe G^i.

6.2 Static bundling algorithm choice

Streaming graphs: For this case, KDEEB is a good solution: KDEEB works for general graphs, produces bundles with little clutter even for complex graphs, and is robust and simple to use. However, most important point is that KDEEB allows to *incrementally* update the graph *during* the bundling. In contrast, most other bundling methods need a full bundling when the input graph changes. This is due to various technical factors, *e.g.*, use of spatial search data structures and compatibility metrics that need reinitialization upon graph changes [17, 12, 25, 38], or encoding the bundle polylines separately from the input graph's straight-line edges [25, 37, 48]. FDEB comes closest to KDEEB in flexibility, as it represents (partially) bundled edges as a set of unstructured polyline curves, so it can be used for incremental smooth bundling upon graph changes. However, KDEEB's linear complexity in the input graph size makes it more suitable than FDEEB which is quadratic in the same input size.

Graph sequences: Here, any bundling algorithm can be technically used. However, KDEEB proved better than alternatives. Figure 5 shows the differences between HEB (top row) *vs* KDEEB (bottom row). HEB produces less structured and compact bundles. A similar effect can be seen in StreamEB [41]. Figure 6 shows the differences between using SBEB [17] (top row) *vs* KDEEB (bottom row). SBEB produces actually too much structure – the bundles have too many branches. This is explained by the fact that SBEB needs to discretely partition the input graph edges into clusters of similar edges, which are next bundled separately. Since clustering is done per keyframe, SBEB cannot guarantee that clusters vary continuously between keyframes. In contrast, KDEEB produces less clutter than SBEB, but more structure than HEB, thereby offering a good visual balance.

6.3 Parameters

Our streaming-based visualization uses the same edge sampling, smoothing, kernel size, and density-map resolution parameters as KDEEB [31]. The parameters added by our streaming method are the size of the time-window Δt and time-step δt for sliding this window (see Alg. 1). Δt controls how much one sees in one animation frame: Larger Δt values show more (bundled) edges, but inherently smooth out the dynamics of the animation. Smaller values show more of the instantaneous graph $G(t)$, but make short-lived edges (dis)appear faster. In our examples, we used a Δt corresponding to a 5% change in the number of edges in $\tilde{G}$, so that animation goes faster over uninteresting time periods, similarly to [41]. δt controls the ratio between the animation speed and the stream's own speed *and* also the bundling tightness. Large δt values subsample the stream, *i.e.* make the animation go faster and show less tight bundles, since, as outlined in Sec. 3.1, bundling occurs in sync with the stream time. Smaller δt values supersample the stream, *i.e.* make the animation go slower and also create tighter bundles. In practice, getting tight bundles with KDEEB requires roughly $I = 5..10$ iterations [31]. Hence, we set δt to $1/I$ of the average edge lifetime in the stream. A good side-effect of this setting is that bundling reflects the edge lifetime: Short-lived edges, likely outliers, do not strongly bundle. Long-lived edges, which contribute to the coarse-scale structure of the graph, get strongly bundled. Apart from Δt and δt, our algorithm has no other parameters.

6.4 Limitations

Currently, we showed that we can bundle graph streams and sequences in a fast, smooth, and clutter-free manner, and that such animations help assessing connection stability and spot fast-changing bundles (Secs. 3.2 and 4.2). However, the animation and visual mapping metaphors, *i.e.* speed, shape, tightness, and shading of bundles, would need to be adapted to support seeing finer-grained events of interest such as bundle splitting, or merging; similar bundles in far-apart time frames; and separating bundles based on additional edge attributes. Also, a quantitative and qualitative measurement of the effectiveness of animated bundles is needed.

When graph edges encode relevant spatial information, bundling introduces the risk of misinterpreting this information in the final (bundled) image. For dynamic graphs, as compared to static graph bundling, this risk increases. Indeed, since bundling displaces edges from their actual positions, dynamic bundling will create edge-motion patterns which can be far from the actual edge-motion patterns in the data. However, this does not imply that one should never bundle graphs in such situations. We see here a gradation of this degree of risk, depending on how this information is precisely used:

A. No relevance: Edge positions do not encode any information besides relations. This is the case of the software graphs in Sec. 4.2. Here, dynamic bundling has a low risk of conveying 'wrong' insight. The key dynamic patterns of interest are bundle splitting and merging, and not the precise location or precise motion speed of bundles.

B. Indirect relevance: Edge positions encode relevant information. However, this information is used only indirectly. This is the case of the eye-tracking trails in Sec. 5.3, where edges actually describe the trails of the subject's gaze. By bundling, we loose the ability to follow the track of the subject's gaze. However, we gain the ability to see coarse-scale patterns such as groups of fixation points, how these are related to each other by visual scanning, and whether similar scanning patterns exist in the image. Since eye-motion analysis in the context of usability and human-machine interaction relies mainly on such patterns rather than the fine-scale tracking of eye movements, we argue that bundling is a low-risk and useful instrument in this scenario.

C. Direct relevance: Edge positions encode information directly pertaining to the questions of interest. This is the case of some scenarios for the flight trails in Sec. 3.2. If we are interested, *e.g.*, to find how flight path spatial *distribution* changes over several days, we cannot use dynamic bundling, as this method only shows how the local spatial *mean* changes over time.

Using animation, texturing, and color mapping, we can show up to three attributes, *e.g.* flight height, flight direction, and flight speed atop of both unbundled and streaming bundles (Sec. 5.2). However, we acknowledge that such techniques have limitations. Pulse animation along trails works relatively well for reasonably crowded areas (Fig. 9 b), but would result into unreadable high clutter if applied to bundled graphs, like the ones in Figs. 9 (e,f). Also, the flight dataset we studied so far (French flights) changes relatively slowly and continuously in time. As such, users can follow the corresponding color, texture, and animation changes to decode the displayed attribute values. For graphs with much higher dynamics, however, such solutions may not work, and further study is required.

7 Conclusion

We have presented two algorithms for the animated visualization of graph streams and sequences. By exploiting the smoothness, stability, speed, and incremental nature of the recent KDEEB image-based bundling algorithm, we succeed in creating streaming graph animations which exhibit the same desirable properties. Next, we use the same algorithm to generate sequence-based graph visualizations where edge appearance and disappearance events are emphasized. We apply our techniques on several large datasets from air traffic monitoring, software engineering, and eye tracking, and present evidence that supports our choice for KDEEB as underlying layout.

Future work can address animation, visualization, and interaction refinements to emphasize finer-grained events of interest, such as bundle merging and splitting, and support tasks such as detecting graph patterns that match problem-specific questions. Also, user evaluations can help in validating and refining the design choices presented here.

References

[1] N. Andrienko and G. Andrienko. *Exploratory analysis of spatial and temporal data: a systematic approach.* Springer, 2006.

[2] F. Beck and S. Diehl. On the impact of software evolution on software clustering. *Empir. Softw. Eng.*, 2012. DOI: 10.1007/s10664-012-9225-9.

[3] J. Bertin. *Sémiologie Graphique. Les diagrammes, les réseaux, les cartes.* Gauthier-Villars, 1967.

[4] C. Binuccia, U. Brandes, G. D. Battista, W. Didimo, M. Gaertler, P. Palladino, M. Patrignani, A. Symvonis, and K. Zweig. Drawing trees in a streaming model. *Inform. Process. Lett.*, 112(11):418–422, 2012.

[5] I. Boyandin, E. Bertini, and D. Lalanne. A qualitative study on the exploration of temporal changes in flow maps with animation and small-multiples. *CGF*, 31(3):1005–1014, 2012.

[6] M. Burch, F. Beck, and S. Diehl. Timeline trees: Visualizing sequences of transactions in information hierarchies. In *Proc. AVI*, pages 75–82, 2008.

[7] M. Burch and S. Diehl. TimeRadarTrees: Visualizing dynamic compound digraphs. *CGF*, 27(3):823–830, 2008.

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

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

[10] A. Çötelkin, S. I. Fabrikant, and M. Lacayo. Exploring the efficiency of users visual analytics strategies based on sequence analysis of eye movement recordings. *Int. J. Geogr. Inf. Sci.*, 24(10):1559–1575, 2010.

[11] T. Couronné. *De la prise d'information visuelle à la formation d'impressions: apports de l'oculométrie pour l'étude des processus de la perception et de la cognition visuelle des objets manufacturés*. PhD thesis, Univ. Grenoble, France, 2007.

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

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

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

[15] EAB. Bureau d'enquêtes et d'analyses pour la sécurité de l'aviation civile, 2013. www.bea.aero.

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

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

[18] C. Erten, S. Kobourov, V. Le, and A. Navabi. Simultaneous graph drawing: Layout algorithms and visualization schemes. In *Proc. Graph Drawing*, pages 437–449, 2004.

[19] S. I. Fabrikant and K. Goldsberry. Thematic relevance and perceptual salience of dynamic geovisualisation displays. In *Proc. ICA/ACI Intl. Cartographic Conference*, pages 9–16, 2005.

[20] S. I. Fabrikant, S. Rebich-Hespanha, N. Andrienko, G. Andrienko, and D. R. Montello. Novel method to measure inference affordance in static small-multiple map displays representing dynamic processes. *The Cartographic Journal*, 45(3):201–215, 2008.

[21] C. Fish, K. Goldsberry, and S. Battersby. Change blindness in animated choropleth maps: An empirical study. *Cartogr. Geogr. Inform.*, 38(4):350–362, 2011.

[22] D. Forrester, S. Kobourov, A. Navabi, K. Wample, and G. Yee. Graphael: A system for generalized force-directed layouts. In *Proc. Graph Drawing*, pages 454–464, 2004.

[23] Y. Frishman and A. Tal. Online dynamic graph drawing. In *Proc. EuroVis*, pages 75–82, 2007.

[24] K. Fukunaga and L. Hostetler. Estimation of the gradient of a density function with applications in pattern recognition. *IEEE TIF*, 21:32–40, 1975.

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

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

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

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

[29] M. Huang, P. Eades, and J. Wang. On-line animated visualization of huge graphs using a modified spring algorithm. *JVLC*, 9(6):623–645, 1998.

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

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

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

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

[34] J. Kim, S. A. Palmisano, A. Ash, and R. S. Allison. Pilot gaze and glideslope control. *ACM Trans. Appl. Perception*, 7(3), 2010.

[35] J. B. Krygier, C. Reeves, D. DiBiase, and J. Cupp. Multimedia in geographic education: Design, implementation, and evaluation. *J. Geogr. Higher Educ.*, 21(1):1739, 1997.

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

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

[38] S. Luo, C. Liu, and K. L. Ma. Ambiguity-free edge-bundling for interactive graph visualization. *IEEE TVCG*, 18(5):810–821, 2012.

[39] Mozilla. Firefox repository, 2012. www.mozilla.org/en-US/firefox.

[40] Q. Nguyen, P. Edges, and S.-H. Hong. StreamEB results, 2012. rp-www.cs.usyd.edu.au/~qnguyen/streameb.

[41] Q. Nguyen, P. Edges, and S.-H. Hong. StreamEB: Stream edge bundling. In *Proc. Graph Drawing*, pages 324–332, 2012.

[42] J. R. Pate, R. Tairas, and N. A. Kraft1. Clone evolution: A systematic review. *J. Soft. Maint. Evol. Res. Pract.*, 2012. DOI:10.1002/smr.579.

[43] Pertech Inc. Eye tracker system, 2013. en.pertech.fr.

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

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

[46] D. Reniers, L. Voinea, O. Ersoy, and A. Telea. The Solid* toolset for software visual analytics of program structure and metrics comprehension: From research prototype to product. *Sci. Comput. Program.*, 2012. doi:10.1016/j.scico.2012.05.002.

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

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

[49] T. Shipley, S. I. Fabrikant, and A. Lautenschütz. Creating perceptually salient animated displays of spatiotemporal coordination in events. In *Cognitive and Linguistic Aspects of Geographic Space*, pages 259–270. Springer, 2013.

[50] H. A. Simon and J. H. Larkin. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11:65100, 1987.

[51] SolidSource. SolidSDD clone detector, 2012. www.solidsourceit.com.

[52] Statistical Computing. US flights dataset, 2012. stat-computing.org/dataexpo/2009/the-data.html.

[53] B. Tatler, N. Wade, H. Kwan, J. Findlay, and B. Velichovsky. Yarbus, eye movements, and vision. i-*Perception*, 1:7–27, 2010.

[54] A. Telea and D. Auber. Code flows: Visualizing structural evolution of source code. *CGF*, 27(3):831–838, 2008.

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

[56] E. R. Tufte. *Envisioning information*. Graphics Press, 1990.

[57] B. Tversky, J. Morrison, and M. Betrancourt. Animation: Can it facilitate? *Intl. J. Human Computer Studies*, 57:247–262, 2002.

[58] K. van de Merw, H. van Dijk, and R. Zon. Eye movements as an indicator of situation awareness in a flight simulator experiment. *Intl. J. Aviation Psychology*, 22(1):78–95, 2012.

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

[60] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, and D. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *CGF*, 30(6):1719–1749, 2011.

[61] C. Ware and R. Bobrow. Motion to support rapid interactive queries on node-link diagrams. *ACM. TAP*, 1(1):3–18, 2004.

[62] N. Weibeland, A. Fouse, C. Emmenegger, S. Kimmich, and E. Hutchins. Let's look at the cockpit: exploring mobile eye-tracking for observational research on the flight deck. In *Proc. Symp. on Eye Tracking Research and Applications*, pages 107–114. ACM, 2013.

[63] Wicket. Apache Wicket, 2012. wicket.apache.org.

[64] J. Yang, Q. Kennedy, J. Sullivan, and R. Fricker. Pilot performance: Assessing how scan patterns and navigational assessments vary by flight expertise. *Aviation, Space, and Environ. Medicine*, 84(2):116–124, 2013.

[65] A. L. Yarbus. *Eye Movements and Vision*. Plenum Press, 1967.

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

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