Zuerst ersch. in : IEEE Transactions on Visualization and Computer Graphics : T-VCG. - 18 (2012), 12. - S. 2486-2495
http://dx.doi.org/10.1109/TVCG.2012.238

Interactive Level-of-Detail Rendering of Large Graphs

Michael Zinsmaier, Ulrik Brandes, Oliver Deussen, and Hendrik Strobelt

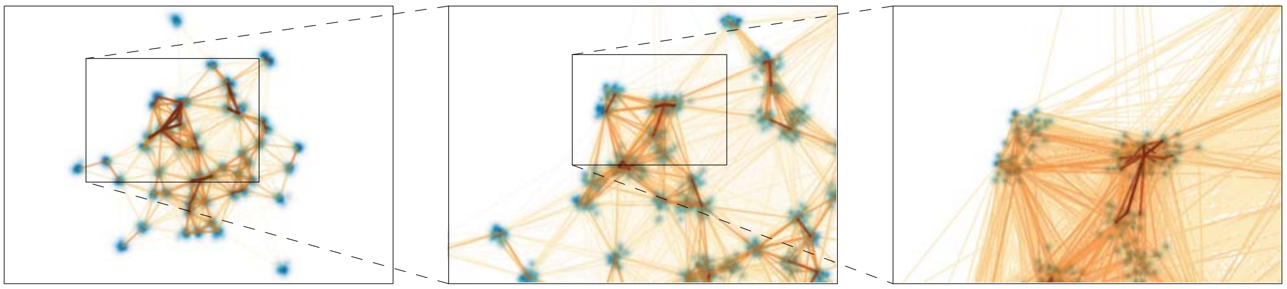

Fig. 1. Application of our visualization technique on a hierarchical data set, zooming from overview (left) to a region of interest (right). The density-based node aggregation field (blue color) guides edge aggregation (orange/red color) to reveal visual patterns at different levels of detail.

Abstract— We propose a technique that allows straight-line graph drawings to be rendered interactively with adjustable level of detail. The approach consists of a novel combination of edge cumulation with density-based node aggregation and is designed to exploit common graphics hardware for speed. It operates directly on graph data and does not require precomputed hierarchies or meshes. As proof of concept, we present an implementation that scales to graphs with millions of nodes and edges, and discuss several example applications.

Index Terms—Graph visualization, OpenGL, edge aggregation.

1 Introduction

We present methods for the interactive visualization of large graphs. We say a graph is large if it fits into video memory but cannot be rendered as node link diagram without significant over-plotting, thus we define size relative to the computing environment. For the interactive exploration of such graphs fast node and edge aggregation is needed in combination with efficient rendering in different levels of detail (LOD). Both is presented in the following. Our techniques enable us to show graphs with up to $\sim 10^7$ nodes and up to $\sim 10^6$ edges at interactive rates.

Lampe and Hauser [20] describe a method for rendering large graphs as density fields based on a GPU implementation of *Kernel Density Estimation (KDE)*. Our method extends their technique for node aggregation by a two-pass seed point rendering that significantly reduces geometry and scales to large graphs. Furthermore we present a fast edge aggregation method that derives start- and endpoints of multi-edge representatives from the underlying node density field in image space. Our method is able to render a graph with a given layout without preprocessing. No auxiliary data structures such as meshes or node hierarchies are needed.

Our techniques allow to render large graphs on common graphics hardware at interactive rates. In combination with a zoom based user interface and flexible KDE bandwidth manipulations efficient graph exploration is possible as well as interactive and intuitive node labeling. Additionally, our techniques provides flexible mechanisms to define the representation of aggregates by modifying the normalization function and used colors as described below.

The paper is organized as follows. First we describe and discuss the related work in Section 2. A detailed description of our methods is given in Section 3, performance considerations are discussed in Section 4. An interactive system based on the proposed techniques is described in Section 5. We present its interaction paradigms and some example applications. Finally, we summarize and propose future work in Section 6.

• *all authors are with University of Konstanz;*
E-mail: <surname.name>@uni-konstanz.de.

2 Related Work

We divide the problem of rendering large graphs on (comparatively) small displays into two main problems: dense regions of nodes and cluttering of edges. While the first is the general problem of dense point sets commonly faced in visualization and computer graphics, the second problem is more closely related to structure-aware methods from information visualization and graph drawing.

2.1 Node Visualization Methods

Nodes can be displayed as small elements, at minimum a single pixel can represent one node. Nevertheless, clutter and over-plotting are common challenges in point data and scatterplot visualizations. One solution to the problem is to slightly displace the nodes to reduce over-plotting on the whole dataset [18]. Alternatively, optical distortion techniques and zoomable user interfaces can improve the visualization in a local area [6]. While both methods preserve the distinct node representation they either distort the shape of node groups [18, 6] or display only parts of the dataset [6].

Density-based visualizations avoid these problems and can be applied to large datasets. Density can be measured by dividing the visualization into bins and counting the number of data points that fall into them. This approach is closely related to histograms and color codes or symbols can be used to represent the local density [8]. For instance alpha blended scatterplots are an implicit form of this technique with a bin sizes equal to one pixel and a color encoded density representation. However similar to histograms the bin sizes and binning borders introduce a bias to the visualization. Instead of such discrete density representations basis functions can be used to accumulate the point influences into a continuous density field. Leeuw and Liere [28] propose a GPU-accelerated technique that visualizes accumulated Gaussian basis functions as continuous field

Konstanzer Online-Publikations-System (KOPS)
URL: http://nbn-resolving.de/urn:nbn:de:bsz:352-246910

similar to a heat map. Cao et.al. [7] use Kernel Density Estimation on nodes to derive a cluster hierarchy which is input to hierarchical edge bundling [14].

2.2 Edge Visualization Methods

In many cases, graphs are visualized as node link diagrams. In addition to node clutter, such representations suffer also from over-plotting of edges, since such elements cannot be displayed as efficiently. Node layout algorithms serve as a global solution to untangle node link diagrams while distortion techniques provide local improvements, e.g. fisheye lenses [24] generate more screen space for interesting areas. Furthermore, the drawing paradigm can be altered to be more space efficient. Becker [5] suggests to draw the links only half the way to their targets. These “half lines” reduce edge clutter and crossings but make tracking edges more difficult.

Apart from such improvements for edge rendering it is common to shift the focus from single edges to edge aggregates. Such aggregates can be loose edge bundles, edge replacements through meta edges or edge density patterns. In either case single connections become harder to read but overall patterns become more prominent. The aggregation strategies are commonly classified into hierarchical methods, confluent drawing, edge bundling techniques, and density based methods.

Hierarchical methods define a pyramid of simplified versions of a graph. This pyramid can be constructed by merging nodes based on graph theoretic measures [13] or geometric clustering [22]. The edges are merged to meta edges, which are explicitly defined and represented. The approaches are often designed as interactive systems that allow the user to browse different aggregation levels by expanding/collapsing nodes. Reducing the amount of visible elements allows exploration of datasets with several million edges [4] on current displays. The creation of the aggregates requires offline computations, except for datasets that already contain natural hierarchies.

Confluent Drawing represents graphs without edge crossings. Dickerson et al. [10] present for instance a heuristic algorithm that replaces clique and biclique edges with an explicit “traffic circle” aggregation metaphor. However confluent drawing can not be applied on general graphs and “the complexity of deciding whether a general graph is confluent or not still remains an open problem” [10].

Edge Bundling techniques share a common metaphor: they join similar parts of edges to bundles. Single nodes and edges remain in the visualization, while visual bundles highlight higher level patterns. The aggregates are often defined explicit by a hierarchy [14], a mesh [9, 19] or based on clustering techniques [12, 11, 21]. However, the standard representation here is implicit with loose bundles of variable diameter indicating the aggregates [15, 11, 14]. Still there are some exceptions like edge to edge force based implicit bundling strategies [15] or flow like explicit representations [21] and clustering based post processing techniques [26]. Edge bundling scales for graphs of medium size ($\sim 1-10$k edges), the algorithms are designed to optimize bundling with allowance for non realtime execution. Gansner et.al. [12] use a proximity graph and an “ink saving” measure to allow fast hierarchical edge bundling of large graphs. Recently, Hurter et.al. [17] described an iterative approach which operates on a hight field created from applying KDE on edge sample points. In each iteration step, the sample points are moved along the gradients towards local maxima. With each repetition, the kernel bandwidth is decreased, which sharpens the density field and results in tighter and more separated bundles.

Density Based approaches transform the discrete node link diagram into a continuous representation. Leeuw and Liere [28] represent graphs with node density fields and Lampe and Hauser [20] extend the method to edges, using *line kernels*. Density based techniques use a pure implicit aggregation that exists only as visual pattern in the visualization. The implicit aggregation allows fast rendering without preprocessing and the techniques can be applied to streaming data [20].

2.3 Discussion of Related Work

Bundling approaches, like Gansner et.al. [12] or Hurter et.al. [17], reduce visual complexity by grouping edge segments to bundles. Lampe and Hauser [20] use a line kernel method to visualize an edge density field. Although using different metaphors, these approaches aim at highlighting edge patterns. In contrast, we focus on relationship patterns between node density clusters. And thus extend the well known density representation [20, 28] of point data sets with a closely coupled visualization of inter-cluster connections. FacetAtlas [7] relates to this idea, but addresses smaller magnitudes of elements.

Similar to Lampe and Hauser our approach operates in image space for fast processing using OpenGL. The method generates *implicit edge aggregates* which appear as a series of pixel values and do not link back to the edges they are aggregating. Object space approaches (like [12]) retain this mapping between *explicit edge aggregates* and edges and benefit from additional rendering options (e.g. changes of aggregate geometry or per aggregate shading). But image space algorithms can reduce the computational complexity and enable us to achieve the required rendering rates for interactive exploration.

3 Methods

The core of our contribution for visualizing large graphs is a new density-based node *and* edge-aggregate representation that addresses the cluttering problem. Our approach works completely in image space, does not use complex spatial data structures and utilizes various features of today’s graphics hardware.

We start by describing the generation of continuos node density fields in Subsection 3.1. Such fields provide an overview of large amounts of data without neglecting the influence of a single node. Then we propose a hill climbing approach that uses the obtained node density field to guide the positioning of aggregated edges in combination with the aggregated nodes. Furthermore, Subsection 3.2 addresses a common problem of overdraw-based edge aggregation: colors change at edge crossings and distort the visual appearance. We solve this problem by a new drawing routine that takes edge orientation into account.

In the following we refer to nodes and node coordinates in object space, while using pixel and pixel positions to describe elements in image space (screen space).

3.1 Kernel Density Estimation for Node Aggregation

As mentioned above, we use density fields to visualize node aggregates. Lampe and Hauser [20] propose a fast algorithm that generates such fields by utilizing the features of todays graphics hardware. Their approach is inspired by kernel density estimation (KDE) and has the advantage of very naturally aggregating the node details, as Silverman says “..the data will be allowed to speak for themselves...” [25]. In the following we introduce their method with our adaption and describe our extensions that allow scaling to large node sets.

The $2D$ kernel density estimator for n data points x_i $i \in 1,2,..,n$ is defined as:

$$f_{K_H}(x) = \frac{1}{n}\sum_{i=1}^{n} K_H(x-x_i) \tag{1}$$

with kernel function K_H. We adjust the formula such that each point contributes a weight of one at its position and a decreasing influence to its neighborhood. Similar to other publications [20, 28] we choose a *2D Normal Kernel* for K with influence matrix H. For simplicity we use an isotrope influence of the kernel in all directions and thus can restrict H to multiples of the identity matrix. A scalar parameter h is used to control the bandwidth of the kernel function. Additive blending of textured rectangles is used to determine $f_{K_H}(x)$ for each pixel. Each node coordinate x_i is mapped from object space to a pixel position x_i' in image space. For *every* node a *geometry shader* creates a rectangle at position x_i'. The rectangle is filled with a precomputed floating point texture that approximates the kernel function. The contribution of the nodes to the density field is accumulated with additive blending of the rectangles in the rasterization pipeline (Figure 2 (a)).

For large node sets, however, the described method is time inefficient because the amount of textured rectangles depends linearly on the number of nodes within the screen area. Therefore, we extend the method and significantly reduce rendering time by binning nearby node coordinates into pixels. Instead of rasterizing multiple rectangles at the same position, a single weighted rasterization step per bin is then sufficient to create the same density field.

Typically, binning is done using spatial data structures such as a *Point Region Quadtree (PRQ)* [23] which provides fast access queries while (offline) construction is cheap (Figure 2 (b)). PRQs represent a point aggregation hierarchy by recursively dividing the viewport. When querying the Quadtree, the hierarchy is cut at an appropriate level of detail and the next smaller aggregation level is returned. However, in most cases the Quadtree regions do not exactly fit pixel sizes, thus causing geometric overhead for non-merged nodes. Furthermore, the induced storage requirements for meta nodes add up to significant amounts for large graphs.

Therefore we propose the **Seed Point Method**, a technique that uses two render-passes: one to merge the nodes in their pixel-sized bins and one to create the actual density field. In the first pass the node coordinates x_i are mapped to pixel positions x_i' of an *Accumulation Field*. The Accumulation Field is a frame buffer object (FBO) in screen resolution. Node coordinates that fall into the same pixel position are cumulated by additive blending.

After the first render pass, the value of each pixel therefore represents the amount of accumulated nodes within its screen space area. The second render pass binds the Accumulation Field as a texture and creates a weighted textured rectangle for every pixel of value ≥ 1 using a Geometry Shader (Figure 2 (c)). Doing so, our approach determines for a given viewport resolution the optimal input set to create a density field and minimizes the overhead during the kernel blending. Moreover it does not need any additional storage and fits directly into the OpenGL pipeline.

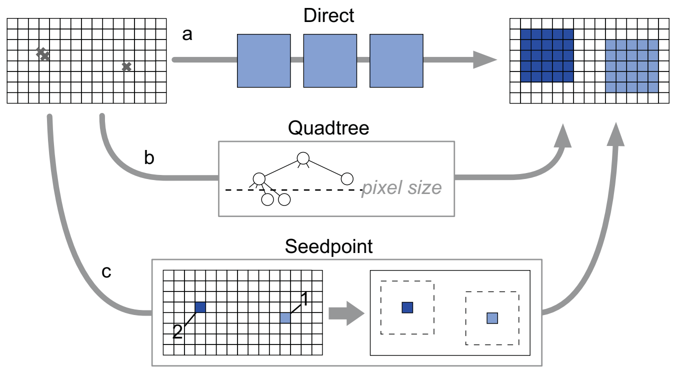

Fig. 2. Rendering KDE kernel textures for points in a dataset to create a continuous density field. a) the direct approach: place a texture for each point; b) use a pre-calculated Point Region Quadtree to bin points; c) the *Seed Point* method uses a two-pass rendering to first accumulate points per pixel and secondly uses a geometry shader to render parametrized kernel textures.

3.2 Hill Climbing for Edge Aggregation

In the next step we use the node *Accumulation Field* as input for a new edge aggregation approach that is visually coupled to aggregated node clusters. Our method bases on the idea that the aggregation of edges between peaks of the node field is an intuitive visual metaphor. In contrast to other edge aggregation techniques our technique operates in image space and therefore does not rely on precomputing supporting data structures.

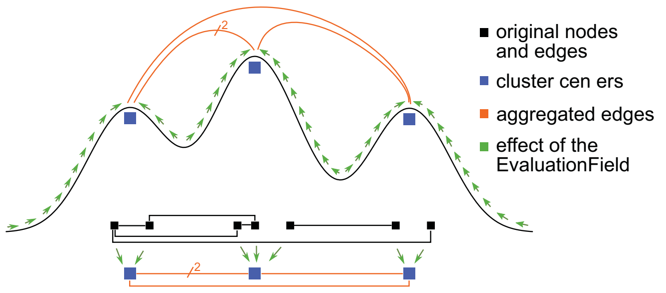

Fig. 3. The hill climbing method is shown as a lateral cut of a height field (our interpretation of the node density). The original edge points (indicated in black at the bottom of the figure) are translated to the nearest hilltops (blue). The edges cumulate to aggregates (orange) which connect the cluster centers (hilltops).

We interpret the node density field as height field and use a hill climbing algorithm to move starting and end points of the edges to the highest point of their visual cluster (the *local maximum*). The edges whose start points fall into the same cluster and whose end points share another cluster are implicitly aggregated. As result, inner-cluster connections are vanished while inter-cluster connections are emphasized. Figure 3 demonstrates the effect. The graph is shown in the lower part of the figure. The original edges are given in black while the aggregated versions are in orange. The density field is shown as hills in the upper part. The blue points are the aggregate edge points, the orange edges the remaining aggregate edges. The edge points correspond directly to the highest peaks in the node aggregate visualization.

Generally, hill climbing moves points towards a *local maximum* which is what we want. If the density field contains several small local maxima within a large plateau, the corresponding “inner-cluster” edges are drawn. To avoid this line clutter within plateaus, the KDE bandwidth h can be adjusted to retrieve less edges.

As mentioned above, hill climbing is implemented in OpenGL: The *Accumulation Field* (see *Seed Points* method) is bound as texture t_{AF} and a *fragment shader* operates on this texture as shown in algorithm 1. Within the *Accumulation Field* for every start pixel the associated hilltop-pixel is searched by iteratively tracking 3x3 neighborhoods. As starting points we use every pixel that accumulates at least one node. The result of this step serves as input for a *geometry shader* that repositions the points on the GPU towards the hilltop positions. The outcome of the process is an edge aggregation texture where each pixel value represents the number of edges that pass through it.

Algorithm 1 Hill climbing in image space

```
Require: texture $t_{AF}$ of the Accumulation Field
  for all pixel positions $p \in t_{AF}$ do
    $p_{temp} \leftarrow p$
    if $value(p_{temp}) \neq 0$ then
      $run \leftarrow true$
      while $run$ do
        $K \leftarrow$ 3x3 neighborhood of $p_{temp}$
        $p_{max} \leftarrow$ pixel position in $K$ with highest value
        if $p_{max} \neq p_{temp}$ then
          $p_{temp} \leftarrow p_{max}$
        else
          $run \leftarrow false$
        end if
      end while
    end if
    assign $p_{temp}$ as hilltop to $p$
  end for
```

It is important to note that the algorithm does not return explicit meta edges but aggregation results for the edges in form of pixel values. Along with these aggregation results, three visualization challenges arise: I) handling of edges that leave the viewport, II) the rendering of appropriate line widths, and III) avoiding visual hotspots on edge crossings. While the first problem only relates to our method, the latter two are of general interest for density based methods. We present solutions for all of them in the following.

Edges that leave the viewport: In a simple implementation the Accumulation Field would not be defined outside the viewport and thus edges that leave the visible area would not be aggregated. The rendering subsequently becomes unstable and panning operations may alter edge aggregates. These problems are reduced by aggregating nodes beyond the viewport borders: we suggest combining the visible Accumulation Field with an off screen field four times the visible area. Since this context is not visible and only used for aggregation we can work with a reduced resolution of the Accumulation Field outside the screen window. Using half of the resolution outside the visible window only doubles the total node aggregation costs but is already sufficient to stabilize the edge renderings of most graphs against panning and feathering out at the visible borders.

Rendering of thick edges: Edge coloring based on overdrawing is a common way to visualize aggregate weights in implicit edge representations. In contrast to explicit techniques the geometry of the aggregate cannot be altered here. In particular, it is not possible to render important edge aggregates with thicker lines because they consist of many over-plotted single edges and the rendering pipeline is not able to generate links between edges and aggregates. Therefore all edges are treated equally and are rendered with the same line width.

We therefore propose a post-processing step using a *fragment shader* that alters line thickness on the edge aggregate texture and creates the illusion of explicit aggregate definitions. We define a distance function $dist(p_i, p_j)$ between two pixels p_i and p_j and a function $width(value(p_x))$ which maps a pixel value to a specific edge width. To determine the color of a pixel p_i the shader inspects a $w \times w$ neighborhood S of pixel p_i. All pixels $p_j \in S$ which fulfill the following condition are further considered as candidates for coloring:

$$dist(p_i, p_j) \leq \frac{width(value(p_j))}{2} \tag{2}$$

Among the candidates the shader chooses the p_j with the highest value to use it for coloring of p_i. Using the maximum as selection criterion implies that important lines are rendered on top of others. The parameter w defines the size of the neighborhood and thresholds the maximal line-width. We chose a value of 5 to be sufficient for our approaches.

A natural choice for a distance function $dist(d_i, d_j)$ would be the Euclidean distance. However, the symmetric character only allows uneven edge widths of one, three, five, etc. Therefore we propose an asymmetric pseudo Euclidean distance function that shifts the edge centers slightly from their intended positions but complements the visualization with even edge widths. Figure 4 describes the distance functions.

For example, given a pixel at position (x_e, y_e) and an attached edge width of two. Measuring with Euclidean distance, the pixel to the left $(x_e - 1, y_e)$ and to the right $(x_e + 1, y_e)$ would be colored, resulting in a width of three. Using our pseudo Euclidean distance, only the left pixel $(x_e - 1, y_e)$ would be colored. The distance of the right pixel $(x_e + 1, y_e)$ is two and therefore it does not satisfy the initial condition of equation 2: $2 \not\leq 1$.

Avoiding hotspots on edge crossings: Measuring the overdraw for computing edge widths and colors works well for isolated edges. If multiple aggregates cross the same pixel, however, all involved edges accumulate. Edge crossings therefore become hotspots and produce disturbing artifacts in the visualization. Moreover the density peaks distort the normalization for color mapping and width assignment and constrain the visualization space for actual edges (Figure 5 left).

We propose to render edges separated by angles to reduce artifacts of edge crossings. The result is a set of textures that is merged with a max value filter to eliminate the crossing artifacts (Figure 5 middle and right).

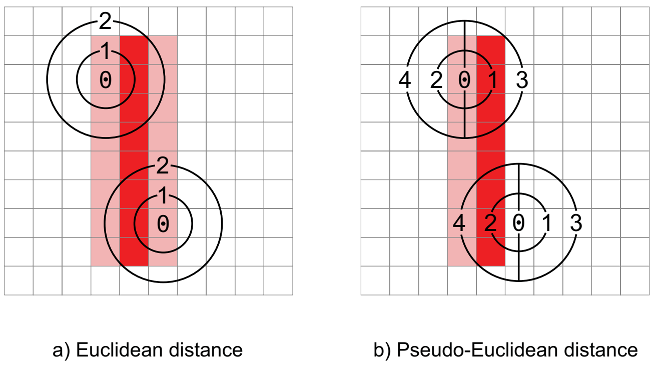

Fig. 4. Euclidean and Pseudo Euclidean distance functions used for widening edges. The aimed line width in the given example is two, i.e. $width(value(red_{pixel})) = 2$. For the symmetric Euclidean distance the left and right pixels are colored (they have distance 1). For the Pseudo Euclidean distance only the left pixel is colored, the right pixel has distance 2 and fails the selection criterion $2 \not\leq \frac{2}{2}$.

This technique is implemented in OpenGL by adding two render passes. In the first pass the edges of four different angle segments are rendered into the four separate color channels of a texture. The second pass uses a maximum filter to merge the obtained texture with the edge aggregation texture. We suggest using 180 angle segments of one degree (45 texture runs) for an undirected graph.

4 Measures & Complexity

The presented visualization system is based on image operations that are executed in parallel on the GPU. Moreover, many render operations build upon cheap texture transformations and in particular only three rendering steps depend on the input data. For each node a single point has to be added to the AccumulationField, each edge has to be rendered to the edge aggregation texture and the kernel rectangles have to be created. The last step depends on the number of Seed Points, which is limited by $min(\#pixels, \#nodes)$. We denote the constant render cost for one Seed Point with R_s, for one rectangle with R_r and for one edge with R_e. The worst case complexity for p pixels, n nodes and m edges is defined as:

$$O(n \cdot R_s + min(n, p) \cdot R_r + m \cdot R_e) \tag{3}$$

Thus we are able to maintain a linear dependency towards the size of the input data. Moreover, the architecture of GPUs compensates for growing data sizes through parallelization. In practice, nearly constant render times can be achieved for graphs with up to $\sim 10^6$ edges on common hardware (Figure 6).

Table 1 gives an overview of the test datasets and measured results. The US air-traffic and US migration graph are well known examples for edge bundling algorithms. The US census [27] data set provides more migration data with weighted edges and at a bigger scale. WikiVote, Net50 and Net150 are available at the Sparse Matrix Collection [2] and have been laid out with SFDP[16] to allow fair comparison to results of MINGLE. The 4.5M graph and H3 graph (Figure 1) are artificial test datasets for edge aggregation and the Europe dataset serves as node rendering benchmark. The latter one has been extracted from OpenStreetMap [1] and includes one point for each tagged building.

Figure 6 compares the presented approach to the results of MINGLE [12] and KDEEB [17], two of the, to our knowledge, fastest algorithms for edge bundling. To allow a comparison to KDEEB on large data sets we include a random graph of the same size as described in [17]. The random graph is marked explicitly because of it's unusual properties: two times more nodes than edges and quasi-equal node distribution . Overall the diagram shows a roughly linear correlation between the number of edges and the render time for MINGLE. The results for KDEEB are more difficult to interpret but the authors

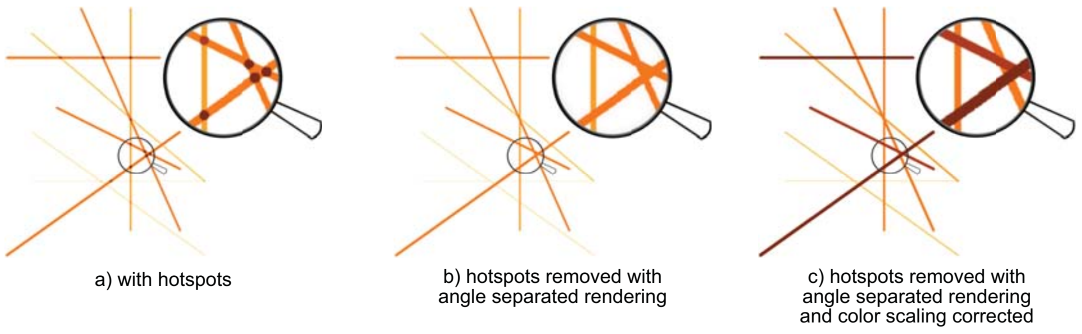
a) with hotspots
b) hotspots removed with angle separated rendering
c) hotspots removed with angle separated rendering and color scaling corrected

Fig. 5. (a) Edge crossings become hotspots and distort the coloring and edge widening. (b) Angle separated rendering removes the artifacts, (c) color scaling and edge widths assignment is corrected.

Table 1. Rendering times in seconds. The marked columns (*) have been reported in [12] (CPU approach) and in [17] (GPU GTX 580). The remaining columns measure the average GPU time to create an overview of the whole graph (left) for 32px and 128px kernel size. The worst rendering times found by manual inspection are given at the right columns. Resolution is 1500x750 pixel.

	description			overview				bad case	
dataset	**source**	**#nodes**	**#edges**	**MINGLE***	**KDEEB***	**32px**	**128px**	**32px**	**128px**
US air-traffic	[3]	275	1,925	0.1	0.5	0.3	0.3	0.3	0.3
US migration	/	1702	9780	1.0	1.5	0.2	0.3	0.4	0.4
Wiki-Vote	[2]	8,436	103,660	18.4	/	0.3	0.4	0.3	0.5
random graph	/	200,000	100,000	/	18.0	0.4	2.2	0.4	2.2
net50	[2]	16,320	464,440	87.1	/	0.4	0.4	0.4	0.4
US census	[27]	3075	545,882	/	/	0.5	0.5	0.6	0.6
net150	[2]	43,517	1,538,840	355.0	/	0.4	0.4	0.7	0.8
4.5M	/	99,965	4,551,564	/	/	1.2	1.2	2.1	2.5
Europe	[1]	37,612,093	/	/	/	0.2	0.5	0.4	2.4

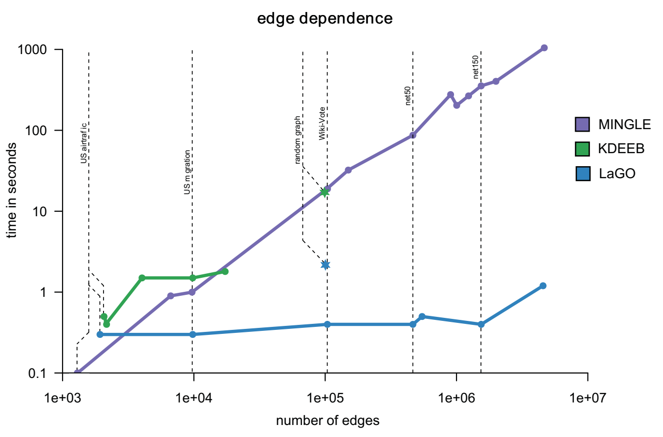

Fig. 6. Log-log scale comparison of the results reported for MINGLE [12] and KDEEB [17] with the presented approach (128px kernel). The GPU implementation allows our approach to maintain nearly constant rendering times below one second up to a graph size of $\sim 10^6$ edges on real world data. The two marked data points refer to the artificial random dataset with special properties.

state that they achieve about the same speed as MINGLE for larger graphs and a linear correlation can thus also be assumed. In contrast, our algorithm achieves nearly constant results for all datasets $\leq 10^6$ edges.

Our test system consists of a high end consumer graphics board (Nvidia GTX590) and an i7 - 2600K CPU. The critical factor, however, is the GPU performance and similar results can be achieved with several ATI graphics cards (HD6950 +).

An analysis of our measurements, the shader code and the outlined data dependencies shows that the rasterization of the kernel rectangles and the render costs for edges have the highest impact on the overall performance and we therefore discuss these factors in the following:

Kernel Rasterization: The costs for kernel rasterization depend on the number of Seed Points and the size of the rectangles. The amount of Seed Points is difficult to estimate because it depends on the current viewport and the image resolution. The above discussed border of $min(\#pixels, \#nodes)$ is used as upper limit. In practice, real world datasets normally do not have a uniform node distribution (i.e. not all pixels contain nodes) and the actual number of rectangles is thus often much smaller than the amount of pixels or nodes. In the following we will speak of a weak linear dependency that is relevant for some data sets like the artificial random graph but often overestimates the problem (e.g. Europe data set). We provide measurements of an “overview” and a “bad case” viewport to illustrate the viewport effects on our test data (Table 1). The second factor (the size of the rectangles) relates quadratically to the selected bandwidth h and determines the texturing costs for the kernels. Table 1 and figure 6 show results for a small kernel with diameter of 32px and a large variant with 128px.

Rendering of Edges: The texture for the edge aggregation is created by the already described angle-separated rendering of all edges of the current viewport. The costs for edge rendering depend linearly on the amount of edges m with a constant factor for the overhead caused by our multi-pass rendering technique.

Despite the weak (nodes) and strong (edges) linear dependencies on the data size the image based design of our approach allows us to maintain near constant render times for large graphs and also point datasets. At some point, however, the hardware limits are reached and parallelization is no longer able to compensate the growth of data. On today’s consumer hardware our approach scales well to graphs with $\sim 10^6$ edges or point datasets with $\sim 10^7$ nodes. Additionally it would be straight forward to distribute the render steps on multiple GPUs to deal with even bigger problem spaces.

5 Applications

In this section we describe LaGO (Large Graph Observer), an implementation of our methods and give examples of data sets visualized using the tool.

5.1 LaGO

LaGO provides zoom and pan operations to support the interactive exploration of visualized graphs. Similar to related density based approaches [20, 28] the bandwidth parameter h is bound to the viewport to support semantic zooming. For instance increasing the zoom level shrinks the viewport such that a smaller subset of the data is displayed and reduces the bandwidth (world space) to decrease the aggregation level. However, we not only shrink and expand the node clusters but also adjust the closely integrated edge aggregates.

The interactive exploration leads to large variations of the visualized density values and fixed color mapping cannot address sparse and crowded viewports equally. We therefore provide an automatic normalization of the densities with the maximal value of the current viewport. The normalized density values are in the interval $[0..1]$ and we use them as texture coordinates such that the density to color mapping can be defined with free choosable color schemes.

The automatic normalization is convenient, however in some cases more control may be desirable for instance to maintain a consistent color mapping during pan operations. We therefore provide an interface for manual fine control that can be used to lock the normalization devisor or to set it to a specific value. If a fixed devisor overshoots the normalization interval the system clips the results and indicates the potentially misleading color-mapping to the user by highlighting the lock symbol.

The distribution of aggregated items might be uneven and hotspots skew the normalization. We therefore provide a graphical user interface that allows the application of scaling functions to the linear normalized values. Normalization, scaling and the value to color mapping can be used in combination to filter the data and balance weak against strong visualization elements.

The described viewport control and filter mechanisms allow the exploration of large graphs at variable levels of detail. However apart from geometry and connectivity information graph datasets often contain node names. LaGO therefore provides a detail on demand implementation that allows the interactive labeling of the node density field. The hill climbing approach from the edge aggregation can be used to group the nodes according to their cluster center and provides an intuitive cluster definition for prominent areas in the visualization. The user selects a cluster by clicking onto it and the system retrieves the cluster nodes and sorts them according to their degree or a dedicated weight. Subsequently a list of the most prominent labels is displayed and the user selects labels from the list to annotate the cluster nodes (Figure 7).

5.2 Examples

US air-traffic data

Figure 7 shows an application of our method to the US air-traffic data set. We selected all flights in the US from July 2011 ($\geq$ 500.000) and boiled them down to 1925 distinct connections. These connections are interpreted as undirected weighted edges to create the visualized graph. The weight of an edge is equivalent to the the number of flights on the connection and the weight of a node is the sum of the connected edge weights. The color scaling for edge and node aggregates is non linear, as can be seen on the scaling bars in the right lower corner. Furthermore the open lock, above the scaling bars, indicates that the automatic normalization of data values is active. The applied color scheme focusses on strong edges, while small edges are attenuated by transparency. The prominent flight hubs (Atlanta, Chicago, LA, San Francisco, East coast) appear immediately and strong connections like LA $\Leftrightarrow$ San Francisco are marked by color and width. When zooming in (i.e. reducing the bandwidth) the aggregates split and smaller airports like San Diego in the LA area or Sacramento near San Francisco appear (Figure 8). The most detailed view reveals finally the three different airports of the San Francisco Bay Area and their individual connections to LA, which sum up to approximately one percent of all flights in July 2011. In contrast to the edge density visualization of Lampe and Hauser [20], the integration of nodes has the advantage that airports and their relative importance can be clearly spotted in our visualization and edge crossings with high densities cannot be misinterpreted as airports.

US migration data

Figure 9 and figure 10 visualize US migration data. The first data set is commonly used to depict the effects of edge bundling algorithms (e.g. [17]) and consists of 9,780 unweighted edges. In contrast the second data set is much larger with 545,882 weighted edges and 3075 nodes. It is based on the Census 2000 and contains all county-to-county migration flows in the United States between 1995 and 2000 [27]. Figure 9 shows strong migration flows at the East Coast and in the South West of the United States. Moreover there are strong connections near big cities like Chicago, Denver, Atlanta and Housten. While these patterns get also highlighted in most edge bundling visualizations there remain interesting differences. The presented approach aggregates edges solely based on their start and end points and near parallel edges are therefore often not aggregated. For instance the East to West pattern which is strongly highlighted in bundling approaches like FDEB [15] and Winding Roads [19] is less obvious in our visualization. However the clear mapping between aggregates and their start and end clusters makes it easier to verify actual connections like New York $\Leftrightarrow$ Florida. The second migration data set (Figure 10) has been rendered with edge weights equal to the number of total movements between the different counties. Interestingly the weights emphasize short cluster connections like those near Los Angeles, Dallas and Boston. One explanation for these patterns could be a tendency towards local movements. For instance nine of the top ten connections are between neighboring counties.

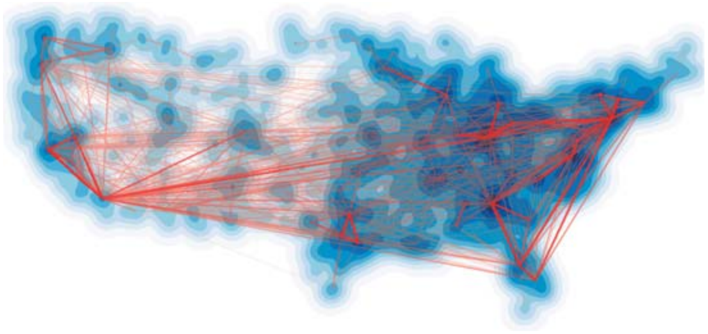
Fig. 9. US migration data set of recent publications visualized in LaGO.

Net 50 data

To enable a visual comparison with MINGLE; we applied our visualization to the Net50 data set (Figure 11). Our method reveals that the half-ring on the right side (I) connects homogeneously to cluster II. No significant connection from the upper part of the half-ring (I) connects to the lower part of the cluster and apart from one outlier all connections end on the right side of cluster II. The cluster itself has two highly interconnected peaks that seem to be the end points for most of the connections from the rectangular cluster (III). Within cluster III the connections reveal a cross pattern, the edges entering from the right side are connected to the left half of the field and vice versa. Additionally, the small outer clusters on the right side of the half-ring (I) are not equally distributed and the main connections go to the half-ring (I) and not to other parts of the graph. The described patterns are harder to read from the MINGLE visualization [12] because the x-shaped edge aggregates hinder tracking of connections and pure bundling cannot sufficiently reduce the complexity of dense edge clusters. For instance the homogenous characteristics of the half ring cannot be spotted and cluster II appears as crowded heat balls in the visualization.

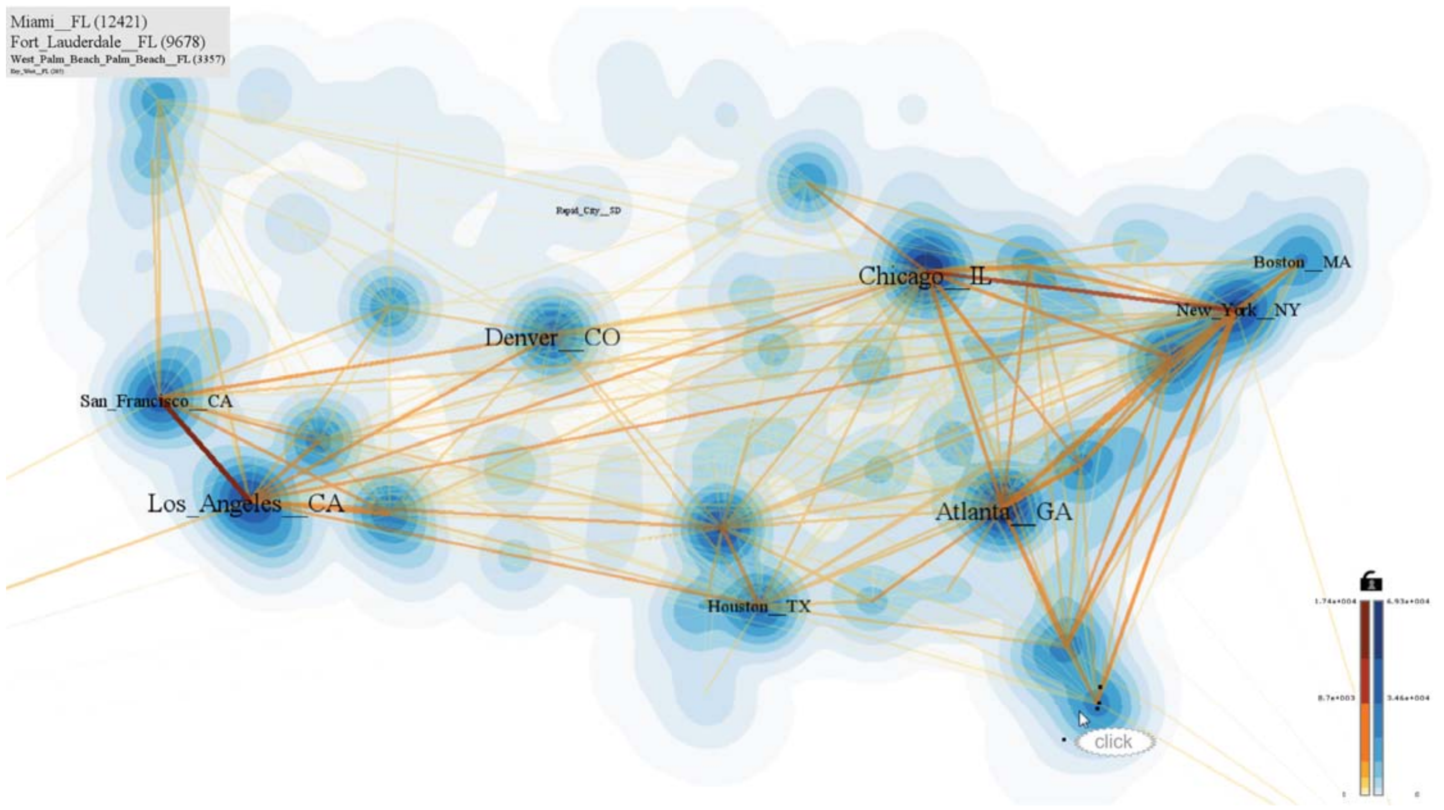

Fig. 7. US air traffic data set. The node aggregation highlights important flight hubs, while edge aggregation shows e.g. a dense connection between Los Angeles and San Francisco. A click in the Miami area (low right) highlights important nodes and a label list on the top left. From the list the user can choose interesting labels, that are placed within the visualization. The color mapping scale is shown on the bottom right.

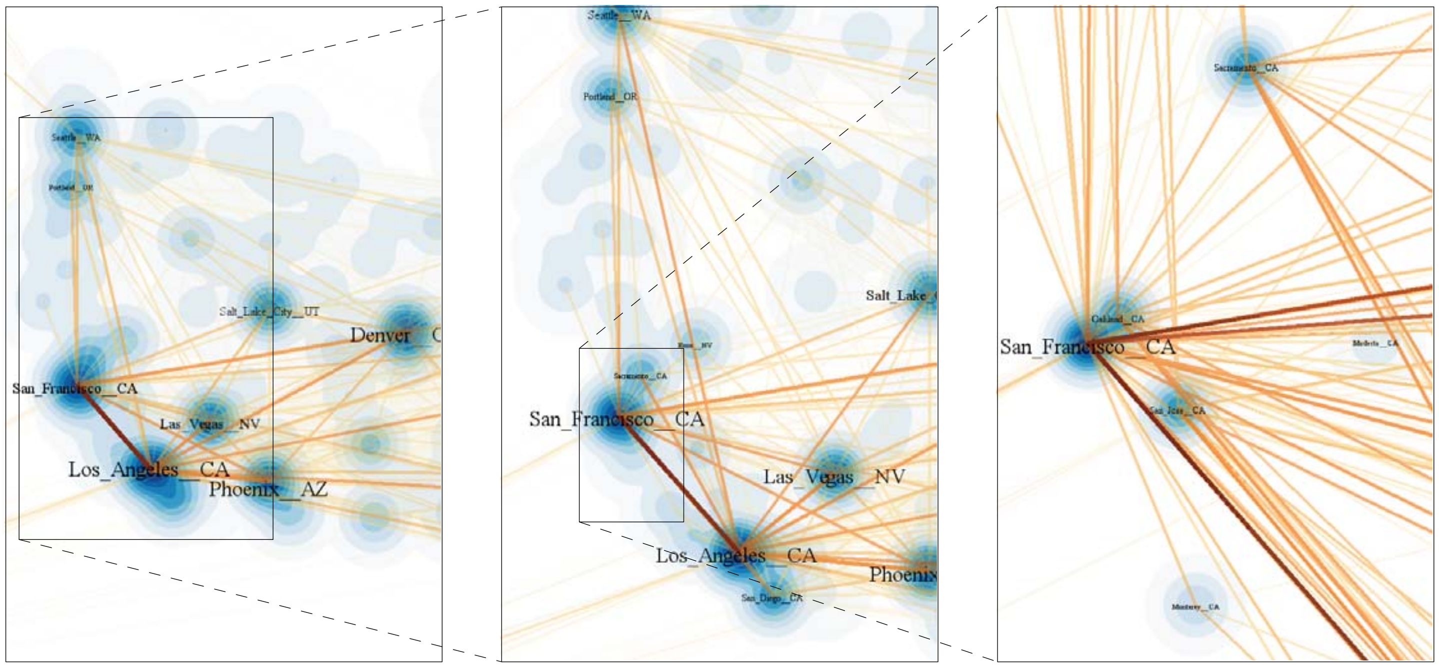

Fig. 8. Semantic zoom into the San Francisco Bay Area. The leftmost picture displays a high aggregation level where all airports in the region around San Francisco and LA get merged. Zooming in reveals smaller nearby airports like Sancramento and San Diego however the main connection remains between the two big cities. The rightmost picture shows finally the three Bay Area airports and their individual connections.

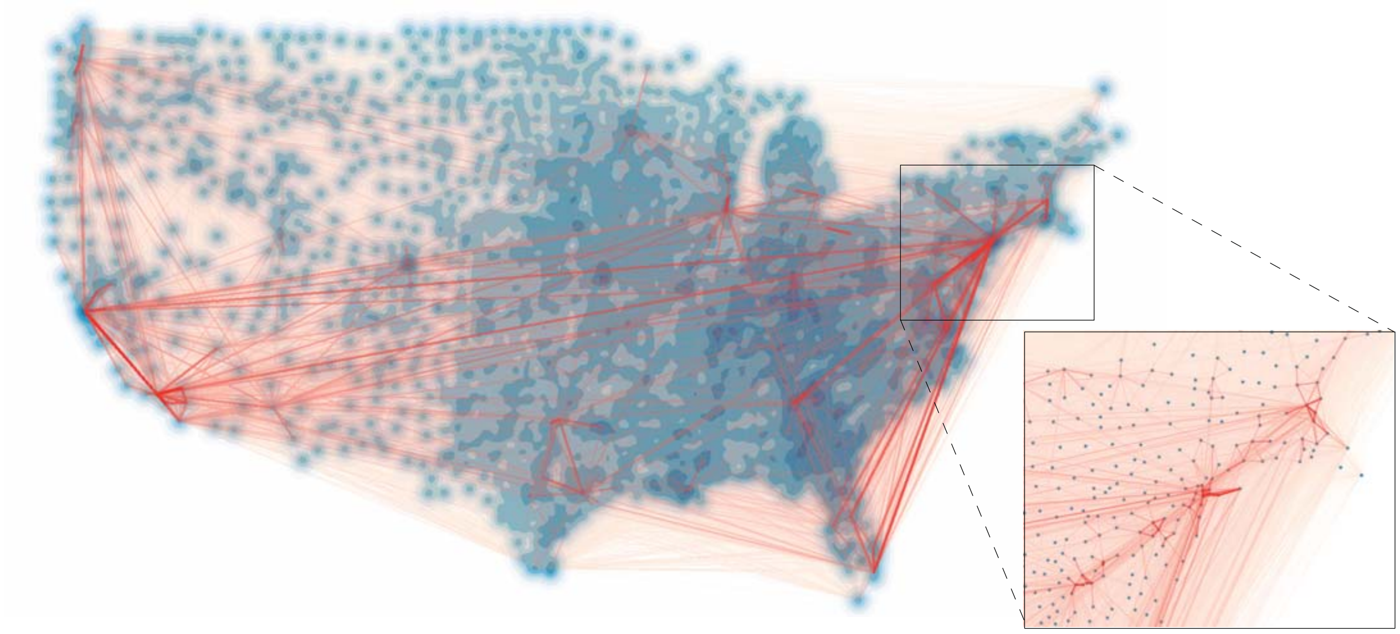

Fig. 10. US census data set with 545,882 weighted edges depicting the county-to-county migration flow between 1995 and 2000. A zoomed perspective (right lower corner) shows connections between single counties while the overview aggregates nearby clusters and allows the observation of higher level patterns like the connection between New York and Florida.

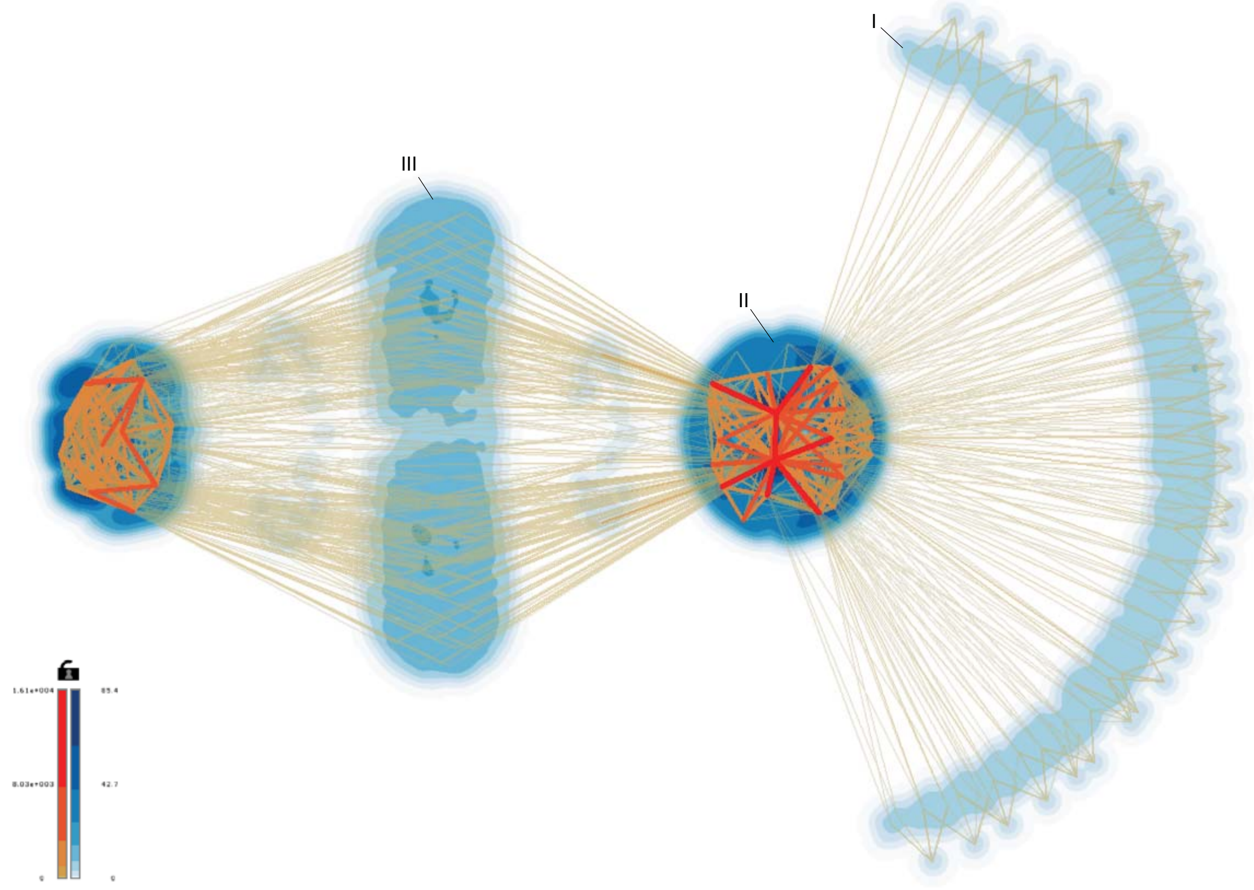

Fig. 11. The Net 50 data set with the color scheme provided bottom left. The visualization reveals several interesting properties of the dataset. For instance the homogeneously structure of the connection from the half-ring (I) to cluster II, the edge cross patterns in cluster III or the two highly interconnected central peaks in cluster II.

Random data

Figure 12 shows a random graph with 200,000 nodes and 100,000 edges which is similar to an example given in Hurter et.al. [17]. Two snapshots are given for smaller (left) and larger (right) KDE bandwidth. It can be seen, that in both aggregation levels no obvious pattern occur. Since the graph is random and reflects a quasi-equal distribution, Fig. 12 can be seen as a valid representation which avoids false positive graph patterns. The graph is also an extreme case for our method as its nodes are nearly equally distributed over the canvas. This reduces rendering speed, because nearly every node occupies an own pixel and therefore becomes a seed point for a KDE texture (see Section 4 and Table 1).

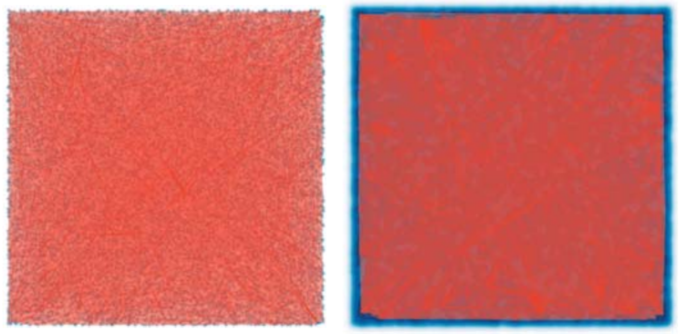

Fig. 12. A random graph with 200,000 nodes and 100,000 edges (similar to the example in [17]). While bandwidth h is increased no patterns occur; an effect wanted to avoid false positive patterns.

OpenStreetMap data

Figure 13 visualizes the Europe point dataset that has been extracted from OpenStreetMap. The data contains one node for each tagged building in Europe, a total of 37.6 million distinct nodes. The discussed improvements of the node rendering approach allow the interactive exploration of this dataset even on a low end consumer graphics card (ATI HD5770).

6 Conclusion & Future Work

We have presented an new approach that integrates node density aggregation and meta edge generation for the visualization of large graphs at different levels of detail. The methods are optimized for fast processing in OpenGL to allow interactive rates when using common graphics hardware. We showed that by using a two-pass rendering method, node aggregation can be accelerated without significant memory cost. The edge aggregation on top of the node density field and the proposed methods for post-processing allow fast rendering of implicit edge aggregates with the look of explicit meta edges.

The methods are implemented in a software tool that is used for applying our approach to large datasets . The tool allows panning and zooming at interactive rates and is highly configurable w.r.t color mapping and normalization scaling functions.

For future work, we plan to investigate if for a given zoom level a best kernel size can be estimated dynamically. Therefore we want to conduct a user study to evaluate what will be a best visualization for different zoom levels. We furthermore plan to release a free version of the described software tool.

Acknowledgments

We thank Arlind Nocaj, Mirza Klimenta, and Christophe Hurter for support with data sets. This work has been funded in part by Deutsche Forschungsgemeinschaft (DFG) under grant GK-1042, Explorative Analysis and Visualization of Large Information Spaces, Konstanz.

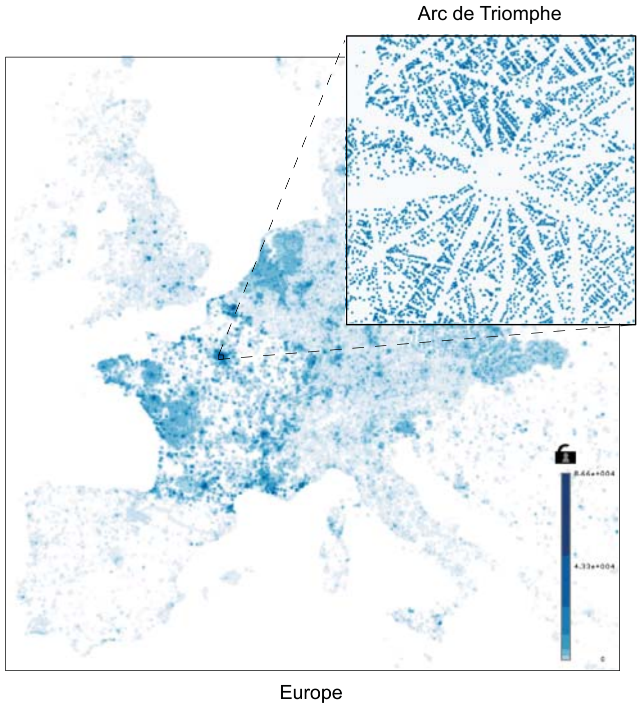

Fig. 13. Interactive zoom into the Europe dataset with 37.6 million nodes that represent all tagged buildings from OpenStreetMap. The highest aggregation level shows interesting data quantity differences at state borders. Zooming into Arc de Triomphe area reveals scenic and architectural patterns (e.g. street structure).

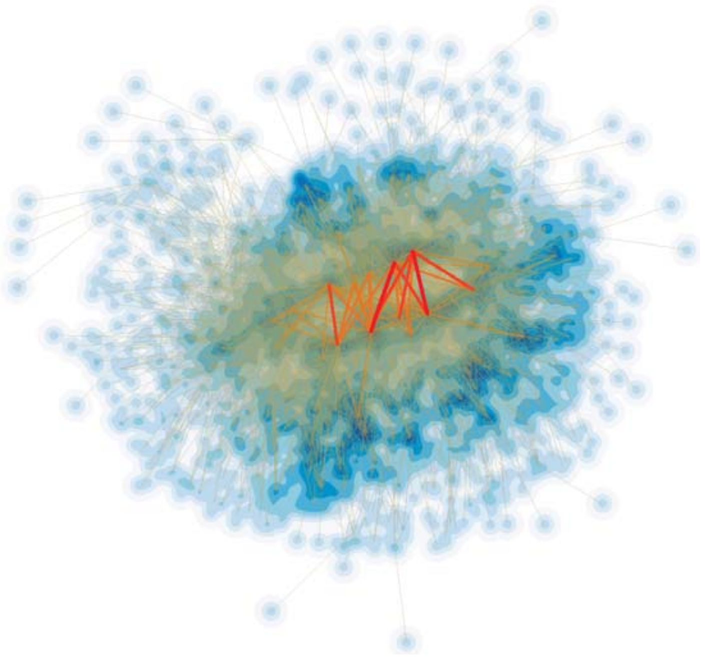

Fig. 14. Wiki-Vote data set laid out with an MDS algorithm. Note that our rendering reveals a prominent bipartite core rather than producing the usual clutter in the center of a small-world graph.

References

[1] Openstreetmap. `http://openstreetmap.de/`, Jul 2011.

[2] Sparse matrix collection. `http://www.cise.ufl.edu/research/sparse/matrices/`, Jan 2012.

[3] Us airtraffic dataset. `http://www.transtats.bts.gov/`, Jan 2012.

[4] J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):669–676, 2006.

[5] R. A. Becker, S. G. Eick, and A. R. Wilks. Visualizing network data. *IEEE Transactions on Visualization and Computer Graphics*, 1:16–28, 1995.

[6] T. Buering, J. Gerken, and H. Reiterer. User interaction with scatterplots on small screens - a comparative evaluation of geometric-semantic zoom and fisheye distortion. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):829–836, Sept. 2006.

[7] N. Cao, J. Sun, Y. Lin, D. Gotz, S. Liu, and H. Qu. Facetatlas: Multifaceted visualization for rich text corpora. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1172–1181, 2010.

[8] D. B. Carr, R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):pp. 424–436, 1987.

[9] W. Cui, H. Zhou, H. Qu, P. C. Wong, and X. Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277–1284, Nov. 2008.

[10] M. Dickerson, D. Eppstein, M. Goodrich, and J. Meng. Confluent drawings: Visualizing non-planar diagrams in a planar way. In G. Liotta, editor, *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin / Heidelberg, 2004. 10.1007/978-3-540-24595-7_1.

[11] O. Ersoy, C. Hurter, F. Paulovich, G. Cantareiro, and A. Telea. Skeleton-based edge bundling for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2364–2373, Dec. 2011.

[12] E. R. Gansner, Y. Hu, S. North, and C. Scheidegger. Multilevel agglomerative edge bundling for visualizing large graphs. In *Proceedings of the 2011 IEEE Pacific Visualization Symposium*, PACIFICVIS '11, pages 187–194, Washington, DC, USA, 2011. IEEE Computer Society.

[13] E. R. Gansner, Y. Koren, and S. C. North. Topological fisheye views for visualizing large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 11(4):457–468, July 2005.

[14] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12:2006, 2006.

[15] D. Holten and J. J. Van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[16] Y. F. Hu. Efficient and high quality force-directed graph drawing. *The Mathematica Journal*, 10:37–71, 2005.

[17] C. Hurter, O. Ersoy, and A. Telea. Graph bundling by kernel density estimation. *Computer Graphics Forum (EuroVis 2012)*, 31(3):865 – 874, 2012.

[18] D. A. Keim, M. C. Hao, U. Dayal, H. Janetzko, and P. Bak. Generalized scatter plots. *Information Visualization*, 9:301–311, December 2010.

[19] A. Lambert, R. Bourqui, and D. Auber. Winding roads: Routing edges into bundles. *Computer Graphics Forum*, 29(3):853–862, 2010.

[20] O. D. Lampe and H. Hauser. Interactive visualization of streaming data with kernel density estimation. In *Proc. IEEE Pacific Visualization Symp. (PacificVis)*, pages 171–178, 2011.

[21] D. Phan, L. Xiao, R. Yeh, P. Hanrahan, and T. Winograd. Flow Map Layout. *Proceedings of the 2005 IEEE Symposium on Information Visualization*, 0:29+, 2005.

[22] A. Quigley and P. Eades. FADE: Graph Drawing, Clustering, and Visual Abstraction. In *GD '00: Proceedings of the 8th International Symposium on Graph Drawing*, pages 197–210, London, UK, 2001. Springer-Verlag.

[23] H. Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16:187–260, June 1984.

[24] M. Sarkar and M. H. Brown. Graphical fisheye views of graphs. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '92, pages 83–91, New York, NY, USA, 1992. ACM.

[25] B. Silverman. *Density estimation for statistics and data analysis*. Monographs on statistics and applied probability. Chapman and Hall, 1986.

[26] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *Computer Graphics Forum*, 29(3):843–852, 2010.

[27] U.S. Census Bureau. County-to-county migration flow files. `http://www.census.gov/population/www/cen2000/ctytoctyflow/`, June 2012.

[28] R. van Liere and W. de Leeuw. Graphsplatting: Visualizing graphs as continuous fields. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):206–212, Apr. 2003.