Topological Fisheye Views for Visualizing Large Graphs

Emden Gansner*
AT&T Labs — Research

Yehuda Koren*
AT&T Labs — Research

Stephen North*
AT&T Labs — Research

ABSTRACT

Graph drawing is a basic visualization tool. For graphs of up to hundreds of nodes and edges, there are many effective techniques available. At greater scale, data density and occlusion problems often negate its effectiveness. Conventional pan-and-zoom, and multiscale and geometric fisheye views are not fully satisfactory solutions to this problem. As an alternative, we describe a topological zooming method. It is based on the precomputation of a hierarchy of coarsened graphs, which are combined on-the-fly into renderings with the level of detail dependent on the distance from one or more foci. We also discuss a related distortion method that allows our technique to achieve constant information density displays.

CR Categories: I.3.5 [Computer Methodologies]: Computer Graphics—Computational Geometry and Object Modeling H.5.2 [Information Systems]: Information Interfaces and Presentation—User Interfaces

Keywords: topological fisheye, large graph visualization

1 INTRODUCTION

Drawing graphs is an important information visualization technique. Common layout styles include force directed, *k*-layered ("hierarchical"), orthogonal and circular [10, 5]. In this paper, we focus on force directed layouts.

For graphs of modest size, with dozens of nodes, there are various layout methods available. In addition, these layouts can be viewed with ordinary document viewers or graph viewers having pan and zoom controls. As graphs become larger, however, problems arise. The first problem is the algorithmic complexity of computing the layout. In recent years, efficient algorithms have been developed. Useful approaches include virtual physical models [5] and techniques from linear algebra and statistics [8, 13].

Given the graph layout, visual complexity remains a problem. Practical data sets can have thousands or even millions of objects. Even at the low end of that scale, it is not realistic to expect layouts with text labels of all the objects to be readable, and navigation is difficult for humans unless the graph has some special structure, such as a tree. At the high end, the number of objects can even exceed the number of screen pixels. Clearly, additional techniques are needed.

Here we consider the problem of the display and interactive exploration of large graphs. We seek a way of reducing the number of displayed objects while preserving structural information that is essential to understanding the graph. In particular, we present a new technique for browsing large graphs whose characteristics include: (1) the efficient use of available display area; (2) informative detail-in-context displays; and (3) a variable degree of abstraction while preserving the graph's structure.

2 RELATED WORK

Distortion viewing is an obvious candidate for dealing with the problems of browsing large graphs. Variants of 2-D geometric fisheye distortion have been described by Lamping and Rao [14], Sarkar and Brown [20], and Carpendale [2]. These do a good job of

highlighting a focus area, but make the global structure even more obscure since it is packed into a much smaller area by the nonlinear transformation.

We illustrate this method using the 4elt graph [22]. This graph, of 15,606 nodes and 45,878 edges, is a mesh created to study fluid flow around a 4 element airfoil. It exhibits extreme variations in nodal density [23]. A drawing of this graph computed by the Kamada-Kawai method [12] is shown in Fig. 1. In Fig. 2, we zoom in on its right hand side using a fisheye lens [2]. This portion is properly enlarged, at the expense of the rest of the graph that becomes very crowded and obscure.
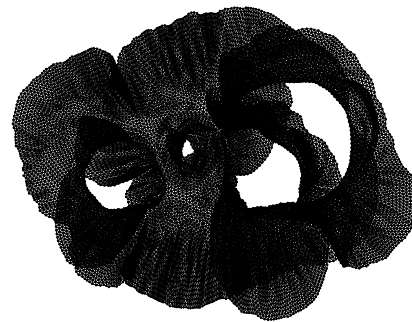

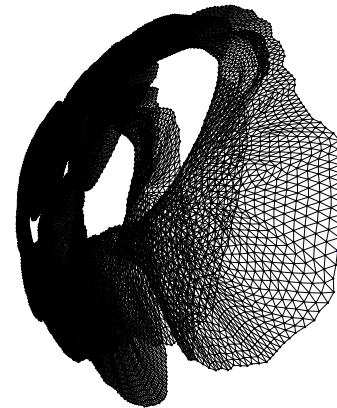Figure 1: The 4elt graph, |V|=15,606, |E|=45,878.

Figure 2: A fisheye view of the 4elt graph focused on its right hand side portion.

Another approach, proposed by Munzner, employs 3-D hyperbolic projection in an interactive viewer with high guaranteed frame-rate rendering [17]. This yields highly scalable viewing of huge trees with a 3-D fisheye effect. Display complexity is limited by tree depth and rendered node size. The technique is not intended for general graphs, and its limitations in obscuring areas of the graph that are distant from the focus are similar to the 2-D fisheye approach.

Multiscale graph abstraction has also been considered; see, e.g., [1, 9, 18]. Such layouts show the global structure well, but navigation is less flexible due to the explicit expansion and contraction of clusters and, for some methods, the clusters themselves must be externally provided. Also, the fine structure across distinct clusters may not be easy to see. In Fig. 3 we show three different levels of

*e-mail:{erg,yehuda,north}@research.att.com

175

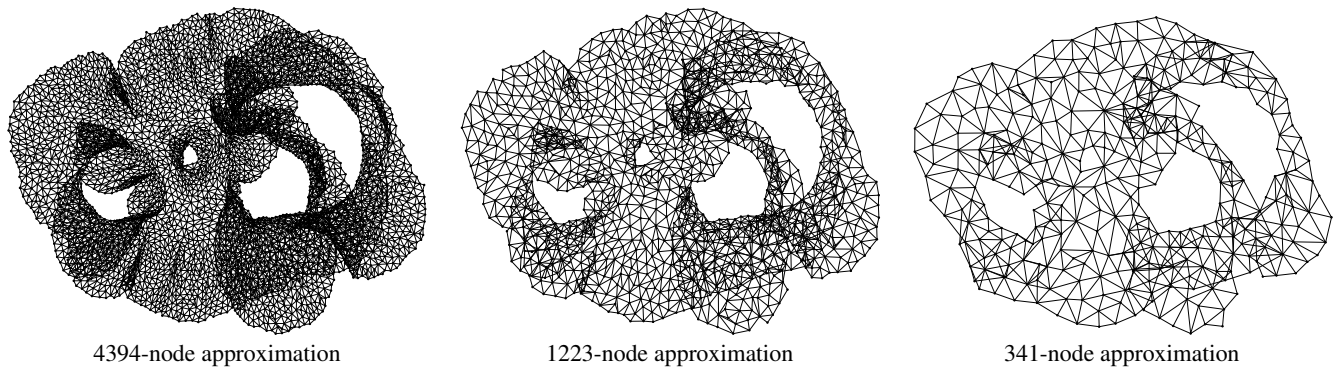| 4394-node approximation | 1223-node approximation | 341-node approximation |

Figure 3: Approximating the 4elt graph at three different scales of decreasing size and accuracy.

abstraction of the 4elt graph, clarifying its structure at the expense of hiding some of the local details.

3 TOPOLOGICAL FISHEYE VIEWS

Our technique combines static multiscale display, which excels in conveying the global structure of graphs, with fisheye display for the exploration of small regions.

We follow the general concept underlying all fisheye views of dense, overcrowded data sets. The display shows a detailed view of a region around a focus that the user chooses, while providing fewer details as the distance from the focus increases. However, in contrast with fisheye viewing that relies on a pure geometric transformation, we perform topological, or combinatorial, operations on the abstract graph. This is a specialized type of semantic zooming [6].

Specifically, we construct a hierarchy, containing graphs of decreasing sizes that approximate the original graph at various levels of accuracy. For example, consider Fig. 3 again, where a part of such a hierarchy is presented for the 4elt graph. Given a focal node, we merge all the graphs in the hierarchy into a single superposition called *the hybrid graph*, where the region of interest is taken from the original graph, and other regions belong to coarser graphs. The exact graph from which each node is taken depends on its distance to the focal node.

An example for the 4elt graph is seen in Fig. 4(a). As in the fisheye example of Fig. 2, we zoomed in on the right hand-side portion of the graph. The nodes and the edges are colored in a red-to-green scale depending on their level in the hierarchy: the red area around the focal node is directly from the original graph, while the green section is from the coarsest graph. We can see that the complexity of the graph is greatly reduced, making closer examination of the focus region possible. Moreover, the picture does not have overly dense areas (as frequently happens with geometric distortion; compare Fig. 2) and the global structure of the layout is preserved. In one click, the user can move the focus, to expand other areas of the graph and obtain new displays; see Fig. 4(b,c).

Fig. 5 shows two topological fisheye views of a published Internet map [3]. It is a large tree ($|V|$=87,931, $|E|$=87,930) made by tracing connections from a central probe to all reachable IP routers. The full layout of this graph (Fig. 5(a)) is too dense to read. In contrast, our approach yields useful views by focusing on different portions of the graph; Fig. 5(b,c).

In the following sections, we explain how to construct such views. Our major design goals were to achieve efficient display of the graph showing its overall structure, to facilitate quick interactive navigation, and to keep the various levels of display consistent so a mental map of the full graph is preserved.

We assume that we are given a graph whose nodes already have assigned coordinates, generated by an external graph drawing algorithm or some other means. The initial layout is a faithful drawing of the full graph. In Section 4, we explain how to create a multiscale representation of the graph, which is a hierarchy of graphs approx-

imating the original graph at different resolutions. This construction has a central role in the precision and reliability of our views. The multiscale representation is constructed once as a preprocessing step. The user then can browse the graph, picking different areas for more detail inspection. Changing the focus triggers redrawing, which is done by computing a new hybrid graph, merging different graphs from the multiscale hierarchy, as explained in Section 5. The layout of the hybrid graph is derived from the original layout by distortion to account for its multiple scales, as explained in Section 6. The method is easily extended to handle several simultaneous focal nodes; we discuss the key adaptations in Section 7. We conclude by discussing possible directions for further work in Section 8.

4 MULTI-SCALE REPRESENTATION OF THE GRAPH

The key step in our method is the construction of a multiscale representation of the graph. Given a graph with coordinates, we produce a hierarchy containing *coarse graphs* of decreasing sizes, approximating the original graph at multiple levels of precision. For simplicity, we concentrate on the basic step in which a single coarse graph is created from an input *fine graph*. To create the full hierarchy, we apply this step recursively, taking the previously computed coarse graph as the next input. Thus we construct a hierarchy of coarser and coarser graphs until the graph size drops below some threshold.

There are several ways to perform coarsening. We follow the common approach where nodes of a coarse graph induce a partitioning of the fine graph, meaning that each coarse node represents a cluster of fine nodes. Thus, a wide range of clustering algorithms can be applied to construct the hierarchy. However, when choosing an algorithm, we need it to satisfy certain special requirements:

1. Coarse graphs should preserve the topological properties of the fine graph. For example, we should avoid connecting nodes that are distant in the graph-theoretic sense, and never create cycles that do not exist in the fine graph.

2. The partitioning of fine nodes induced by a coarse graph should yield clusters of fairly uniform sizes. In other words, all nodes of a coarse graph should have about the same "size." Otherwise, the coarse graph might be a misleading representation of the original graph.

3. Layout of the coarse graph should preserve the geometry of the fine graph. Thus, the layouts of all graphs in the hierarchy are tightly related, allowing smooth transitions between them.

4. Since we are dealing with large graphs, it is important to use efficient algorithms, ideally with linear running time.

Note that the first two demands disqualify geometric approaches based on partitioning the space into cells.

Of course these requirements may conflict, so they cannot be optimized simultaneously. Our way of trying to satisfy them is to select a maximal set of disjoint node pairs and *contracting* (or, *matching*) these pairs. Variants of this approach are in common use
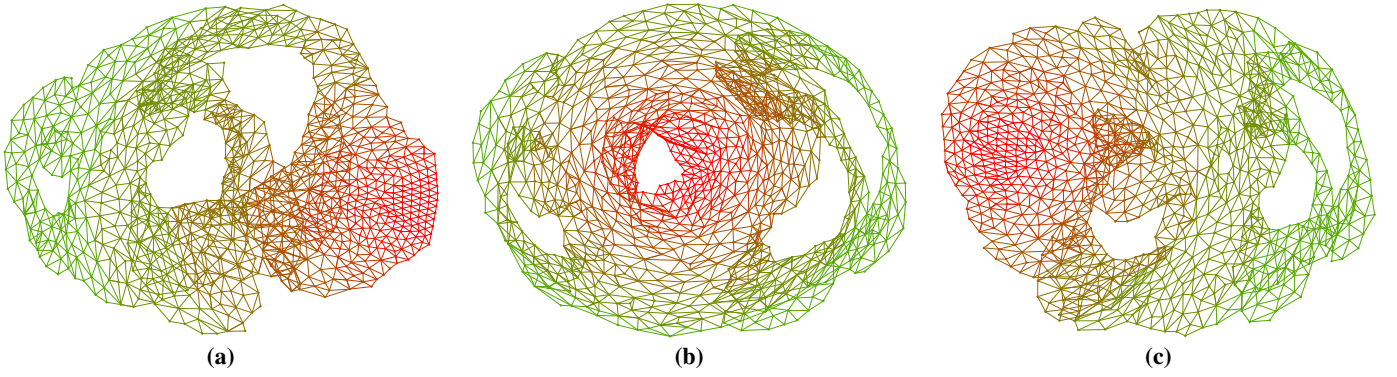
Figure 4: Topological fisheye views of the 4elt graph. Views are based on "hybrid graphs" formed by superposition of several approximations of the graph. Levels are colored red-to-green, so the focus area from the finest graph is in red. The figure shows three examples, focusing on the right hand side **(a)**, the small central hole **(b)**, and the left hand side **(c)**.

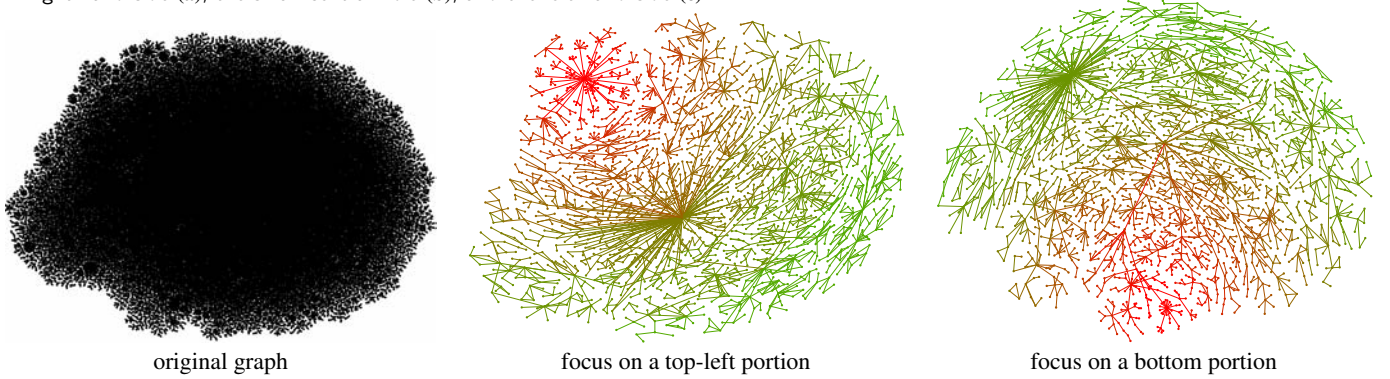original graph   focus on a top-left portion   focus on a bottom portion

Figure 5: This Internet map (($|V|$=87,931, $|E|$=87,930)) is too large to visualize as a flat structure. Two topological fisheye views are shown. The focused sections in red are the original graph. Peripheral areas, in green, are simplified.

[7, 11, 23]. Contraction of two nodes is done in the usual graph-theoretic sense: the nodes are identified and the merged node is incident to the union of all edges of the two nodes. Any edge between the two nodes is deleted.

Before we elaborate on the method, we introduce some definitions and notation. Henceforth, we assume that we are given a weighted undirected graph $G(V = \{1,\ldots,n\}, E, w)$, with no multiedges or loops, and where $w(i,j) \in \mathbb{R}^+$ is the weight of edge $\langle i,j \rangle \in E$. As we shall see, the weight of an edge reflects the desired similarity of the two endpoints. Each node i is associated with point p_i (in the layout). The Euclidean distance between p_i and p_j is denoted by $\|p_i - p_j\|$. The neighborhood of node i is $N_i \overset{\text{def}}{=} \{j \in V \mid \langle i,j \rangle \in E\}$. Similarly, $N_i^* \overset{\text{def}}{=} N_i \cup \{i\}$. The degree of i, $\deg_i$, is $|N_i|$. Each node i of G represents a cluster of nodes of the original graph. We denote by size_i the cardinality of this cluster. Consequently, when G is the original graph itself, $\text{size}_i = 1$ for each $i \in V$. Also, if node k is the result of contracting i and j, we have $\text{size}_k = \text{size}_i + \text{size}_j$. In addition, if i and j are both connected to a node l, we have $w(k,l) = w(i,l) + w(j,l)$ in the coarse graph.

Coarsening by contraction is performed in two steps. First, we construct a *candidate set* S of node pairs. Usually, $|S| = O(|V| + |E|)$, and S contains only those pairs that best qualify for contraction according to the special requirements we have stated above. The set S serves both for filtering out unsuitable node pairs and for accelerating computation by reducing the quadratic number of possible pairs to a number linear in the graph size. (Usually $|E| = O(|V|)$, i.e., the graph is sparse. Large dense graphs are inherently intractable.) The second step is a selection of a maximal subset of disjoint pairs from S that will actually be contracted. By "disjoint pairs," we mean that we do not allow the same node to appear in two selected pairs. Here again, we try to choose the best pairs from S according to the requirements stated above.

This strategy directly addresses two of the four requirements stated above: computational efficiency and uniformity of cluster sizes. Efficiency is achieved since the selection process will be performed in time $O(|S|)$. Uniformity is also achieved fairly well; by contracting disjoint pairs, the clusters are of sizes 1 or 2, and by choosing a maximal subset of S, we hope that very few clusters are of size 1. To achieve the two other requirements, namely, preservation of topological and geometric properties, we need to make smart decisions about which pairs to contract, both when constructing S and when selecting the contracted pairs out of it.

Our approach to preserving the topological and geometric properties of the graph is to contract pairs that are close both in the graph-theoretic sense and in the geometric sense. Contracting nodes with large graph-theoretic distance will join two separate areas of the graph, which causes a dramatic change in the graph's topology. For example, consider Fig. 6, where the contraction of the two gray nodes causes an abrupt change in the topology, introducing a cycle in the graph's structure. In fact, if we contract only nodes whose (unweighted) graph-theoretical distance is 1 or 2, we can guarantee that no new artificial cycles will appear in the coarse graph. However, contracting two nodes of distance 3 or more can introduce a misleading cycle in the coarse graph. It is also obvious that closeness in the geometric sense is essential; contracting two nodes that are distant in the layout (meaning, placing them at the same point) must always substantially alter the layout, violating the geometric consistency requirement.

A good strategy might have been choosing the contracted pairs only from the set of edges, so only adjacent nodes can be contracted and $S = E$. In fact, most contraction-based methods use this restriction [7, 11, 23]. However, we have found that the edge set does not provide us enough freedom and hence is not suitable. For example, consider a star-like structure in the graph. Since the contracted
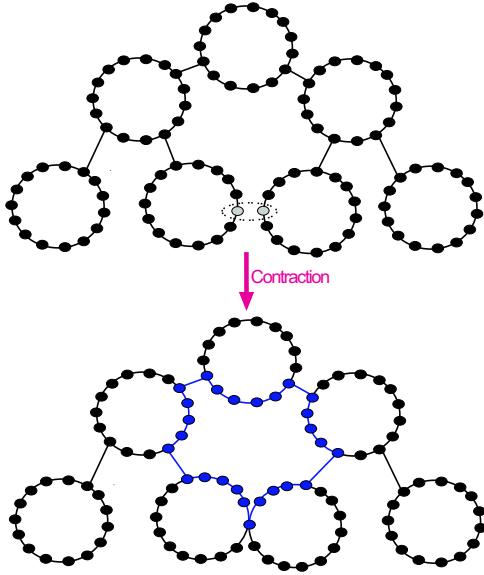
Figure 6: The effect of contracting nodes with large graph-theoretic distance: contracting the two gray nodes in the top graph yields a fundamentally different graph, with a large, new cycle (colored blue).

pairs must be disjoint, only a single pair out of this star can be contracted. Consequently, the coarsening rate will be extremely slow. Also, large deviations in node size will inevitably appear. To account for such a situation, we must add more candidate pairs to S. One possibility is to add all pairs whose graph-theoretic distance is less than 2 or 3, but then we cannot ensure that $|S| = O(|V| + |E|)$. However, thanks to the fact that we have geometric coordinates, we can use a more efficient solution: add all nodes that are neighbors in a *proximity graph*.

A proximity graph is a graph derived from the geometry of a set of points. There are several variants of proximity graphs [4]. They all attempt to capture the concept of adjacency relations between the points. In our case, we work with two such graphs:

- The *Delaunay triangulation (DT)*. Two points are neighbors in the DT if and only if there exists a sphere such that its boundary contains these points, and no other point exists in its interior. An alternative definition of DT is as the dual of the Voronoi diagram [4], so two points are adjacent in the DT if their respective Voronoi cells are adjacent.

- The *relative neighborhood graph (RNG)*. Points p_i and p_j are neighbors in the RNG if and only if for every point p_k:

$$\|p_i - p_j\| \leqslant \max\{\|p_i - p_k\|, \|p_j - p_k\|\}$$

where $\|p_i - p_j\|$ denotes the (Euclidean) distance between p_i and p_j. In other words, p_i and p_j are neighbors in the RNG when there is no other point that is both closer to p_i than p_j and closer to p_j than p_i. Interestingly, if two points are neighbors in the RNG, then they are also neighbors in the DT.

For 2-D geometry, both the DT and RNG are planar graphs and can be computed in $O(n \log n)$ time [4, 15]. Several good implementations are available [16, 21]. Fig. 7 shows examples of both DT and RNG.

The proximity graphs are well suited to coarsening. Our experiments show that contraction of these graphs provides a kind of "space-decomposition" of the points, where cluster sizes are usually uniform, and clusters tend to correspond to compact regions in the drawing. Also, the coarsening rate is good, and typically a coarse graph is of half the size of the fine graph. Therefore, we also use the edges of the proximity graph as candidates for contraction (in addition to the original graph edges). Since neighbors in

a proximity graph might be undesirably far in the graph-theoretic sense, we remove all pairs whose graph-theoretic distance is above k, where k is usually 2 or 3.

Which proximity graph should be chosen: the larger DT or the smaller RNG? The RNG seems more appropriate as it captures a very appealing notion of closeness. However, computationally, the DT can be obtained more easily. Also, after removing all nodes that are distant in the graph-theoretic sense, the difference between the two graphs is not that significant. Therefore, we cannot provide a definitive recommendation as to which of these two graphs should be favored. As a compromise, we construct a graph that is contained in the DT and contains the RNG, using the following fast method: we construct the DT, and then we remove any DT-edge $\langle i, j \rangle$ if there is some k adjacent to i or j (in the DT) such that $\|p_i - p_j\| > \min\{\|p_i - p_k\|, \|p_j - p_k\|\}$.

To summarize, we choose the node pairs to be contracted from a candidate set S comprised of two kinds of pairs:

1. The edges of the graph

2. The edges of a proximity graph (excluding edges whose endpoints are far in the graph-theoretic sense)

Notice that since the proximity graph is planar, its edge set contains at most $3n - 6$ edges. Therefore, the cardinality of the candidate set is still linear in the graph size.

After constructing the set of candidate pairs, we want to contract a maximal number of disjoint pairs from it. While we could apply optimal algorithms for maximal matching of candidate pairs, we prefer a linear time heuristic because computational performance is critical.

We use the following procedure; related techniques are described elsewhere [7, 11]. Iterating over all nodes, for each unmatched node i, we look at the set of candidate pairs containing it, and match it with an unmatched node j that maximizes a weighted sum of the following measures:

1. Geometric proximity: $\frac{1}{\|p_i - p_j\|}$

2. Cluster size: $\text{size}_i + \text{size}_j$

3. Normalized connection strength: $\frac{w(i,j)}{\sqrt{\text{size}_i \cdot \text{size}_j}}$, where $w(i,j) = 0$ if $\langle i, j \rangle \notin E$.

4. Similarity of neighborhood: $\frac{|N_i^* \cap N_j^*|}{|N_i^* \cup N_j^*|}$

5. Degree: $\frac{1}{\deg_i \cdot \deg_j}$

Since the different measurements are in different scales, we normalize them all to the range $[0, 1]$ as part of the weighting scheme.

The first measure is aimed at preserving the graph's geometry. The second encourages uniform node sizes. The third and fourth measures help to preserving the graph's topology by contracting pairs that are tightly related in the graph. The fourth measure also encourages sparser coarse graphs, which are better both aesthetically and computationally. The last two measures inhibit the formation of high degree nodes, which are formed by contracting two nodes of high degree into a single node of even higher degree. High degree nodes are undesirable for several reasons. First, they form a salient topological feature which we do not want to create artificially. Second, they obscure readability. And, third, they make further recursive coarsening more difficult.

Since we deal with graphs having a layout, we need to assign coordinates to the nodes of the coarse graph as well. We assign a coarse node the average coordinates of the nodes in its associated cluster. Specifically, when contracting nodes i and j, with corresponding points p_i and p_j, respectively, the point associated with the new node is:

$$\frac{\text{size}_i \cdot p_i + \text{size}_j \cdot p_j}{\text{size}_i + \text{size}_j}$$
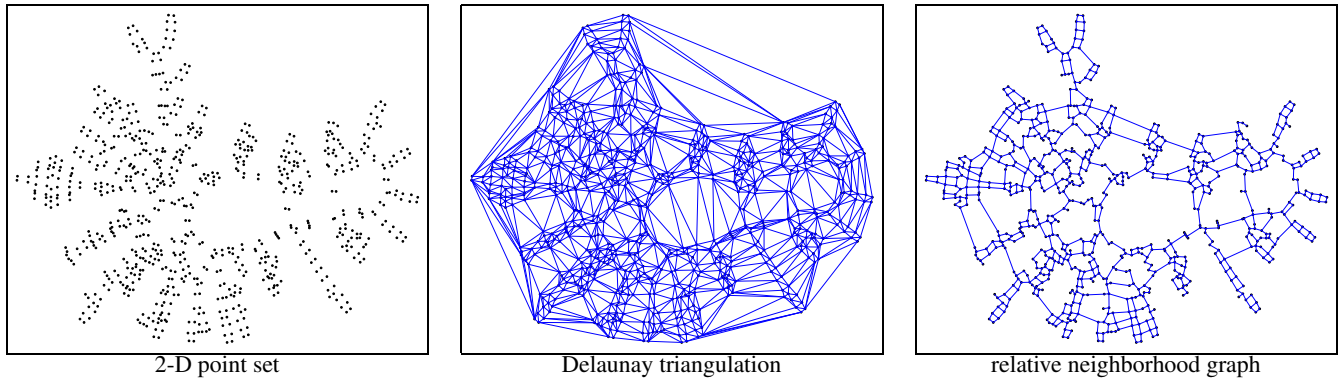
2-D point set   Delaunay triangulation   relative neighborhood graph

Figure 7: The Delaunay triangulation and relative neighborhood graph of a point set.

Coarsening a single graph takes linear time $O(n+|E|)$. As previously explained, to build a full hierarchy, we execute this process recursively until the graph size drops below some threshold (say, 20). If each coarsening cuts the graph size about in half, we get $O(\log n)$ levels in the hierarchy and the total coarsening time is linear. We should add to this the $O(n \log n)$ time needed to construct the proximity graph. In our implementation, this graph is constructed only once at the original, finest level, and for other levels we estimate it in linear time by coarsening the current proximity graph. If the coarsening does not significantly reduce the number of nodes, it is an indication of a degenerate process (as happens with a star-like graph when contracting only neighbors), and we terminate it regardless after a predetermined number of levels (50, in our implementation). This degenerate behavior can be an indication that we were given a low quality layout as input.

In practice, the coarsening phase is clearly the computationally most expensive part of our method. Typical running times are about 1-4 seconds for graphs with around million nodes on a Pentium-4 PC. However, it is performed only once in the preprocessing stage that precedes the user interaction.

5   THE HYBRID GRAPH

After the user selects a focal node, we create the hybrid graph, merging all the graphs in the hierarchy so the focus area is presented in detail, while more distant portions have a less detailed representation.

We represent the hierarchy of the coarse graph as a binary tree; see Fig. 8. Each tree node corresponds to a (coarse) node representing a set of graph's nodes. Each leaf corresponds to a unique graph node, hence level 0 of the tree represents the original graph. Higher levels of the tree represent coarser graphs, up to the root of the tree that represents the full node set (or a single node coarse graph). This way, each graph node is contained in all the tree nodes on the path from its corresponding leaf to the root of the tree.

Horizontal slices   While each tree level represents a single coarse graph, we use more elaborated horizontal slices to merge several coarse graphs. For example, consider Fig. 9, where we show such a slice in the hierarchy tree. This slice implies that nodes E and F are represented by the level-1 node $\{E,F\}$. Nodes A and D are represented by their corresponding level-0 leaves. And, nodes C,B,G and H, are represented by their shared level-2 node $\{C,B,G,H\}$. We use the term *active nodes* to describe the tree nodes that are immediately above the horizontal slice; in our case: $\{E,F\},A,D,\{C,B,G,H\}$. These active nodes constitute the node set of the hybrid graph. Regarding the edge set, we form an edge between two active nodes when there exists an edge in the original graph between two nodes, one in each cluster. Thus, the hybrid graph in our example contains 4 nodes and 4 edges and is shown in Fig. 9.
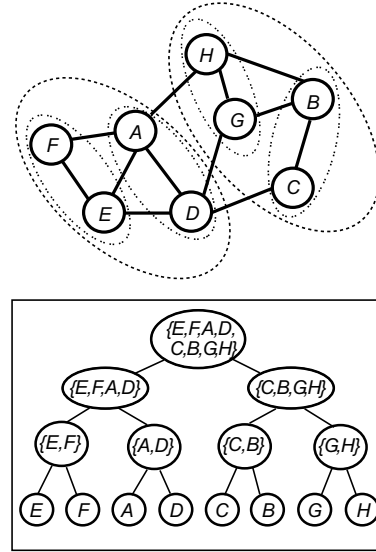

Figure 8: An 8-node clustered graph and its corresponding hierarchy tree. Each level in the tree corresponds to a single coarse graph.
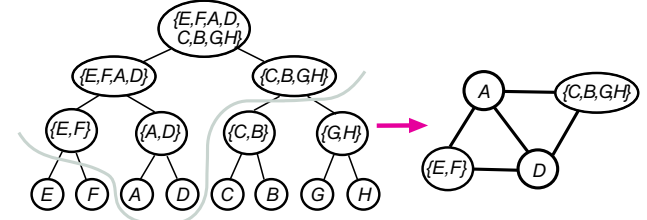

Figure 9: An horizontal slice through the hierarchy tree of Fig. 8. This induces a 4-node hybrid graph that approximates the original graph at various levels of accuracy simultaneously.

Determining active nodes   By construction, the hybrid graph is fully characterized by defining the active nodes in the tree. This is computed by a three-phase process, which we now describe.

Phase 1 - the wish list   The first step is to iterate through all nodes in the original graph and decide in which level of the tree we would like to represent it. To do so, each of the tree levels has an associated capacity. We denote the capacity of level i by c_i. The purpose of these capacities is to define how many nodes we want to represent at each level. Note that as the node is closer to the focus, we want it to be represented by a lower level tree node, to give it a finer representation. Therefore, we sort all nodes according to their distance from the focus, in ascending order. Then, we take the first

179

c_0 nodes in the ordering and associate them with level 0, the next c_1 nodes are associated with level 1, and so on. Thus, we obtain for each node the desired level in which we would like to represent it. However, as we will see, this does not yield a final representation as inconsistencies may arise. Regarding actual level sizes, based on implementation experience we recommend that c_0 be 50–100, so 50–100 nodes are individually represented in the hybrid graph and form the "focal region" of the graph. For the other levels we use the rule $c_{i+1} = C \cdot c_i$, where $2 \leqslant C \leqslant 3$.

Phase 2 - making decisions After associating each node with its desired representation level, we determine the active nodes (or where the horizontal slice will pass). This is done by a bottom-up traversal of the hierarchy tree which resolves all potential inconsistencies.

How can we get such inconsistencies? Note that the association of desired levels completely ignored the structure of the hierarchy tree. Therefore, we may encounter a situation where two contracted nodes have different desired levels. For example, consider the case when nodes A and B are contracted into a single level-1 node $\{A, B\}$. If, in the previous phase, A was associated with level 3, and B was associated with level 4, there is no possible way to achieve both associations, as the contraction binds the two nodes to the same representation level. Our way of resolving such conflicts is to associate two conflicting nodes with the lower of their two levels. This way we never associate a node with a representation level higher than its desired one.

Therefore, we determine active nodes using a bottom-up traversal of the tree. Assume that at some point of the traversal, we encounter two nodes A and B in level i whose parent is node C (of level $i+1$). The desired representation levels of A and B are l_A and l_B, respectively. Without loss of generality, assume $l_A \leqslant l_B$. The behavior of the algorithm is characterized by the following cases:

- $l_A = i+1$
  Make C an active node. Do not traverse the ancestors of C.

- $l_A > i+1$
  Set $l_C = l_A$. One of C's ancestors will be an active node.

- $l_A \leqslant i$ and $l_B \geqslant i+1$
  Make B an active node (A is already an active node). Do not traverse the ancestors of C.

In case A is the only child of C, we set $l_C = l_A$; C is an active node if $l_A = i+1$.

Phase 3 - propagating decisions Each node below the horizontal slice is represented by its ancestor active node. (We ignore nodes above the active nodes as they are irrelevant at this point.) Therefore, we perform a top-down traversal of the graph, and point from each node to its active ancestor node.

The time complexity of this entire process is dominated by the sort-by-distance operation which takes $O(n \log n)$ time. In practice, the execution cost was negligible for all graphs we experimented with.

Our application does not explicitly construct the hybrid graph, as for our needs it can be efficiently derived from the coarse graphs and the active nodes. Specifically, we need to support two operations: (1) list all nodes of the hybrid graph and (2) list all edges of the hybrid graph. When we need to list all nodes, we traverse the hierarchy tree top-down until reaching the active nodes. When we need to list all edges, we iterate over all active nodes. For each active node A of level i, we use the coarse graph associated with level i and scan A's neighbors in this coarse graph. Then, for each edge $\langle A, B \rangle$ (B is a node of the level-i coarse graph) there are three possibilities:

1. If B is an active node, add edge $\langle A, B \rangle$.

2. If B points to an ancestor, C, which is an active node, add edge $\langle A, C \rangle$.

3. If B has descendants which are active nodes, no edge should be added. The edges will be added when visiting B's descendants.

6 DISTORTING THE LAYOUT

The final step of our method is the actual display of the hybrid graph. In this section, we explain how its layout is computed. In fact, this graph already possesses a *default layout*, as described in Section 4, where each of its nodes is assigned the average location of the nodes in its cluster. However, this layout is not always satisfactory, because the hybrid graph integrates multiple scales. Consequently, different portions of the original graph get various detail levels and the layout has a highly nonuniform density – the detailed focus portion will be very dense, while the peripheral portions will be much sparser. A typical example of this behavior can be seen for the Crack graph [19] as shown in Fig. 10(a). When we focus on one of its central nodes, we get the default layout of the hybrid graph shown in Fig. 10(b). The detailed focus region (colored red) is very dense. As the distance from the focus increases, the layout becomes sparser.

While the default layout is not satisfactory, it certainly embodies several qualities that originated in the layout of the related fine graph. In fact, we found that default layouts serve very well for single-scale coarse graphs. Only with a multiscale hybrid graph did we encounter the nonuniform density problem. In what follows, we explain how a simple radial distortion can fix this density problem. The result of applying this distortion to the default layout of the Crack graph is shown in Fig. 10(c). It is clear that the result is much more uniform and readable than the original layout in Fig. 10(b).

We will work with *polar coordinates*, which allows cleaner expressions as compared with *Cartesian coordinates*. Assume that the node set of this graph is $\{1, \ldots, m\}$ and its layout is $(r_1, \theta_1), \ldots, (r_m, \theta_m)$, where the origin is the focal point. To simplify notation, we assume that the nodes are sorted according to their distance from the focus, so $r_i \leqslant r_{i+1}$.[1] We are looking for some distortion function $\mathscr{D}$, so that the distorted layout $\mathscr{D}(r_1, \theta_1), \ldots, \mathscr{D}(r_m, \theta_m)$ has a "uniform density."

We now make the following observation. The hybrid graph merges several coarse graphs. By our construction of this graph (see Section 5), it is reasonable to assume that nodes with the same distance from the focus belong to the same coarse graph. Therefore, each circle centered at the focus constitutes an equi-density region. In other words, we expect all points in the default layout lying at the same distance from the focus to have the same density. Therefore the density depends only on the radial coordinate, and is independent of the angular coordinate. This simplifies construction of the distortion function, which we restrict to act only on the radial coordinate, so $\mathscr{D}(r, \theta) = (\mathscr{F}(r), \theta)$, where $\mathscr{F} : \mathbb{R} \to \mathbb{R}$ is the radial distortion function.

At this point, we need to formulate the notion of density. There are many ways of measuring density at a point, such as counting how many points lie inside a small circle centered at the point, or finding the distance to its closest point. However, working with a small circle is scale-dependent, as the radius of the circle will influence the value. Also, measuring the distance to the single closest point is not robust. Our way of obtaining a robust and scale-independent measure is based on the RNG of the m points (see Section 4); other proximity graphs could be used as well. Note that in dense regions, the RNG edges will be shorter than in sparse regions. Thus, we define d_i as the average length of an edge adjacent to i in the RNG. The value d_i measures the average distance between i and its neighbors. The smaller d_i, the denser the neighborhood of i.

We need only consider the radial component, obtaining a 1-D problem. Since we can directly measure density only on the m given points, we discretize our space and assume uniform density inside

---

[1]Note that $r_1 = 0$ since the focal point is always associated with a node.

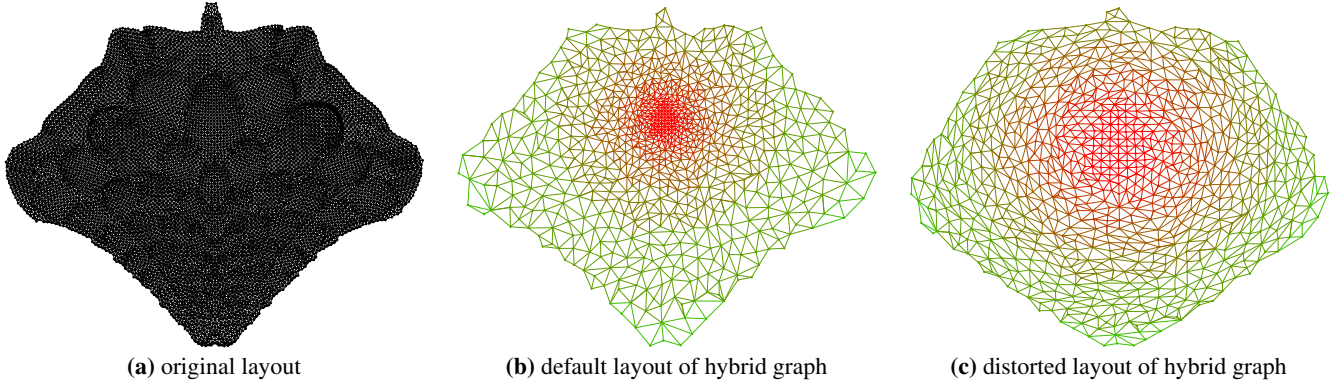**(a)** original layout   **(b)** default layout of hybrid graph   **(c)** distorted layout of hybrid graph

Figure 10: The Crack graph [19], |V|=10,240, |E|=30,380. In the hybrid graph, the focus region is colored red. Its default layout is derived directly from the original layout, and exhibits a very dense focus region. The distorted layout is a uniformly-scattered version of the default layout obtained by radial distortion.

intervals between consecutive points. These intervals are defined by:

$$\Delta_i = [r_{i-1}, r_i) \quad i = 1, \ldots, m$$

where $r_0 \overset{\text{def}}{=} 0$.

The density within interval Δ_i is interpolated from d_{i-1} and d_i or, more robustly, we average the densities $d_{i-p}, \ldots, d_{i+p-1}$ for some p. Thus, we define

$$d_{\Delta_i} = \frac{d_{i-p} + \cdots + d_{i+p-1}}{2p}. \tag{1}$$

In our implementation, $p = 20$. (Near the extremities (i.e., 0 and m) this definition should be altered, truncating the summation so that the subscripts remain in the range $[0, m]$).

Now we can define a radial distortion function which stretches segments according to their density

$$\mathscr{F}(r_i) = \sum_{j=1}^{i} \frac{|\Delta_j|}{d_{\Delta_j}}$$

where $|\Delta_j| = r_j - r_{j-1}$.

Or, equivalently

$$\mathscr{F}(r_i) = \mathscr{F}(r_{i-1}) + \frac{|\Delta_i|}{d_{\Delta_i}}.$$

Note that $\mathscr{F}$ is an increasing function, therefore our distortion function $\mathscr{D}(r, \theta) = (\mathscr{F}(r), \theta)$ relates the distorted layout to the original layout with two properties:

1. Angular coordinates do not change.

2. The order of radial coordinates do not change (so the order of distances from the origin remains the same).

Sometimes, we want to overemphasize the focal region, and to enlarge it at the expense of peripheral regions. This can be done by slightly changing the definition of the radial distortion function, introducing the *distortion factor* α:

$$\mathscr{F}(r_i) = \mathscr{F}(r_{i-1}) + \frac{|\Delta_i|}{(d_{\Delta_i})^\alpha}.$$

Using $\alpha > 1$ makes the focus region less dense than other regions, yielding an effect similar to a classic geometric fisheye view [2]. In our default setting, we seek uniform density over the whole layout, so $\alpha = 1$. (All single-focus layouts shown in this paper were made this way.) Note that $\alpha = 0$ means no distortion at all.

The time complexity of computing the distortion is $O(m \log m)$ and is dominated by the time to sort the radial component and the construction of the RNG. The time complexity is independent of n (the original number of points), and therefore the distortion takes negligible time.

7 WORKING WITH SEVERAL FOCI

Our algorithm can easily be extended to handle the case of selecting multiple foci nodes. Two changes are needed.

The first change is to the hybrid graph construction described in Section 5. Recall that this construction was based on a three-phase process. We adjust only the first phase, in which nodes are ordered and distributed to the different levels. When we have several foci, we sort the nodes according to their distance from the *set of foci* (i.e., the distance between each node and its closest focus). Also, regarding the level capacities, $c_1, \ldots, c_m$, we recommend multiplying all capacities by the number of foci in order to accurately describe more nodes.

The second change is in the distortion algorithm described in Section 6. We must now handle several disjoint dense areas, which invalidates our assumption of equi-density around circles centered at the focus. Our way of solving this is to distort the layout for each focal point separately (each time considering only a single focal point), and then average all of the layouts into a single final layout. We found this approach to be effective. In the multi-foci case, we recommend emphasizing the foci areas by slightly increasing the distortion factor α (defined in Section 6).

Fig. 11 shows views with 2 and 3 foci of three graphs which appeared previously in this work with the same red-green scale showing level of detail. In these examples, we used $\alpha = 1.5$.

8 FUTURE WORK

The main contribution of our topological fisheye technique is to integrate the display of the global structure of a graph with a way to interactively examine local areas in detail. This technique can be extended in various ways. Although we have only experimented with 2-D layouts, the ideas can be extended naturally to 3-D. We also intend to add animation between successive views to help preserve the user's mental map.

Our implementation is only one possible way of making topological fisheye views, with many alternatives worth considering. One interesting possibility is to combine the display tool with layout creation. Our design is based on separating layout creation from the display tool, as we assumed we are given a graph whose nodes already have assigned coordinates. This separation not only makes the whole design easier and cleaner, but also facilitates fast navigation, because no costly layout calculations are needed during interaction. Also, all viewpoints are based on the same underlying layout, which helps to preserve the user's mental map. However, the effectiveness of our method strongly depends on the quality of

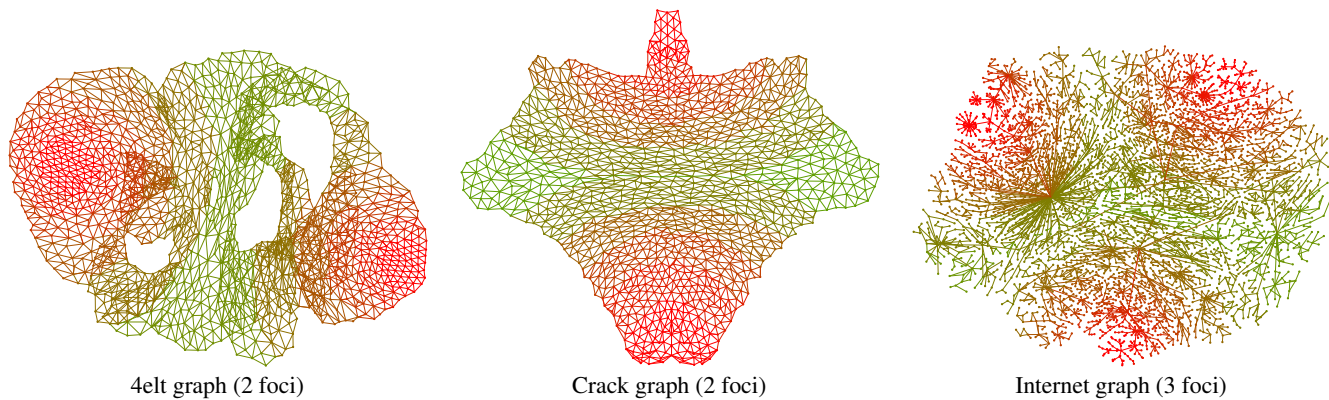| 4elt graph (2 foci) | Crack graph (2 foci) | Internet graph (3 foci) |

Figure 11: Viewing graphs with multiple foci

the given layout, and making a good layout of a huge graph can be very challenging. Therefore, an alternative would be to tightly integrate the display tool with the layout algorithm, so no global layout is needed in advance. Instead, the layout of each hybrid graph could be computed from scratch. This avoids the problem of drawing the full large graph, as only the much smaller hybrid graphs are drawn. Of course, this change affects our whole algorithm, as the layout of the original graph is used extensively in all stages of the process. Such an integration of the display tool with layout computation will challenge the interactivity of the tool, as response time will necessarily be longer. Also, as there is no single base layout, layouts of different hybrid graphs could differ significantly, affecting the smoothness of transitions and the user's mental map.

REFERENCES

[1] D. Auber, Y. Chiricota, F. Jourdan and G. Melançon. Multiscale Visualization of Small World Networks. *Proc. IEEE Symposium on Information Visualization*, 75-81, 2003.

[2] M. S. T. Carpendale, D. J. Cowperthwaite and F. D. Fracchia. Making Distortions Comprehensible. **Visual Languages**, 36-45, 1997.

[3] W. Cheswick and H. Burch. Mapping the Internet. *IEEE Computer* **32**:4, 97-98,102, April 1999.

[4] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf. **Computational Geometry – Algorithms and Applications**, Springer Verlag, 1997.

[5] G. Di Battista, P. Eades, R. Tamassia and Ioannis G. Tollis. **Graph Drawing: Algorithms for the Visualization of Graphs**. Prentice-Hall, 1999.

[6] G. W. Furnas and B. B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. *Proc. Human Factors in Computing Systems*, ACM, 234-241, 1995.

[7] R. Hadany and D. Harel. A Multi-Scale Method for Drawing Graphs Nicely. *Discrete Applied Math.* **113**, 3-21, 2001.

[8] D. Harel and Y. Koren Graph Drawing by High-Dimensional Embedding. *Proc. 10th International Sympoisum on Graph Drawing*, Springer-Verlag LNCS **2528**, 207-219, 2002.

[9] J. Huotari, K. Lyytinen and M. Niemel. Improving Graphical Information System Model Use With Elision and Connecting Lines. *ACM Transactions on Computer-Human Interaction* **11**:1, 26-58, March 2004.

[10] M. Jünger and P. Mutzel, eds. **Graph Drawing Software**, Springer-Verlag, 2004.

[11] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* **20**, 359-392, 1998.

[12] T. Kamada and S. Kawai. An Algorithm for Drawing General Undirected Graphs. *Info. Processing Letters* **31**:1 7-15, 1989.

[13] Y. Koren, L. Carmel and D. Harel. Drawing Huge Graphs by Algebraic Multigrid Optimization. *Multiscale Modeling and Simulation* **1**:4, 645-673, SIAM 2003.

[14] J. Lamping and R. Rao. The Hyperbolic Browser. A Focus+Context Technique for Visualizing Large Hierachies. *Journal of Visual Languages and Computing* **6**:4, 1995.

[15] A. Lingas. A Linear-Time Construction of the Relative Neighborhood Graph From the Delaunay Triangulation. *Computational Geometry* **4**:4, 199-208, 1994.

[16] K. Mehlhorn and S. Näher. **The LEDA Platform of Combinatorial and Geometric Computing**, Cambridge University Press, 1999.

[17] T. Munzner. H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space. *Proc. IEEE Symposium on Information Visualization*, 2-10, 1997.

[18] G. Parker, G. Franck and C. Ware. Visualization of Large Nested Graphs in 3D: Navigation and Interaction. *Journal of Visual Languages and Computing*, 299-317, 1998.

[19] J. Petit. www.lsi.upc.es/~jpetit/MinLA/Experiments

[20] M. Sarkar and M. H. Brown. Graphical Fisheye Views of Graphs. *Proc. Conference on Computer-Human Interaction Conference*, ACM, 83-91, 1992.

[21] J.R. Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. *First Workshop on Applied Computational Geometry*, ACM, 124-133, 1996. Also, www.cs.cmu.edu/~quake/triangle.html

[22] C. Walshaw. www.gre.ac.uk/~c.walshaw/partition

[23] C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. *Journal of Graph Algorithms and Applications* **7**:3, 253-285, 2003.