Topological Fisheye Views for Visualizing Large Graphs

Emden R. Gansner, Yehuda Koren, *Member*, *IEEE*, and Stephen C. North, *Senior Member*, *IEEE*

Abstract—Graph drawing is a basic visualization tool that works well for graphs having up to hundreds of nodes and edges. At greater scale, data density and occlusion problems often negate its effectiveness. Conventional pan-and-zoom, multiscale, and geometric fisheye views are not fully satisfactory solutions to this problem. As an alternative, we propose a topological zooming method. It precomputes a hierarchy of coarsened graphs that are combined on-the-fly into renderings, with the level of detail dependent on distance from one or more foci. A related geometric distortion method yields constant information density displays from these renderings.

Index Terms—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 [6], [15]. In this paper, we consider viewing force directed layouts.

For graphs of modest size, with dozens or hundreds of nodes, there are various useful layout algorithms. The resulting layouts can be viewed with ordinary document viewers or graph viewers having pan and zoom controls. Problems arise, though, when the graphs are larger. The first problem is the algorithmic complexity of computing the layout. In recent years, sophisticated efficient algorithms have been proposed. They include virtual physical models [6] and techniques from linear algebra and statistics [13], [18].

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 even exceeds 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.

• *The authors are with AT&T Research, AT&T Shannon Labs, 180 Park Ave., Florham Park, NJ 07932.*
E-mail: {erg, yehuda, north}@research.att.com.

Manuscript received 15 Sept. 2004; revised 17 Dec. 2004; accepted 30 Dec. 1004; published online 10 May 2005.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0104-0904.

This paper extends prior work published in the *Proceedings of IEEE InfoVis 2004* [11]. We introduce several important extensions. One of the most significant is the animation of transitions between successive graph views. Such animation, as compared with discontinuous changes, is necessary to a user's maintaining a sense of context while exploring graphs. We also describe key details in the implementation of the coarsening and geometric distortion phases. We include detailed examples and figures to illustrate the overall process of computing topological fisheye views.

2 Related Work

Distortion viewing is an obvious candidate for dealing with the problems of browsing large graphs. Variants of 2D geometric fisheye distortion were described by Lamping and Rao [19], Sarkar and Brown [26], and Carpendale et al. [3]. These do a good job of showing the detail in a focus area, but make the global structure, if anything, even more obscure since it is packed into a much smaller area through nonlinear transformation.

We illustrate this with the 4elt graph [28]. 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 variation in the spatial density of nodes [29]. A drawing of this graph computed by the Kamada-Kawai method [17] is shown in Fig. 1. In Fig. 2, we zoom in on its right-hand side using a fisheye lens [3]. This portion is properly enlarged at the expense of the rest of the graph, which becomes very crowded and obscure.

Another approach, proposed by Munzner, employs 3D hyperbolic projection in an interactive viewer with high guaranteed frame-rate rendering [22]. This yields highly scalable viewing of huge trees with a 3D fisheye effect. Display complexity is limited by tree depth and rendered node size. The technique is not intended for general graphs and the problem of distorting areas of the graph distant from the focus are similar to the 2D fisheye.

Clustered graph abstraction has also been considered previously [1], [14], [24]. Such layouts can show the global

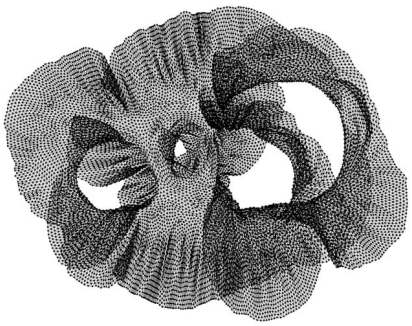

Fig. 1. The 4elt graph, $|\mathrm{V}| = 15,606$, $|\mathrm{E}| = 45,878$.

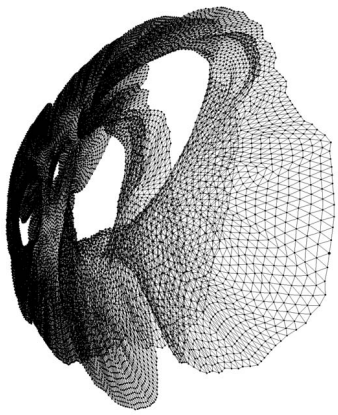

Fig. 2. A fisheye view of the 4elt graph focused on the right-hand portion.

structure well, but navigation is less flexible due to explicit expansion and contraction of clusters. Also, the fine structure across distinct clusters may not be easy to see and, for some methods, the clusters must be externally defined somehow.

In general, such approaches embody Furnas' idea of varying the logical level of detail in many types of focus-in-context displays [9]. The goal of our work is to find effective ways of doing this with general graphs, where the graph's layout and structure must be considered simultaneously to make views that that faithfully represent the graph's structure at all times.

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 [10].

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, 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. 4a. 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 an orange-to-blue scale depending on their level in the hierarchy: The orange area around the focal node is directly from the original graph, while the blue 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 to 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, as in Fig. 4b and Fig. 4c.

An easier to follow example involves the relatively small Qh768 graph ($|V| = 768$, $|E| = 1,322$) [2]; see Fig. 5. In Fig. 6, we provide a series of layouts resulting from gradually changing the focus counterclockwise from the 12 o'clock position to the 9 o'clock position.

Fig. 7 shows two topological fisheye views of a published Internet map [4]. It is a large tree ($|\mathrm{V}| = 87,931$, $|\mathrm{E}| = 87,930$) made by tracing connections from a central probe to all

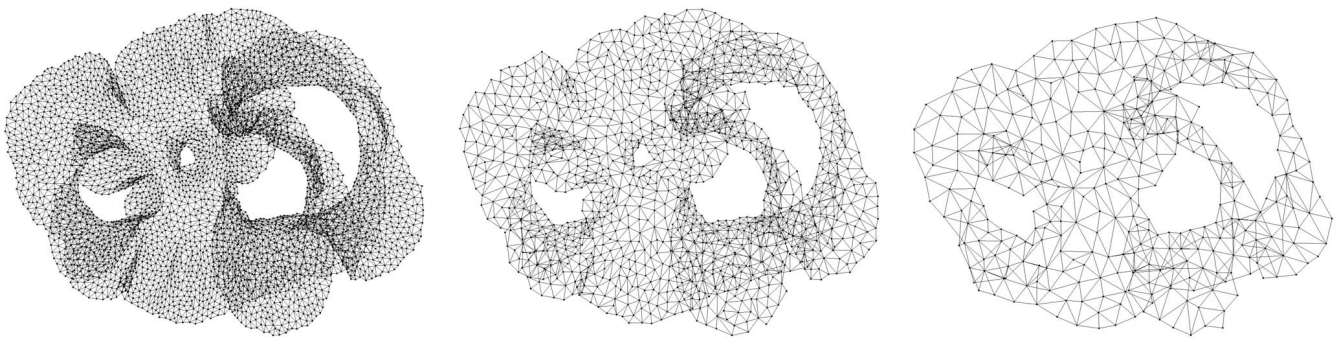

Fig. 3. Approximating the 4elt graph at three different scales of decreasing size and accuracy.

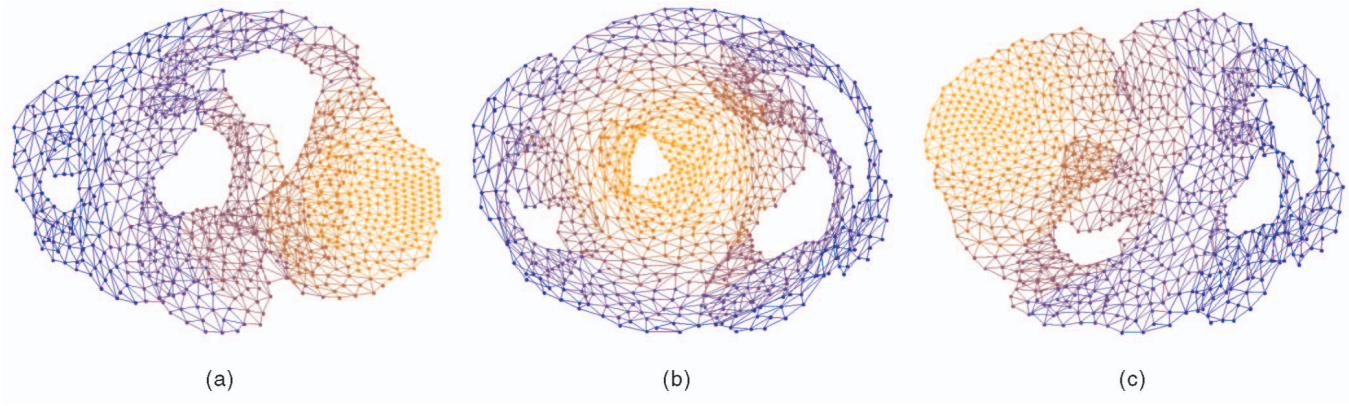

Fig. 4. Topological fisheye views of the 4elt graph. Views are based on "hybrid graphs" formed by superposition of several approximations of the graph. Levels are colored orange-to-blue, where the focus area from the finest graph is in orange. The figure shows three examples, focusing on (a) the right-hand side, (b) the small central hole, and (c) the left-hand side.

reachable IP subnets. The full layout of this graph in Fig. 7a is too dense to read. In contrast, our approach, seen in Fig. 7b and Fig. 7c, yields useful views by focusing on different portions of the graph.

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 [7] 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 approximating 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 various areas for further inspection. Moving the focus triggers redrawing, which is done by computing a new hybrid graph from 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 compensate for its multiple scales, as explained in Section 6. A flowchart of the overall process is shown in Fig. 8. The method is easily extended to handle several simultaneous focal nodes; we discuss the key adaptations in Section 7, along with a short discussion on animating the process. We conclude by discussing possible directions for further work in Section 8.

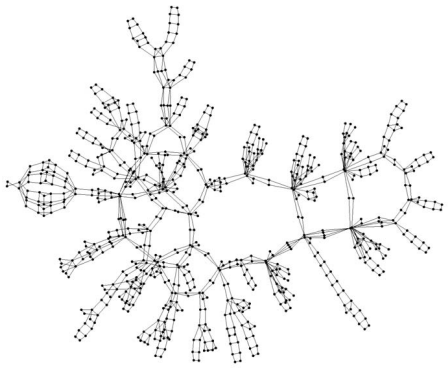

Fig. 5. The Qh768 graph ($|V| = 768$, $|E| = 1,322$) [2].

4 Multiscale Representation of the Graph

The key step in our method is the construction of a multiscale representation of the graph. Given a graph with layout coordinates, we compute a hierarchy of *coarse graphs* of decreasing sizes that approximate 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 a common approach in which nodes of a coarse graph induce a partitioning of the fine graph, meaning that coarse nodes represent disjoint clusters of fine nodes. Thus, a wide range of clustering algorithms may be applied to construct the hierarchy. However, for our purposes, the algorithm should satisfy certain additional requirements:

1. The 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. The 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.

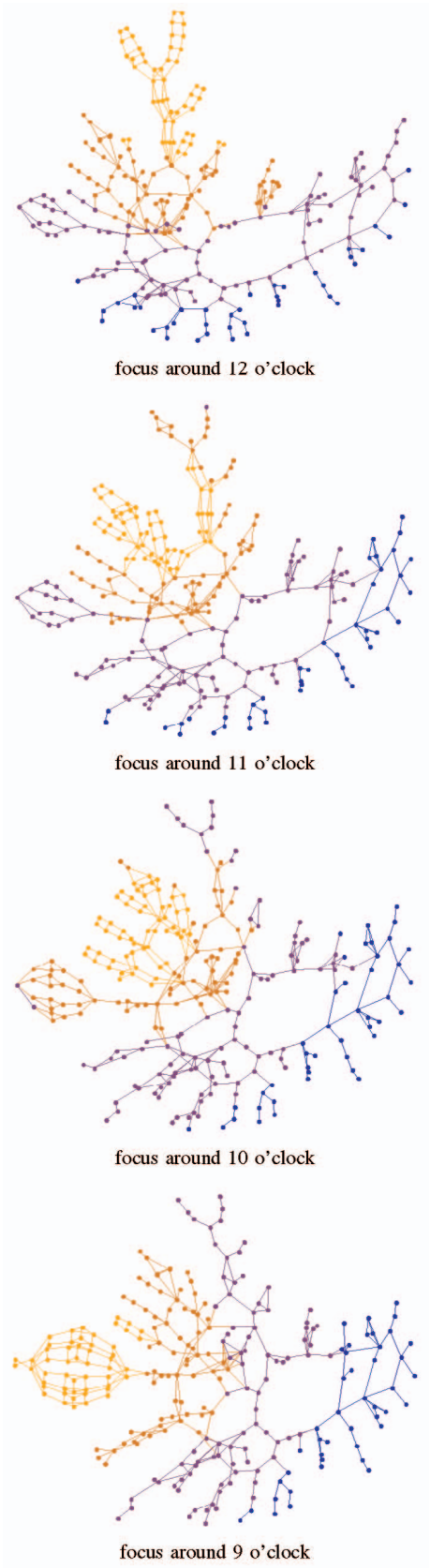

Fig. 6. A sequence of topological fisheye views of the Qh768 graph. The orange nodes belong to the original graph, forming the regions of interest. The other nodes are taken from coarser graphs.

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 [12], [16], [29]. Contraction of node pairs 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. Merged edges are assigned the sum of the weights of the component edges. Any edge between the two nodes is deleted. Fig. 9 gives an example of this operation.

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 $\langle i,j \rangle \in E$. 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 $N_i \stackrel{\text{def}}{=} \{j \in V | \langle i,j \rangle \in E\}$. Similarly, $N_i^* \stackrel{\text{def}}{=} N_i \cup \{i\}$. 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 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,

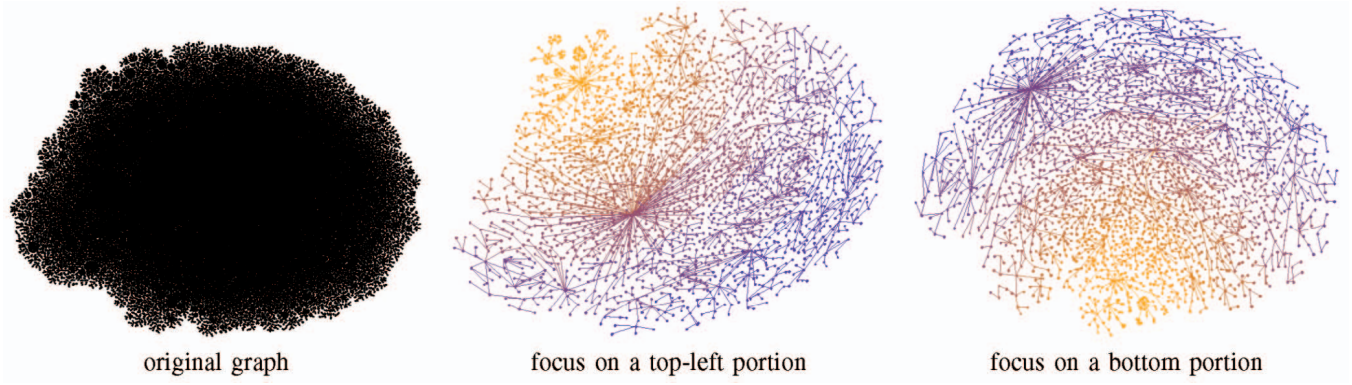

Fig. 7. This Internet map ($(|\mathrm{V}| = 87,931,\ |\mathrm{E}| = 87,930)$) is too large to visualize as a flat structure. Two topological fisheye views are shown. The focused sections in orange are the original graph. Peripheral areas, in blue, are simplified.

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 joins separate parts of the graph, causing dramatic changes in the graph's topology. For example, consider Fig. 10, where contraction of the two gray nodes causes an abrupt change in the topology, introducing a cycle in the graph's structure. Fortunately, if we contract only nodes whose unweighted graph-theoretic distance is 1 or 2, we can guarantee that no 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, combining 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 [12], [16], [29]. However, we have found that the edge set does not provide us enough freedom and, hence, is not suitable. For example, consider the star-like subgraph in Fig. 11. Since the contracted pairs must be disjoint, only a single pair out of this star can be contracted. Consequently, the coarsening

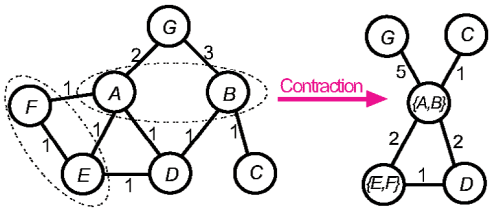

Fig. 9. Contracting the two node pairs $\{A, B\}$ and $\{E, F\}$.

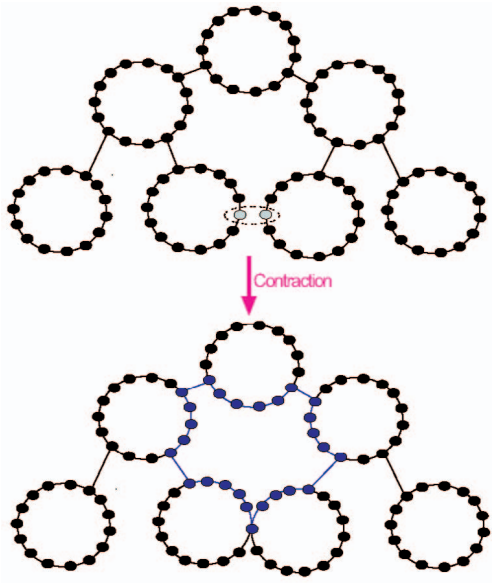

Fig. 10. The effect of contracting nodes with large graph-theoretic distance: Contracting the two gray nodes in the top graph yields a fundamentally different graph, with a large, new cycle (colored blue).

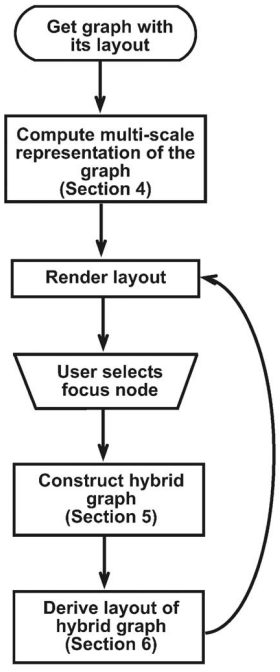

Fig. 8. A flowchart of the topological zooming tool.

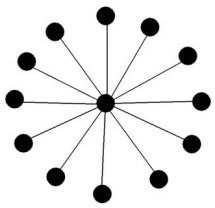

Fig. 11. A star-like graph.

rate is very 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 [5]. 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 [5], 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:

$$\|p_i - p_j\| \leq \max\{\|p_i - p_k\|, \|p_j - p_k\|\}.$$

  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 2D geometry, both the DT and RNG are planar graphs and can be computed in $O(n \log n)$ time [5], [20]. Several good implementations are available [21], [27]. Fig. 12 shows examples of both the 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 about 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 away in the graph-theoretic sense, we remove all pairs whose graph-theoretic distance is above k, where k is usually 2 or 3. Note that if 3 is used, one introduces the possibility of creating a (trivial) cycle when the nodes are merged. The user may decide this is acceptable.

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 $\|p_i - p_j\| > \max\{\|p_i - p_k\|, \|p_j - p_k\|\}$.

(Other proximity graphs, such as Gabriel graphs [5], might be considered as alternatives. Because the performance of the intermediate proximity graph defined above is satisfactory in practice, we have not explored this further.)

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 and
2. the edges of a proximity graph, excluding those whose endpoints are too distant 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

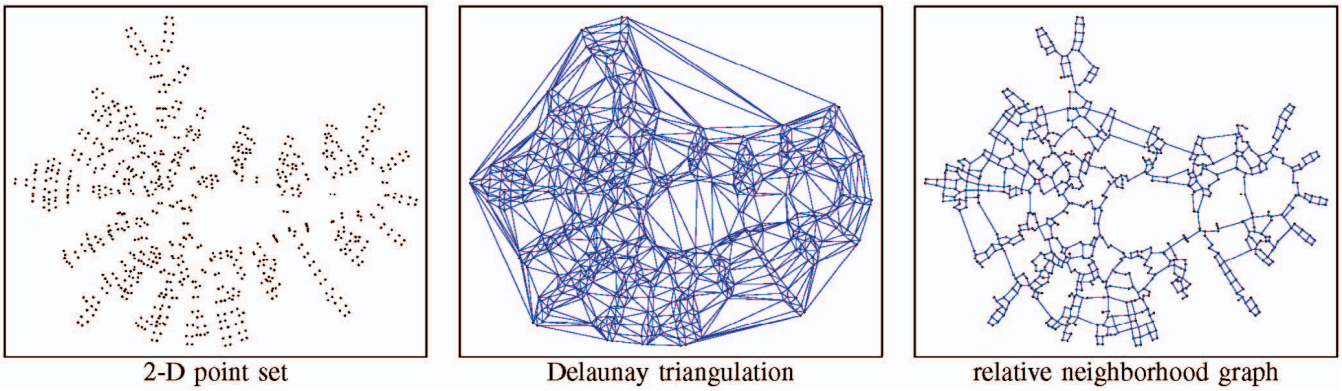

Fig. 12. The Delaunay triangulation and relative neighborhood graph of a point set.

TABLE 1
Running Time (Sec) of the Three Phases of the Zooming Technique, Measured on a 2.6 GHz Pentium-4 PC

Graph name	$\|V\|$	$\|E\|$	hierarchy creation	hybrid graph	distortion
Crack	10,240	30,380	0.08	0.005	0.0
4elt	15,606	45,878	0.14	0.01	0.0
Internet graph	87,931	87,930	0.87	0.05	0.015
Sierpinski, depth 10	88,575	177,147	0.75	0.04	0.015
Grid 317×317	100,489	200,344	0.87	0.04	0.015
Grid 500×500	250,000	499,000	2.43	0.11	0.015

of candidate pairs, we prefer a linear time heuristic because computational performance is critical.

We use the following procedure; related techniques are described elsewhere [12], [16]. 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: $\frac{1}{\text{size}_i + \text{size}_j}$.
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: $\frac{1}{\deg_i \cdot \deg_j}$.

The first measure is aimed at preserving the graph's geometry. The second encourages uniform node sizes. The third and fourth measures help to preserve 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.

Because the various measurements are in different scales, we normalize them all to the range $[0, 1]$ as part of the weighting scheme. In our implementation, we apply weights 3, 0, 1, 1, and 1, respectively, to the normalized values. (This weighting ignores cluster size, but they are empirically almost always 1 or 2 and, therefore, uniform anyway. A nonzero value could be helpful for coping with pathological input graphs in which some points are not clustered on any level, though we have not seen that experimentally.)

Since we deal with graphs having layouts, 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:

$$\frac{\text{size}_i \cdot p_i + \text{size}_j \cdot p_j}{\text{size}_i + \text{size}_j}.$$

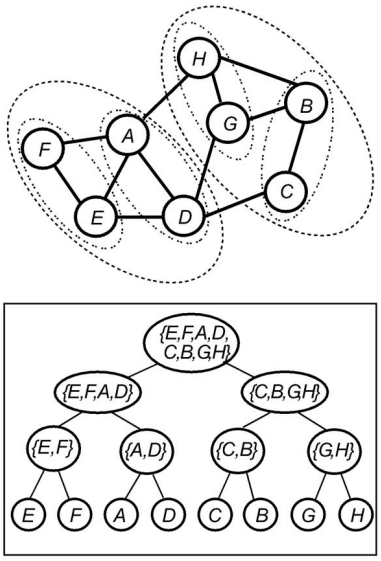

Fig. 13. An 8-node clustered graph and its corresponding hierarchy tree. Each level in the tree corresponds to a single coarse graph.

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, the hierarchy has $O(\log n)$ levels and the total coarsening time is linear. It also takes $O(n \log n)$ time 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 with a star-like subgraph when contracting only neighbors) and we terminate it, regardless, after a predetermined number of levels (50, in our implementation). This degenerate behavior may signify 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 one million nodes on a Pentium-4 PC; see under "hierarchy creation" in Table 1 for measurements. However, it is performed only once in the preprocessing stage that precedes the user interaction, so it does not inhibit quick response when exploring data.

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. 13. Each tree node corresponds to a coarse node representing a set of the 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

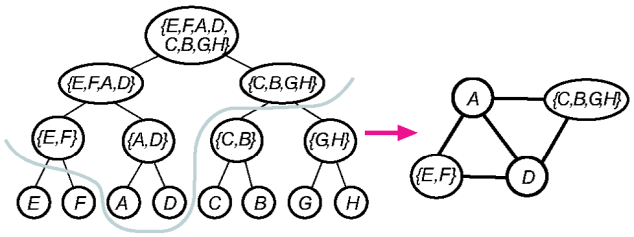

Fig. 14. A horizontal slice through the hierarchy tree of Fig. 13. This induces a 4-node hybrid graph that approximates the original graph at various levels of accuracy simultaneously.

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. 14, 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 four nodes and four edges and is shown in Fig. 14.

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.

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 geometric distance from the focus, in ascending order. Then, we take the first 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 representation level. However, as we will see, this does not yield a final representation as inconsistencies may arise. Regarding level sizes, from implementation experience, we recommend that c_0 be 50-100, so 50-100 nodes are individually represented in the hybrid graph, forming the "focal region" of the graph. For the other levels, we use the rule $c_{i+1} = C \cdot c_i$, where $2 \leq C \leq 3$.

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

How can such inconsistencies arise? Note that the association of desired levels completely ignores 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 by 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 in level $i+1$. The desired representation levels of A and B are l_A and l_B, respectively. Without loss of generality, assume $l_A \leq l_B$. 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.
- $l_A \leq i$ and $l_B \geq i + 1$. 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 is negligible for all graphs we experimented with; experimental running times are listed under "hybrid graph" in Table 1.

Our application does not need to explicitly construct the hybrid graph as, for our purposes, 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.

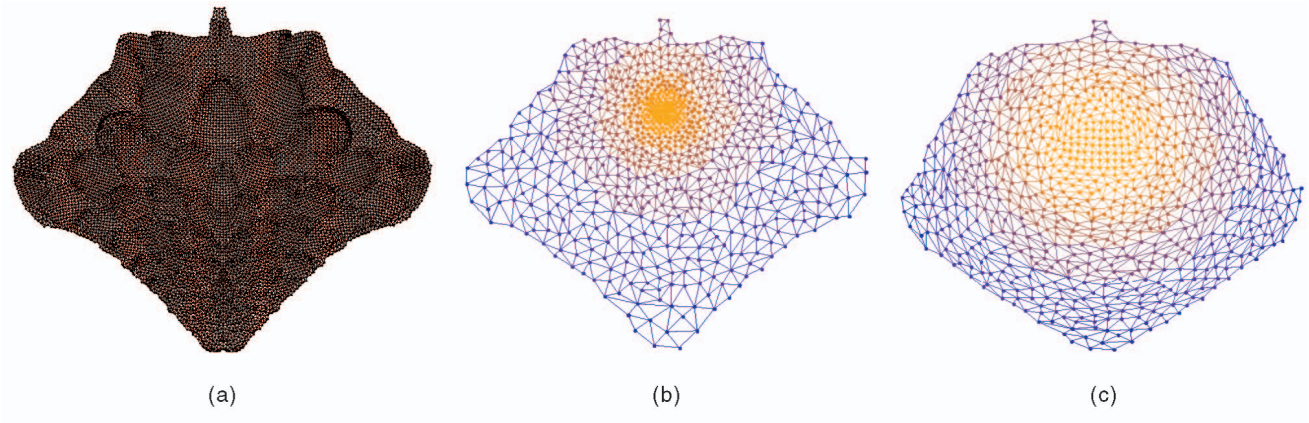

Fig. 15. The Crack graph [25], $|\mathrm{V}| = 10,240$, $|\mathrm{E}| = 30,380$. In the hybrid graph, the focus region is colored orange. Its default layout is derived directly from the original layout and exhibits a very dense focus region. The distorted layout is a uniformly scattered version of the default layout obtained by radial distortion. (a) Original layout. (b) Default layout of hybrid graph. (c) Distorted layout of hybrid graph.

6 Distorting the Layout

The final step of our method is the actual display of the hybrid graph. In this section, we will explain how its layout is computed. Notice that 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 usually not 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 [25] in Fig. 15a. When we focus on a central node, we get the default layout of the hybrid graph shown in Fig. 15b. The focus region (in orange) is very dense. As distance from the focus increases, the layout becomes sparse. This may hinder exploring details near the focus.

Although the default layout is not satisfactory, it does preserve several qualities of the related fine graph layout. 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 the following, we explain how simple radial distortion can correct nonuniform density. The result of applying this distortion to the default layout of the Crack graph is shown in Fig. 15c. It is clear that the result is much more uniform and readable than the original layout in Fig. 15b.

We will work with *polar coordinates*, which allows cleaner expressions as compared with *Cartesian coordinates*.[1] 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 by distance from the focus, so $r_i \leq r_{i+1}$.[2] We are looking for some distortion function $\mathcal{D}$ so that the distorted layout $\mathcal{D}(r_1, \theta_1), \ldots, \mathcal{D}(r_m, \theta_m)$ has a "uniform density."

1. Recall that a 2D point P is described by the polar coordinates (r, θ). The *radial coordinate* r denotes the distance of the point from the origin O ($r = \|OP\|$), while the *angular coordinate* θ is the angle between OP and the positive x-axis.

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

Observe that the hybrid graph merges several coarse graphs. By our construction of this graph (see Section 5), it is reasonable to assume that nodes at the same distance from the focus belong to the same coarse graph. Therefore, each circle centered at the focus constitutes a region of equal density. 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, 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 $\mathcal{D}(r, \theta) = (\mathcal{F}(r), \theta)$, where $\mathcal{F} : \mathbb{R} \rightarrow \mathbb{R}$ is the radial distortion function.

At this point, we need to formulate a 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 1D problem. Since we can directly measure density only on the m given points, we discretize our space and assume uniform density inside intervals between consecutive points. These intervals are defined by:

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

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

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

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

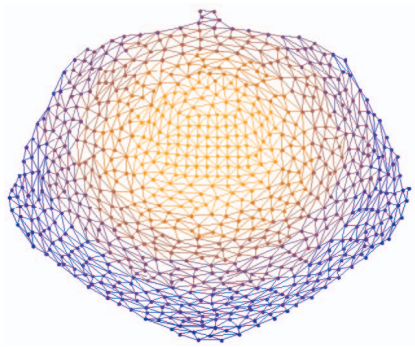

Fig. 16. Overemphasizing the focal region: We set $\alpha = 1.5$ to grant more area to the focal region of the Crack graph of Fig. 15.

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:

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

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

Or, equivalently,

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

The result is that denser circular segments are allocated more area, making the density uniform overall. Note that $\mathcal{F}$ is an increasing function, therefore our distortion function $\mathcal{D}(r, \theta) = (\mathcal{F}(r), \theta)$ relates the distorted layout to the original layout with two properties:

1. Angular coordinates do not change.
2. The order of radial coordinates does 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:

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

Setting $\alpha > 1$ makes the focus region less dense than other regions, yielding an effect similar to a classic geometric fisheye view [3]. In Fig. 16, we show the Crack graph again, with the same focal region as Fig. 15, but with $\alpha = 1.5$ to give the focal region more area.

In our default setting, we prefer uniform density over the whole layout, so $\alpha = 1$. (All single-focus layouts shown in this paper (except Fig. 16) were made this way.) Note that $\alpha = 0$ means no distortion at all.

To summarize, we sketch the steps of the distortion process.

Function Distortion $\{P = (r_1, \theta_1), \ldots, (r_m, \theta_m)\}$
% Input:
% m points represented by polar
% coordinates, in ascending radial distance
% Output:
% distorted coordinates with uniform density

Const $\alpha = 1$
Construct RNG of P
for $i = 1$ to m
  Compute d_{Δ_i} according to (1)
end for
$\hat{r}_0 \rightarrow 0$
for $i = 1$ to m
  $\hat{r}_i \leftarrow \hat{r}_{i-1} + \frac{r_i - r_{i-1}}{(d_{\Delta_i})^{\alpha}}$
end for
return $(\hat{r}_1, \theta_1), \ldots, (\hat{r}_m, \theta_m)$

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 value of m is significantly smaller than n (the original number of points) and, therefore, the distortion takes negligible time; see under "distortion" in Table 1.

7 Extensions

We extended the basic algorithm in two ways:

7.1 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 by 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 allow several disjoint dense areas, invalidating our assumption of constant density around circles centered at the focus. Our solution 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 one final layout. We found this approach to be effective. In the multifoci case, we recommend emphasizing the foci areas by slightly increasing the distortion factor α (defined in Section 6).

Fig. 17 shows views with two and three foci of three graphs which appeared previously in this work with the same orange-blue scale showing level of detail. In these examples, $\alpha = 1.5$.

7.2 Animation

Animation between successive views helps preserve the user's mental map when graph structure is changed by moving the focus.

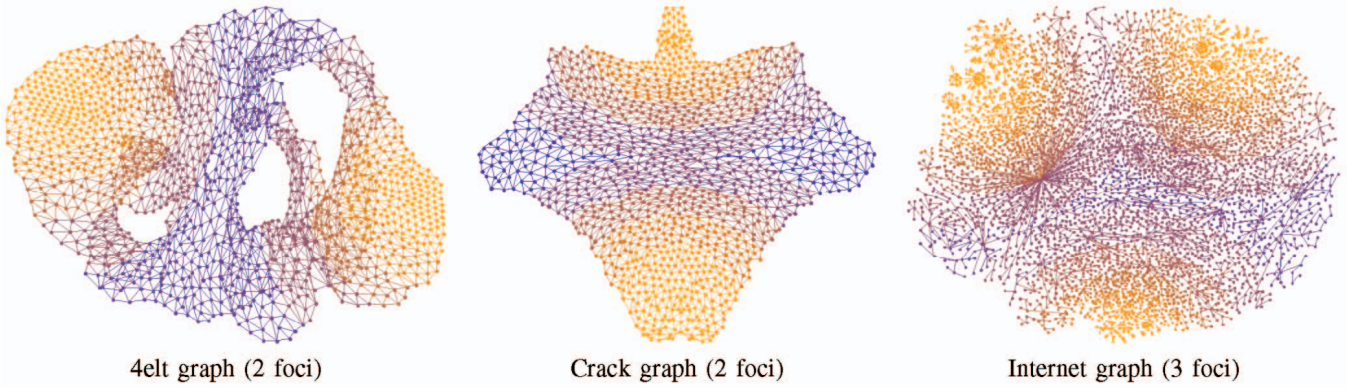

Fig. 17. Viewing graphs with multiple foci.

To enable animation, we maintain two sets of active nodes: the old active nodes generated by the previous focus node and the new active nodes generated by the new focus node. This way we construct three different hybrid graphs. The first is the old hybrid graph induced by the old active nodes. Its coordinates are defined in the usual manner; see Section 6. The second is the new hybrid graph induced by the new active nodes; its coordinates are defined the same way.

The third graph is induced by merging the two sets of active nodes. This merged set is formed by taking the union of the old and new active nodes and removing each node that has a descendant (according to the hierarchy) in the set. This yields a valid horizontal slice through the hierarchy. For example, consider Fig. 18. The gray horizontal slice defines the old active nodes $\{E, F\}, A, D, \{C, B, G, H\}$. The new active nodes $\{E, F, A, D\}, \{C, B\}, G, H$ are defined by the blue horizontal slice. Consequently, the merged set is defined by the solid (nondashed) horizontal slice and contains $\{E, F\}, A, D, \{C, B\}, G, H$.

Each node in the merged set corresponds to a single old active node, either itself or one of its ancestors. Similarly, each node in the merged set corresponds to a single new active node, also itself or one of its ancestors. Thus, we can relate each node of the new graph to two points: the "source point," where its old active node resides, and a "destination point," where its new active node resides.

Therefore, the "merged graph" aligns visually with the old hybrid graph when taking the source points as the node coordinates and aligns visually with the new hybrid graph when taking the destination points as the node coordinates. Consequently, we can animate the transition from the old hybrid graph to the new one by moving each node of the merged graph along the line from its source to destination points.

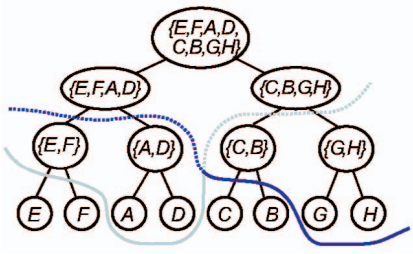

Fig. 18. Combining the gray and blue horizontal slices into the merged horizontal slice (solid line).

As pointed out in Section 5, none of these hybrid graphs is explicitly constructed. Instead, we simply maintain, for each node in the hierarchy, pointers to its ancestor active nodes from the old, new, and merged sets, whenever these ancestors exist.

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 2D layouts, the ideas can be extended naturally to 3D.

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 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 Symp. Information Visualization,* pp. 75-81, 2003.

[2] R.F. Boisvert et al., "The Matrix Market: A Web Resource for Test Matrix Collections," *Quality of Numerical Software, Assessment and Enhancement,* R.F. Boisvert, ed., pp. 125-137, Chapman Hall, 1997, math.nist.gov/MatrixMarket.

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

[4] W. Cheswick and H. Burch, "Mapping the Internet," *Computer,* vol. 32, no. 4, pp. 97-98, 102, Apr. 1999.

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

[6] G. DiBattista, P. Eades, R. Tamassia, and I.G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, 1999.

[7] P. Eades, W. Lai, K. Misue, and K. Sugiyama, "Layout Adjustment and the Mental Map," *J. Visual Languages and Computing,* vol. 6, no. 2, pp. 183-210, 1995.

[8] K.M. Fairchild, S.E. Poltrock, and G.W. Furnas, "SemNet: Three-Dimensional Graphic Representations of Large Knowledge Bases," *Cognitive Science and Its Applications for Human Computer Interaction,* R. Guindon, ed., pp. 201-233, Hillsdale, N.J.: Lawrence Erlbaum, 1988. Reprinted in *Information Visualization: Using Vision to Think,* S.K. Card et al., eds. Morgan Kaufmann, 1999.

[9] G.W. Furnas, "Generalized Fisheye Views," *Proc. Human Factors in Computing Systems,* pp. 16-23, 1986.

[10] G.W. Furnas and B.B. Bederson, "Space-Scale Diagrams: Understanding Multiscale Interfaces," *Proc. Human Factors in Computing Systems,* pp. 234-241, 1995.

[11] E. Gansner, Y. Koren, and S. North, "Topological Fisheye Views for Visualizing Large Graphs," *Proc. IEEE Symp. Information Visualization,* pp. 175-182, 2004.

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

[13] D. Harel and Y. Koren, "Graph Drawing by High-Dimensional Embedding," *Proc. 10th Int'l Symp. Graph Drawing,* pp. 207-219, 2002.

[14] J. Huotari, K. Lyytinen, and M. Niemel, "Improving Graphical Information System Model Use with Elision and Connecting Lines," *ACM Trans. Computer-Human Interaction,* vol. 11, no. 1, pp. 26-58, Mar. 2004.

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

[16] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM J. Scientific Computing,* vol. 20, pp. 359-392, 1998.

[17] T. Kamada and S. Kawai, "An Algorithm for Drawing General Undirected Graphs," *Information Processing Letters,* vol. 31, no. 1, pp. 7-15, 1989.

[18] Y. Koren, L. Carmel, and D. Harel, "Drawing Huge Graphs by Algebraic Multigrid Optimization," *Multiscale Modeling and Simulation,* vol. 1, no. 4, pp. 645-673, 2003.

[19] J. Lamping and R. Rao, "The Hyperbolic Browser: A Focus+Context Technique for Visualizing Large Hierarchies," *J. Visual Languages and Computing,* vol. 6, no. 4, 1995.

[20] A. Lingas, "A Linear-Time Construction of the Relative Neighborhood Graph from the Delaunay Triangulation," *Computational Geometry,* vol. 4, no. 4, pp. 199-208, 1994.

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

[22] T. Munzner, "H3: Laying Out Large Directed Graphs in 3D Hyperbolic Space," *Proc. IEEE Symp. Information Visualization,* pp. 2-10, 1997.

[23] E.G. Noik, "Exploring Large Hyperdocuments: Fisheye Views of Nested Networks," *Proc. ACM Conf. Hypertext,* pp. 102-205, 1993.

[24] G. Parker, G. Franck, and C. Ware, "Visualization of Large Nested Graphs in 3D: Navigation and Interaction," *J. Visual Languages and Computing,* vol. 9, no. 5, pp. 299-317, 1998.

[25] J. Petit , www.lsi.upc.es/jpetit/MinLA/Experiments, 2005.

[26] M. Sarkar and M.H. Brown, "Graphical Fisheye Views of Graphs," *Proc. Conf. Computer-Human Interaction,* pp. 83-91, 1992.

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

[28] C. Walshaw www.gre.ac.uk/c.walshaw/partition, 2005.

[29] C. Walshaw, "A Multilevel Algorithm for Force-Directed Graph Drawing," *J. Graph Algorithms and Applications,* vol. 7, no. 3, pp. 253-285, 2003.

Emden R. Gansner received the PhD degree in mathematics from the Massachusetts Institute of Technology in 1978. He is a technology consultant in information visualization research at the AT&T Shannon Laboratory. His research interests include graph drawing, graph theory, information visualization, graphical user interfaces, and programming tools and languages.

Yehuda Koren received the BA degree in computer science from the Open University, Israel, in 1997, and the MSc and PhD degrees from the Weizmann Institute of Science, Israel, in 1999. He completed his PhD in the Department of Computer Science and Applied Mathematics of the Weizmann Institute of Science in 2003. Currently, he works at AT&T Research. Among his primary research interests are algorithms for drawing large graphs, data analysis and visualization, multiscale optimization, and clustering algorithms. He is a member of the IEEE and the IEEE Computer Society.

Stephen C. North received the PhD degree in computer science from Princeton University in 1986. He is director of Information Visualization Research at the AT&T Shannon Laboratory. His research interests include graph layout and visualization for spatial data mining. He is a member of the ACM, a senior member of IEEE, and a member of the IEEE Computer Society.