Interactive Manipulation of Large Graphs

Peterson Trethewey* Brynjar Gretarsson† John O'Donovan‡ Svetlin Bostandjiev§
Tobias Höllerer¶
Department of Computer Science
University of California, Santa Barbara.

Abstract

This paper presents two techniques for interacting with large scale graphs of various types. The methods draw on ideas from 3D modeling, mesh deformation, and static graph drawing to promote discovery of hidden information across a wide variety of graph types. The key innovation of this work is the application of methods traditionally reserved for automated graph layout, to produce useful layout through dynamic interactions. Our first technique uses a multigrid method for modeling and animating large 3D graphs, the second employs ideas from a simpler mesh deformation scheme together a user interface to control region of influence. We show how these techniques along with a set of basic graph refinement tools can be used interactively to produce informative visualizations based on graph connectivity alone. We demonstrate the techniques working with diverse graph types including meshes, trees, and small-world networks. Our techniques run at interactive frame rates on a standard desktop or laptop computer, for graphs up to hundreds of thousands of nodes. An analysis and discussion of timing results are presented for a range of graph types and sizes, in addition to images of layouts produced by a user interactions with the techniques.

1 Introduction

Algorithms for drawing a graph in 2D or 3D based on vertex connectivity alone, i.e. in absence of an initial spatial layout of the vertices, have been well explored [9]. In this paper, we focus on the problem of interactively manipulating existing graph layouts.

We propose and develop fast interaction techniques that allow a user to explore, improve and create, meaningful graph layouts. This process occurs interactively in real time. In this context, "informative" and "meaningful" layouts are those from which users can gain new insights into the underlying data, such as identification of strong and weak ties, or cliques in a social network; identification of graph distance between cliques, or the identification of hub nodes and outliers in the data. The techniques presented are scalable to very large graphs with hundreds of thousands of vertices, and are capable of quickly providing the user with a "big picture" of the broader data universe.

Through interaction, we enable a user to mold a graph into his/her own mental model of the underlying data, this has positive implications for both comprehension and recollection. [14] For large graphs, interactivity has commonly been limited to multi-scale representations, folding entire node clusters into a single vertex for manipulation and display purposes [19, 15, 1]. For small to medium-sized graphs, interactive manipulation has been shown to provide useful insights into graph structure and topology [20]. We claim that for large graphs the same is true if we assume sufficient screen space, a scheme for deforming the entire graph in a natural way given adjustments to only a few vertices, and interaction methods that run in real time. This paper presents such interaction methodology. Similar solutions arise in the field of 3D modeling where a user deforms a large polygon mesh by manipulating handles on vertices or patches of vertices. Many techniques have been proposed for deforming large meshes in this way. Mesh deformation and static graph drawing often make use of similar numerical calculations such as solving laplacian systems on the vertices.

We propose two techniques which apply ideas from both fields to the following general problem: Given an arbitrary, undirected, unweighted graph with no self loops or multiple edges, draw the entire graph in 3D with a straight line representing each edge, and then allow the user to modify the layout to explore the graph's properties, extract more information, or make the layout more aesthetic. Ideally the user is able to modify the graph by adjusting only a few vertices, and the entire graph deforms to match the user's adjustments in a manner that reflects inherent graph properties. We demonstrate that for mesh and tree-like structures, the technique usually produces a layout that is a good general representation of the entire structure. For more complex graphs

*e-mail:peterson@math.ucsb.edu
†e-mail:brynjar@cs.ucsb.edu
‡e-mail:jod@cs.ucsb.edu
§e-mail:alex@cs.ucsb.edu
¶e-mail:holl@cs.ucsb.edu

such as those with small-world properties [4], layout derived from interactions can provide useful information about the relationship between the set of moved nodes and the overall data universe. As we demonstrate in our evaluation, the technique produces useful, informative layouts on all of these graph types.

1.1 Related Work

The problem of drawing an arbitrary graph in an informative way given no initial layout information has received much attention in the last decade. Many algorithms for drawing a graph in the plane with a straight line representing each edge have been developed. Some of these methods are compared in running time and aesthetic quality in [9]. More recently, Brandes and Pich presented a study that highlights the benefits of multidimensional scaling [5].

One of the observations from existing evaluations is that spectral analysis graph drawing algorithms, which are algebraically motivated, scale better than classical force-directed models such as [7], especially when sped up via a multigrid approach, as done by the ACE method [12]. Multilevel, physically-based models have also been developed [8] which scale better than classical ones. However, ACE still beats them in speed and often produces comparably aesthetic results [9]. In Section 2.2 we present a multigrid method which solves a system like the one in ACE but aimed at the application of graph layout manipulation rather than graph drawing.

The ACE method uses an algebraic multigrid scheme to solve a laplacian system minimizing a quadratic energy function on the vertices. Many techniques in computer graphics have been developed which use laplacian systems to produce natural looking mesh deformations [2] [22] [11]. The ACE method is similar in spirit to the method described in [16] which uses a scheme based on Ruge-Stuben AMG to achieve natural looking deformations of surfaces at interactive rates.

In [21], Zayer et al. propose a mesh deformation scheme which computes a scalar harmonic field in an initial calculation to determine interpolation coefficients, and then in subsequent iterations, approximates solutions by interpolating. In [16], Shi et al adopt a similar scheme and show that for their application, results are visually indistinguishable. This technique inspires our interpolation technique described in Section 3, but rather than using harmonic coefficients, we compute coefficients based on the graph metric.

Networks with small world structure are abundant in nature, occurring in social networks, biological network and many others [13]. Over the years there have been various techniques developed to visualize small world networks [3, 4, 10, 17, 8, 9]. It is generally agreed [9] that finding a good layout for a small world network is non-trivial due to potentially complex connectivity which is an inherent property of such networks. Auber et al. [3] present a multiscale approach which uses semantic zooming to develop meaningful visualizations of small world data. Heer et al. use a similar interactive approach to this problem in *Vizter*, which employs a simple force-directed layout algorithm, coupled with connectivity highlighting based on user-input. We posit that our interpolation technique can be applied to a small world network to provide natural, intuitive, and meaningful visualizations that rely only on graph structure to function well.

2 Multigrid Method

Multigrid methods are fast linear solvers that work best on geometrically motivated problems such as partial differential equations. In this section, we present a brief overview of the general multigrid approach we follow. For a more complete discussion of algebraic multigrid methods, see [6].

2.1 Classical AMG

Geometric multigrid methods were first developed to solve numerical PDEs. These multigrid schemes work by discretizing the domain of the PDE in a hierarchy of coarser and coarser grids. Solutions on coarse grids are prolongated to get good initial guesses on fine grids. These multigrid schemes are particularly well equipped to solve equations of the form:

$$\Delta\varphi = f$$

The Algebraic Multigrid Method, AMG, was developed to solve arbitrary graph laplacian systems, i.e. systems of the form:

$$Lx = b$$

where L is the $n \times n$ laplacian matrix of a graph (with adjustments for constraints). AMG is mechanically similar to geometric multigrid, but motivated by linear algebra. It assumes that a matrix L is given and uses the underlying graph structure of L to compute a matrix P which acts like the prolongation operator in geometric multigrid. Then it approximates a solution to the system:

$$P^T LPx' = P^T b$$

The vector Px' gets used as an initial guess in an iterative scheme to solve $Lx = b$. In general, P is selected to have about half as many columns as L, so AMG recursively defines smaller and smaller systems until it reaches one that can be solved directly.

2.2 Our Multigrid Method

If L is SPD (symmetric and positive definite) then $P^T LP$ is SPD as well, provided P has full rank. The matrix $P^T LP$ is not always the laplacian matrix of a graph, but since it is SPD, it can be expressed as the sum of a laplacian matrix of a weighted graph and a diagonal matrix. In our implementation, rather than compute that sparse matrix at every level, we compute the appropriate weighted graph G. The matrix $P^T LP$ is encoded as G together with an array representing the entries of the diagonal matrix. The Gauss-Seidel smoothing step of the multigrid scheme can be done by iterating through this graph.

To determine P at each level, we pick a subset S of the vertices of the graph G. Entries in x' correspond to values at vertices in S, and entries in x correspond to values at vertices in all of G. We then pick P to be the operator assigning to each vertex v in G the value already there if $v \in S$ and the weighted average of the one-ring neighbors of v which are in S if $v \notin S$. For most graphs, a maximal independent set makes a good choice for S. If S is a maximal independent set, then every vertex in $G - S$ has at least one neighbor in S so the weighted average is never singular.

Our method is inspired by the mesh deformation scheme presented by Shi et al in [16]. Their method allows a user to constrain the position, normal and binormal vectors at some of the vertices of a mesh, and the entire mesh deforms to meet those constraints. This method performs two passes per iteration. The first pass computes a harmonic quaternion field which is used to find smoothly changing orthonormal frames on all of the vertices. This allows normal and binormal constraints to propagate over the entire model. For a graph layout, there is no concept of a surface normal, so we eliminate this step. Instead, we perform one pass of multigrid solving a 3-dimensional vector valued laplacian system and interpret each vector as the literal position of the vertex. Assuming an initial layout of the graph, we form the right-hand-side of the laplacian system from the laplacian of the original embedding. To improve interactivity, we perform just one v-cycle of the method per frame rather than wait in each frame for the method to converge. This way the frame rate stays uniform, and the method often converges in few enough frames that any lag of the graph behind the cursor is negligible to the user.

Solving a laplacian system on a graph requires that at least one vertex's value be constrained, otherwise the system is rank-deficient. In our software, we present the user with a graph in some initial layout, whereupon each click on a vertex adds a position constraint on that vertex and we call the vertex *fixed*. (Note that *fixed* does not mean the vertex cannot move, it means multigrid uses the position of that vertex to build the right-hand-side of the system, it does not solve for it.) We run iterations of multigrid only when at least one vertex is fixed. Vertices which are not fixed are solved for to attain discrete laplacian equal to what it was in the embedding just before the click. This way, the vertex getting dragged moves exactly where the user intends, fixed vertices not being dragged remain in position, and every other vertex's position relative to its neighbors changes minimally. To free the user from the job of unfixing vertices, we limit the number of fixed vertices to some number m (≈ 10). Each click adds a fixed point, and if as many as m fixed points already exist, the least recently fixed point gets unfixed automatically. In practice we have observed that for big enough m this happens transparently to the user, because by the time m vertices have been fixed, the user has often moved on to a different part of the graph. At the same time, vertices recently moved into position remain in place, affording the user a good amount of control over position.

3 Interpolation Method

In [21], Zayer et al employ a short cut using a scalar-valued laplacian system in an initial pass to compute interpolation weights, and then interpolate in subsequent iterations to approximate solutions. For our second graph layout manipulation technique, we propose a similar scheme except that instead of using a harmonic scalar field for interpolation weights, we derive weights from the combinatorial distance from the vertex the user clicks on.

Whenever the user clicks on a vertex v, we perform a breadth-frist search originating from v and record the combinatorial distance to each other vertex in the graph. Then we apply an appropriate affine function to get weights in the interval $[0,1]$ (see Figure 1). We then apply another function f to the weights to improve the appearance. By default, f is the cubic s-curve $y = 3x^2 - 2x^3$, but since applying f only requires one pass through the vertices, we give the user control over the function f. This allows the user to adjust the radius of influence of each click (see Figure 2).

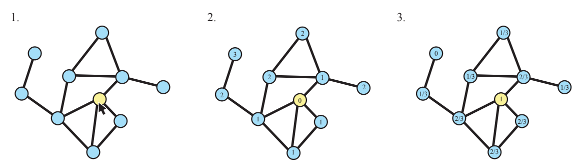


Figure 1: The user clicks on the indicated vertex above in panel 1. In panel 2, the DFS computes combinatorial distances to each other vertex. The distance to the vertex the user clicked on is 0. In panel 3, an affine function is applied to get interpolation weights in $[0,1]$. The weight 1 on the vertex the user clicked on means that that vertex will follow the cursor most. The weight 0 means the vertex stays put.

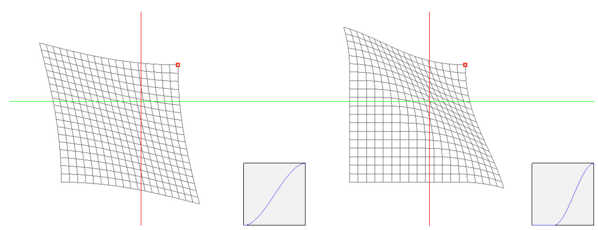


Figure 2: A square grid is deformed by the same movement of a vertex but with different functions applied to the interpolation weights. In each panel, a graph of the function in the unit square is shown in the lower right.

Let $v_1 \dots v_n$ denote the positions of the n vertices of the graph just before a vertex gets clicked. Let $w_1 \dots w_n$ be the weights computed by the above scheme. In each iteration of the mouse-drag that follows, denote by u the vector pointing from the original location of the clicked-on vertex and the dragged location of the clicked-on vertex. Then the new positions of the vertices $p_1 \dots p_n$ are determined in each iteration by the formula:

$$p_i = v_i + w_i u$$

One advantage of this manner of interpolation is that after the initial computation of weights, each iteration of the drag requires only a linear pass through the *vertices* of the graph, not the edges. This method also completely frees the user from having to think about fixed points, the user simply clicks and drags vertices and the graph deforms in a natural looking way.

4 Metric

Given an embedding of a graph G, we propose a metric for how good the embedding is, or rather how bad it is, since for an embedding with uniform edge lengths, our metric is close 0. Let E denote the set of edges in G. Let $l(e)$ denote the length of the edge e in the embedding, and let $\bar{l}$ denote the average length of all edges in the embedding. Let σ^2 denote the variance of the edge lengths given by the following formula.

$$\sigma^2 = \frac{1}{|E|} \sum_{e \in E} (l(e) - \bar{l})^2$$

Let R be the maximum distance (in $\mathbb{R}^3$) from a vertex v the centroid of the vertices. Then we measure the badness M of the embedding by the following formula:

$$M = \frac{\sigma^2}{R}$$

Some properties of this metric:

- It is scale invariant.
- For a random embedding of a graph of any size, M is typically between 1 and 2.
- If all edges are of the same non-zero length, $M = 0$.

Note that M is undefined when all vertices are in the same position. Also, some embeddings with $M = 0$ are not intuitively the best. If G is a tree for instance, the vertices can be placed on integer points on the x-axis such that all edge lengths are 1, but this is not a good way to present the graph. In the case of a tree, however, there are certainly other embeddings which have uniform edge lengths and which are more human-readable.

5 Other Tools

5.1 Perturbing Interpolation Weights

In the interpolation coefficients method, it sometimes happens that vertices stick together simply because they are the same distance from the dragged vertex. This can lead to deceiving layouts where many vertices are in the same location. To fix this problem, we propose artificially perturbing the combinatorial distance to each vertex by a random amount that does not exceed $\frac{1}{2}$ in absolute value. In this manner, interpolation weights are not limited to a discrete set, but rather they are uniformly distributed across an interval. With this random perturbation, it is likely that every vertex moves at a slightly different rate as the user drags the clicked vertex, so individual vertices are more visible.

5.2 Gauss-Seidel Smoothing

A single iteration of Gauss-Seidel on the system $Lx = 0$ with no constraints can make the embedding much more aesthetic. It is well known that Gauss-Seidel converges after a large number of iterations. Convergence in this case means that every vertex moves closer to some constant point, but the first few iterations only eliminate high-frequencey noise. High-frequency noise includes artifacts of the random perturbation described above, creases in mesh like graphs that should be flat and spikes in the graph resulting from isolated fixed points in the multigrid scheme.

5.3 Metric-Driven Layout

Using the metric described in Section 4, we implemented an automated scheme which switches between Gauss-Seidel smoothing and pulling random vertices in random directions using the interpolation method in Section 3. The automated scheme is meant to mimic what in practice we observed to be a typical procedure for producing layouts interactively: pulling vertices in three orthogonal directions, and then smoothing to reduce visual artifacts. Each iteration, we either pull or smooth and favor the layout if M is strictly smaller. If M for the new layout is not smaller, we throw it out and try again. For many graphs this can make M drop dramatically after only a few iterations, but it usually fails to produce as aesthetic a layout as human interaction can.

6 Test Data

The methods presented in this paper draw on ideas from the field of 3D modeling, specifically the modeling of mesh structures. However, the application of our techniques are not by any means limited to mesh structures. In the following sections we evaluate our techniques in terms of visualization quality, speed and scalability. Firstly we define a diverse set of test data which consists of tree structures, small-world data, meshes and graphs of various density. In our evaluation, particular focus is placed on small world data [13], as it occurs frequently in a broad range of sources such as social networks, internet link topology, protein structures, biological networks and many others. If we can demonstrate that our technique is useful for manipulation of small world graphs, it can serve as a useful tool in a broad variety of applications. Three graphs (sw-Xk) were generated using the Barabasi-Albert (BA) model [4] for construction of small world networks. To verify that we have in fact generated data which exhibits small-world properties, the distribution of degree of connectivity was plotted in log-log scale for each graph. Each graph exhibited the distinct power-law distribution which is inherent in small world networks [4]. Details of each graph is shown in Table 1. Additional graphs used in our evaluations were sourced from Chris Walshaw's *Graph Partitioning Archive* [18]. For those graphs, the name given is the name of the file on that site. The binary tree, torus and grid examples were produced procedurally.

7 Evaluation and Results

We now present an evaluation of our techniques in three phases. Firstly, we present an example of our application working in a real-world system to display connections based on citation data with small world properties. Phase two is an comparison of methods which assess both approaches in terms of interaction and aesthetic quality of the resulting visualizations for our diverse set of test graphs. Phase three discusses a timing and scalability experiment which test the limits of our techniques in terms of graph size and interaction speeds.

7.1 Real-World Example

To highlight the real world applicability and scope of our interpolation technique, we show how it can be used to produce clear and meaningful graphs in real-time from a randomly laid-out small-world network of citation data. The term "graph visualization" was entered into a citation search engine and the resulting articles were used as seeds for an expanded query, which returned all articles and authors within a distance of three (articles) from the initial seed. The data was compiled into a graph which exhibited small-world properties [4] for nodes connected to each seed. (We note that due to spelling mistakes etc in the original citation data, there is an increased amount of authors with only a single connection and therefore a slight reduction in the percentage of cycles in the graph.)

Figure 3 shows three visualizations of this data. 3(a.) is subset of this data laid out randomly, and Figure 3(b.) shows the same data laid out by a standard Fruchterman-Reingold [7] force directed technique, which tends to be slow for larger graphs. [9]. This image is similar to many small world network visualizations on the web [17] [7]. For the purpose of this discussion, a target author has been highlighted in all images. Figure 3(c.) shows the layout produced by our interpolation algorithm in real time when the target node is dragged across the screen with the mouse. From this graph, we can clearly see all of the target authors publications and co-authors, which are clustered together where there are cycles between them. The layout categorises all elements in the small world graph by degree of separation from the target node. Given that small world networks are notoriously difficult to lay out with standard algorithms because of high clustering [17], use of interaction in the layout can be considered as a solution, and our technique arrives at a comprehensible layout based on a target node in real time. While we are not claiming that the technique can guarantee good layout for complex small world graphs, it can produce a dynamic transitions such as those from 3(a.) and 3(b.) to 3(cv.) and back. This is an improvement over simply using a tree-based layout technique to arrive at 3(c.) based on a selected node in one of the other layouts, since the user is in direct control of the method through mouse interaction and can 'guide' the interpolation method to arrive at a more informative layout. To facilitate graph-based data exploration the interpolation algorithm can be used to produce ephemeral or ad-hoc layouts. If the method is activated on a mouse-down and drag input, which for example would transition to a view similar to 3(c.) from most other graph embeddings, then upon release of the mouse button, the graph can spring-back to its original embedding after the user has attained the required information about the target node in 3(c.). In this example, the required data might be the set of publications written by co-authors of the target person in 3(c.).

7.2 Comparison of Methods

As a comparison between the multigrid and interpolation techniques, Figure 5(a) shows how similar motions with the two methods can yield visibly different results. In this example, a simple 50×50 square-laid grid is shown with flat initial configuration. The next two panels show the result of dragging a particular vertex upward one with the multigrid method and one with the interpolation method. In the multigrid method, to achieve a

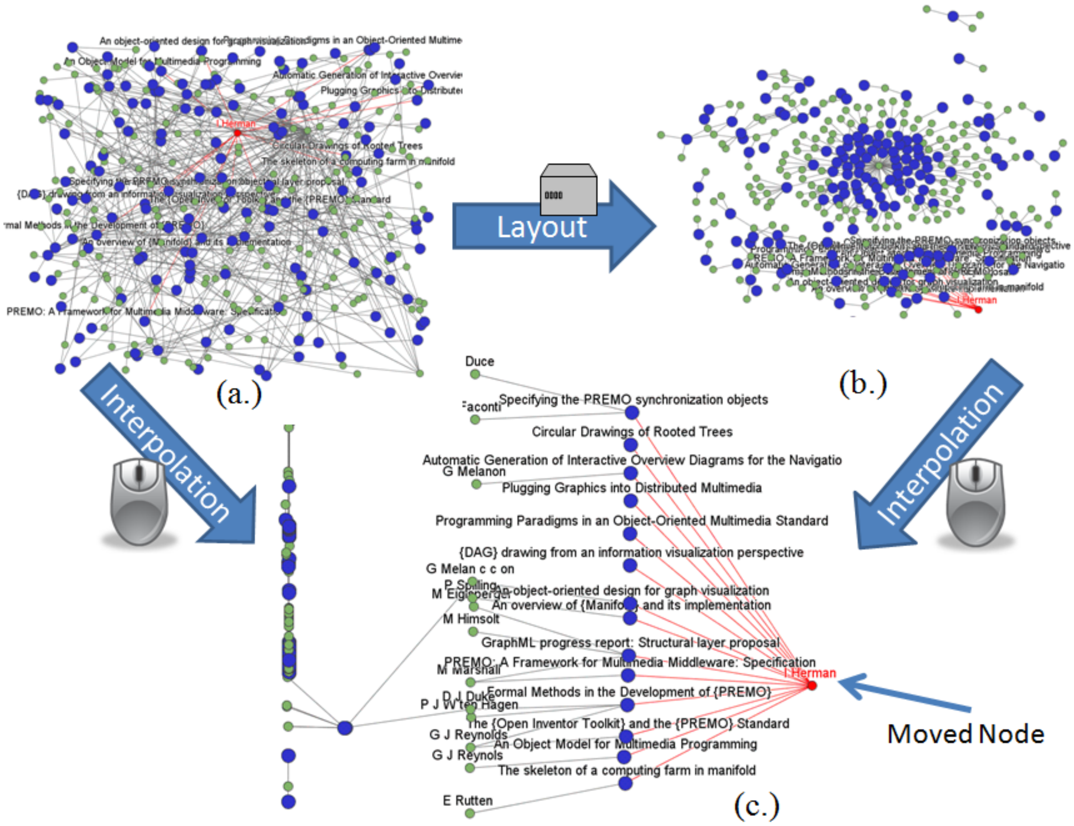


Figure 3: Three different layouts of (small world) citation data related to graph visualization. Authors are shown as larger blue nodes and articles are represented as smaller green nodes. (a.) Random node positioning. (b.)Positioning after Fruchterman-Reingold force directed algorithm. (c) Interpolated positioning, acquired on this small-world network from a single mouse interaction input to our interpolation method. The target author, Ivan Herman, is highlighted in all three sub-figures. Directional arrows highlight the fact that view (c.) can be reached in a single drag from any embedding of the graph.

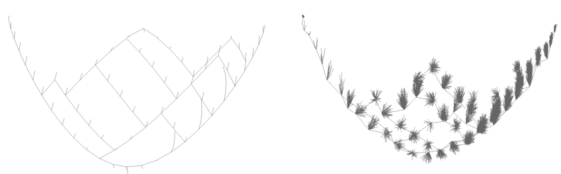
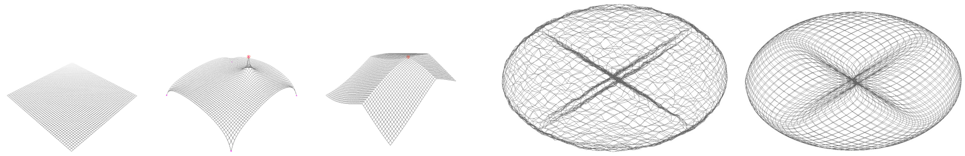


Figure 4: A very tree-like graph laid out using the interpolation technique in three interactive steps once without perturbing the interpolation weights, and once with. In the layout on the left, perturbation is turned off, the valence 1 endpoints of spurs stick together and wind up hidden in the resulting layout. The layout on the right shows the result with perturbation turned on. Numerous spurs are visible yet the global layout remains similar.

comparable effect, the four corners of the square grid are fixed into position first, otherwise dragging just one vertex would translate the entire graph. The result of the multigrid method appears smoother. Creases appear in the interpolation method which make it perhaps less aesthetic, but recall that interpolation weights are computed based on the distance from the dragged point. The n-ring neighborhood of a vertex in the graph metric is diamond shaped in the embedding, so the creases actually reflect a property of the graph metric.

The effect of perturbing interpolation weights can be seen in figures 4 and 5(b). In Figure 4, a graph representing a network of scientific papers and their authors is shown. This graph is almost a tree, it has only a few cycles, and some vertices have numerous twigs attached (by a twig, we mean an edge connecting a vertex to another vertex of valence 1). When perturbation is turned off, all vertices of valence one connected by twigs to the same center vertex move together. As a result, the twigs are completely hidden. When perturbation is turned on, each vertex moves at a slightly different rate, and all the twigs become visible. In Figure 5(b), the graph used is a wireframe torus with 50 meridians and 50 parallels. The first panel shows the embedding generated when



(a) Initial random layout (b) After one interaction (X to Y)

Figure 5: (a) A 50×50 grid is shown with one vertex displaced using the multigrid method and then the interpolation method for comparison. The initial layout appears on the left, the multigrid method is shown in the middle, and the interpolation method is on the right. In the multigrid method, to get a comparable effect, the four corners of the grid are fixed into position before dragging the interior vertex upward., (b) A wireframe torus is laid out using the interpolation technique in three clicks with perturbation of the interpolation weights turned on. The panel on the left shows the resulting embedding and the panel on the right shows the artifacts of the random perturbation removed by three passes Gauss-Seidel smoothing.

perturbation is turned on. Artifacts of the random perturbation are visible. These quickly disappear, however, after only a few iterations of Gauss-Seidel smoothing.

The interpolation method can be used to reveal global topological information about a graph. In Figure 6, we show a wireframe torus with 50 meridians and 50 parallels getting deformed using the interpolation method starting from a random embedding. In just a few pulls, the inherently round nature of the graph becomes apparent, and by the end, the handle of the torus can be seen. A few Gauss-Seidel smooths in the last couple panels massage away residual artifacts of the original random embedding. In examples farther on, we deform graphs by a similar process, but figures show fewer panels.

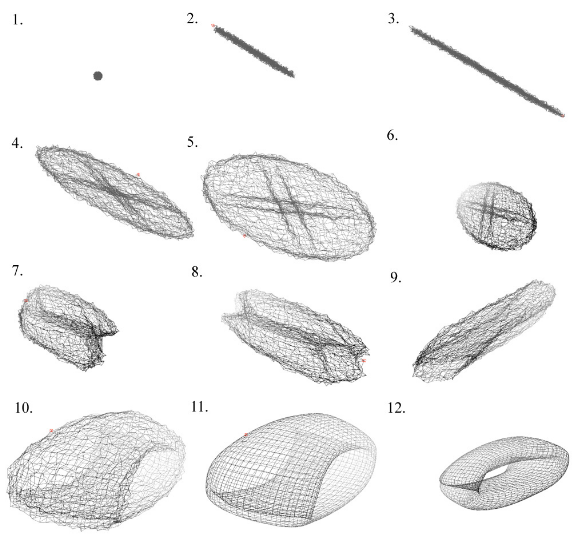


Figure 6: The wireframe torus graph from Figure 5(b) is manipulated using the interpolation technique to produce an informative layout starting from a random initial layout. In panels 1 through 5, various vertices of the graph are pulled outward to spread out the vertices. In panel 6 the viewing angle is changed and in panel 7, the embedding is pulled out of the plane. In panels 8, 9 and 10 the layout is adjusted further. In the panel 11, an iteration of Gauss-Seidel is applied to smooth out residual artifacts of the random initial embedding. In the panel 12, the handle of the torus is apparent.

For a less contrived example than a torus, the graph called “3elt” from [18] is shown in figure 7. In this example, the graph is first pulled into place using the interpolation method. A few smooths get performed to eliminate artifacts of the random embedding, and then in figure 8, the multigrid method gets used to stretch out potions of the graph which were folded up by the first few motions in the interpolation method. As a result, more detail can be seen in those areas.

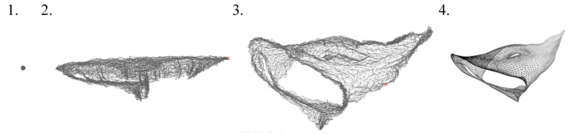


Figure 7: Four intermediate steps of interactive manipulation of the graph 3elt starting from a random initial layout. The penultimate panel shows the graph after a few vertices have been pulled into position, and the last panel shows the result of a a few more adjustments plus one Gauss-Seidel smooth.

In Figure 9, we show the graph “t60k” getting modified using the interpolation technique only. This example shows the utility of being able to control the function that gets applied to the interpolation weights. As the

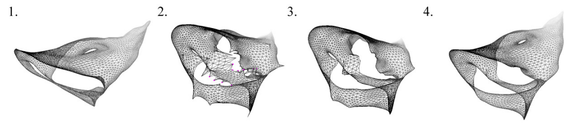


Figure 8: The graph 3elt is shown with initial embedding gotten from the interpolation method as in Figure 7. Here, the graph layout is fine tuned by positioning fixed points using the multigrid method. In panel 2, the graph is rotated to bring the thin part in the lower left to the front, and the thin part is stretched out to expose details. In panel 3, the pointy artifacts the fixed points leave behind get smoothed out with an iteration of Gauss-Seidel. In panel 4, the graph is shown from a viewing angle closer to panel 1, but with the new detail showing.

embedding begins to look better on a global level, the user shortens the radius of influence of each click so that moving vertices only affect the embedding locally. Also in this example we show how the metric M described in Section 4 rapidly decreases as the user improves the embedding and then stays small as the user refines the embedding.

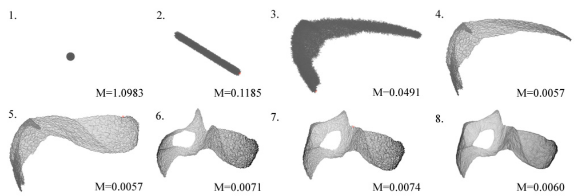


Figure 9: The graph t60k is laid out using the interpolation technique only but with varying functions applied to the weights to attain different radii of influence for each click. The first three panels show the results of a few pulls with the function set to the default s-curve (see Section 3). Panel 4 shows shows the result of smoothing. In panel 5, the radius of influence is adjusted to be closer to half the diameter of the graph. And the right half of the embedding is modified to pull apart sections of the graph which were compressed in the initial interaction. In panels 6 through 8, the graph is viewed from a different angle, the radius of influence is reduced even further and the graph is manipulated to show even more detail. Each panel shows the value of M for the embedding shown.

Figure 10, shows six graph layouts created using the multigrid and interpolation methods together. These examples show the diversity of graphs that the two methods can handle when applied in tandem. The graph entitled "Binary Tree" is a procedurally generated binary tree, the rest of the graphs are from [18]. The binary tree and graphs cs4 and add32 differ from previous examples in that they are not simply wireframe meshes. In particular, cs4 looks mesh-like in the figure but it has a complex, internal 3-dimensional structure.

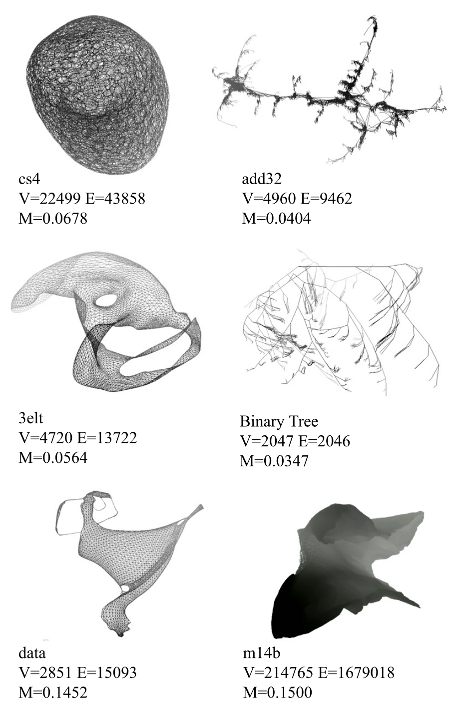
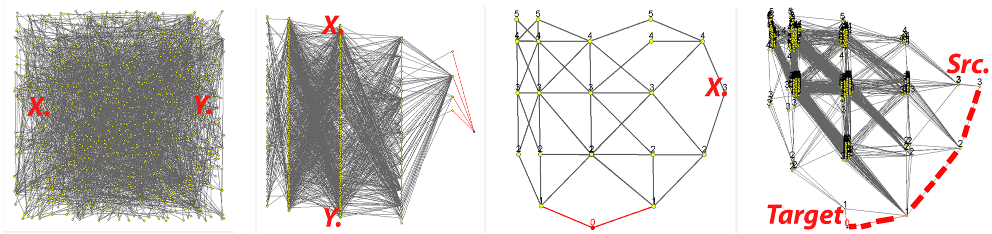


Figure 10: Layouts for a diverse collection of graphs created using our two methods. Each layout is labeled with the name of the graph (This is the name of the file if the graph comes from [18].), the number of vertices V, the number of edges E and the value of the metric M for the embedding shown.

Figure 11 shows an example of three interactions performed on the the sw1k graph, which is a small world network. Figure 11(a) shows an initial random layout. A user clicked on a node at position X and dragged it



(a) Initial random layout (b) After one interaction (X to Y) (c) After two interactions (X to Y) (d) After some disturbance of the layout.

Figure 11: (a), (b) and (c) show operation of the interpolation method on a small-world network. The red line in (c) highlights the shortest path between the two nodes that were moved from X to Y in the previous steps. Furthermore, all other nodes in the network are ordered by their degree of separation from the source and target nodes along horizontal and vertical axes respectively.

to position Y, triggering the interpolation method. The resulting layout from this single interaction is shown in Figure 11(b), where nodes form clusters based on their degree distance from the moved node (which we call "source" for the sake of discussion). Figure 11(b) shows a second node (which we call "target") moved from position X to position Y. The resulting layout from this second interaction is shown in Figure 11(c). This is an interesting layout, where nodes are clustered along the horizontal and vertical axes based on their degrees of separation from the source and target nodes. This form of interaction can have many useful applications. Consider a social networking site for example, using this deformation technique it is possible to visually determine degree of separation between any two people in two real-time mouse motions. This is highlighted by the dashed line in Figure 11(d) which indicates the shortest path between source and target. To arrive at the layout in Figure 11(d), a few nodes were slightly perturbed to show the size of the clusters from Figure 11(c). Furthermore, all nodes in this layout are clustered based on their distance from both source and target. Nodes in the top left corner being farthest away, at 5 hops. Applied to the social networking example, we can now easily determine the degree of separation visually of *any* node in the graph to both source and target with just a few mouse gestures.

7.3 Timing

Table 1 presents timing results in seconds for both the interpolation and multigrid methods on a collection of graphs of varying sizes. Tests were run on a Lenovo ThinkPad T61 laptop with 2.0GHz Intel Core 2 Duo T7300 processor, 1GB of RAM, and an NVIDIA NVS 140M graphics chip with 128MB of video memory. Our program runs on Ubuntu 7.10 (Gutsy Gibbon). Each test was conducted 20 times by an automated test harness, and running times were averaged to get the numbers in the table. For each test, we allowed the graph to draw, but the times presented in the table do not include drawing, only computational time. For the interpolation technique, we separately list the time taken to compute the interpolation weights (has to be performed once on mouse-down) and the average time taken by one frame of dragging a vertex. For the multigrid technique, we simply list the time taken by one v-cycle of the method. The reported time information clearly allows for interactive exploration of graphs with hundred of thousands of nodes and over one million edges: for the first five graphs we achieve frame rates of 12.5 ms per frame or higher for the interpolation technique and 40fps or higher for the multigrid method. The last two graphs can still be explored at about 25 ms/frame or higher with the interpolation scheme, while the multigrid method slows down to 250-500 ms/frame. All these update rate calculations do not count the time necessary to draw the edges and vertices, but a state-of-the-art graphics card can handle such numbers in real time. Two implementations of the interpolation technique have been implemented. The initial section of the video accompanying this paper demonstrates a desktop-based version of the program written in C++ and running on an Apple Macintosh G5. The latter part of the video shows the technique running natively in a web browser[1], controlled with a server-side Java implementation.

The timing results in Table 1 show that the method performs very well on both the artificially generated and real small world datasets. Weights are computed in 98 ms for the 100,000 node graph, while interpolation takes 22 ms per frame.

8 Discussion

Because static graph layout generation is a well studied problem, we focused our attention on interactive graph layout generation and refinement allowing the user to manipulate the graph by hand to attain an intuitive layout

[1] http://eire.mat.ucsb.edu

Graph	Type	V	E	Weights	Interp	V-cycle
add32	tree	4960	9462	.0019	.0007	.0022
data	mesh	2851	15093	.0011	.0004	.0014
3elt	mesh	4720	13722	.0020	.0007	.0021
uk	mesh	4824	6837	.0017	.0007	.0037
t60k	mesh	60005	89440	.0298	.0128	.0253
m14b	mesh	214765	1679018	.2360	.0515	.2111
auto	mesh	448695	3314611	.5807	.1067	.5872
sw1K	s-world	1000	2000	.0003	.0001	.0008
sw10K	s-world	10000	20000	.0048	.0017	.0193
sw100K	s-world	100000	200000	.0981	.0227	.5528
citeseer2	s-world	1385	2503	.0006	.0003	.0014
citeseer3	s-world	6291	11671	.0032	.0018	.0277
citeseer4	s-world	11624	13303	.0051	.0029	.6879

Table 1: Running times for both methods. Weights and interpolation times in are shown in seconds for the interpolation method, while v-cycle times are shown in seconds for the multigrid method.

even on graphs with hundreds of thousands of nodes. Although our methods can be used to create informative layouts for a diverse range of simple graphs, for graphs with multiple edges, weighted edges or self loops, extending our method would be a non-trivial task. For the multigrid method, weighted edges are implicit in each level of the graph, so the method can take edge weights into account, but we found the multigrid method useful mostly for fine tuning layouts generated by the interpolation method to reveal more detail in congested areas. In the interpolation scheme, on the other hand, the natural extension of the method to account for weighted edges would require computing the distance from vertex v to each other vertex in a weighted graph. This is an inherently more difficult problem and might not scale as well as our method. Plus, it is not necessarily the case that the weights in an arbitrary weighted graph are best interpreted as distances. Our methods still struggle with the inherently difficult problem of laying out a highly connected graph at a very general level, however, the technique can be used to elicit useful information visually about groups of nodes. We tested the interpolation scheme with a graph of Wikipedia sites with average valence near 40. With this graph, meaningful global connectivity information was difficult to attain, but through after interpolation we could visually determine proximity between topics much more easily (in terms of link topology) than before the interaction process was applied. It is generally accepted that graphs of this form are, however, difficult to draw in an informative way in general [9].

Despite the simplicity of our interpolation technique, it can produce surprisingly informative layouts. We found that the interpolation method works best for revealing global connectivity data of a graph, especially when that graph has some inherent topology such as: a planar graph, a tree, the 1-dimensional skeleton of a surface mesh or a graph with a natural 3-dimensional embedding like cs4 in Panel 1 of Figure 10. For graphs of this nature, the method can expose information in only a few interactive steps. The benefit of our interpolation scheme is also quantifiable as pulling vertices in the interpolation method to make the embedding more visually pleasing tends also to make our metric M decrease. Gradually reducing the radius of influence of each click can allow more localized editing, and this tends to keep M low. The multigrid method is often even more useful for local editing since each vertex dragged stays put and therefore gives the user more control over positioning. Smoothing out the graph with Gauss-Seidel is a very simple operation and therefore runs quite fast. It eliminates high frequency noise which has the effect of making the graph layout more aesthetic and in general decreases M as well. This makes a good finishing touch on a layout.

Our methods scale well up to graphs of hundreds of thousands of vertices, especially the interpolation technique. We contend that because drawing and manipulation of the entire graph occurs without appealing to supergraphs, where a vertex represents many vertices of the graph of interest, our methods have the potential to visualize very large graphs on high resolution displays showing all vertices and edges.

9 Conclusions and Future Work

We have shown two methods for interactive graph layout manipulation which can be used together to generate embeddings of large graphs in 3D with a manageable amount of interaction. Our techniques draw the entire graph and allow the user to control the layout by moving only a few vertices. In general, we envision improvements to the software which benefit performance by localizing operations which are currently always performed on the whole graph. In our implementation, the multigrid method recomputes the entire grid hierarchy whenever the user clicks a fixed point, it would help to implement an incremental scheme which changes only the part of the grid hierarchy affected by the click. The interpolation technique has similar overhead on the first click due to the initial search that must be performed to compute interpolation weights. In the case where the user has decreased the effective radius of interaction by adjusting the function applied to the interpolation weights, we still compute weights for every vertex, even though many of them are zero. Pruning distant vertices in that search would yield a performance benefit and better equip the method for much larger graphs. We are also in the process of extending our algorithm to support weighted edges as well as edge and node annotations. We have provided visual examples and descriptions of the method providing informative layouts based on a few mouse interactions for a variety of graphs including meshes, trees and small world graphs.

References

[1] D. Archambault, T. Munzner, and D. Auber. Topolayout: Multi-level graph layout by topological features. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):305–317, March/April 2007.

[2] O. K.-C. Au, C.-L. Tai, L. Liu, and H. Fu. Dual laplacian editing for meshes. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):386–395, 2006.

[3] D. Auber, Y. Chiricota, F. Jourdan, and G. Melancon. Multiscale visualization of small world networks. pages 75–81, 2003.

[4] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

[5] U. Brandes and C. Pich. An experimental study on distance-based graph drawing. In *Graph Drawing*, pages 218–229, 2008.

[6] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial (2nd ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[7] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software - Practice and Experience*, 21(11):1129–1164, 1991.

[8] S. Hachul and M. Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Graph Drawing*, pages 285–295, 2004.

[9] S. Hachul and M. Jünger. An experimental comparison of fast algorithms for drawing general large graphs. In P. Healy and N. S. Nikolov, editors, *Graph Drawing, Limerick, Ireland, September 12-14, 2005*, pages pp. 235–250. Springer, 2006.

[10] J. Heer and D. Boyd. Vizster: Visualizing online social networks. *Information Visualization, IEEE Symposium on*, 0:5, 2005.

[11] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum. Subspace gradient domain mesh deformation. *ACM Trans. Graph.*, 25(3):1126–1134, 2006.

[12] Y. Koren, L. Carmel, and D. Harel. Ace: A fast multiscale eigenvectors computation for drawing huge graphs. Technical Report MCS01-17, The Weizmann Institute of Science, 2001.

[13] S. Milgram. The small world problem. *Psychology Today*, (2):60–67, 1967.

[14] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210, June 1995.

[15] J. F. Rodrigues, H. Tong, A. J. M. Traina, C. Faloutsos, and J. Leskovec. Gmine: a system for scalable, interactive graph visualization and mining. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 1195–1198. VLDB Endowment, 2006.

[16] L. Shi, Y. Yu, N. Bell, and W.-W. Feng. A fast multigrid algorithm for mesh deformation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1108–1117, New York, NY, USA, 2006. ACM.

[17] F. van Ham and J. J. van Wijk. Interactive visualization of small world graphs. In *INFOVIS '04: Proceedings of the IEEE Symposium on Information Visualization*, pages 199–206, Washington, DC, USA, 2004. IEEE Computer Society.

[18] C. Walshaw. The graph partitioning archive, availabile at http://staffweb.cms.gre.ac.uk/ wc06/partition/.

[19] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In J. Marks, editor, *Proc. 8th Int. Symp. Graph Drawing, GD*, volume 1984, pages 171–182. Springer-Verlag, 20–23 2000.

[20] P. C. Wong, G. Chin, H. Foote, P. Mackey, and J. Thomas. Have green – a visual analytics framework for large semantic graphs. *Visual Analytics Science And Technology, 2006 IEEE Symposium On*, pages 67–74, Oct. 2006.

[21] R. Zayer, C. Rössl, Z. Karni, and H.-P. Seidel. Harmonic guidance for surface deformation. In M. Alexa and J. Marks, editors, *The European Association for Computer Graphics 26th Annual Conference : EUROGRAPHICS 2005*, volume 24 of *Computer Graphics Forum*, pages 601–609, Dublin, Ireland, 2005. Eurographics, Blackwell.

[22] K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo, and H.-Y. Shum. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph.*, 24(3):496–503, 2005.