

# Dynamic Multilevel Graph Visualization

Todd L. Veldhuizen\*

July 9, 2018

## Abstract

We adapt multilevel, force-directed graph layout techniques to visualizing dynamic graphs in which vertices and edges are added and removed in an online fashion (i.e., unpredictably). We maintain multiple levels of coarseness using a dynamic, randomized coarsening algorithm. To ensure the vertices follow smooth trajectories, we employ dynamics simulation techniques, treating the vertices as point particles. We simulate fine and coarse levels of the graph simultaneously, coupling the dynamics of adjacent levels. Projection from coarser to finer levels is adaptive, with the projection determined by an affine transformation that evolves alongside the graph layouts. The result is a dynamic graph visualizer that quickly and smoothly adapts to changes in a graph.

## 1 Introduction

Our work is motivated by a need to visualize dynamic graphs, that is, graphs from which vertices and edges are being added and removed. Applications include visualizing complex algorithms (our initial motivation), ad hoc wireless networks, databases, monitoring distributed systems, realtime performance profiling, and so forth. Our design concerns are:

- D1. The system should support *online* revision of the graph, that is, changes to the graph that are not known in advance. Changes made to the graph may radically alter its structure.
- D2. The animation should appear smooth. It should be possible to visually track vertices as they move, avoiding abrupt changes.

---

\*Dept. of Electrical and Computer Engineering, University of Waterloo, Canada. Email: [tveldhui@acm.org](mailto:tveldhui@acm.org)

- D3. Changes made to the graph should appear immediately, and the layout should stabilize rapidly after a change.
- D4. The system should produce aesthetically pleasing, good quality layouts.

We make two principle contributions:

1. We adapt multilevel force-directed graph layout algorithms [Wal03] to the problem of dynamic graph layout.
2. We develop and analyze an efficient algorithm for dynamically maintaining the coarser versions of a graph needed for multilevel layout.

### 1.1 Force-directed graph layout

Forced-directed layout uses a physics metaphor to find graph layouts [Ead84, KK89, FLM94, FR91]. Each vertex is treated as a point particle in a space (usually  $\mathbb{R}^2$  or  $\mathbb{R}^3$ ). There are many variations on how to translate the graph into physics. We make fairly conventional choices, modelling edges as springs which pull connected vertices together. Repulsive forces between all pairs of vertices act to keep the vertices spread out.

We use a potential energy  $V$  defined by<sup>1</sup>

$$V = \underbrace{\sum_{(v_i, v_j) \in E} \frac{1}{2} K \|\mathbf{x}_i - \mathbf{x}_j\|^2}_{\text{spring potential}} + \underbrace{\sum_{v_i, v_j \in V, v_i \neq v_j} \frac{f_0}{\epsilon_R + \|\mathbf{x}_i - \mathbf{x}_j\|}}_{\text{repulsion potential}} \quad (1)$$

where  $\mathbf{x}_i$  is the position of vertex  $v_i$ ,  $K$  is a spring constant,  $f_0$  is a repulsion force constant, and  $\epsilon_R$  is a small constant used to avoid singularities.

To minimize the energy of Eqn (1), one typically uses ‘trust region’ methods, where the layout is advanced in the general direction of the gradient  $\nabla V$ , but restricting the distance by which vertices may move in each step. The maximum move distance is often governed by an adaptive ‘temperature’ parameter as in annealing methods, so that step sizes decrease as the iteration converges.

One challenge in force-directed layout is that the repulsive forces that act to evenly space the vertices become weaker as the graph becomes larger. This results in large graph layouts converging slowly, a problem addressed by multilevel methods.

---

<sup>1</sup>Somewhat confusingly, we use  $V$  for potential energy as well as the set of vertices. This is for consistency with Lagrangian dynamics.

Multilevel graph layout algorithms [Wal03, KCH02] operate by repeatedly ‘coarsening’ a large graph to obtain a sequence of graphs  $G_0, G_1, \dots, G_m$ , where each  $G_{i+1}$  has fewer vertices and edges than  $G_i$ , but is structurally similar. For a pair  $(G_i, G_{i+1})$ , we refer to  $G_i$  as the finer graph and  $G_{i+1}$  as the coarser graph. The coarsest graph  $G_m$  is laid out using standard force-directed layout. This layout is interpolated (projected) to produce an initial layout for the finer graph  $G_{m-1}$ . Once the force-directed layout of  $G_{m-1}$  converges, it is interpolated to provide an initial layout for  $G_{m-2}$ , and so forth.

## 1.2 Our approach

Roughly speaking, we develop a dynamic version of Walshaw’s multilevel force-directed layout algorithm [Wal03].

Because of criterion D3, that changes to the graph appear immediately, we focused on approaches in which the optimization process is visualized directly, i.e., the vertex positions rendered reflect the current state of the energy minimization process.

A disadvantage of the gradient-following algorithms described above is that the layout can repeatedly overshoot a minima of the potential function, resulting in zig-zagging. This is unimportant for offline layouts, but can result in jerky trajectories if the layout process is being animated. We instead chose a dynamics-based approach in which vertices have momentum. Damping is used to minimize oscillations. This ensures that vertices follow smooth trajectories (criterion D2).

We use standard dynamics techniques to simultaneously simulate all levels of coarseness of a graph as one large dynamical system. We couple the dynamics of each graph  $V_i$  to its coarser version  $V_{i+1}$  so that ‘advice’ about layouts can propagate from coarser to finer graphs.

Our approach entailed two major technical challenges:

1. How to maintain coarser versions of the graph as vertices and edges are added and removed.
2. How to couple the dynamics of finer and coarser graphs so that ‘layout advice’ can quickly propagate from coarser to finer graphs.

We have addressed the first challenge by developing a fully dynamic, Las Vegas-style randomized algorithm that requires  $O(1)$  operations per edge insertion or removal to maintain a coarser version of a bounded degree graph (Section 4).

We address the second challenge by using coarse graph vertices as inertial reference frames for vertices in the fine graph (Section 3). The projection from coarser to finer graphs is given dynamics, and evolves simultaneously with the vertex positions, converging to a least-squares fit of the coarse graph onto the finer graph (Section 3.2.1). We introduce time dilations between coarser and finer graphs, which reduces the problem of the finer graph reacting to cancel motions of the coarser graph (Section 3.2.2).

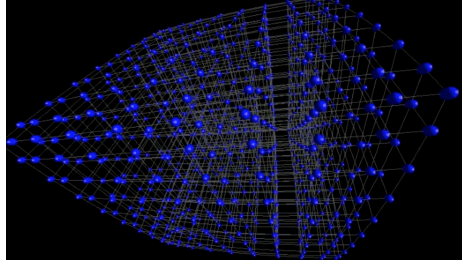
### 1.3 Demonstrations

Accompanying this paper are the following movies.<sup>2</sup> All movies are real-time screen captures on an 8-core Mac. Unless otherwise noted, the movies use one core and 4th order Runge-Kutta integration.

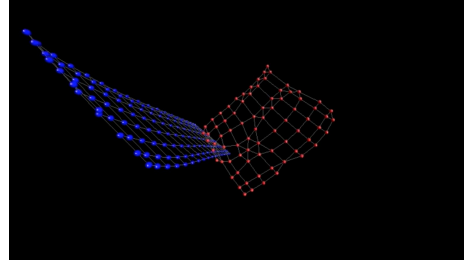
- `ev1049_cube.mov`: Layout of a 10x10x10 cube graph using single-level dynamics (Section 3.1), 8 cores, and Euler time steps.
- `ev1049_coarsening.mov`: Demonstration of the dynamic coarsening algorithm (Section 4).
- `ev1049_twolevel.mov`: Two-level dynamics, showing the projection dynamics and dynamic coarsening.
- `ev1049_threelevel.mov`: Three-level dynamics showing a graph moving quickly through assorted configurations. The coarsest graph is maintained automatically from modifications the first-level coarsener makes to the second graph.
- `ev1049_compare.mov`: Side-by-side comparison of single-level vs. three-level dynamics, illustrating the quicker convergence achieved by multilevel dynamics.
- `ev1049_multilevel.mov`: Showing quick convergence times using multilevel dynamics (4-6 levels) on static graphs being reset to random vertex positions.
- `ev1049_randomgraph.mov`: Visualization of the emergence of the giant component in a random graph (Section 6). (8 cores, Euler step).
- `ev1049_tree.mov`: Visualization of rapid insertions into a binary tree (8 cores, Euler step).

---

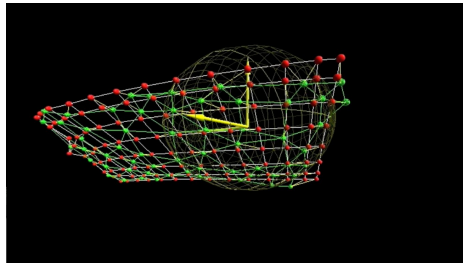
<sup>2</sup> They may be downloaded from <http://ubiety.uwaterloo.ca/ubigraph/ev/>, or by following the hyperlinks from Figure 1.



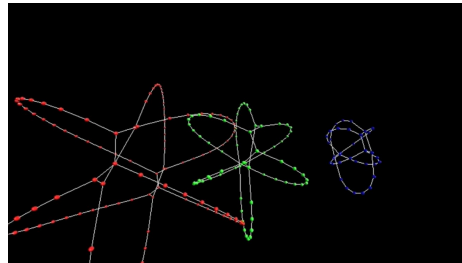
(a) ev1049\_cube



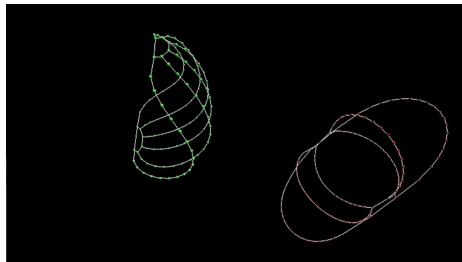
(b) ev1049\_coarsening



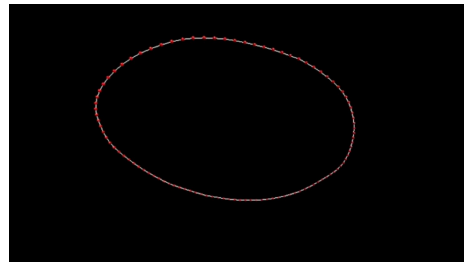
(c) ev1049\_twolevel



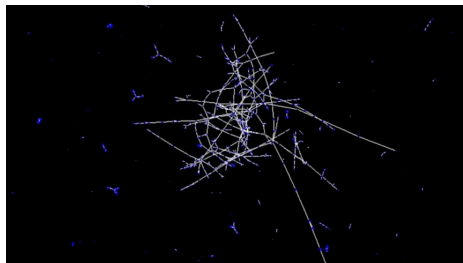
(d) ev1049\_threelevel



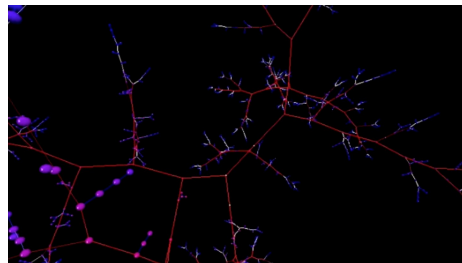
(e) ev1049\_compare



(f) ev1049\_multilevel



(g) ev1049\_randomgraph



(h) ev1049\_tree

Figure 1: Still frames from the demonstration movies accompanying this paper.

## 2 Related work

Graph animation is widely used for exploring and navigating large graphs (e.g., [HMM00].) There is a vast amount of work in this area, so we focus on systems that aim to animate changing graphs.

Offline graph animation tools develop an animation from a sequence of key frames (layouts of static graphs). Such systems find layouts for key frame graphs, and then interpolate between the key frames in an appropriate way (e.g. [FE02]).

In contrast, our system tackles the problem of *online* or *incremental* graph layout. We use the term online in the sense of *online algorithms*: one receives a sequence of requests (in our case, changes to a graph), and each request must be processed without foreknowledge of future requests.

The key frame approach can be adapted to address the online problem by computing a new key frame each time a request arrives, and then interpolating to the new key frame. For example, Huang et al [HEW98] developed a system for browsing large, partially known graphs, where navigation actions add and remove subgraphs. They use force-directed layout for key frame graphs, and interpolate between them.

Another approach is to take an existing graph layout algorithm and incrementalize (or dynamize) it. For example, the Dynagraph system [NW01, EGK\*04] uses an incrementalized version of the batch Sugiyama-Tagawa-Toda algorithm [STT81].

Our basic approach is to develop an incrementalized version of Walshaw’s multilevel force-directed layout algorithm [Wal03]. We perform a continuously running force-directed layout. When a graph change request arrives, the graph is immediately updated. This changes the potential function, so that the current layout is no longer an equilibrium position. The layout immediately starts to seek a new equilibrium. If the changes come rapidly enough, the graph will be in continuous motion. One advantage of this approach is that changes to the graph are shown immediately, without delaying to compute a key frame.

## 3 Layout Dynamics

We use Lagrangian dynamics to derive the equations of motion for the simulation. Lagrangian dynamics is a bit excessive for a simple springs-and-repulsion graph layout. However, we have found that convergence times are greatly improved by dynamically adapting the interpolation between

coarser and finer graphs. For this we use generalized forces, which are easily managed with a Lagrangian approach.

As is conventional, we write  $\dot{\mathbf{x}}_i$  for the velocity of vertex  $i$ , and  $\ddot{\mathbf{x}}_i$  for its acceleration. We take all masses to be 1, so that velocity and momentum are interchangeable.

In addition to the potential energy  $V$  (Eqn (1)), we define a kinetic energy  $T$ . For a single graph, this is simply:

$$T = \sum_{v_i \in V} \frac{1}{2} \|\dot{\mathbf{x}}_i\|^2 \quad (2)$$

Roughly speaking,  $T$  describes channels through which potential energy (layout badness) can be converted to kinetic energy (vertex motion). Kinetic energy is then dissipated through friction, which results in the system settling into a local minimum of the potential  $V$ . We incorporate friction by adding extra terms to the basic equations of motion.

The equations of motion are obtained from the Euler-Lagrange equation:

$$\left[ \frac{d}{dt} \frac{\partial}{\partial \dot{\mathbf{x}}_i} - \frac{\partial}{\partial \mathbf{x}_i} \right] L = 0 \quad (3)$$

where the quantity  $L = T - V$  is the Lagrangian.

### 3.1 Single level dynamics

The coarsest graph has straightforward dynamics. Substituting the definitions of (1,2) into the Euler-Lagrange equation yields the basic equation of motion  $\ddot{\mathbf{x}}_i = \mathbf{F}_i$  for a vertex, where  $\mathbf{F}_i$  is the net force:

$$\ddot{\mathbf{x}}_i = \underbrace{\sum_{v_i, v_j} -K(\mathbf{x}_i - \mathbf{x}_j)}_{\text{spring forces}} + \underbrace{\sum_{v_j \neq v_i} \frac{f_0}{(\epsilon_R + \|\mathbf{x}_i - \mathbf{x}_j\|)^2} \cdot \frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}}_{\text{repulsion forces}} \quad (4)$$

We calculate the pairwise repulsion forces in  $O(n \log n)$  time (with  $n = |V|$ , the number of vertices) using the Barnes-Hut algorithm [BH86].

The spring and repulsion forces are supplemented by a damping force defined by  $\mathbf{F}_i^d = -d\dot{\mathbf{x}}_i$  where  $d$  is a constant.

Our system optionally adds a ‘gravity’ force that encourages directed edges to point in a specified direction (e.g., down).

### 3.2 Two-level dynamics

We now describe how the dynamics of a graph interacts with its coarser version. For notational clarity we write  $\mathbf{y}_i$  for the position of the coarse vertex corresponding to vertex  $i$ , understanding that each vertex in the coarse graph may correspond to multiple vertices in the finer graph.<sup>3</sup>

In Walshaw’s static multilevel layout algorithm [Wal03], each vertex  $\mathbf{x}_i$  simply uses as its starting position  $\mathbf{y}_i$ , the position of its coarser version. To adapt this idea to a dynamic setting, we begin by defining the position of  $\mathbf{x}_i$  to be  $\mathbf{y}_i$  plus some displacement  $\delta_i$ , i.e.,:

$$\mathbf{x}_i = \delta_i + \mathbf{y}_i$$

However, in practice this does not work as well as one might hope, and convergence is faster if one performs some scaling from the coarse to fine graph, for example

$$\mathbf{x}_i = \delta_i + a\mathbf{y}_i$$

A challenge in doing this is that the appropriate scaling depends on the characteristics of the particular graph. Suppose the coarse graph roughly halves the number of vertices in the fine graph. If the fine graph is, for example, a three-dimensional cube arrangement of vertices with 6-neighbours, then the expansion ratio needed will be  $\approx 2^{1/3}$  or about 1.26; a two-dimensional square arrangement of vertices needs an expansion ratio of  $\approx \sqrt{2}$  or about 1.41. Since the graph is dynamic, the best expansion ratio can also change over time. Moreover, the optimal amount of scaling might be different for each axis, and there might be differences in how the fine and coarse graph are oriented in space.

Such considerations led us to consider affine transformations from the coarse to fine graph. We use projections of the form

$$\mathbf{x}_i = \delta_i + \alpha\mathbf{y}_j + \beta \tag{5}$$

where  $\alpha$  is a linear transformation (a 3x3 matrix) and  $\beta$  is a translation. The variables  $(\alpha, \beta)$  are themselves given dynamics, so that the projection converges to a least-squares fit of the coarse graph to the fine graph.

---

<sup>3</sup> The alternative to ‘ $\mathbf{y}_i$ ’ would be to write e.g.  $\mathbf{x}_{c(i)}^{l+1}$ , meaning the vertex in level  $l + 1$  corresponding to vertex  $i$  in level  $l$ .



### 3.2.1 Frame dynamics

We summarize here the derivation of the time evolution equations for the affine transformation  $(\alpha, \beta)$ . Conceptually, we think of the displacements  $\delta_i$  as “pulling” on the transformation: if all the displacements are to the right, the transformation will evolve to shift the coarse graph to the right; if they are all outward, the transformation will expand the projection of the coarse graph, and so forth. In this way the finer graph ‘pulls’ the projection of the coarse graph around it as tightly as possible.

We derive the equations for  $\ddot{\alpha}$  and  $\ddot{\beta}$  using Lagrangian dynamics. To simplify the derivation we pretend that both graph layouts are stationary, and that the displacements  $\delta_i$  behave like springs between the fine graph and the projected coarse graph, acting on  $\alpha$  and  $\beta$  via ‘generalized forces.’ By setting up appropriate potential and kinetic energy terms, the Euler-Lagrange equations yield:

$$\ddot{\alpha} = \frac{1}{n} \sum_i \delta_i \mathbf{y}_i^T + \mathbf{y}_i \delta_i^T \quad (6)$$

$$\ddot{\beta} = \frac{1}{n} \sum_i \delta_i \quad (7)$$

To damp oscillations and dissipate energy we introduce damping terms of  $-d_\alpha \dot{\alpha}$  and  $-d_\beta \dot{\beta}$ .

### 3.2.2 Time dilation

We now turn to the equations of motion for vertices in the two-level dynamics. The equations for  $\dot{\delta}_i$  and  $\ddot{\delta}_i$  are obtained by differentiating Eqn (5):

$$\mathbf{x}_i = \delta_i + \underbrace{\beta + \alpha \mathbf{y}_i}_{\text{proj. position}} \quad (8)$$

$$\dot{\mathbf{x}}_i = \dot{\delta}_i + \underbrace{\dot{\beta} + \dot{\alpha} \mathbf{y}_i + \alpha \dot{\mathbf{y}}_i}_{\text{proj. velocity}} \quad (9)$$

$$\ddot{\mathbf{x}}_i = \ddot{\delta}_i + \underbrace{\ddot{\beta} + \ddot{\alpha} \mathbf{y}_i + 2\dot{\alpha} \dot{\mathbf{y}}_i + \alpha \ddot{\mathbf{y}}_i}_{\text{proj. acceleration}} \quad (10)$$

Let  $\mathbf{F}_i$  be the forces acting on the vertex  $\mathbf{x}_i$ . Substituting Eqn (10) into  $\ddot{\mathbf{x}}_i = \mathbf{F}_i$  and rearranging, one obtains an equation of motion for the displacement:

$$\ddot{\delta}_i = \mathbf{F}_i - \underbrace{\left( \ddot{\beta} + \ddot{\alpha}\mathbf{y}_i + 2\dot{\alpha}\dot{\mathbf{y}}_i + \alpha\ddot{\mathbf{y}}_i \right)}_{\text{proj. acceleration}} \quad (11)$$

The projected acceleration of the coarse vertex can be interpreted as a ‘pseudoforce’ causing the vertex to react against motions of its coarser version. If Eqn (11) were used as the equation of motion, the coarse and fine graph would evolve independently, with no interaction. (We have used this useful property to check correctness of some aspects of our system.)

The challenge, then, is how to adjust Eqn (11) in some meaningful way to couple the finer and coarse graph dynamics. Our solution is based on the idea that the coarser graph layout evolves similarly to the finer graph, *but on a different time scale*: the coarse graph generally converges much more quickly. To achieve a good fit between the coarse and fine graph we might slow down the evolution of the coarse graph. Conceptually, we try to do the opposite, speeding up evolution of the fine graph to achieve a better fit. Rewriting Eqn (8) to make each variable an explicit function of time, and incorporating a time dilation, we obtain

$$\mathbf{x}_i(t) = \delta_i(t) + \underbrace{\beta(t) + \alpha(t)\mathbf{y}_i(\phi t)}_{\text{proj. position}} \quad (12)$$

where  $\phi$  is a time dilation factor to account for the differing time scales of the coarse and fine graph. Carrying this through to the acceleration equation yields the equation of motion

$$\ddot{\delta}_i = \mathbf{F}_i - \underbrace{\left( \ddot{\beta} + \ddot{\alpha}\mathbf{y}_i + 2\dot{\alpha}\phi\dot{\mathbf{y}}_i + \alpha\phi^2\ddot{\mathbf{y}}_i \right)}_{\text{proj. acceleration}} \quad (13)$$

If for example the coarser graph layout converged at a rate twice that of the finer graph, we might take  $\phi = \frac{1}{2}$ , with the effect that we would discount the projected acceleration  $\ddot{\mathbf{y}}_i$  by a factor of  $\phi^2 = \frac{1}{4}$ . In practice we have used values of  $0.1 \leq \phi \leq 0.25$ . Applied across multiple levels of coarse graphs, we call this approach *multilevel time dilation*.

In addition to the spring and repulsion forces in  $\mathbf{F}_i$ , we include a drag term  $\mathbf{F}_i^d = -d\dot{\delta}_i$  in the forces of Eqn (13).

### 3.3 Multilevel dynamics

To handle multiple levels of coarse graphs, we iterate the two-level dynamics. The dynamics simulation simultaneously integrates the following equations:

- The equations of motion for the vertices in the coarsest graph, using the single-level dynamics of Section 3.1.
- The equations for the projection  $\alpha, \beta$  between each coarser and finer graph pair (Section 3.2.1).
- The equations of motion  $\ddot{\delta}_i$  for the displacements of vertices in the finer graphs, using the two-level dynamics of Section 3.2.

In our implementation, the equations are integrated using an explicit, fourth-order Runge-Kutta method. (We also have a simple Euler-step method, which is fast, but not as reliably stable.)

### 3.4 Equilibrium positions of the multilevel dynamics

We prove here that a layout found using the multilevel dynamics is an equilibrium position of the potential energy function of Eqn (1). This establishes that the multilevel approach does not introduce spurious minima, and can be expected to converge to the same layouts as a single-level layout, only faster.

**Theorem 3.1.** *Let  $(X, \dot{X})$  be an equilibrium position of the two-level dynamics, where  $X = (\delta_1, \delta_2, \dots, \alpha, \beta, y_1, y_2, \dots)$ , and  $\ddot{X} = \dot{X} = 0$ . Then,  $(x_1, x_2, \dots, x_n)$  is an equilibrium position of the single-level dynamics, where  $x_i = \delta_i + \alpha y_i + \beta$ , and the single-level potential gradient  $\nabla V = 0$  (Eqn (1)) vanishes there.*

*Proof.* Since  $\dot{X} = 0$ , the drag terms vanish from all equations of motion. Substituting  $\dot{X} = 0$  and  $\ddot{X} = 0$  into Eqn (13) yields  $F_i = 0$  for each vertex. Now consider the single-level dynamics (Section 3.1) using the positions  $x_i$  obtained from  $x_i = \delta_i + \alpha y_i + \beta$  (Eqn (5)). From  $\ddot{x}_i = F_i$  we have  $\ddot{x}_i = 0$  for each  $i$ . The Euler-Lagrange equations for the single level layout are (Eqn (3)):

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{x}_i} - \frac{\partial}{\partial x_i} L = 0$$

Since  $\frac{d}{dt} \frac{\partial L}{\partial \dot{\mathbf{x}}_i} = \ddot{\mathbf{x}}_i = 0$ , we have  $-\frac{\partial}{\partial \mathbf{x}_i} L = 0$ . Using  $L = T - V$  and that the kinetic energy  $T$  does not depend on  $\mathbf{x}_i$ , we obtain

$$\frac{\partial}{\partial \mathbf{x}_i} V = 0$$

for each  $i$ . Therefore  $\nabla V = 0$  at this point.  $\square$

This result can be applied inductively over pairs of finer-coarser graphs, so that Theorem 3.1 holds also for multilevel dynamics.

## 4 Dynamic coarsening

As vertices and edges are added to and removed from the base graph  $G = (V, E)$ , our system dynamically maintains the coarser graphs  $G_1, G_2, \dots, G_m$ . Each vertex in a coarse graph may correspond to several vertices in the base graph, which is to say, each coarse graph defines a partition of the vertices in the base graph. It is useful to describe coarse vertices as subsets of  $V$ . For convenience we define a finest graph  $G_0$  isomorphic to  $G$ , with vertices  $V_0 = \{\{v\} : v \in V\}$ , and edges  $E_0 = \{(\{v_1\}, \{v_2\}) : (v_1, v_2) \in E\}$ . Coarser graphs are obtained by merging vertices via set union. The sequence  $V_0, V_1, \dots, V_m$  of coarse graph vertices forms a chain in the lattice of partitions of  $V$ ; we need to maintain this chain as changes are made to the base graph  $V_0$ .

We have devised an algorithm that efficiently maintains  $G_{i+1}$  in response to changes in  $G_i$ . By applying this algorithm at each level the entire chain  $G_1, G_2, \dots, G_m$  is maintained.

We present a fully dynamic, Las Vegas-style randomized graph algorithm for maintaining a coarsened version of a graph. For graphs of bounded degree, this algorithm requires  $O(1)$  operations on average per edge insertion or removal.

Our algorithm is based on the traditional matching approach to coarsening developed by Hendrickson and Leland [HL95]. Recall that a matching of a graph  $G = (V, E)$  is a subset of edges  $M \subseteq E$  satisfying the restriction that no two edges of  $M$  share a common vertex. A matching is *maximal* if it is not properly contained in another matching. A maximal matching can be found by considering edges one at a time and adding them if they do not conflict with an edge already in  $M$ . (The problem of finding a maximal matching should not be confused with that of finding a *maximum cardinality* matching, a more difficult problem.)

## 4.1 Dynamically maintaining the matching

We begin by making the matching unique. We do this by fixing a total order  $<$  on the edges, chosen uniformly at random. (In practice, we compute  $<$  using a bijective hash function.) To produce a matching we can consider each edge in ascending order by  $<$ , adding it to the matching if it does not conflict with a previously matched edge. If  $e_1 < e_2$ , we say that  $e_1$  has priority over  $e_2$  for matching.

Our basic analysis tool is the *edge graph*  $G^* = (E, S)$  whose vertices are the edges of  $G$ , and  $e_1 S e_2$  when the edges share a vertex. A set of edges  $M$  is a matching on  $G$  if and only if  $M$  is an independent set of vertices in  $G^*$ . From  $G^*$  we can define an *edge dependence graph*  $\mathcal{E} = (E, \rightarrow)$  which is a directed version of  $G^*$ :

$$e_1 \rightarrow e_2 \equiv (e_1 < e_2) \text{ and } e_1 S e_2 \text{ (share a common vertex)}$$

Since  $<$  is a total order, the edge dependence graph  $\mathcal{E}$  is acyclic. Figure 2 shows an example.

Building a matching by considering the edges in order of  $<$  is equivalent to a simple rule:  $e$  is matched if and only if there is no edge  $e' \in M$  such that  $e' \rightarrow e$ . We can express this rule as a set of *match equations* whose solution can be maintained by simple change propagation. Let  $m : E \rightarrow \{\perp, \top\}$  indicate whether an edge is matched:  $m(e) = \top$  if and only if  $e \in M$ . The match equation for an edge  $e \in E$  is:

$$m(e) = \bigwedge_{e' : e' \rightarrow e} \neg m(e')$$

where by convention  $\bigwedge \emptyset = \top$ .

To evaluate the match equations we place the edges to be considered for matching in a priority queue ordered by  $<$ , so that highest priority edges are considered first. The match equations can then be evaluated using a straightforward change propagation algorithm:

While the priority queue is not empty:

1. Retrieve the highest priority edge  $e = (v_1, v_2)$  from the queue and evaluate its match equation  $m(e)$ .
2. If the solution of the match equation has changed, then:
  - (a) If  $m(e) = \top$ , then  $\text{match}(v_1, v_2)$ .
  - (b) If  $m(e) = \perp$ , then  $\text{unmatch}(v_1, v_2)$ .

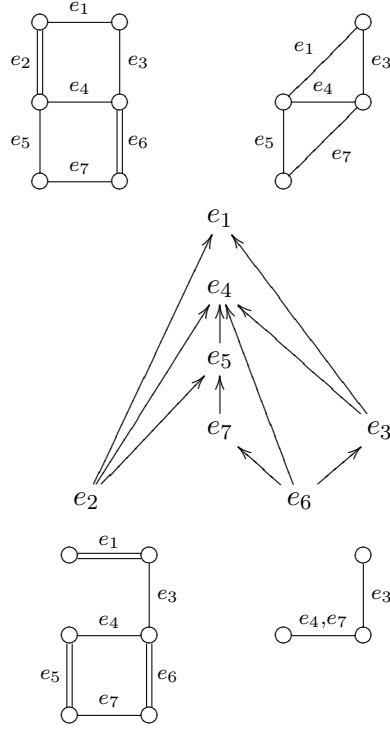


Figure 2: Illustration of dynamic coarsening. Top left: A graph with double lines indicating matched edges; Top right: its coarsened version, obtained by contracting matched edges; Middle: the edge dependence graph for the initial graph, using the (arbitrarily chosen) order  $e_6 < e_2 < e_3 < e_7 < e_5 < e_1 < e_4$ . Initially the matching is  $\{e_2, e_6\}$ . Removing edge  $e_2$  (bottom left) triggers re-evaluation of the match equations for  $e_5$ ,  $e_1$ , and  $e_4$ . The new matching is  $\{e_6, e_5, e_1\}$ , resulting in changes to the coarsened graph (bottom right).

Both  $\text{match}(e)$  and  $\text{unmatch}(e)$  add the dependent edges of  $e$  to the queue, so that changes ripple through the graph.

Figure 3 summarizes the basic steps required to maintain the coarser graph  $(V', E')$  as edges and vertices are added and removed to the finer graph.

## 4.2 Complexity of the dynamic matching

The following theorem establishes that for graphs of bounded degree, the expected cost of dynamically maintaining the coarsened graph is  $O(1)$  per edge inserted or removed in the fine graph. The cost does not depend on the number of edges in the graph.

**Theorem 4.1.** *In a graph of maximum degree  $d$ , the randomized complexity of  $\text{addEdge}(e)$  and  $\text{removeEdge}(e)$  is  $O(de^{2d})$ .*

We first prove a lemma concerning the extent to which updates may need to propagate in the edge dependence graph. As usual for randomized algorithms, we analyze the complexity using “worst-case average time,” i.e., the maximum (with respect to choices of edge to add or remove) of the expected time (with expectation taken over random priority assignments). For reasons that will become clear we define the priority order  $<$  by assigning to each edge a random real in  $[0, 1]$ , with 1 being the highest priority.

**Lemma 4.2.** *Let  $G^* = (E, S)$  be a graph and  $\rho : E \rightarrow [0, 1]$  an injective function chosen uniformly at random assigning to each vertex a real priority in  $[0, 1]$ . If  $G^*$  has maximum degree  $k$ , the expected number of vertices reachable from any vertex in  $E$  following only edges from higher to lower priority vertices is  $O(e^k)$ .*

*Proof.* It is helpful to view the priority assignment  $\rho$  as inducing a linear arrangement of the vertices, i.e., we might draw  $G^*$  by placing its vertices on the real line at their priorities. We obtain a directed graph  $(E, \rightarrow)$  by making edges always point toward zero, i.e., from higher to lower priorities (cf. Figure 4). Note that vertices with low priorities will tend to have high indegree and low outdegree.

We write  $\mathbb{E}[\cdot]$  for expectation with respect to the random priorities  $\rho$ . For  $e \in E$ , let  $N(e)$  be the number of vertices of  $G^*$  reachable from  $e$  by

---

### DYNAMIC-MATCH operations

We use  $\bar{v}$  to indicate the coarse version of a vertex  $v$ .

- **addEdge( $e$ ):** Increase the count of  $(\bar{v}_1, \bar{v}_2)$  in  $E'$  (possibly adding edge if not already present). Add  $e$  to the queue.
  - **deleteEdge( $e$ ):** If  $e$  is in the matching then  $\text{unmatch}(e)$ . Decrease the count of  $(\bar{v}_1, \bar{v}_2)$  in  $E'$ ; if this count is 0 then delete this edge from  $E'$ . If either  $v_1$  or  $v_2$  have no edges except  $e$ , then remove  $\bar{v}_1$  (resp.  $\bar{v}_2$ ) from the coarse graph. Add all edges  $e'$  with  $e \rightarrow e'$  to the queue.
  - **deleteVertex( $v$ ):** for each edge  $e$  incident on  $v$ ,  $\text{deleteEdge}(e)$ . Remove  $v$  from  $G$ .
  - **addVertex( $v$ ):** no action needed.
  - **match( $e$ ), where  $e = (v_1, v_2)$ :** For each edge  $e'$  where  $e \rightarrow e'$ , if  $e'$  is matched then  $\text{unmatch}(e')$ . Delete vertices  $\bar{v}_1$  and  $\bar{v}_2$  from the coarser graph. Create a new vertex  $v_1 \cup v_2$  in  $G'$ . For all edges  $e = (v, v')$  in  $G$  incident on  $v_1$  or  $v_2$  (but not both), add a corresponding edge to or from  $v_1 \cup v_2$  in  $G'$ . For each  $e'$  such that  $e \rightarrow e'$ , add  $e'$  to the queue.
  - **unmatch( $e$ ), where  $e = (v_1, v_2)$ :** delete any edges in  $G'$  incident on  $v_1 \cup v_2$ . Delete the vertex  $v_1 \cup v_2$  from  $G'$ . Add new vertices  $\bar{v}_1$  and  $\bar{v}_2$  to  $G'$ . For each edge incident on  $v_1$  or  $v_2$  in  $G$ , add a corresponding edge to  $G'$ . For each  $e'$  such that  $e \rightarrow e'$ , add  $e'$  to the queue.
- 

Figure 3: Basic operations of the dynamic coarsening algorithm.

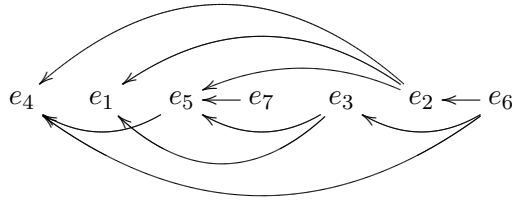


Figure 4: A linear arrangement of the edge dependence graph of Figure 2.



following paths that move from higher to lower priority vertices. We bound the expected value of  $N(e)$  given its priority  $\rho(e) = \eta$ : we can always reach  $e$  from itself, and we can follow any edges to lower priorities:

$$\mathbb{E}[N(e) \mid \rho(e) = \eta] \leq 1 + \sum_{e' : eSe'} \underbrace{\Pr(\rho(e') < \eta)}_{=\eta} \cdot \mathbb{E}[N(e') \mid \rho(e') < \eta] \quad (14)$$

Let  $f(\eta) = \sup_{e \in E} \mathbb{E}[N(e) \mid \rho(e) = \eta]$ . Then,

$$\mathbb{E}[N(e') \mid \rho(e') < \eta] \leq \int_0^\eta \eta^{-1} f(\alpha) d\alpha \quad (15)$$

where the integration averages  $f$  over a uniform distribution on priorities  $[0, \eta]$ . Since the degree of any vertex is  $\leq k$ , there can be at most  $k$  terms in the summation of Eqn (14). Combining the above, we obtain

$$f(\eta) = \sup_{e \in E} \mathbb{E}[N(e) \mid \rho(e) = \eta] \quad (16)$$

$$\leq \sup_{e \in E} \left( 1 + \sum_{e' : Se} \eta \mathbb{E}[N(e') \mid \rho(e') < \eta] \right) \quad (17)$$

$$\leq 1 + k\eta \int_0^\eta \eta^{-1} f(\alpha) d\alpha \quad (18)$$

$$\leq 1 + k \int_0^\eta f(\alpha) d\alpha \quad (19)$$

Therefore  $f(\eta) \leq g(\eta)$ , where  $g$  is the solution to the integral equation

$$g(\eta) = 1 + k \int_0^\eta g(\alpha) d\alpha \quad (20)$$

Isolating the integral and differentiating yields the ODE  $g(\eta) = k^{-1}g'(\eta)$ , which has the solution  $g(\eta) = e^{\eta k}$ , using the boundary condition  $g(0) = 1$  obtained from Eqn (20). Since  $0 \leq \eta \leq 1$ ,  $g(\eta) \leq e^k$ . Therefore, for every  $e \in E$ , the number of reachable vertices satisfies  $\mathbb{E}[N(e)] \leq e^k$ .  $\square$

Note that the upper bound of  $O(e^k)$  vertices reachable depends only on the maximum degree, and not on the size of the graph.

We now prove Theorem 4.1.

*Proof.* If a graph  $G = (V, E)$  has maximum degree  $d$ , its edge graph  $G^*$  has maximum degree  $2(d-1)$ . Inserting or removing an edge will cause us to reconsider the matching of  $\leq e^{2(d-1)}$  edges on averages by Lemma 4.2.

If a max heap is used to implement the priority queue,  $O(de^{2d})$  operations are needed to insert and remove these edges. Therefore the randomized complexity is  $O(de^{2d})$ .  $\square$

In future work we hope to extend our analysis to show that the entire sequence of coarse graphs  $G_1, G_2, \dots, G_m$  can be efficiently maintained. In practice, iterating the algorithm described here appears to work very well.

## 5 Implementation

Our system is implemented in C++, using OpenGL and pthreads. The graph animator runs in a separate thread from the user threads. The basic API is simple, with methods `newVertex()` and `newEdge( $v_1, v_2$ )` handling vertex and edge creation, and destructors handling their removal.

For static graphs, we have so far successfully used up to six levels of coarsening, with the coarsened graphs computed in advance. With more than six levels we are encountering numerical stability problems that seem to be related to the projection dynamics.

For dynamic graphs we have used three levels (the base graph plus two coarser versions), with the third-level graph being maintained from the actions of the dynamic coarsener for the first-level graph. At four levels we encounter a subtle bug in our dynamic coarsening implementation we have not yet resolved.

### 5.1 Parallelization

Our single-level dynamics implementation is parallelized. Each frame entails two expensive operations: rendering and force calculations. We use the Barnes-Hut tree to divide the force calculations evenly among the worker threads; this results in good locality of reference, since vertices that interact through edge forces or near-field repulsions are often handled by the same thread. Rendering is performed in a separate thread, with time step  $t$  being rendered while step  $t + \delta t$  is being computed. The accompanying animations were rendered on an 8-core (2x4) iMac using OpenGL, compiled with g++ at -O3.

Our multilevel dynamics engine is not yet parallelized, so the accompanying demonstrations of this are rendered on a single core. Parallelizing the multilevel dynamics engine remains for future work.

## 6 Applications

We include with this paper two demonstrations of applications:

- The emergence of the giant component in a random graph: In Erdős-Renyi  $G(n, p)$  random graphs on  $n$  vertices where each edge is present independently with probability  $p$ , there are a number of interesting phase transitions: when  $p < n^{-1}$  the largest connected component is almost surely of size  $\Theta(\log n)$ ; when  $p = n^{-1}$  it is a.s. of size  $\Theta(n^{2/3})$ , and when  $p > n^{-1}$  it is a.s. of size  $\Theta(n)$ —the “giant component.” In this demonstration a large random graph is constructed by preassigning to all  $\binom{n}{2}$  edges a probability trigger in  $[0, 1]$ , and then slowly raising a probability parameter  $p(t)$  from 0 to 1 as the simulation progresses, with edges ‘turning on’ when their trigger is exceeded.
- Visualization of insertions of random elements into a binary tree, with an increasingly rapid rate of insertions.

In addition, we mention that the graph visualizer was of great use in debugging itself, particularly in tracking down errors in the dynamic matching implementation.

## 7 Conclusions

We have described a novel approach to real-time visualization of dynamic graphs. Our approach combines the benefits of multilevel force-directed graph layout with the ability to render rapidly changing graphs in real time. We have also contributed a novel and efficient method for dynamically maintaining coarser versions of a graph.

## References

- [BH86] BARNES J. E., HUT P.: A hierarchical  $O(N \log N)$  force-calculation algorithm. *Nature* 324, 6270 (1986), 446–449.
- [Ead84] EADES P.: A heuristic for graph drawing. *Congressus Numerantium* 42 (1984), 149–160.
- [EGK\*04] ELLSON J., GANSNER E. R., KOUTSOFIOS E., NORTH S. C., WOODHULL G.: Graphviz and dynagraph – static and dynamic

- pgraph drawing tools. In
- Graph Drawing Software*
- , Jünger M., Mutzel P., (Eds.), Mathematics and Visualization. Springer, 2004, pp. 127–148.
- [FE02] FRIEDRICH C., EADES P.: Graph drawing in motion. *J. Graph Algorithms Appl* 6, 3 (2002), 353–370.
  - [FLM94] FRICK A., LUDWIG A., MEHLDAU H.: A fast adaptive layout algorithm for undirected graphs. In *Graph Drawing* (1994), Tamassia R., Tollis I. G., (Eds.), vol. 894 of *Lecture Notes in Computer Science*, Springer, pp. 388–403.
  - [FR91] FRUCHTERMAN T. M. J., REINGOLD E. M.: Graph drawing by force-directed placement. *Softw. Pract. Exper.* 21, 11 (1991), 1129–1164.
  - [HEW98] HUANG M. L., EADES P., WANG J.: On-line animated visualization of huge graphs using a modified spring algorithm. *J. Visual Languages and Computing* 9, 6 (Dec. 1998), 623–645.
  - [HL95] HENDRICKSON B., LELAND R.: An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing* 16, 2 (Mar. 1995), 452–469.
  - [HMM00] HERMAN, MELANÇON G., MARSHALL M. S.: Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics* 6, 1 (Jan./Mar. 2000), 24–43.
  - [KCH02] KOREN Y., CARMEL L., HAREL D.: ACE: A fast multiscale eigenvectors computation for drawing huge graphs. In *INFOVIS '02: Proceedings of the IEEE Symposium on Information Visualization (InfoVis'02)* (Washington, DC, USA, 2002), IEEE Computer Society, p. 137.
  - [KK89] KAMADA T., KAWAI S.: An algorithm for drawing general undirected graphs. *Inf. Process. Lett.* 31, 1 (1989), 7–15.
  - [NW01] NORTH S. C., WOODHULL G.: Online hierarchical graph drawing. In *Graph Drawing* (2001), Mutzel P., Jünger M., Leipert S., (Eds.), vol. 2265 of *Lecture Notes in Computer Science*, Springer, pp. 232–246.

- [STT81] SUGIYAMA K., TAGAWA S., TODA M.: Methods for visual understanding of hierarchical systems. *IEEE Transactions on Systems, Man, and Cybernetics* 11, 4 (1981), 109–125.
- [Wal03] WALSHAW C.: A multilevel algorithm for force-directed graph drawing. *J. Graph Algorithms Appl.* 7, 3 (2003), 253–285.