The Marey Graph Animation Tool Demo

Carsten Friedrich and Peter Eades

The University of Newcastle
University Drive
Callaghan, NSW 2308
{friedric,eades}@cs.newcastle.edu.au Australia

Abstract Enabling the user of a graph drawing system to preserve the mental map between two different layouts of a graph is a major problem. In this paper we present Marey, a system that can smoothly transform one drawing of a graph into another without any restrictions to the class of graphs or type of layout algorithm.[1]

1 Introduction

Graphs are a common way to communicate information. In many applications these graphs are not static but change their structure and layout according to user and application actions. Preserving the mental map during these changes has been identified to be crucial for the usability of a system [2]. There are two possible approaches to this problem. Either develop graph drawing algorithms that try to minimize changes [3] or to communicate the changes in the form of an animation [8,6,7], i.e. a smooth transition from the old drawing to the new drawing.

While specialized algorithms work quite well in practice, general animation techniques tend to fail to actually improve usability in many situations. Figure 1 on page 397 shows an example of a bad animation[2].

In this paper we introduce Marey, a system which implements a variety of different animation techniques which can be used to display the changes in the graph structure or the graph layout. Furthermore we try to identify criteria for a good animation and measures to evaluate the quality of an animation.

We try to achieve two goals with our system. First we want to use the tool in further studies and experiments to better understand the properties of good animation. And secondly, we want to provide a set of tools in the form of a Java package to the graph drawing community which can be used to create applications using graph animations not restricted to specific layout algorithms or classes of graphs.

[1] This work was supported by DSTO Australia

[2] All examples, in the form of mpeg videos, and a free mpeg player for MS Windows NT as well as references to free players for Unix are available from `http://www.cs.newcastle.edu.au/~friedric/gd00/`

J. Marks (Ed.): GD 2000, LNCS 1984, pp. 396–406, 2001.

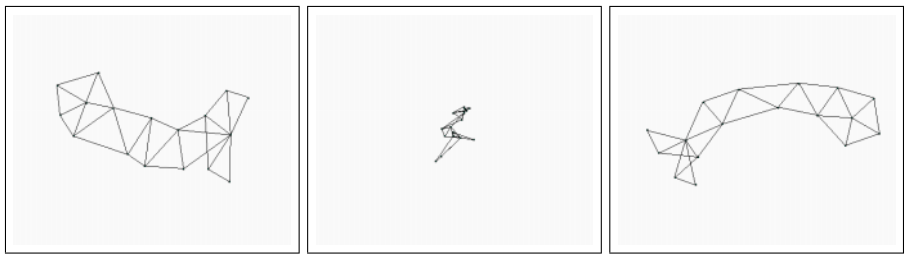

Figure 1. Example of a bad animation. The nodes from the drawing on the left are moved to their new position in the drawing on the right using a linear interpolation. The drawing in the middle is a snapshot of the animation where nodes lie very close to each other. Individual node movements are difficult to follow at this stage. The mpeg file is `http://www.cs.newcastle.edu.au/~friedric/gd00/a.mpg`

2 Model for a Good Animation

An animation is a sequence of images. This sequence is characterized by subtle but highly structured changes between consecutive images over space and over time. The changes are perceived as movement of the corresponding objects in the image by the human brain. A detailed analysis of these mechanisms is beyond the scope of this paper. For an introduction to human perception of moving pictures see e.g. [1].

In the case of a graph animation, the images are drawings of graphs. The changes in the drawings are changes in the positions of the nodes and edges.

The animation should help the user to maintain the mental map of a changing graph. Major changes to the drawing of a graph usually occur when the user applies a layout algorithm which provides a different view of the graph or when the structure of the graph changes in a way which makes it necessary to recompute the layout. Examples for structural changes in graphs are collapsing or expanding sub graphs in clustered graphs, navigation in infinite graphs[3] or graphs such as graph A of the 1999 Graph Drawing Contest [4] which represents the changes in the cast of a soap opera.

2.1 General Goals

The following general goals have been identified for a good animation between two graph layouts.

1. Preserve the mental map
2. Communicate the structural changes in the graph

Apart from preserving the mental map between two drawings, in the case where the new layout was triggered by changes to the graph structure it is important to communicate that the graph changed and what these changes were to the user.

[3] E.g. www-based graphs

2.2 Criteria for a Good Animation

The following aesthetic criteria have been identified to characterize a good animation.

The movements of nodes and edges should be easy to follow.

The movements of the graph should be structured. The more structure the user can identify in the movements, the easier it is to preserve the mental map of the graph. E.g. if nodes of a sub-graphs do not change their positions in relation to each other, the quality of the animation increases significantly if these nodes move in a uniform way.

The transition from source to destination should be smooth. The movements should be performed in small steps and adequately fast to help the human brain to perceive and interpret the movement.

Displaying non-existing structures should be avoided. An often neglected problem in graph drawings is the case where the drawing displays some structure which does not exist in the graph [12]. Figure 2 on page 398 shows an example for two layouts of the same graph, where the second layout could lead the user wrongly to assume that the graph is a simple path. Similar problems can occur easily during an animation, as the human brain tends to be quite imaginative when it tries to interpret moving images [11].

`http://www.cs.newcastle.edu.au/~friedric/gd00/d.mpg` shows an example for an animation which wrongly suggests the transformation to be the rotation of a three-dimensional graph.

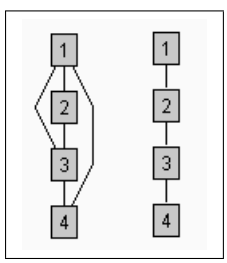

Figure 2. Example of a misleading layout

The individual drawings should satisfy aesthetic criteria. such as minimizing edge crossings, maximizing the smallest angle, etc; see [2] for details.

Some of these criteria are NP hard to optimize, e.g. crossing minimization, and often not all criteria can be achieved in one solution. E.g. it might be necessary to move nodes along a non-optimal path to avoid edge crossings, or it might be preferable to accept some edge crossings instead of watching the graph untangle in a complex and confusing way.

2.3 Measures for a Good Animation

To be able to actually evaluate how well an animation matches our criteria we derived the following possible measures.

Minimize temporary edge crossings. Edge crossings in a graph drawing generally reduce readability. It seems that this is also valid for animations. If the animation avoids introducing unnecessary edge crossings on the way then it is easier for the user to follow the movements.

Maintain a minimal distance between nodes which do not move uniformly. If nodes lie close to each other, it is more difficult to follow their individual movements than if they are further apart. Of course, if two nodes lie next to each other in the source drawing and in the target drawing, it would be better to move them uniformly close to each other to their destination than to separate them first.

Maximize symmetries in movement. Symmetry in a drawing helps the user to understand the structure of a graph. In an animation symmetry of movement makes it easier to understand the structure of the movement. A formal measure for symmetric node movement can be derived by extending the model in [9].

Minimize the length of the path of a node. A node which moves on a straight line between its source and destination clearly moves a minimal distance. However some criteria may prevent straight line movement. In such cases we require a minimum distance movement to help the user to follow and anticipate the node movement.

Provide smooth transitions and adequate speed. Smooth transitions and an adequate speed are obvious criteria for a good animation. If the transition steps are to big the user is no longer able to perceive a movement. If the speed is too slow the user will become impatient and stop following the animation. If it is too fast the user will not be able to keep track of the nodes and comprehend the movements.

Formal metrics can be derived from the above. E.g. it is possible to count the number of temporary edges or compute the length of the node paths.

3 Architecture of Marey

As our tool was implemented using Java, the architecture of the system was developed following object oriented design patterns.

The graph itself, the layout algorithms, the graph modifying algorithms, and the animation engine are implemented as separate modules. They exchange information and trigger actions using a defined API.

In a typical use case the user invokes some action which results in major changes in the graph drawing, e.g. applying a layout algorithm or changing the graph structure followed by a layout algorithm. Figure 3 on page 400 shows the data-flow for these cases.

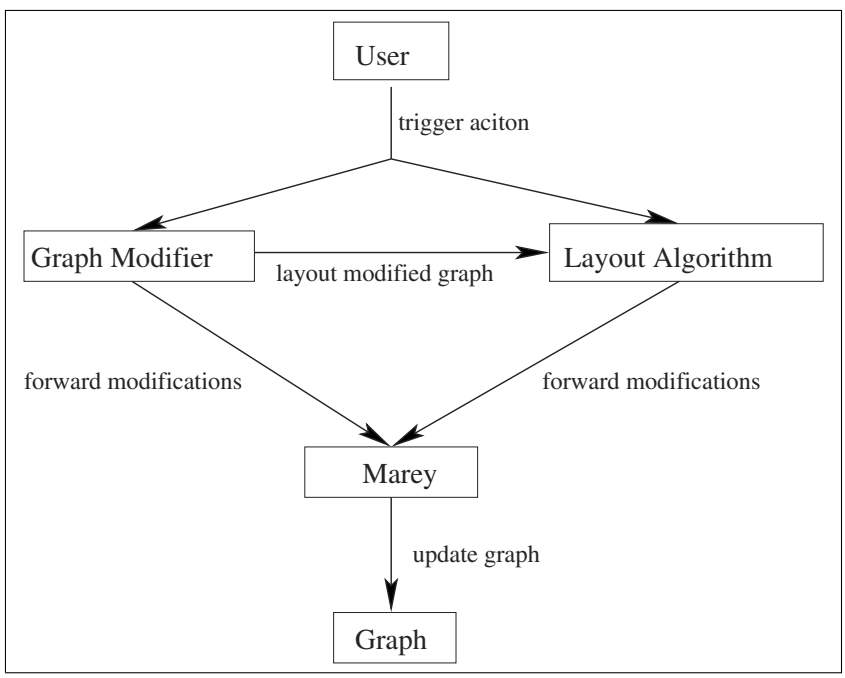

Figure 3. Architecture

Note that there is usually no need for the user to directly access the animation module. In our prototype the user is able to access the Marey module to take snapshots of graphs and generate an animation between them. Figure 4 on page 401 shows the control panel of the Marey module.

A special Java class is used to store the changes which should be animated[4].

Modules can either provide instances of that class to the animation module or tell the animation module to take a snapshot of the current state of the graph and compute the differences to the last snapshot.

[4] Currently only node movements and visibility of nodes and edges can be animated. The system is easily extensible and more features are currently being developed

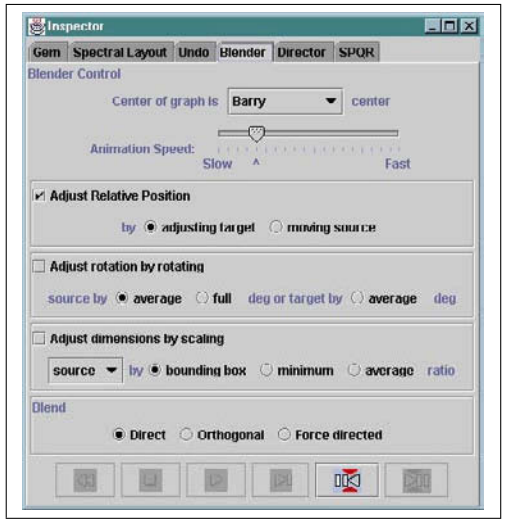

Figure 4. Control Panel

3.1 The Animation Process

When the animation module has enough information in the form of snapshots or change-objects the animation itself can be invoked. The animation between two states consists of six steps.

1. Hide vanishing nodes and edges
2. Adjust absolute positions (optional).
3. Rotate graph (optional).
4. Scale graph (optional).
5. Move nodes to their final position
6. Show newly visible elements

1. Hide vanishing nodes. It is difficult for an animation engine to animate the removal of graph elements, as they are usually no longer existent in the graph when the animation is generated. Our proposed work around for the modifying module is to only hide the elements, generate the animation and after the animation remove them from the graph. As these nodes do not have a destination, hiding happens at the beginning of the animation process.

They can simply be hidden after the first step, or slowly fade out.

The basic idea behind the next three steps is to minimize unstructured movement during the animation. We try to break up the animation into a uniform transformation, a uniform scaling, a uniform rotation to all elements, and a final individual movement for each node. The three uniform steps can always be easily performed without destroying the mental map and thereby minimize the movements in the more critical individual movements.

2. Adjust absolute positions. To minimize node movements during the animation the layouts are adjusted using a uniform translation of all nodes to reduce distances between old and new positions.

Either the barycenter or the center of the bounding boxes of the graphs is computed. The layouts are then transformed to overlap the two centers.

There are two possible approaches to this step. In most cases the absolute position of a graph is irrelevant. In this case time can be saved by just moving the new layout towards the old layout and not animate this operation at all.

Alternatively, as there are cases where the absolute position of a graph actually matters, the old layout can be moved towards the new layout in an animation.

Furthermore, in the case where only a few nodes move between two layouts, it might be preferable not to adjust the absolute positions at all as this would cause all nodes to move during the animation. Therefore this step is optional.

3. Rotate graph. We compute the average rotation of all nodes around the barycenter or the center of the bounding box of the graph, depending on the last step. We then rotate the graph at this angle.

The time needed to display the rotation may be relatively long in a simple animation. The extra time can out-weight the improvement achieved by the rotation in this case. This step is therefore optional.

If the orientation of the target drawing does not matter the target drawing can be rotated to match the orientation of the source drawing without animation.

4. Scaling. There are different approaches to compute a scaling. The graph can be scaled so that the bounding boxes of the two layouts have the same dimensions or it could be scaled by the average, minimal or maximal scaling factor over all nodes.

Again, there are cases where scaling does not improve the animation and this step is therefore optional.

If the size of the target drawing does not matter, the target drawing can be resized to match the source drawing without animation.

5. Move nodes to their final positions. For the final movements several algorithms are available. The easiest approach is to move the nodes the remaining way to their new positions on a linear path. Alternatively these movements can be restricted to certain general directions at a time or broken up in uniform movements, e.g. first move to the x-positions and then to the y-positions. The advantage of these approaches is that the movements can be computed very fast. The disadvantage is that none of our criteria for a good animation is enforced.

However more sophisticated approaches are more appropriate in some cases. We have developed an adaptation of the force directed layout approach described in [5] to move the nodes to their final positions. The repulsive forces are similar to the static version, whereas instead of attracting edges, nodes are attracted

to their destination. This approach provides a minimal distance between nodes at all times in most cases and thereby increase traceability. Problems with this approach are to update the forces fast enough to be able to provide a fast and smooth animation and secondly to guarantee an efficient movement of the nodes to their final destination.

A modification of [3] might be able to avoid new edge crossings at the same time.

Other approaches are imaginable, and are currently being developed, which try to minimize our measures for good animation directly.

Show newly visible elements. The last step adds graph elements to the drawing which did not exist or were invisible at the start of the animation. As for the first step this can be done within one step or using a slower fade-in.

4 Examples

The following examples show typical use cases of the animation module. The black and white printed pictures in this section try to capture the animation process. Of course this is only possible to a certain degree. All examples are available as mpeg videos on the following web site:

`http://www.cs.newcastle.edu.au/~friedric/gd00`

Note that some freely available mpeg display programs do not support the features necessary to play the movies. References to free and working mpeg-players for Windows NT and Unix are provided on this page.

4.1 Applying a New Layout

The first example shows the animation between two force directed layouts. The structure of the drawing does not change much apart from a rotation of the graph of about 180 degree. A direct movement of the nodes from the source to the target would result in a confusing animation as shown in figure 1. Figure 5 shows how Marey breaks up the movement into a rotation and a final movement of the nodes.

The second example shows the transition from an hierarchical type layout to an force directed layout of a graph. Snapshots of the animation are shown in figure 6.

4.2 Sub-graph Expansion

The example in figure 7 shows a clustered graph. The center node, which represents a group of nodes is expanded and a new layout is computed using a force directed algorithm. The animation shows the change from the old layout to the new layout.

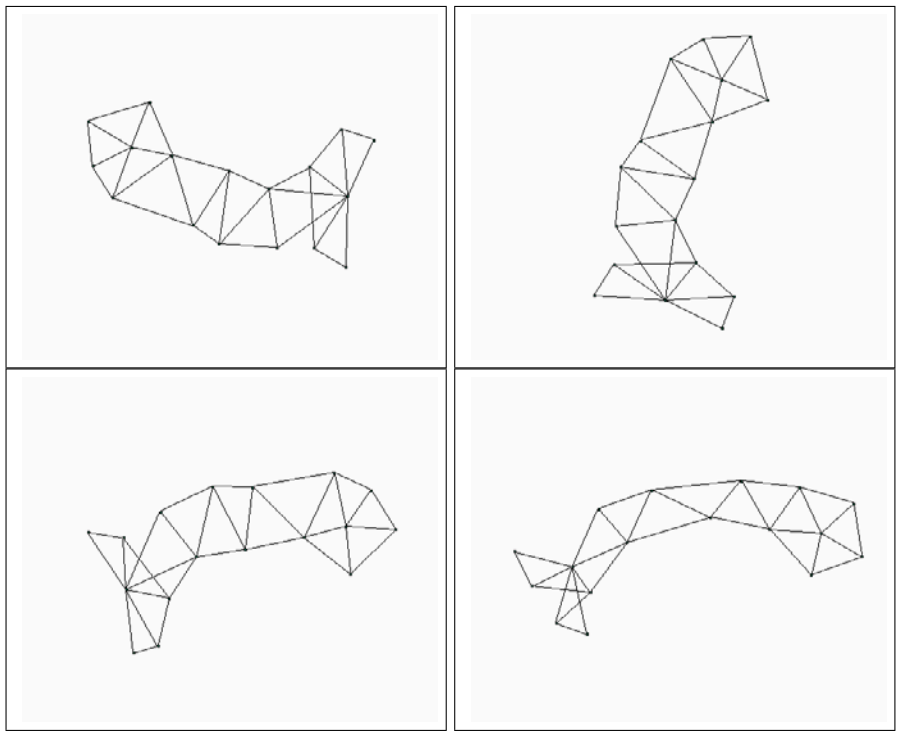

Figure 5. The drawing in the upper left corner is the initial drawing, the drawing in the lower right corner is the target drawing. The animation is broken into a rotation of the graph and a subsequent movement of the node to their final position. The mpeg file is `http://www.cs.newcastle.edu.au/~friedric/gd00/b.mpg`

5 Status and Future Plans

The system currently implements a basic set of animation techniques and further algorithms are being developed. The tool is scheduled to be released as a freely available Java package in the second half of 2000. A prototype is currently integrated into the InVision framework[10].

To gain a deeper understanding of the criteria for a good animation user experiments are planned for the near future. We hope the results of these experiments will enable us to develop new and better animation techniques and algorithms to be integrated into our tool.

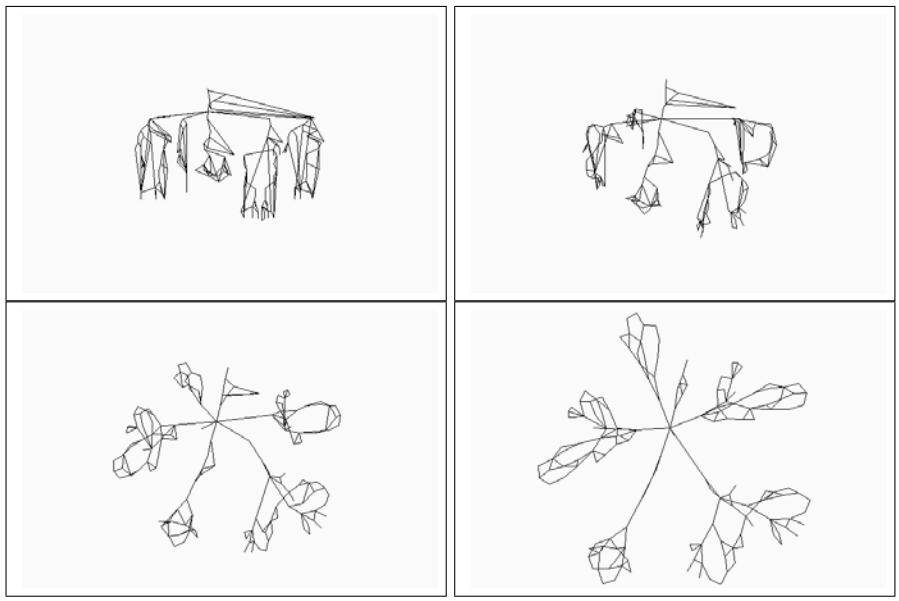

Figure 6. Morphing between a hierarchical layout and a force directed layout using a linear interpolated path for the nodes. The mpeg file is `http://www.cs.newcastle.edu.au/~friedric/gd00/c.mpg`

References

1. Edward Adelson. Mechanisms for motion perception. *Optics and Photonics News*, August:24–30, 1991.
2. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph drawing: algorithms for the visualization of graphs.* Prentice-Hall Inc., 1999.
3. F. Bertault. A force-directed algorithm that preserves edge-crossing properties. *Information Processing Letters*, 74(1–2):7–13, 2000.
4. Franz J. Brandenburg, Michael Jünger, Joe Marks, Petra Mutzel, and Falk Schreiber. Graph-drawing contest report. In *Proc. of the 7th Internat. Symposium on Graph Drawing (GD'99)*, pages 400–409, 1999.
5. Frick, Ludwig, and Mehldau. A fast adaptive layout algorithm for undirected graphs. In *Proc. of the DIMACS International Workshop on Graph Drawing (GD'94)*, 1994.
6. C. Friedrich. The ffGraph library. Technical Report 9520, Universität Passau, Dezember 1995.
7. http://www.mpi sb.mpg.de/AGD/. *The AGD-Library User Manual Version 1.1.2.* Max-Planck-Institut für Informatik.
8. Mao Lin Huang and Peter Eades. A fully animated interactive system for clustering and navigating huge graphs. In Sue H. Whitesides, editor, *Proc. of the 6th Internat. Symposium on Graph Drawing (GD'98)*, pages 374–383, 1998.
9. J. Manning. *Geometric Symmetry in Graphs.* PhD thesis, Purdue University, Department of Computer Sciences, 1990.

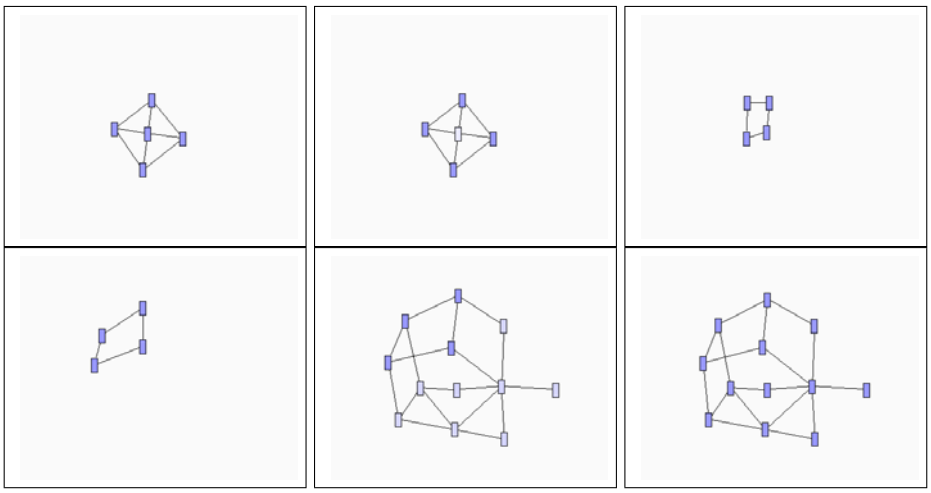

Figure 7. Sub-graph expansion. The node in the middle of the first image is expanded to a sub-graph. The sequence shows how the node is faded out, how the remaining nodes are moved to their new position, and how the new nodes are faded in. The mpeg file is `http://www.cs.newcastle.edu.au/~friedric/gd00/expand.mpg`

10. T.R. Pattison, R.J. Vernik, D.P.J. Goodburn, and M.P. Phillips. Rapid assembly and deployment of domain visualisation solutions. In *Submitted to IEEE Visualisation 2000*, 2000.
11. Robert Sekuler and Randolph Blake. *Perception.* McGraw-Hill Publishing company, 1990.
12. Richard J. Webber. *Finding the Best Viewpoints for three-dimensional graph drawings.* PhD thesis, University of Newcastle (Australia), 1998.