Journal of Graph Algorithms and Applications

http://jgaa.info/

vol. 6, no. 3, pp. 353–370 (2002)

Graph Drawing in Motion

Carsten Friedrich *Peter Eades*

Basser Department of Computer Science
The University of Sydney
Australia
http://www.it.usyd.edu.au/
carsten@it.usyd.edu.au peter@it.usyd.edu.au

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 methods that smoothly transform one drawing of a graph into another without any restrictions to the class of graphs or type of layout algorithm.

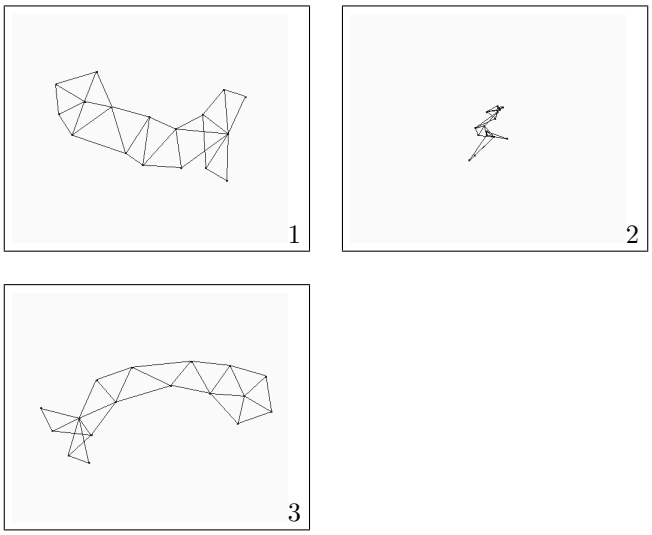
Communicated by M. Kaufmann: submitted February 2001; revised February 2002.

Research supported in part by DSTO Australia.

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 [7, 9, 10], that is, 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 shows an example of a bad animation[1].



http://www.it.usyd.edu.au/~carsten/gd00/a.mpg

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.

We introduce requirements for animations of graph drawings and methods which partially satisfy these requirements. These techniques have been implemented in the Marey system which will also be briefly introduced.

[1] All examples, in the form of mpeg videos, and a free MPEG player for Windows-NT/95/98 as well as references to free players for Unix are available from http://www.it.usyd.edu.au/~carsten/gd00/.

2 Model for a good animation

An animation is a sequence of images called *frames*. This sequence is characterized by subtle but highly structured changes between consecutive frames 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 perceptional mechanisms is beyond the scope of this paper. For an introduction to human perception of moving pictures see, for example, [1, 5, 15].

In the case of a graph animation, the frames 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 of structural changes in graphs are collapsing or expanding sub graphs in clustered graphs, navigation in infinite graphs[2] 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 Criteria for a good animation

We try to achieve the following goals with animation between two graph layouts.

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

   Firstly the system needs to tell the user that the graph has changed. Secondly, the system needs to indicate the nature of the change.

The following aesthetic criteria have been identified to be critical to achieve these goals.

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

2.1.2 The movements of the graph should be structured.

The human brain contains highly optimized mechanisms for recognizing and interpreting certain special kinds of movements. These movements include

- Uniform movement.

  If the relative position of nodes in the initial frame is similar to those in the final frame, then it should be so in the intermediate frames.

- Symmetrical movement.

  As with static images the brain shows a predisposition for conceiving symmetry in movement.

[2] For example www-based graphs

- Two-dimensional projections of movements of three-dimensional rigid objects.

  Humans can see three-dimensional movement in a two-dimensional moving image. Although humans are able to perceive three-dimensional images directly using stereo vision, the loss of the third dimension hardly poses any problems for us in everyday live. Even in situations where we only have access to a two-dimensional projection of the world around us, for example if using only one eye, or on a TV or computer screen, we have no problems identifying three-dimensional objects and their movement in all three dimensions. For a detailed discussion of these effects see [14].

The more we can exploit these mechanisms for which the brain is optimized, the easier it is for the user to preserve the mental map of the graph.

2.1.3 The transition from source to destination should be smooth.

The movements should be performed in small steps and adequately quickly to help the human brain to perceive and interpret the movement.

2.1.4 Display of non-existing structures should be avoided.

An often neglected problem in graph drawing is the case where the drawing suggests some structure which does not exist in the graph [17]. Figure 2 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 [14].

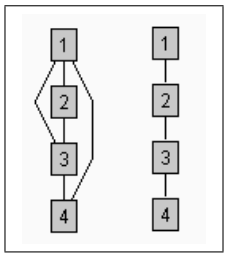


Figure 2: Example of a misleading layout

2.1.5 The individual frames should satisfy the classical aesthetic criteria for good graph drawings

such as minimizing edge crossings, maximizing the smallest angle, etc; see [2] for details.

2.2 Measures for a good animation

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

2.2.1 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.

2.2.2 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.

2.2.3 Maximize structured movements.

- Maximize uniform movement

  Maximizing uniform movement can be done by measuring how much the relative positions of a nodes change from the initial to the final frame.

- Maximize symmetry

  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 [11].

- Maximize movement interpreted as movement of a rigid object

  The human is very good at interpreting two dimensional projections of movements of three dimensional rigid objects in $\mathbb{R}^3$. We should therefore try to do as much of the animation as possible in such a way. A detailed description of how to do this follows below.

2.2.4 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.

2.2.5 Provide smooth transitions and adequate speed.

Smooth transitions and an adequate speed are obvious criteria for a good animation. If the transition steps are too big then 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. For example it is possible to count the number of temporary edges or compute the length of the node paths.

Formalizations of some of these metrics are NP hard to optimize and often not all criteria can be achieved in one solution. For example 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.

3 The animation process

The animation process we propose here splits the animation into two main stages.

In the first stage we try compute an animation which moves the graph as close as possible to its destination in a way that is perceived as the motion of a rigid three-dimensional object in space.

We then move the nodes the rest of the way in a direct linear interpolation.

In addition we also animate changes in the structure of the graph and changes to graphical attributes like color or visibility.

To make the animation process as independent as possible to a specific application we only require a very small and general set of information to be available from the embedding system. This information consists of the coordinates of the nodes in the first and final frame, how graphical attributes such as color and visibility change, as well as how the structure of the graph changes.

The animation consists of the following steps.

1. Hide vanishing nodes and edges
2. The rigid motion stage
3. The linear interpolation stage
4. Show newly visible elements

3.1 Hide vanishing nodes and edges

Nodes and edges which do not exist in the final frame because they were either deleted or became invisible are hidden at the beginning. Various effects, for example fade out, are used to do this.

3.2 The rigid motion stage

In this stage we move the nodes and edges as close as possible to the final frame using only movements which are perceived as if the graph was moving as a rigid object in three-dimensional space.

3.2.1 Computing the rigid transformation.

Projecting tree-dimensional movements of rigid objects on two dimensions is identical to applying the operations translation, rotation, scaling and shearing in $\mathbb{R}^2$ to the projection of the object. We interpret a shearing angle α of more than 90 degree as a flip followed by a shear of $\alpha - 90$ degree.

This set of operations is precisely the set of two dimensional affine linear transformations.

As these movements are very easy to understand for humans we try to do as much of the total animation as possible in this stage.

We use linear regression to find the best affine linear transformation of the graph between the first and final frame. This can be done in linear time as follows.

Given a graph G and the coordinates (x_v, y_v) for each node $v \in G$ the linear transformation has the general form

$$f(v) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_v \\ y_v \end{pmatrix} + \begin{pmatrix} a_{13} \\ a_{23} \end{pmatrix}$$

In order to determine the linear function which transforms the graph as close as possible to the final frame we have to determine the a_{ij} which minimize the distance between the positions in the final frame and the positions after the linear transformation. We use the average the error between the individual node distances as a good approach in respect to the goal of preserving the mental map; that is we minimize the sum of the squared Euclidean distances.

Given (x'_v, y'_v) as the original coordinates of a node v we aim to minimize the error function

$$e = \sum_{v \in G} |f(v) - (x'_v, y'_v)|$$

with $||$ being the squared Euclidean norm, that is

$$e = \sum_{v \in G} (a_{11} * x_v + a_{12} * y_v + a_{13} - x'_v)^2 + (a_{21} * x_v + a_{22} * y_v + a_{23} - y'_v)^2 \tag{1}$$

Since (1) is always non negative, minimizing this quadratic function is done by deriving with respect to the a_{ij} and setting the resulting expressions to equal 0. We solve the system of equations for the a_{ij}. Assuming at least 3 non co-linear pairs of node coordinates this will give us a unique solution for each a_{ij}:

Let

$$denom = (\sum_{v\in G} x_v^2) * (n * \sum_{v\in G} y_v^2 - (\sum_{v\in G} y_v)^2) - n * (\sum_{v\in G} x_v * y_v)^2 + (\sum_{v\in G} x_v) *$$

$$(2 * (\sum_{v\in G} y_v) * \sum_{v\in G} x_v * y_v - (\sum_{v\in G} x_v) * \sum_{v\in G} y_v^2).$$

Then

$$\begin{aligned}
a_{11} &= -(\frac{(\sum_{v\in G} x_v * y_v) * (n * \sum_{v\in G} x'_v * y_v - (\sum_{v\in G} x'_v) * \sum_{v\in G} y_v)}{denom} + \\
&\quad \frac{(\sum_{v\in G} x'_v * x_v) * ((\sum_{v\in G} y_v)^2 - n * \sum_{v\in G} y_v^2)}{denom} + \\
&\quad \frac{(\sum_{v\in G} x_v) * ((\sum_{v\in G} x'_v) * \sum_{v\in G} y_v^2 - (\sum_{v\in G} y_v) * \sum_{v\in G} x'_v * y_v)}{denom}), \\
a_{12} &= \frac{(\sum_{v\in G} x_v^2) * (n * \sum_{v\in G} x'_v * y_v - (\sum_{v\in G} x'_v) * \sum_{v\in G} y_v)}{denom} + \\
&\quad \frac{(\sum_{v\in G} x_v * y_v) * ((\sum_{v\in G} x'_v) * \sum_{v\in G} x_v - n * \sum_{v\in G} x'_v * x_v)}{denom} + \\
&\quad \frac{(\sum_{v\in G} x_v) * ((\sum_{v\in G} y_v) * \sum_{v\in G} x'_v * x_v - (\sum_{v\in G} x_v) * \sum_{v\in G} x'_v * y_v)}{denom}, \\
a_{13} &= \frac{(\sum_{v\in G} x_v^2) * ((\sum_{v\in G} x'_v) * \sum_{v\in G} y_v^2 - (\sum_{v\in G} y_v) * \sum_{v\in G} x'_v * y_v)}{denom} - \\
&\quad \frac{(\sum_{v\in G} x'_v) * (\sum_{v\in G} x_v * y_v)^2}{denom} + \\
&\quad \frac{(\sum_{v\in G} x_v * y_v) * ((\sum_{v\in G} y_v) * \sum_{v\in G} x'_v * x_v + (\sum_{v\in G} x_v) * \sum_{v\in G} x'_v * y_v)}{denom} - \\
&\quad \frac{(\sum_{v\in G} x_v) * (\sum_{v\in G} y_v^2) * \sum_{v\in G} x'_v * x_v}{denom}.
\end{aligned} \tag{2}$$

We can obtain a_{21}, a_{22}, and a_{23} from the solutions for a_{11}, a_{12}, and a_{13} respectively by replacing x'_v with y'_v.

3.2.2 Quality of the transformation

The sum of the Euclidean squares is known to be statistically biased and to over-emphasize outlying samples. Other distance functions are much harder to minimize on the other hand [8, 13]. To increase statistic stability different approaches are imaginable. The median or centroid of the drawings could be added to the node set with a weight factor. Another promising approach would be to compute different transformations using random subsets of the nodes and using the best one.

3.2.3 Animating the affine linear transformation

After computing the affine linear transformation we have to generate an animation for the graph. The easiest approach would be to do a linear interpolation of each matrix entry from the identity matrix to the computed matrix. However the result would not produce the desired effect of being perceived as the movement of a rigid object in all cases.

This can easily be seen in the case of a 180 degree rotation of the graph. The transformation matrix would be

$$tm = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

interpolating the entries from

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

to tm will produce the matrix

$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

at some stage collapsing the graph to one point. This does not happen in a 180 degree rotation.

Shoemake and Duff [16] show that the rotation part of the transformation is the only part not compatible with linear entry interpolation and propose polar matrix decomposition as an efficient way to separate the rotational part from the transformation. For a 2×2 matrix[3] the decomposition takes constant time.

We can now generate an animation by interpolating the rotation over the angle and at the same time do a linear interpolation of the matrix entries of the non-rotational part.

3.2.4 Adjusting the center of rotation

Using the approach described above still has a small problem. The rotation we compute is always done in respect to the coordinates $(0, 0)$, generating a smaller or bigger arc depending on how the graph lies to the origin. To avoid this we modify our linear transformation before the decomposition to move the center of the graph to the origin at the beginning and back to the former position afterwards. This can easily done by multiplying the corresponding transformation matrices to each side of our matrix. The graph will now rotate around its center. Several definitions of center are possible, for example the center of the bounding box, or the barycenter. In our experience the decision of which definition to use had little influence on the overall animation. Computing the center takes linear time.

[3] For this purpose we can ignore the translation part

3.2.5 Complexity

The following computations have do be done:

- The a_{ij} in (2) can be computed in one iteration over all nodes and is therefore linear in time.
- The polar decomposition is done in constant time.
- Adjusting the center of rotation takes again linear time to compute the center.

All computations have to be done once at the start of the animation. The overall cost of this stage is therefore linear.

3.3 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 into uniform movements. For example, we can first move to the x-positions and then to the y-positions. The advantage of these simple 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 [6] 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 increases 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.

3.4 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

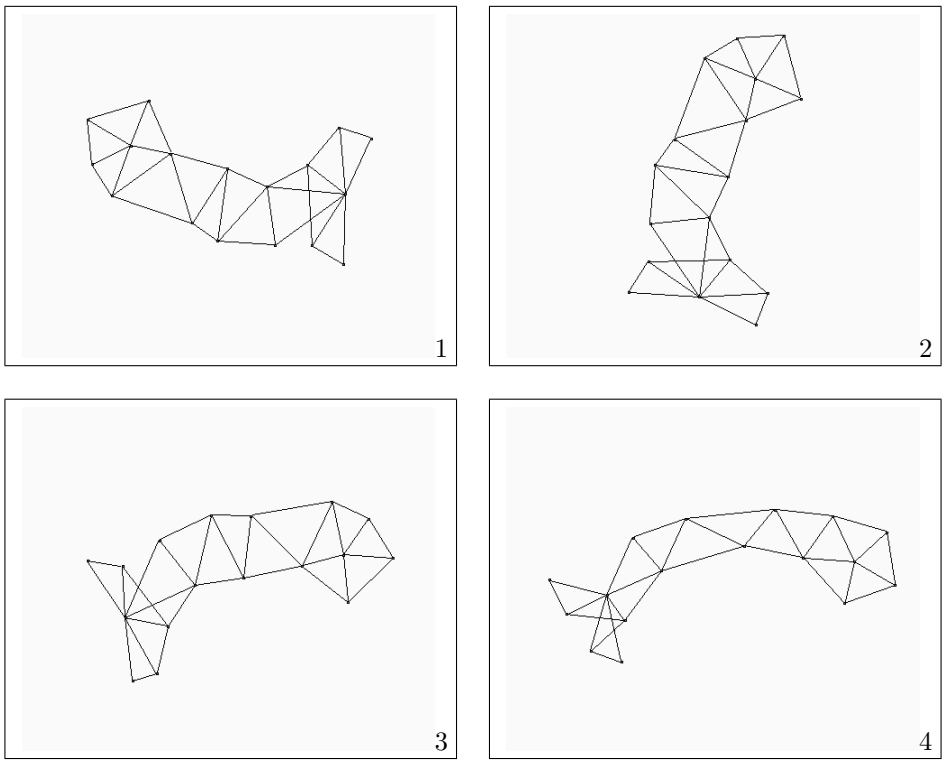
The following examples show typical use cases of the animation module. The static pictures in this section try to capture the animation process. Of course this is only possible to a certain degree. All examples are available as mpeg videos on the following web site:

http://www.it.usyd.edu.au/~carsten/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

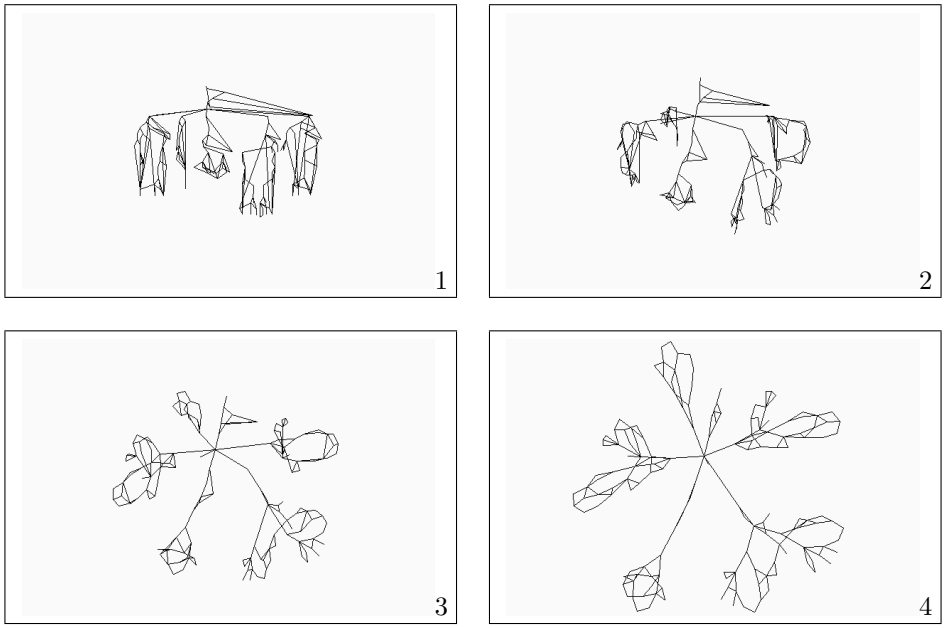
The first example shows the animation between two force directed layouts. The structure of the drawing does not change much apart from a rotation of the graph of about 180 degree. A direct movement of the nodes from the source to the target would result in a confusing animation as shown in figure 1. Figure 3 shows how algorithm breaks up the movement into a rotation and a final movement of the nodes.



http://www.it.usyd.edu.au/~carsten/gd00/b.mpg

Figure 3: The order of the frames is upper left corner, upper right corner, lower left corner, lower right corner. The animation is broken into a rotation of the graph and a subsequent movement of the node to their final position.

The second example shows the transition from a hierarchical type layout to an force directed layout of a graph. Snapshots of the animation are shown in figure 4.



http://www.it.usyd.edu.au/~carsten/gd00/c.mpg

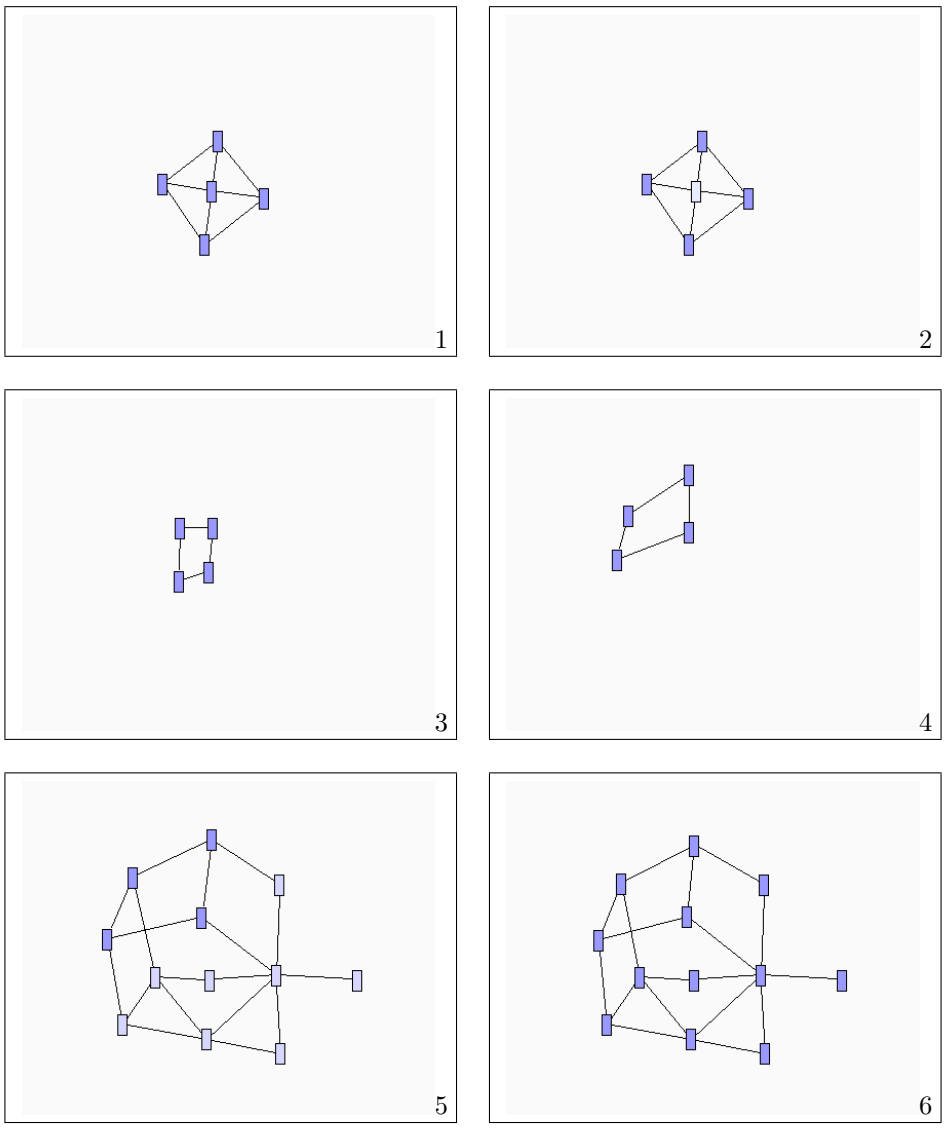
Figure 4: The order of the frames is upper left corner, upper right corner, lower left corner, lower right corner. Morphing between a hierarchical layout and a force directed layout using a linear interpolated path for the nodes.

4.2 Sub-graph expansion

The example in figure 5 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.

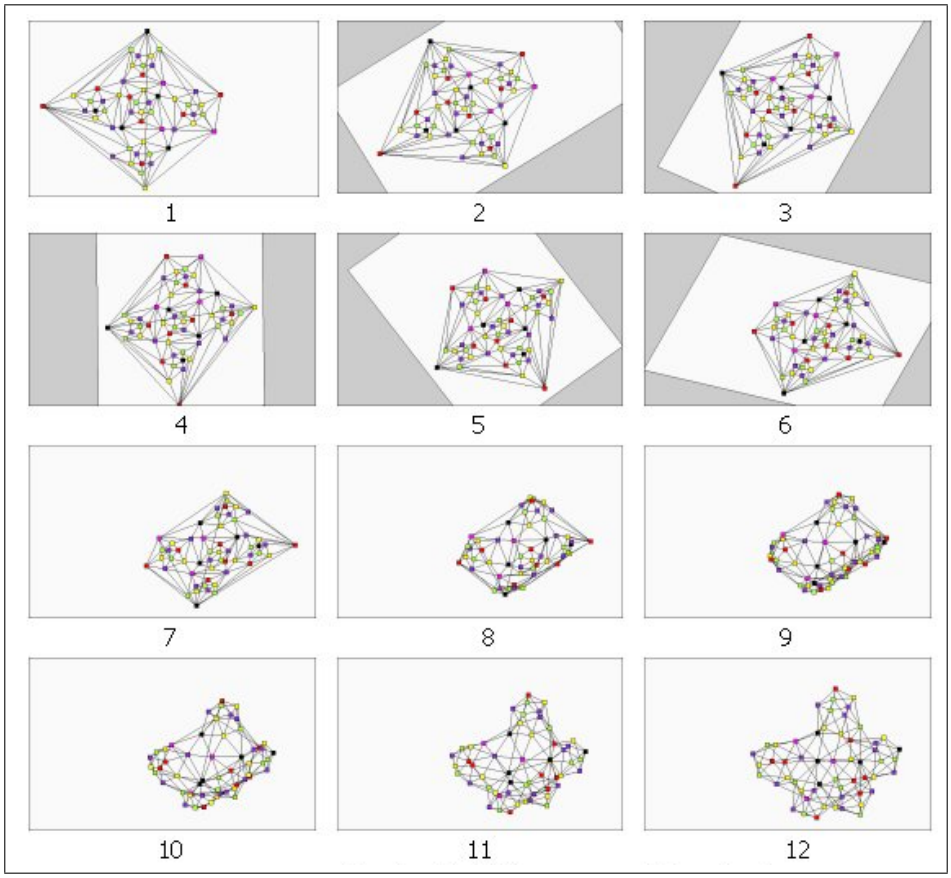
4.3 Adding visual clues

The example in figure 6 shows an animation with added visual effects. The modification of the canvas increases the impression of a movement of a rigid object. After the rigid motion stage the nodes are moved to their final position in the new layout using linear interpolation.



http://www.it.usyd.edu.au/~carsten/gd00/expand.mpg

Figure 5: Sub-graph expansion. The order of the frames is upper row from left to right, then lower row from left to right. 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.



http://www.it.usyd.edu.au/~carsten/gd00/clue.mpg

Figure 6: Adding visual effects can increase the impression of the graph moving as a rigid object in space.

5 Architecture of Marey

The methods described above were implemented as a Java package of the name Marey and integrated in our experimental graph drawing tool JJGraph. 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, for example applying a layout algorithm or changing the graph structure followed by a layout algorithm. Figure 7 shows the data-flow for these cases.

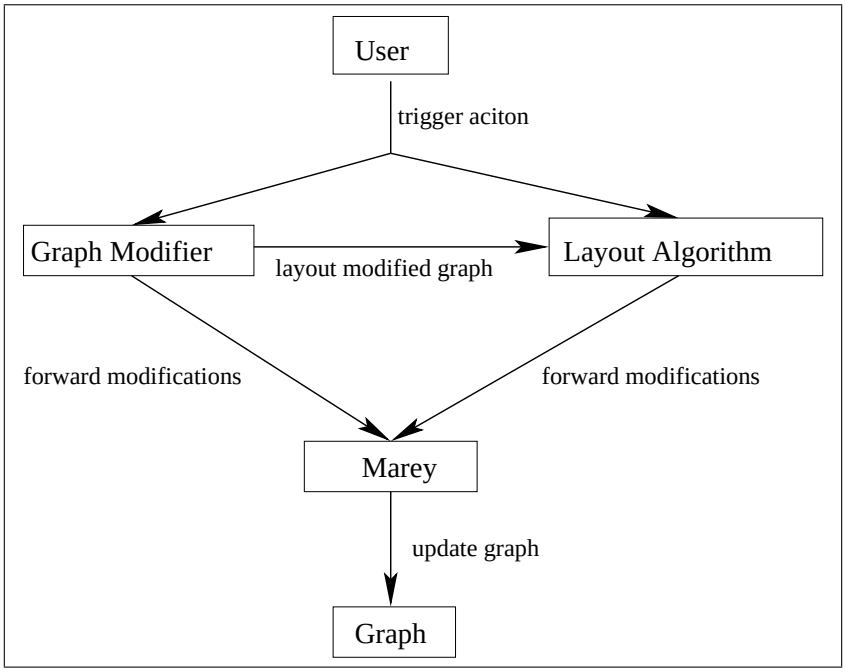


Figure 7: Architecture

Note that there is typically 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 8 on the next page shows the control panel of the Marey module.

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

[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

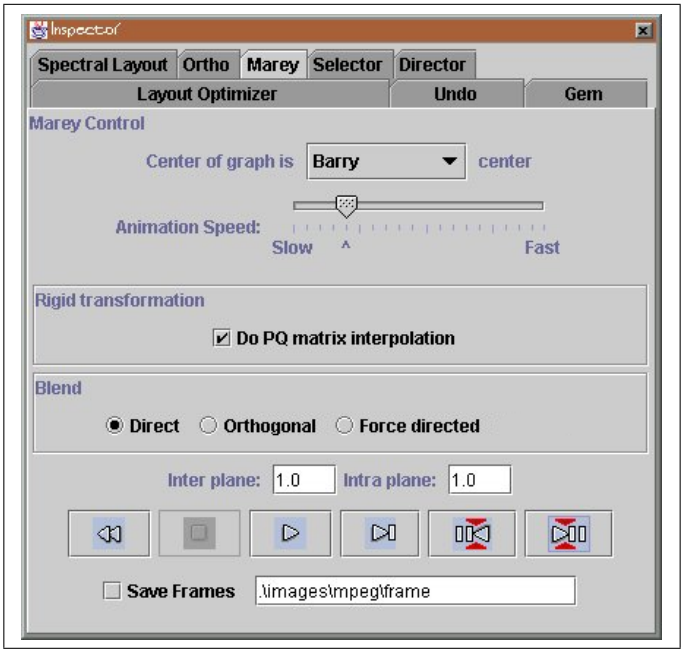


Figure 8: Control Panel

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.

6 Future Plans

Our main focus currently is on developing more sophisticated methods for the non-structured part of the animation, incorporate mechanisms for handling edges with bends, especially the case where the number of bends varies between the initial and the final frame, as well as trying to identify and animate non-linear structured movements.

At the same time we try to extend the Marey system to incorporate these new developments and finally release it as a publicly available Java package. A prototype is currently integrated into the InVision framework[12].

References

[1] Edward Adelson. Mechanisms for motion perception. *Optics and Photonics News*, August:24–30, 1991. 2

[2] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph drawing: algorithms for the visualization of graphs*. Prentice-Hall Inc., 1999. 1, 2.1.5

[3] F. Bertault. A force-directed algorithm that preserves edge-crossing properties. *Information Processing Letters*, 74(1–2):7–13, 2000. 1

[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. 2

[5] Roger N. Shepard Eloise H. Carlton. Psychologically simple motions and geodesic paths. *Journal of Mathematical Psychology*, 34(2):127–228, 1990. 2

[6] 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. 3.3

[7] C. Friedrich. The ffGraph library. Technical Report 9520, Universität Passau, Dezember 1995. 1

[8] F.R. Hampel, E.M. Ronchetti, P.J. Rousseeuw, and W.A. Stahe l. *Robust Statistics: The Approach Based on Influence Functions*. John Willey, NY USA, 1986. 3.2.2

[9] http://www.mpi sb.mpg.de/AGD/. *The AGD-Library User Manual Version 1.1.2*. Max-Planck-Institut für Informatik. 1

[10] Mao Lin Huang and Peter Eades. Navigating clustered graphs using force-dierected methods. *JGAA Special Issue on GD98*, 4(3):157–181, 2000. 1

[11] J. Manning. *Geometric Symmetry in Graphs*. PhD thesis, Purdue University, Department of Computer Sciences, 1990. 2.2.3

[12] 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. 6

[13] P.J. Rousseeuw and A.M. Leroy. *Robust regression and outlier detection*. John Willey, NY USA, 1987. 3.2.2

[14] Robert Sekuler and Randolph Blake. *Perception*. McGraw-Hill Publishing company, 1990. 2.1.2, 2.1.4

[15] Roger N. Shepard. Ecological constraints on internal representation: Resonant kinematics of perceiving, imagining thinking, and dreaming. *Psychological Review*, 91(4):417–447, 1984. 2

[16] Ken Shoemake and Tom Duff. Matrix animation and polar decomposition. In *Proc. of the 1992 Graphics Interface Conference*, pages 245–254, 1992. 3.2.3

[17] Richard J. Webber. *Finding the Best Viewpoints for three-dimensional graph drawings*. PhD thesis, University of Newcastle (Australia), 1998. 2.1.4