Intersection-Free Morphing of Planar Graphs[⋆]

(System Demo)

Cesim Erten[1], Stephen G. Kobourov[1], and Chandan Pitta[2]

[1] Department of Computer Science
University of Arizona
{cesim,kobourov}@cs.arizona.edu
[2] Department of Electrical and Computer Engineering
University of Arizona
chandanp@ece.arizona.edu

Abstract. Given two different drawings of a planar graph we consider the problem of morphing one drawing into the other. We designed and implemented an algorithm for intersection-free morphing of planar graphs. Our algorithm uses a combination of different techniques to achieve smooth transformations: rigid morphing, compatible triangulations, as well as morphing based on interpolation of the convex representations of the graphs. Our algorithm can morph between drawings with straight-line segments, bends, and curves. Our system is implemented in Java and available as an applet at `http://gmorph.cs.arizona.edu`.

1 Introduction

Morphing refers to the process of transforming one shape (the source) into another (the target). Morphing is widely used in computer graphics, animation, and modeling; see a survey by Gomes *et al* [12]. In planar graph morphing we would like to transform a given source graph to another pre-specified target graph. A smooth transformation of one graph into another can be useful for numerous problems from graph drawing [4, 17]. In particular, when dealing with dynamic graphs and graphs that change through time, it is crucial to preserve the mental map of the user. Thus, it is important to minimize the changes to the drawing and to create a smooth transition between consecutive drawings.

In this paper we consider the problem of morphing between two drawings, D_s and D_t, of the same planar graph $G = (V, E)$. We assume that both drawings realize the same embedding of G, have the same outer-face, and are intersection-free. The source drawing D_s and the target drawing D_t can be straight-line drawings, or drawings with bends and curves. The positions of the vertices in the two drawings may be different (as long as the embedding is the same in both). The main objective is to find a morph that preserves planarity throughout the transformation. Secondary objectives include obtaining simple and smooth trajectories for the vertices (and bends) and preserving drawing invariants throughout

[⋆] This work is partially supported by the NSF under grant ACR-0222920.

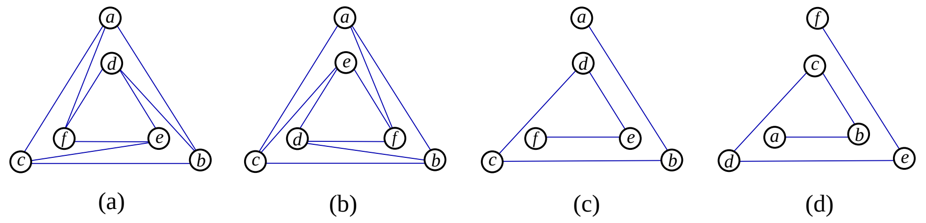


Fig. 1. The drawing from part (a) cannot be morphed into the drawing of part (b) while preserving the edge lengths. In particular, edges $(e, f), (b, d), (c, e)$ will have to shorten and lengthen if we are to preserve planarity. The drawing from part (c) cannot be morphed into the drawing of part (d) using linear trajectories while avoiding crossings.

the transformation. Preservation of drawing invariants refers to the the continuity of the change: for example, a particular edge may shorten throughout the process but it should not shorten and then lengthen repeatedly. Similarly, the morph should avoid shrinking and growing of the graph faces.

We designed and implemented an algorithm for morphing planar graphs which preserves planarity throughout the transformation. It is easy to see that if we want to preserve planarity, some edges may have to lengthen and shorten; see Fig. 1(a-b). Similarly, linear trajectories cannot always be achieved; see Fig. 1(c-d). Thus, our algorithm yields smooth trajectories and preserves edge lengths whenever possible.

2 Previous Work

Morphing has been extensively studied in graphics, animation, modeling and computational geometry, e.g., morphing 2D images [2, 13, 24], polygons and polylines [11, 19–21], 3D objects [14, 16] and free form curves [18].

Graph morphing, refers to the process of transforming a given graph G_1 into another graph G_2. Early work on this problem includes a result by Cairns in 1944 [3] who shows that if G_1 and G_2 are maximally planar graphs with the same embedding, then there exists a non-intersecting morph between them. Later, Thomassen [25] showed that if G_1 and G_2 are isomorphic convex planar graphs with the same outer face, then there exists a non-intersecting morph between them that preserves convexity.

A naive approach to morphing one graph to another is *linear morphing*, where all the vertices move in a straight line at constant velocity from their positions in the source drawing to their final positions in the target drawing [7, 15]. This is the simplest form of morphing but it may result in poor animation as all the trajectories may intersect at a common point, thus shrinking the drawing to a point on the way from the source to the target; see Fig. 2. Another problem with linear morphing is that intermediate graphs may have self-intersections even thought the source and the target are non-crossing; see Fig. 3.

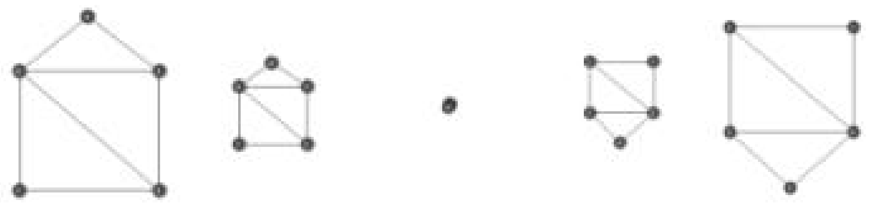


Fig. 2. Linear morphing can result in degenerate intermediate drawings.

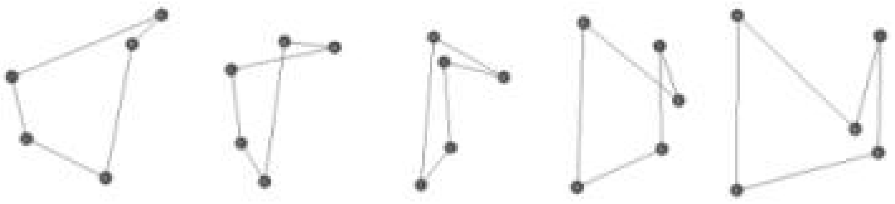


Fig. 3. Linear morphing can create crossings.

Friedrich and Eades [9] present a graph animation technique based on rigid motion and linear interpolation. In the rigid motion stage the trajectories of the vertices are computed by an affine linear transformation. As a result, the source and target vertices are aligned as close as possible. In the linear interpolation stage the vertices travel on straight-line trajectories. While the rigid motion leads to smooth animations, in the interpolation stage, crossings may occur, even if the source and target are intersection-free. Friedrich and Houle [10] modify the algorithm in [9] by clustering groups of nodes that share similar motions in order to create better animations.

Graph morphing is also related to the problem of compatible triangulations. This problem arises when it is necessary to find isomorphic triangulations of two point sets on n vertices, or of two n-sided polygons. Aronov *et al* [1] show that it is always possible to create isomorphic triangulations, provided that $O(n^2)$ additional points (Steiner points) are created. Given two compatible triangulations with the same convex boundaries, Floater and Gotsman [6] and Surazhsky and Gotsman [23] show how to morph between them using convex representation of triangulations using barycentric coordinates, originally described by Tutte in 1963 [26]. A generalization of the same approach is used in [13] for morphing simple planar polygons, while guaranteeing that the intermediate polygons are also simple.

3 Algorithm Overview

We assume that the source drawing D_s and the target drawing D_t are intersection-free, have the same outer-face, and their underlying graphs are isomorphic. If the two drawings are isomorphic but the outer-face is different then there does not

```
Main Algorithm
    1. compute trajectories based on rigid motion
    2. introduce "bend" vertices
    3. compatibly triangulate all faces
    4. compute trajectories based on convex representations
```

Fig. 4. Summary of the algorithm.

exists a transformation that preserves planarity throughout the process. If the graphs are not isomorphic, then nodes and edges that are not in the intersection of the two graphs can be faded in and out as in earlier systems [8, 9].

Our algorithm for intersection-free morphing of planar graphs has four distinct stages. In the first stage the two drawings are aligned using 2-D transformations consisting of translation, rotation, scaling and shearing. That is, we move the source drawing as close as possible to the destination drawing as a rigid object in space. In the second stage we introduce "bend" vertices on all edges with bends. In the case of a curvilinear drawings, we approximate the curves by piecewise linear curves. For every edge in the graph, we ensure that the same number of bend vertices are introduced in both D_s and D_t. In the third stage we identify all faces and compatibly triangulate all corresponding pairs of faces. In this process, we introduce additional triangulation vertices (Steiner points), internal to the faces. There are at most $O(k^2)$ Steiner points, where k is equal to the number of vertices together with the number of bends. In the fourth stage we compute trajectories for all vertices (including the bend vertices) based on convex graph representations and using interpolation of the matrices that represent the two graphs. The four steps of the algorithm are summarized in Fig. 4 and illustrated through an example in Fig. 5.

In the following sections we discuss steps 1, 3 and 4 in detail, leaving out step 2 as it is quite straight-forward.

4 Computing Trajectories Using Rigid Motion

4.1 Affine Matrix of Transformations

The rigid motion in 2-D can be associated with a natural interpolation of four transformations over time: translation, rotation, scaling and shearing. All these transformations can be accommodated by an affine matrix, which can be considered a 2×2 matrix, appended with a translation row:

$$\begin{bmatrix} c_{11} & c_{12} & 0 \\ c_{21} & c_{22} & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Then a point (x, y) which can be represented with the vector $[x\ y\ 1]$ and multiplied on the right by the matrix, is transformed into (x', y') using the

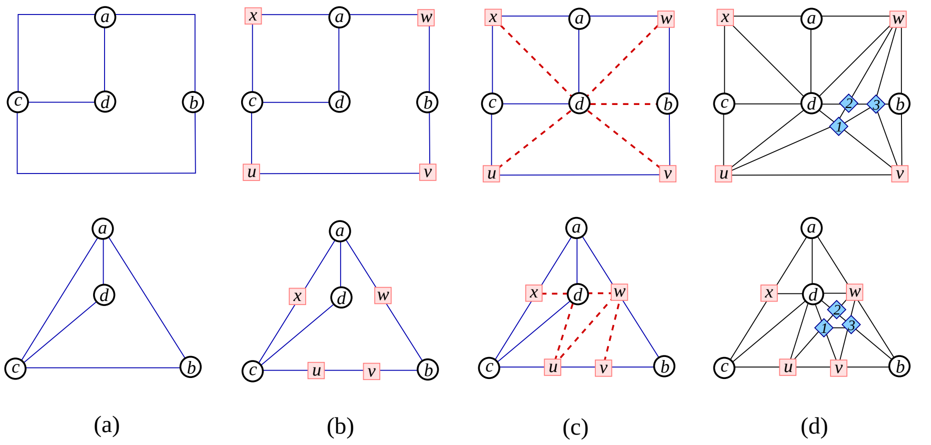


Fig. 5. Part (a) shows the graphs to be morphed: the source is on the top and target is on the bottom. Part (b) shows the addition of the bend vertices u, v, w, x (shown as squares). Part (c) shows the independent triangulation of both graphs (dashed edges). Part (d) shows the compatible triangulation with three Steiner points $1, 2, 3$ (shown as diamonds).

linear equations:

$$x' = c_{11}x + c_{21}y + t_x$$
$$y' = c_{12}x + c_{22}y + t_y$$

Given a point $p_s = (x_s, y_s)$ in D_s and the corresponding target point $p_t = (x_t, y_t)$ in D_t, we want p'_s, the resulting point after the transformations being applied on p_s, to be as close as possible to the target point p_t. Thus, to align the two drawings as best as possible we minimize the sum of squares of such pairwise distances:

$$\sum_{p_s \in D_s} dist^2(p'_s, p_t)$$

where *dist* is the Euclidean distance between two points. Minimizing this sum can be realized by setting the derivative with respect to c_{ij}, t_x, t_y to zero and solving the resulting equations which can be done in linear time.

4.2 Linear Interpolation of the Affine Matrix

Once we find the affine matrix of transformations, M, it is straight-forward to perform a linear interpolation in order to obtain the sequence of matrices throughout the morph in the rigid motion stage: $(1 - t) \times I + t \times M$, where I is the identity matrix, gives us a natural interpolation throughout time. Once again, the linear interpolation can lead to degeneracies, such as the collapse of the drawing to a single point [22]. Consider, for example, a square as D_s and the same square rotated 180° around the center as D_t. If we perform the linear

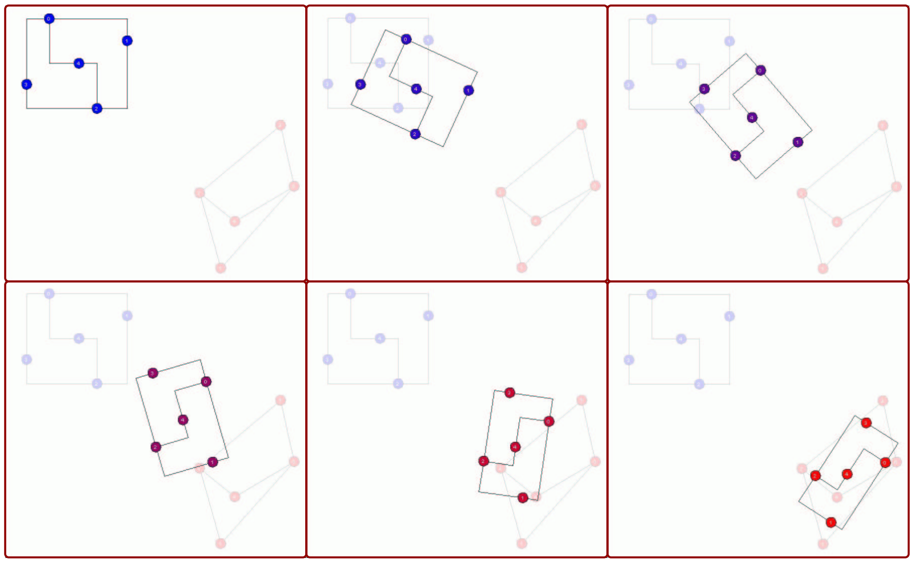


Fig. 6. D_s, at the top-left corner is aligned to D_t, at the bottom-right corner using rigid motion. The transformations include translation, rotation, scaling and shearing. Both D_s and D_t are opaquely drawn in all images.

interpolation from the identity matrix to the rotation matrix the square collapses into a point in the center. Fortunately, rotation is the only rigid transformation that is distorted by matrix interpolation and we can extract the rotation from a given affine matrix of transformations in constant time [22]. Once rotation is extracted from the affine matrix M, the linear interpolation of M does not introduce degeneracies, and the rotation can be applied separately by a linear interpolation of the rotation angle.

When aligning the two drawings using rigid motion we only use the vertices in D_s and D_t. Fig. 6 shows the snapshots when rigid motion is applied to D_s (an orthogonal drawing) to align it with D_t (a straight-line drawing). It is also possible to first introduce all bend vertices and then align D_s and D_t, using all vertices (including the bend vertices). This will correspond to swapping steps 1 and 2 of our algorithm; see Fig. 4. The main reason for aligning the drawings based only on the original graph vertices is that the placement of the bend vertices can be quite arbitrary, as long as the bends are added in the right order along the original graph edges.

5 Compatible Triangulation of the Faces

After introducing all the bend vertices in both D_s and D_t we proceed to the third stage of the algorithm and compatibly triangulate all matching pairs of faces in D_s and D_t. Once we have the embedding of the drawing D_s, i.e., the

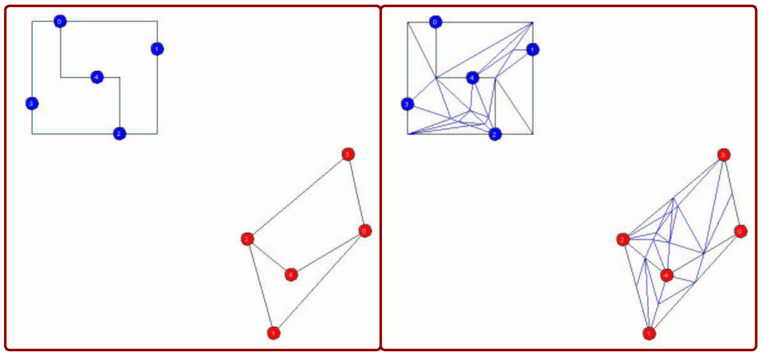

(a) (b)

Fig. 7. Part (a) shows D_s and D_t and part (b) shows the compatible triangulation.

clockwise order of the edges around each vertex in D_s, it is easy to identify the faces. We make each edge bi-directed and traverse through the directed edges each time following a neighboring edge in the clockwise order. This traversal continues until all the edges are traversed in which case we have all the faces identified. A face in D_s and D_t is a simple polygon if the graph is biconnected, or possibly a polygonal subdivision. We first consider the simple polygon case and then the polygonal subdivision case.

Given two corresponding polygons P_1 and P_2, the goal then is to compatibly triangulate the two polygons, i.e. triangulate them in such a way that the resulting triangulations are isomorphic. In general, it is not always possible to compatibly triangulate two simple polygons. However, if we allow the introduction of extra vertices (Steiner points) then we can always find a compatible triangulation, using $O(k^2)$ Steiner points, where k is the number of vertices in each polygon. We use the algorithm of [1] to construct compatible triangulations. First we independently triangulate P_1 and P_2 in $O(k)$ time. Then we overlay the two triangulations on a newly created convex polygon P with k vertices. This overlay introduces intersection points between the triangulation edges of P_1 and those of P_2. These intersections are the Steiner points and it is easy to see that there are at most $O(k^2)$ of them, since every triangulation edge of P_1 can cross at most $O(k)$ triangulation edges of P_2. The overlay of the two triangulations can create faces with more than 3 edges. Fortunately, all these faces are convex and can be easily triangulated by selecting a vertex and adding all the needed chordal edges. The resulting full triangulation is a compatible triangulation of both P_1 and P_2. Fig. 7 shows the compatible triangulations of the D_s (orthogonal edges) and D_t (straight-line edges).

If the graph underlying D_s and D_t is biconnected, then the above approach for compatible triangulation of polygons can be applied to all matching pairs of faces. If the underlying graph is not biconnected, additional complications arise.

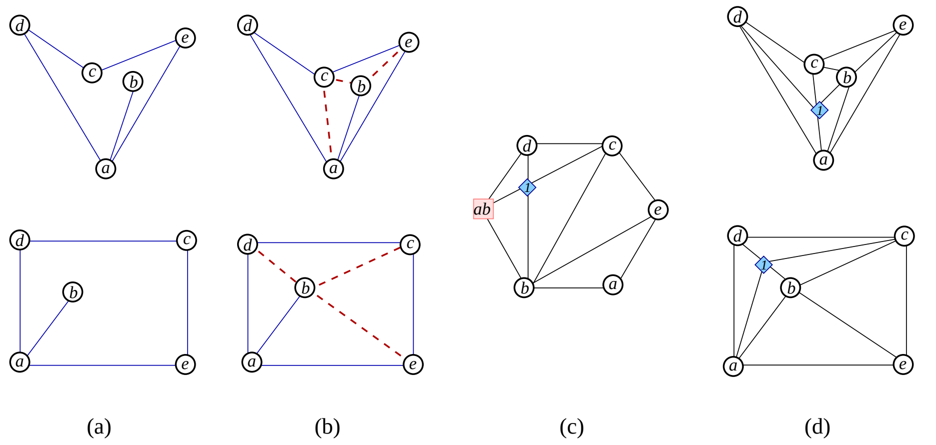


Fig. 8. Dealing with non-simple polygons; (a) two non-simple faces, P_1 (top) and P_2 (bottom); (b) independent triangulation of P_1 and P_2; (c) overlay of triangulations on P. Note that triangulation edge (a, c) from P_1 is replaced with (ab, c) in the overlay; (d) the compatible triangulation of P_1 and P_2.

In this case the polygonal subdivision P, constructed from a particular face in the graph, may not be simple, as some vertex may be repeated; see Fig 8. This problem can be overcome as follows. The triangulations of P_1 and P_2 are obtained independently as before. However, we might need to add extra vertices to P, corresponding to each repeated vertex, before overlaying the triangulations on P. Special care must be taken while overlaying the triangulation edges connected to such vertices. Let a be such a vertex, as in Fig 8. Each repetition must be the result of an edge (a, b) that is traversed in both directions while constructing the face. Denote each such repeated vertex with the corresponding edge, i.e. ab. Let (a, c) be a triangulation edge in P_1 (or P_2) that follows (a, b) in the counter clockwise order. The triangulation edge (a, c) in the top drawing of Fig. 8(b) is such an edge. While overlaying the triangulations in P we create an edge between vertices c and ab, rather than c and a, see Fig. 8(c). We then overlay the resulting triangulation on P_1 and P_2 as before.

6 Computing Trajectories Using Convex Representations

In 1963 Tutte proposed the following *barycentric mapping* to generate straight line drawing of a 3-connected planar graph G: Given an embedding of G, we map the outer face of G onto a convex polygon. Then the locations of interior vertices are determined by their barycentric coordinates:

$$u_i = \sum_{j \in N(i)} \lambda_{ij} \times u_j, \qquad \sum_{j \in N(i)} \lambda_{ij} = 1,$$

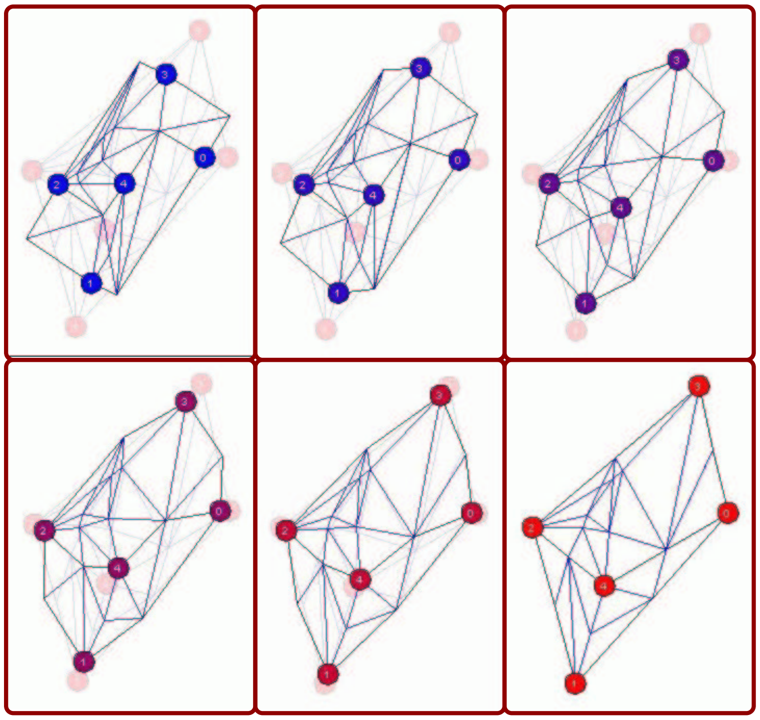


Fig. 9. Convex representation morph. D_t is opaquely drawn in all images.

where λ_{ij} is called a barycentric coordinate of u_i with respect to u_j and $N(i)$ is the set of neighbors of u_i. In Tutte's mapping ,$\lambda_{ij} = 1/d_i$, where d_i is the degree of u_i.

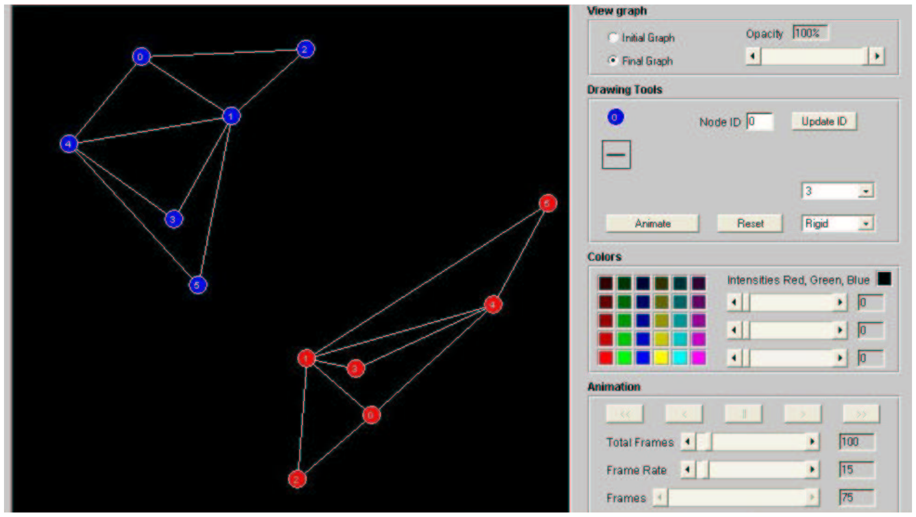
Floater and Gotsman [6] applied the idea to morphing compatible triangulations. The basic idea is to obtain a barycentric representation of source/target triangulations as $n \times n$ matrices (λ_{ij} is the entry at row i and column j), call them M_s and M_t respectively, and apply a linear interpolation from M_s to M_t, $(1-t) \times M_s + t \times M_t$. Since throughout the interpolation each resulting matrix is a barycentric representation the sequence of graphs obtained from these matrices are all planar and the morphing is intersection-free [6].

In order to find proper λ_{ij} values that depend continuously and smoothly on the neighbors of u_i we use the mean value coordinates described in [5]:

$$\lambda_{ij} = \frac{w_{ij}}{\sum_{j \in N(i)} w_{ij}}, \qquad w_{ij} = \frac{tan(\alpha_{ij-1}/2) + tan(\alpha_{ij}/2)}{dist(v_i, v_j)},$$

where α_{ij} is the angle between the segments $v_j v_i$ and $v_i v_{j+1}$. Fig. 9 shows the morph to D_t after computing the trajectory using convex representation.

Note that this approach assumes D_s and D_t share the same outer face, i.e. the outer face vertices are located at exactly the same locations. In our general



Initial Graph [✓ Final Graph]
Opacity 100%
Node ID 0
3
Rigid
0
0
0
Total Frames 100
Frame Rate 15
Frames 75

Fig. 10. A snapshot of the morphing system interface.

setting for planar graphs this usually will not be the case. To handle this problem we embed D_s and D_t inside the same triangle T. It remains to connect an outer face vertex with a vertex of T in D_s and D_t. A simple way to do this is to pick a vertex v_s in D_s that is visible from one of the triangle vertices, t_i. Connect v_s and t_i with a straight-line segment. Find a vertex v_t in D_t that is visible from t_i. Create a path from v_s to v_t in D_t following the outer face edges and connect the path to t_i.

7 System Implementation

We have implemented our morphing algorithm using Java; see Fig. 10 for a snapshot of the system. An applet for this implementation and graph morphing movies can be found at, `http://gmorph.cs.arizona.edu`. Fig. 11 shows the complete morphing sequence of two different drawings of the same graph.

References

1. B. Aronov, R. Seidel, and D. Souvaine. On compatible triangulations of simple polygons. *CGTA: Computational Geometry: Theory and Applications*, 3:27–35, 1993.
2. T. Beier and S. Neely. Feature-based image metamorphosis. volume 26, pages 35–42, July 1992.
3. S. S. Cairns. Deformations of plane rectilinear complexes. *American Math. Monthly*, 51:247–252, 1944.

4. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall, Englewood Cliffs, NJ, 1999.
5. M. Floater. Mean value coordinates. *Comp. Aided Geom. Design*, 20:19–27, 2003.
6. M. Floater and C. Gotsman. How to morph tilings injectively. *Journal of Computational and Applied Mathematics*, 101:117–129, 1999.
7. C. Friedrich. The ffgraph library. Technical Report 9520, Universität Passau, 1995.
8. C. Friedrich and P. Eades. The Marey graph animation tool demo. In *Proceedings of the 8th Symposium on Graph Drawing (GD)*, pages 396–406, 2000.
9. C. Friedrich and P. Eades. Graph drawing in motion. *Journal of Graph Algorithms and Applications*, 6(3):353–370, 2002.
10. C. Friedrich and M. E. Houle. Graph drawing in motion II. In *Proceedings of the 9th Symposium on Graph Drawing (GD)*, pages 220–231, 2001.
11. E. Goldstein and C. Gotsman. Polygon morphing using a multiresolution representation. In *Graphics Interface '95*, pages 247–254, 1995.
12. J. Gomes, L. Darsa, B. Costa, and D. M. Vello. *Warping and Morphing of Graphical Objects.* Morgan Kaufmann, 1999.
13. C. Gotsman and V. Surazhsky. Guaranteed intersection-free polygon morphing. *Computers and Graphics*, 25(1):67–75, Feb. 2001.
14. T. He, S. Wang, and A. Kaufman. Wavelet-based volume morphing. In *Proceedings of the Conference on Visualization*, pages 85–92, 1994.
15. M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In *Proceedings of the 6th Symposium on Graph Drawing (GD)*, pages 374–383, 1998.
16. J. F. Hughes. Scheduled Fourier volume morphing. *Computer Graphics*, 26(2):43–46, July 1992.
17. M. Kaufmann and D. Wagner. *Drawing graphs: methods and models*, volume 2025 of *Lecture Notes in Computer Science.* Springer-Verlag, 2001.
18. T. Samoilov and G. Elber. Self-intersection elimination in metamorphosis of two-dimensional curves. *The Visual Computer*, 14:415–428, 1998.
19. T. W. Sederberg, P. Gao, G. Wang, and H. Mu. 2D shape blending: An intrinsic solution to the vertex path problem. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 15–18, 1993.
20. T. W. Sederberg and E. Greenwood. A physically based approach to 2-D shape blending. volume 26, pages 25–34, July 1992.
21. M. Shapira and A. Rappoport. Shape blending using the star-skeleton representation. *IEEE Computer Graphics and Applications*, 15(2):44–50, Mar. 1995.
22. K. Shoemake and T. Duff. Matrix animation and polar decomposition. In *Proceedings of Graphics Interface '92*, pages 258–264, May 1992.
23. V. Surazhsky and C. Gotsman. Controllable morphing of compatible planar triangulations. *ACM Transactions on Graphics*, 20(4):203–231, Oct. 2001.
24. A. Tal and G. Elber. Image morphing with feature preserving texture. *Computer Graphics Forum*, 18(3):339–348, Sept. 1999. ISSN 1067-7055.
25. C. Thomassen. Deformations of plane graphs. *J. Combin. Theory Ser. B*, 34:244–257, 1983.
26. W. T. Tutte. How to draw a graph. *Proc. London Math. Society*, 13(52):743–768, 1963.

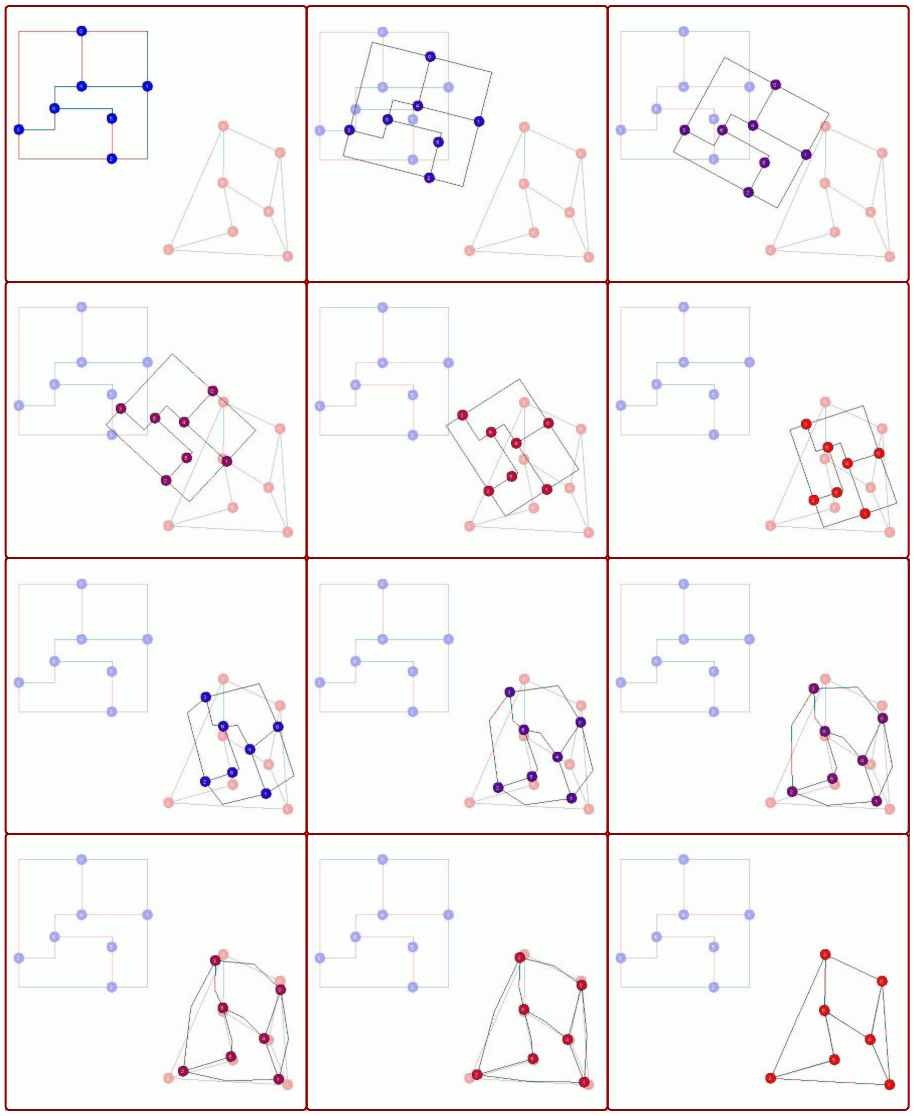


Fig. 11. A complete morph from D_s at the to-left corner, to D_t at the bottom-right.