Morphing Planar Graphs

C. Erten, S. G. Kobourov and C. Pitta
Department of Computer Science
University of Arizona
{cesim,kobourov,chandanp}@cs.arizona.edu
http://gmorph.cs.arizona.edu

1. INTRODUCTION

We describe an efficient algorithm and implementation for morphing planar graphs. The algorithm generalizes the notion of compatible triangulations of polygons to compatible triangulations of planar graphs and applies a combination of rigid-motion morph and barycentric-representation morph to find smooth and crossings-free transformations between different drawings of a given planar graph. The video presentation and our Java applet can be found at `http://gmorph.cs.arizona.edu`.

Morphing refers to the process of transforming one shape (the source) into another (the target). Morphing is widely used in graph drawing, computer graphics, animation, and modeling [2, 3, 4]. In planar graph morphing we would like to transform a given source graph to another pre-specified target graph. Smooth transformation of one graph into another is useful for numerous graph drawing problems. In particular, when dealing with dynamic graphs and graphs that change through time, it is important 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.

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 are crossings-free and realize the same embedding of G (otherwise, a crossing-free morph does not exist). The source drawing D_s and the target drawing D_t can be straight-line drawings, or drawings with bends and curves. 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 the transformation.

2. THE ALGORITHM

Our algorithm has five main steps which we summarize below.

Introduction of Bend Vertices: We begin by introducing all the bend vertices in both D_s and D_t; see Fig. 1.

Introduction of Convex Bounding Box: The morping stages of the algorithm require that D_s and D_t share the same outer face. In the general setting for planar graphs this usually will not be the case. To handle this problem we embed each of D_s and D_t inside bounding boxes, B_s and B_t. We select a vertex v_s in D_s that is visible from one of the bounding box vertices, b_s in B_s. We connect v_s and b_s with a straight-line segment. In D_t we find the vertex v_t (that corresponds to v_s in D_s) and connect it via a path to vertex b_t in D_t (that corresponds to b_s in B_s).

Compatible Triangulations: Given the embedding of D_s, i.e., the clockwise order of the edges around each vertex in D_s, it is easy to identify the faces. We make each edge bidirected 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. Next, given two corresponding faces (polygons) we compatibly triangulate them, 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 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. We describe how we generalize this approach to general planar graphs in [2].

Rigid-Motion Morph: We begin by aligning the two drawings using 2-D transformations consisting of translation, rotation, scaling, and shearing. That is, we move the source drawing as close as possible to the target drawing as a rigid object in space. 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 linear equations: $x' = c_{11}x + c_{21}y + t_x$ and $y' = c_{12}x + c_{22}y + t_y$.

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

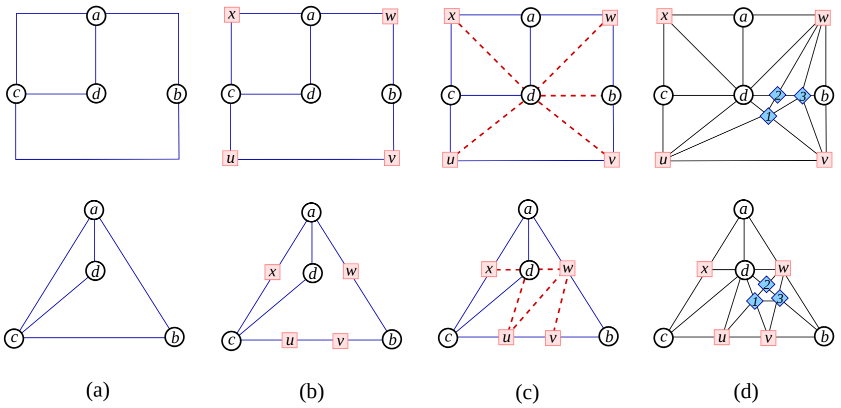

Figure 1: Part (a) shows the graphs to be morphed: the source is on the top and target is on the bottom. Part (b) shows the addition of the bend vertices u, v, w, x (shown as squares). Part (c) shows the independent triangulation of both graphs (dashed edges). Part (d) shows the compatible triangulation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

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 all the 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.

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. However, the linear interpolation can lead to degeneracies, such as the collapse of the drawing to a single point. As rotation is the only rigid transformation that is distorted by matrix interpolation we extract the rotation from a given affine matrix of transformations in constant time. Then the linear interpolation of M does not introduce degeneracies, and the rotation is applied separately by a linear interpolation of the rotation angle. Thus, we obtain the first part of the trajectories for the vertices (the remaining part is computed in the third stage of the algorithm).

Barycentric-Representation Morph: In this stage of the algorithm we compute the remaining part of the trajectories of the vertices. In 1963 Tutte [5] 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:

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

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.

These barycentric coordinates can be used to morph compatible triangulations [3]. We 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 crossings-free.

3. THE VIDEO

The video illustrates the main stages of the algorithm: introduction of bend vertices, introduction of bounding boxes, compatible triangulations, rigid morphing, and barycentric morphing. We have implemented our algorithm in Java. Fig 2 shows a sample snapshot from a morphing sequence. Top-left is the source and bottom-right is the target drawing. The video and the Java applet can be found at `http://gmorph.cs.arizona.edu`.

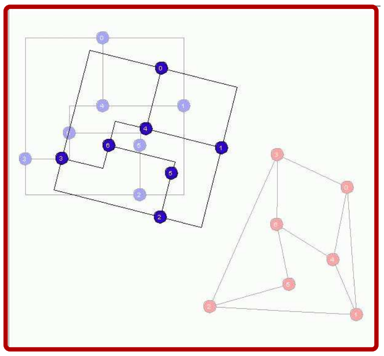

Figure 2: Morphing snapshot

4. 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] C. Erten, S. G. Kobourov, and C. Pitta. Intersection-free morphing of planar graphs. In *Proceedings of the 11th Symposium on Graph Drawing (GD)*, 2003.

[3] M. Floater and C. Gotsman. How to morph tilings injectively. *Journal of Computational and Applied Mathematics*, 101:117–129, 1999.

[4] C. Friedrich and P. Eades. Graph drawing in motion. *Journal of Graph Algorithms and Applications*, 6(3):353–370, 2002.

[5] W. T. Tutte. How to draw a graph. *Proc. London Math. Society*, 13(52):743–768, 1963.