Vis Comput
DOI 10.1007/s00371-009-0396-3

ORIGINAL ARTICLE

An intuitive polygon morphing

Martina Málková · Jindřich Parus · Ivana Kolingerová · Bedřich Beneš

Abstract We present a new algorithm for morphing simple polygons that is inspired by growing forms in nature. While previous algorithms require user-assisted definition of complicated correspondences between the morphing objects, our algorithm defines the correspondence by overlapping the input polygons. Once the morphing of one object into another is defined, very little or no user interaction is necessary to achieve intuitive results. Our algorithm is suitable namely for growth-like morphing. We present the basic algorithm and its three variations. One of them is suitable mainly for convex polygons, the other two are for more complex polygons, such as curved or spiral polygonal forms.

Keywords Morphing · Vector data · Polygon intersection · Computer Graphics

This work was supported by Grant Agency of the Czech Republic—project No. 201/09/0097.

Electronic supplementary material The online version of this article (http://dx.doi.org/10.1007/s00371-009-0396-3) contains supplementary material, which is available to authorized users.

M. Málková (✉) · J. Parus · I. Kolingerová
University of West Bohemia, Plzeň, Czech Republic
e-mail: mmalkov@kiv.zcu.cz

J. Parus
e-mail: jparus@kiv.zcu.cz

I. Kolingerová
e-mail: kolinger@kiv.zcu.cz

B. Beneš
Purdue University, West Lafayette, USA
e-mail: bbenes@purdue.edu

1 Introduction

Morphing is a (non-linear) transformation of a shape into another shape and has practical uses in computer graphics, animation, modeling and design. This area has been thoroughly researched and the existing methods can be classified into two main groups—traditional volume and image morphing and boundary-based morphing. In this article we focus on boundary-based morphing methods. The algorithms designed in this area concentrate on morphing similar shapes, where some common features can be identified. However, in some cases it is not possible to align all the main features of the input shapes (e.g. a head with and without horns). A natural morph usually means growth of the non-aligned part from its aligned neighbor. In this paper, we propose a novel approach for morphing with this "growth-like" nature.

The computation of a morph requires three inputs: two shapes (source and destination), the definition of vertex correspondences between the source and the destination, and the definition of vertex paths together with the dynamics of the transformation. The last input defines the way in which the shapes are morphed in time. The morph can be linear, it can accelerate or decelerate, or its time dependence can be described by a user-defined mapping function. Establishing the correspondence of the vertices and the vertex paths are two independent steps, which can be solved by different algorithms, but both usually require some user interaction. Although there is much to achieve by a suitable vertex path computation, the definition of the correspondence between two shapes remains crucial. Most algorithms offer the user the ability to influence the correspondence by adding some constraints or manually defining corresponding vertices, but such a definition can be complex and sometimes the user cannot predict how the changes will influence the result, so a trial and error technique is usually applied iteratively.

Published online: 18 October 2009

We present an algorithm for easy-to-use and easy-to-define morphing of polygonal objects. Our technique takes two polygonal objects that partially overlap. Then an overlapping area of these two polygonal objects is formed, defining the *core* of the morphing. We suppose that the *core* consists of a single simple polygon, which is a typical situation for such inputs where one would expect a growth-like morph. The morphing simultaneously changes both polygons. The input polygon is absorbed by the core, whereas the output polygon grows out of it. The principal advantage of the shape absorption and the growth is that they can be viewed as the same process with different direction in time. Algorithmically this means that we need to implement only one process. Another advantage of the algorithm is that it offers growth-like morphing. This process could not be easily achieved using traditional algorithms as it would require manually defining many correspondences. Thanks to its growth-like shape change, our algorithm also produces satisfactory results for polygons that are not star-shaped.

The algorithm solves the correspondence of the two input polygons as well as the vertex paths. The final animation is computed by the linear interpolation between the intermediate positions in the vertex paths.

The paper is organized as follows. Section 2 describes related work in the area of polygon morphing. Section 3 describes the method itself and discusses three morphing strategies used. Section 4 shows the results of the algorithm and demonstrates typical uses of the techniques described in Sect. 3. Finally, Sect. 5 describes our conclusions and suggestions for future work.

2 Related work

Digital warping and morphing has a long tradition in Computer Graphics and its full exposition is outside the scope of this paper. We refer readers to [5] for a detailed description. Here we describe the work related to 2D polygon morphing.

Polygon morphing computation can be divided into two parts: (1) defining the vertex-to-vertex correspondence in the source and the destination polygons, and (2) defining the transitions. Point out that the source and the destination polygons are not required to have the same number of vertices.

The problem of vertex-to-vertex correspondence was addressed by Sederberg and Greenwood in [10] by using a physical model. The polygon edges are modeled as elastic connections and the shape transformation involves the calculation of a physical response by minimizing the energy of the system. This algorithm is efficient for similar input polygons and it can also handle cases when the initial shapes are rotated or translated. The algorithm does not address highly dissimilar shapes and self-intersections.

A computation of trajectories was described by Sederberg et al. in [9], where edge lengths or internal angles are interpolated instead of the vertex positions. The polygons are converted to the so-called edge–angle representation [5] that is invariant to rigid transformations. This interpolation scheme avoids edge collapsing and non-monotonic angle changes. This technique was used to generate in-betweens for animation based on keyframes. The concept of interpolation of intrinsic parameters was also further used for morphing of planar triangulations in [12, 13]. The intrinsic interpolation avoids local self-intersections. However, it does not solve the problem of global self-intersections which may occur for highly dissimilar and complicated shapes.

Shapira and Rappoport [11] introduced a method that first decomposes the source and the destination polygons into star-shaped polygons. The skeleton is a planar graph which joins star-points of neighboring star-shaped polygons. The skeletons are interpolated and the intermediate shapes are reconstructed from the interpolated skeletons. The difference between this approach and methods described in [9, 10] is that this approach also considers the interior of the polygon and not only the boundary. The problem with this approach is that it relies on isomorphic star-shaped decomposition which might be difficult to compute, especially in the case of dissimilar shapes.

Alexa et al. introduced *as-rigid-as-possible shape interpolation* in [2]. They compute a compatible triangulation of the input polygons. The compatible triangulation is a dissection of the source and the destination polygons so that the triangulations are isomorphic, i.e., there is one-to-one correspondence between triangles in the source and in the destination. Then, an affine transformation which transforms a source triangle to the destination triangle is computed. Interpolation of the affine transformation defines the morphing. Adjacent triangles are also considered in the interpolation. Similar approaches were also described by Surazhsky and Gotsman in [12, 13]. The principal problem of these methods is the computation of the isomorphic triangulation of the input shapes, as its quality influences the quality of the morphing.

Gomez et al. introduced *2D merging* [5], which is a 2D application of the algorithm that was originally developed for 3D meshes (see for example [1, 7]). First, the input polygons are mapped to a unit disc, then both mappings are merged. The vertices of the first polygon are mapped on the second polygon and vice versa using inverse mapping. This results in polygons with the same number of vertices. A linear interpolation is used to obtain the resulting morphing transition. This technique is suitable for convex, star-shaped or slightly non-convex polygons. This algorithm is not suitable for highly non-convex polygons, as it produces self-intersections during the morphing transition.

Carmel and Cohen-Or [3] showed an algorithm which combines a 2D merging and a polygon evolution. A user

first specifies anchor points that define the correspondence between polygons. Using the anchor points, a warp function is computed that warps the source polygon to the destination polygon. Once the source polygon is warped, a polygon evolution technique is used to evolve the source and the destination polygon to a convex shape.

Johnstone and Wu [6] described an alternative approach to merging polygons by morphing. The 2-to-1 morphing is a fundamental case in the morphing between different numbers of polygons. The basic idea is to merge the two polygons into one and then use the one-to-one polygon morphing technique to morph between the merged polygon and a destination polygon. During the merging the two polygons are morphed towards each other until they meet at a point. Then a curve evolution technique is used to morph the two polygons connected at some point into a more natural shape which is later morphed towards the destination shape.

3 The polygon morphing algorithm

Let us have an ordered set of vertices v_i, $i = 0, \ldots, N-1$. An edge e_i is a line segment with endpoints v_i and v_{i+1}. A polygonal chain Q is a sequence of edges $e_0, e_2, \ldots, e_{N-2}$ with vertices $v_0, v_1, \ldots, v_{N-1}$. A polygon P is a region of the plane bounded by a closed polygonal chain δP, i.e. a polygonal chain where $v_0 = v_{N-1}$. A polygon is simple if there is no pair of nonconsecutive edges sharing a point.

3.1 Algorithm overview

The algorithm takes as input two simple polygons (see Fig. 1), the source polygon A and the destination polygon B which must spatially overlap, $A \cap B \neq \emptyset$. No further condition on the polygons is imposed. However, it is beneficial if the polygon vertices are distributed equidistantly. Therefore, an optional preprocessing step is to resample the input polygons A, B so that their vertices are equidistantly distributed.

The overlapping area of A and B is called *the core* of the morph and it is denoted by $C = A \cap B$. Without loss of generality, we suppose that the core C consists of a single simple polygon.

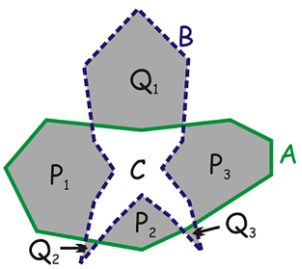

Fig. 1 The description of the polygons. A and B are the input polygons. C is the intersection (the core) and P and Q are polygons that will be absorbed and grow out of the core respectively

The area of polygon A that is clipped out is denoted by $P = A - B$ and analogously the area of the polygon B that is clipped out is denoted by $Q = B - A$. Note that P and Q are not necessarily single polygons. They can be a set of simple polygons so that $P = \bigcup_i P_i$ and $Q = \bigcup_j Q_j$. We suppose that $P \neq \emptyset$ or $Q \neq \emptyset$. During the morphing process, the parts Q_j grow out from the core, while the parts P_i are absorbed into the core. All parts of the polygons grow and are absorbed simultaneously, which results in the effect of morphing two simple polygons. Algorithmically, the process of absorption is an inverse process of growing, so for now we will concentrate only on the description of the absorption of one part P_i.

The boundary δP_i consists of two polygonal chains C_{in}, C_{out}, where C_{in} is a shared polygonal chain of δC and δP_i, C_{out} is the rest of δP_i which remains after removing C_{in} from δP_i. The polygonal chains C_{in}, C_{out} are separated by intersection vertices v_{I0}, v_{I1} (Fig. 2). An intersection vertex lies in the intersection of the closed polygonal chains δA and δB. If $P_i \neq A$ and $P_i \neq B$ then two vertices of δP_i become intersection vertices. By morphing the polygon chain C_{out} to the polygon chain C_{in} we achieve the effect of the part P_i being absorbed into the core C. Hereby, we can express the polygon morphing problem as several polygonal chain morphing problems.

The morphing between a polygonal chain C_{out} and C_{in} can be described in terms of a vertex path. The *vertex path* of a vertex v_i is a list of coordinates describing points in E^2 that the vertex passes through during the morphing sequence. The vertex path is computed for each vertex of the polygonal chain C_{out} excluding the intersection vertices v_{I0} and v_{I1}. It has at least two elements, i.e., the initial position of the vertex at the time $t = 0$ and the final position of the vertex at the time $t = 1$. A set of vertex transformations is obtained by computing intermediate positions of a vertex. The intermediate positions are computed by interpolating the position values p_k along the vertex path. Any interpolation technique such as a piecewise linear interpolation, a cubic spline interpolation or some other interpolation form can be used.

The vertex path is computed using a concept of a *topological distance*. If we define a distance $d(v_i, v_j)$ between vertices v_i and v_j as the minimal number of edges on the polygon between v_i and v_j, the *topological distance* of a vertex v_i is its minimal distance to the intersection vertices $d_{\min}(v_i) = \min(d(v_i, v_{I0}), d(v_i, v_{I1}))$. Figure 2 shows an example of topological distances—the vertices are labeled according to the topological distance from the intersection vertices v_{I0}, v_{I1}. The topological distance establishes the order in which the vertices will grow. The vertices with a smaller topological distance will reach their destination sooner than the vertices with a larger topological distance. This prevents self-intersections during the morphing

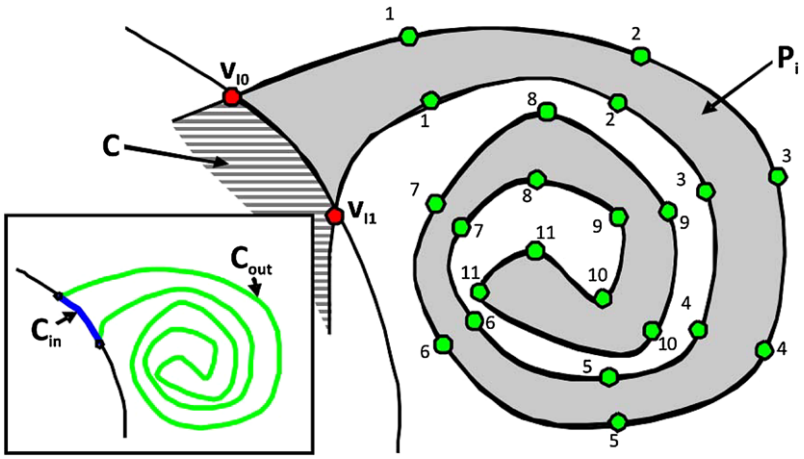

Fig. 2 A highly non-convex polygon P_i is absorbed in the core C (the hatched part). The polygon P_i will be absorbed in the core C by morphing the polygonal chain C_{out} to C_{in}

Input: Two partially overlapping polygons A, B (*Optional*: Resample the input polygons so that their vertices are equidistantly distributed).
Output: A polygon $R = A \cup B$, where each vertex of δR either contains a vertex path determining its behavior over time, or belongs to $A \cap B$.
The algorithm:
1. Compute the core $C = A \cap B$.
2. Compute the polygon sets $P = A - B = \bigcup_i P_i$, $Q = B - A = \bigcup_i Q_i$.
3. $\forall v_i \in \delta P$ and $\forall v_i \in \delta Q$: compute its topological distance d_i.
4. Using a user-selected method (Perimeter, Midpoint, or Projection growing), compute the vertex path of each vertex v_i.
5. Merge the polygon C and the polygon sets P, Q to get the resulting polygon R

Fig. 3 Pseudocode of the algorithm

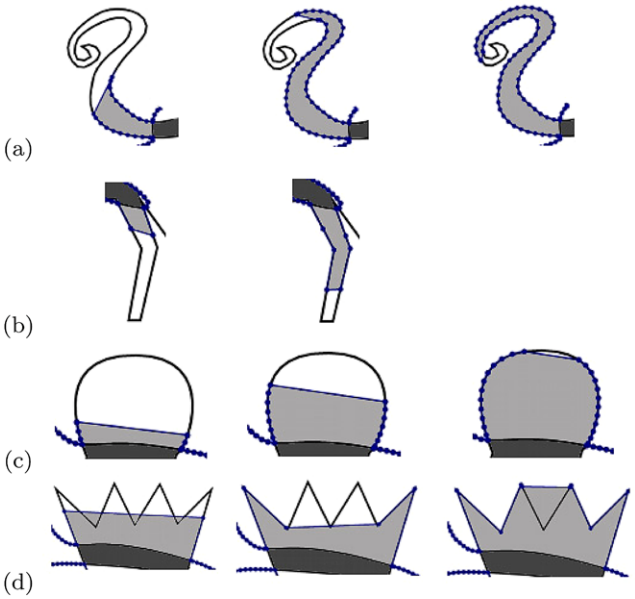

Fig. 4 Cases that are suitable (**a**, **b**) and that are not suitable (**c**, **d**) for the perimeter growing (*dark gray color* indicates the place where the part meets the core, *light gray* represents the part that grows out, the input polygon is indicated by a *thick line*)

process. Moreover, we extend the concept of the topological distance to distinguish between topological distances of vertices lying on the polygonal chains C_{in} and C_{out}. The topological distance should be positive only, but we use positive distance for vertices $v_i \in \delta P_i C_{\text{in}}$ and add the negative sign to the vertices lying on the polygonal chain C_{in}. Then $d_{\min}, d_{\max}$ are minimal and maximal topological distances of the vertices from δP_i.

The pseudocode of the algorithm can be seen in Fig. 3. In step 4 of the algorithm, we use three different methods for computing the vertex paths to morph polygonal chains C_{in} and C_{out} and to simulate a process of absorption for polygon P_i; the Perimeter growing, the Midpoint growing and the Projection growing. These methods will be described below.

3.2 The perimeter growing method

As was described in Sect. 2, many previously introduced algorithms struggle with morphing of curved objects such as those shown in Fig. 4a, b.

We address this problem with a method that we call *Perimeter growing*. The basic idea is that the vertices $v_i \in C_{\text{out}}$ travel along the perimeter of the polygon P_i, meaning that its vertex path contains only positions of the vertices of δP_i. When deciding about the correspondence of the vertices of C_{in} and C_{out}, we need to determine at which vertex v_j with the topological distance d_j is the specific vertex path supposed to end. The vertex path of a vertex v_i with the topological distance d_i contains vertices with topological distances $(d_{i-1}, d_{i-2}, \ldots, d_0, d_{-1}, \ldots, d_j)$ (see Fig. 5). We define two conditions concerning the last vertex of the vertex path, vertex v_j. First, it must belong to the polygonal chain C_{in}. Second, there should be at least one vertex path to end at each vertex that belongs to the polygonal chain C_{in} (to form the shape of the other polygon). Based on these observations, we let the vertex path of the vertex with $d_{\max}$

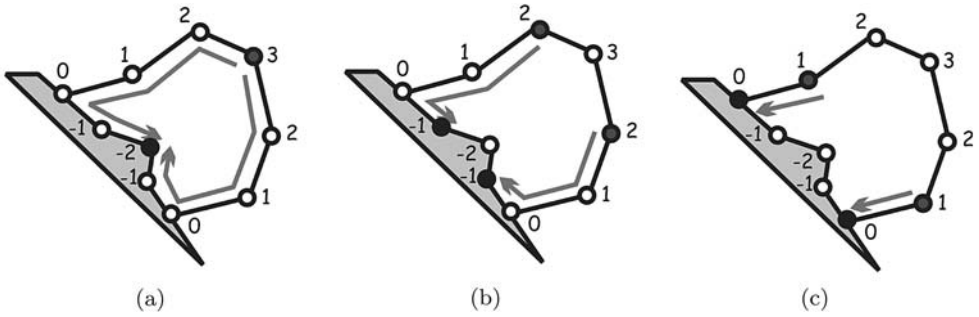

Fig. 5 Vertex paths ($d_{\max} = 3$, $d_{\min} = -2$): (**a**) the vertex path for the vertex with $d = d_{\max}$ ends at the vertex with $d = d_{\min}$ (**b**) for the vertex with $d = d_{\max -1}$ it ends at the vertex with $d = d_{\min +1}$ (**c**), and so on

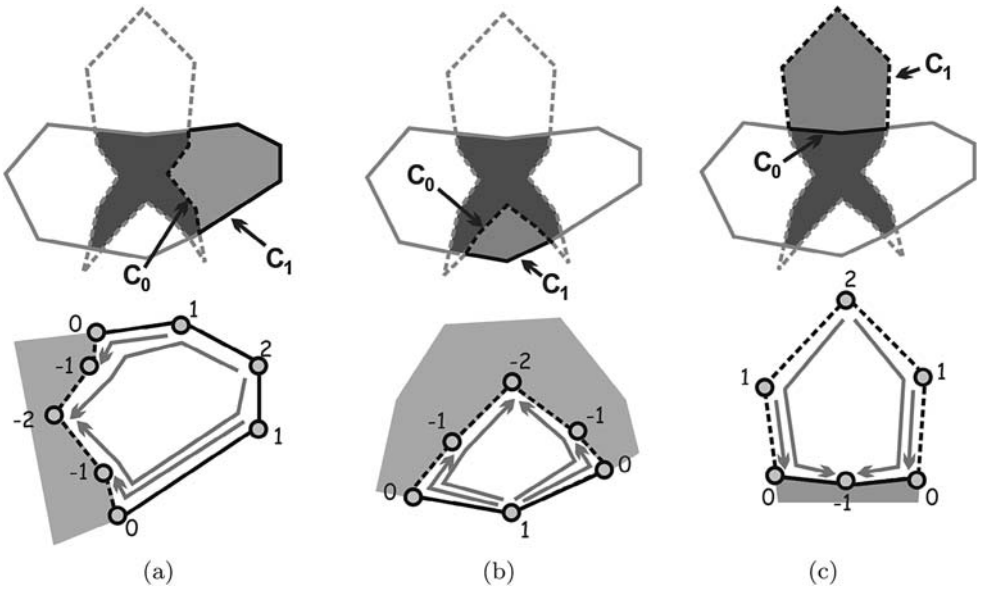

Fig. 6 Three possible input configurations for computing vertex paths: (**a**) $n_{\text{in}} = n_{\text{out}}$, (**b**) $n_{\text{in}} > n_{\text{out}}$, (**c**) $n_{\text{in}} < n_{\text{out}}$. (The *dark gray color* represents the core C, the *grey* represents the selected part, the *light grey* indicates the place where the part meets the core; the first polygon is indicated by a *full line* and the second by a *dashed line*, the vertex paths are outlined by *grey arrows*)

always end at the vertex with $d_{\min}$. If there is only one vertex with $d_{\max}$, we duplicate it to solve the case when there are two vertices with $d_{\min}$. The vertex path of the vertex with $d_{\max -1}$ should end at the vertex with $d_{\min +1}$. Generally, a vertex path of the vertex with $d_{\max -i}$ should end at the vertex with $d_{\min +i}$, as shown in Fig. 5.

However, this situation is a special case that happens only when C_{out} and C_{in} have the same number of vertices. For the other cases, let us denote n_{in} and n_{out} the number of vertices of C_{in} and C_{out} respectively. We can then distinguish the following three cases:

- $n_{\text{in}} = n_{\text{out}}$ (Fig. 6a).
  The vertex path of each vertex of C_{out} ends at one vertex of C_{in}.
- $n_{\text{in}} > n_{\text{out}}$ (Fig. 6b).
  Some vertices of C_{in} do not belong to any vertex path. However, they should have a correspondence. One solution is to duplicate some vertices of C_{out}. We use such vertices of C_{out} that have a topological distance equal to one and duplicate them as many times as is necessary to cover all the vertices of C_{in} that remain.
- $n_{\text{in}} < n_{\text{out}}$ (Fig. 6c) .
  Some vertices of C_{out} do not have any corresponding vertices in C_{in}. In such a case their paths will end at the intersection vertices.

Because the vertex path follows the perimeter of the polygon, the resulting polygon will look like it is growing from the core. The Perimeter growing method works for most of the morphs; however, there are two problematic cases. First, the top of the growing polygon is always a straight line connecting the vertices with the same topological distance. This makes the method unsuitable for polygons, where some vertices with the same topological distance are wide apart, such as those in Fig. 4c. The second problem is that following the shape of the polygon is not always desirable—for example, if we have a polygon as in Fig. 4d, it will first grow from the core and then be absorbed a bit before continuing its growth.

3.3 The midpoint growing method

The *Midpoint growing* attempts to make the top of the growing part less flat than does the Perimeter growing. This method uses the midpoints of the line segments defined by the vertices of the same polygonal chain C_{out} or C_{in} with the

same topological distance (Fig. 7) as the vertices in the vertex paths. Figure 8 shows that the method produces similar results to the Perimeter growing method, the only difference is that the top of the growing part, which is flat in the case of the Perimeter growing method, but sharp for the Midpoint growing method.

Let us denote m_i the midpoint of the line segment $v_i v_j$ where $d_i = d_j$. The vertex path of a vertex v_i with the topological distance d_i contains vertices $(m_{i-1}, m_{i-2}, \ldots, m_0, \ldots, m_{j-1}, v_j)$. The vertex v_j is computed according to the rules described for the Perimeter growing method in Sect. 3.2. The results for all three cases with a different relationship between $n_{\text{in}}, n_{\text{out}}$ are shown in Fig. 9. When $n_{\text{in}} = n_{\text{out}}$ (Fig. 9a), each vertex of C_{out} ends its path at one vertex of C_{in}. If $n_{\text{in}} > n_{\text{out}}$ (Fig. 9b), some of the vertices of C_{out} need to be duplicated (those with the topological distance equal to one). If $n_{\text{in}} < n_{\text{out}}$ (Fig. 9c), some vertices of C_{out} cannot end their paths at the expected vertex (so they end at the intersection vertices).

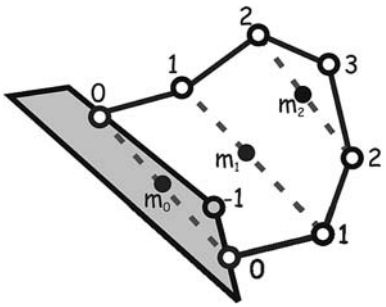

Fig. 7 Midpoints m_0, m_1, m_2 connecting the vertices with the same topological distance (and of the same polygonal chain)

3.4 The projection growing method

Both the Perimeter and the Midpoint growing methods can generate undesired deformations at the top of the growing part—the former "cuts" the top (there are always two vertices connected by a straight line), the latter bends the top a little, but none of the methods indicates the final shape of the top of the part. Therefore, both methods produce better results for narrow and curved parts. To morph parts that are not curved, we created a new method based on a projection and we call it *Projection growing method*. This method equidistantly maps the vertices onto the line segment connecting the two intersection vertices.

The main difference of the Projection growing method from the previously presented methods is that all vertices of C_{out} have the same number of elements in their vertex paths. Let us describe in detail how this method calculates the vertex paths. First, the vertices of C_{out} are mapped (projected) onto the line segment l_i defined by the intersection vertices (i.e., the vertices with a zero topological distance), using an equidistant mapping. The line segment l_i is divided into $n_{\text{out}} + 1$ parts, where n_{out} is the number of the vertices of C_{out}. We assign the vertices of C_{out} sequentially to the new vertices on the line segment (Fig. 10a). The next step is to map the vertices of C_{in} in a similar way (Fig. 10b).

The last step (Fig. 10c) is to sort the projected vertices of C_{in} and C_{out} onto an ordered list in the order in which they appear on l_i. We denote the list $v_{I0} = a_0, a_1, a_2, \ldots, a_{n-1} = v_{I1}$, where a_i is a mapped vertex that originally belongs either to C_{in} or to C_{out}. Then we traverse this list as follows:

1. Start at v_{I0}. Go through the list until a vertex of C_{in} is reached. Add it into the vertex paths of all the vertices of C_{out} that are located before this vertex.

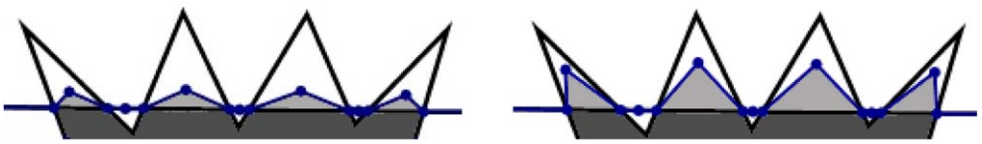

Fig. 8 A suitable input shape for the Midpoint morphing (*dark gray color* indicates the place where the part meets the core, *light gray* represents the part that grows out, the input polygon is marked by a *thick line*)

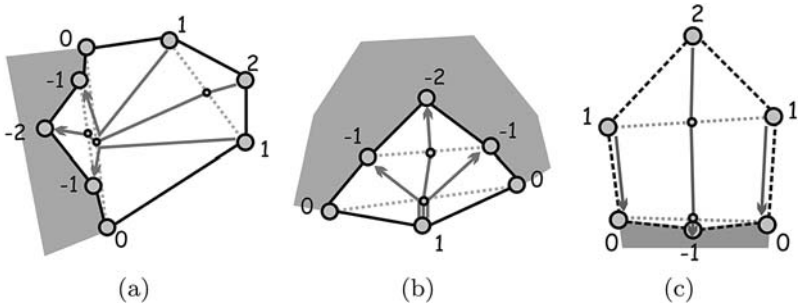

Fig. 9 Three possible inputs for computing vertex paths: (**a**) $n_{\text{in}} = n_{\text{out}}$, (**b**) $n_{\text{in}} > n_{\text{out}}$, (**c**) $n_{\text{in}} < n_{\text{out}}$ (*light grey color* designates the place where the part meets the core, and the vertex paths are outlined by *grey arrows*...)

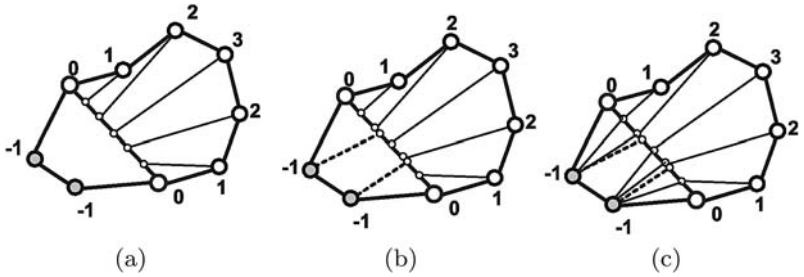

Fig. 10 Computing the vertex paths in the Projection growing method: (**a**) mapping the vertices of C_{out} onto the line segment between the intersection vertices, (**b**) the vertices of C_{in} onto the same line segment, (**c**) choosing the vertices of C_{in} for the vertex paths of the vertices of C_{out} (the mapping is marked by a *dashed line*, the vertex paths are outlined by *thin lines*)

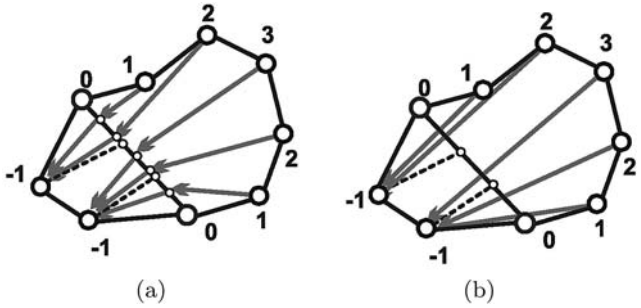

Fig. 11 Vertex paths (*grey*) for (**a**) projection using the auxiliary coordinates in the vertex paths, (**b**) direct projection

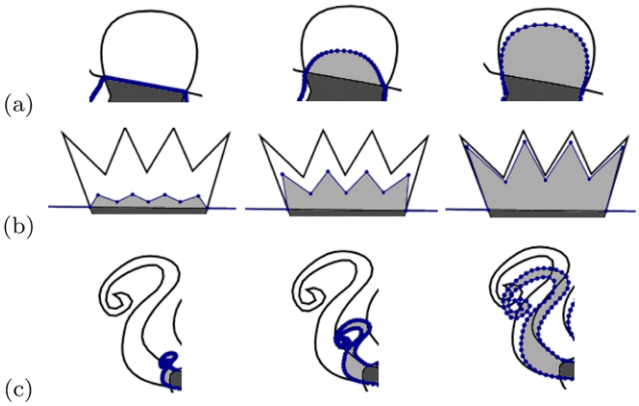

Fig. 12 The Projection growing method provides intuitive results in (**a**, **b**) but fails in the case (**c**) (*dark gray color* indicates the place where the part meets the core, *light gray* represents the part that grows out, and the input polygon is marked by a *thick line*)

2. Continue traversing the list. Each time the vertex C_{out} is reached, add the recent vertex of C_{in} to its vertex path.
3. The traversal is completed when v_{I1} is reached.

The above-mentioned method produces vertex paths containing only two vertices as it defines only the vertex-to-vertex correspondence between C_{out} and C_{in}. There is a possibility of using the position on l_i, where v_i was mapped, in the morphing sequence—so as to include it in its path between the two vertices. This modification brings different results—instead of heading straight towards their corresponding vertices, the vertices of C_{out} first travel towards their positions on l_i. The difference between the two strategies is shown in Fig. 11. The modified Projection morphing using an intermediate point is more suitable when both C_{in} and C_{out} lie on the same side of l_i, whereas the direct vertex-to-vertex morphing provides better results when C_{out} and C_{in} lie on different sides of l_i.

The Projection growing method generates output that appears to be somewhere between the Perimeter (or Midpoint) growing methods and the typical algorithms based on the correspondence between the whole polygons. There is still dependence of the output on the core; however, the vertex paths are not influenced by the shape of the polygon, they only define the correspondence of the vertices. Therefore, the projection is not usable for curled or spiral parts, where it has the same problems as the traditional correspondence-based algorithms. This morphing method is suitable for the cases where the shape of the part is convex or when it is non-convex, but not curled (Fig. 12).

3.5 Merging

The last step of our algorithm (recall Fig. 3) is merging. The polygons P_i and Q_i that were clipped out by the core C are merged into one polygon R. Moreover, each vertex in R has its vertex path that is defined by one of the above described morphing methods. The direction of each vertex path depends on the direction of morphing. Vertex paths of vertices from δQ point into the core as the polygon Q is absorbed, whereas the vertex paths of vertices from δP point out.

The merging process is motivated by Weiler–Atherton algorithm for polygon intersection [4]. It processes each polygon and makes copies of vertices with positive topological distance to the new list of vertices, while preserving the vertex order. The details of the merging process are in Fig. 13.

Input: List of polygons $S = \bigcup_i P_i \cup \bigcup_i Q_j$, lists of vertices of each polygon $l_i = (v_0, \ldots, v_{N-1})$. The lists l_i are circular so the next vertex to v_{N-1} is v_0 and the previous vertex to v_0 is v_{N-1}. Each polygon has a different number of vertices in its list, but each part shares exactly two vertices with two other polygons (the intersection vertices).
Output: One list of vertices containing such vertices v_j from the lists l_i that have $d_j \geq 0$.
The merging algorithm:
1. Choose a polygon S_i from the list of input polygons. Start from the first vertex in S_i. Go through l_i until the first intersection vertex v_j is found. Add v_j to the resulting list (which now contains only v_j).
2. Check the vertex v_{j+1} if its topological distance is positive. If so, continue forward, otherwise backward in l_i. Add each visited vertex to the resulting list until the next intersection vertex v_k is added. Delete the part S_i from the list of polygons.
3. Because v_k was the intersection vertex, either one of the parts in the list contains it (in such a case use this part and continue from step 2), or the list of parts is empty (v_k is the intersection vertex from step 1). In such a case, the algorithm terminates.

Fig. 13 The merging algorithm

4 Implementation and results

In the following examples we demonstrate the different behaviors of the presented methods. In the examples, only one method is used to morph all the parts, to show the behavior of each method, one at a time. Our algorithm is compared to the Carmel and Cohen-Or's algorithm and Sederberg and Greenwod's algorithm, which are well established correspondence-based algorithms.

In the first example we show the morphing of a spiral using the Perimeter growing (Fig. 14). This method provides more intuitive results than the Midpoint and the Projection growing. As can be seen in Fig. 14, the Perimeter growing method produces more intuitive results than both the Carmel and Cohen-Or algorithm and the Sedeberg and Greenwood algorithm, where many unwanted self-intersections occur.

Although the Projection growing method produces unexpected results when we desire a spiral form to grow out of the core, it can sometimes give interesting esthetic results, especially when we fill the polygons with color (because the overlapping parts are filled with the background color). Figure 15 shows an example of an in-between result obtained using the Projection growing method in the case of a snail's shell morphing.

The second example shows the case where the Projection growing method is more suitable for objects that are convex or nearly convex (Fig. 16). Here, a butterfly is morphed into an alien. The body of the butterfly is similar to that of the alien, which makes the objects good candidates for growing and absorption. One would expect the wings of the butterfly to be absorbed in the body of the alien, and the eyes (at the end of the antennae) to grow out from the antennae of the butterfly.

The example in Fig. 17 depicts long straight parts growing and absorption from a complex polygon. When the shape of a morphed part is long and more or less straight, the Projection, the Perimeter, and the Midpoint growing methods can provide comparable results, whereas the projection growing method is not suitable. The Carmel and Cohen-Or algorithm produces comparable results as well as the Sedeberg and Greenwood algorithm, both displaying a few self-intersections (Fig. 17).

The case shown in this particular example is not a case where one would naturally expect a growing behavior, because the octopus and the shark do not have any similar part. However, the growth of the octopus' tentacles is an aesthetically pleasing way to morph from the shark's belly into the octopus' tentacles without any intersections.

We have presented three examples showing the behavior of the three designed methods—the Perimeter growing, the Half-line growing and the Projection growing method. According to the results achieved, the user should use the Projection growing method for convex or nearly convex parts and the Perimeter growing or the Half-line growing for curved parts.

The examples show that our algorithm produces the expected results for the cases where some parts of one polygon are supposed to grow out of the other polygon or be absorbed into it. However, as we can see in the last example, the algorithm can also be used in some cases where growing is

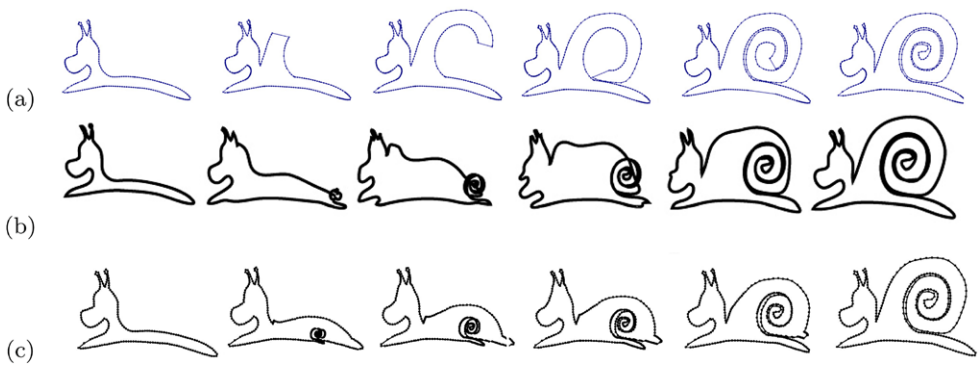

Fig. 14 The Perimeter growing method comparison: (**a**) the new Perimeter growing method, (**b**) the object being morphed using the Carmel and Cohen-Or algorithm, and (**c**) using the Sedeberg and Greenwood algorithm

not expected and it still produces acceptable results. An animation showing the described behavior of our algorithm is available at http://home.zcu.cz/~mmalkov/morphing.avi.

Our algorithm has also some limitations. It is not suitable for morphing of objects that are only rotated or translated, because these transformations cannot be represented as growth. Figure 18 shows how all the three algorithms behave for the case of a rotated fish. None of the algorithms produces a clear rotation, but the Carmel and Cohen-Or algorithm produces acceptable results with some user interaction.

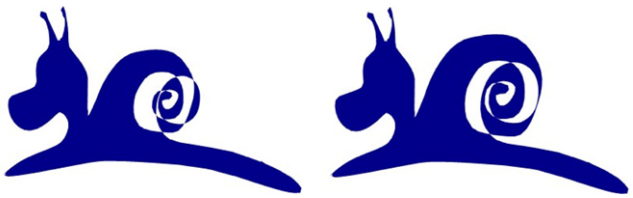

Fig. 15 Example of morphing for esthetic purposes

In the Carmel and Cohen-Or algorithm, the correspondence can be influenced by manually defining some corresponding vertices. Our algorithm offers the user an easier way to influence the correspondence—changing the mutual position of the polygons, while the results are easily predictable. Mostly, the best mutual position is a position where the area of the intersection is the maximally possible. Apparently two different choices of the core will lead to different results. This is demonstrated in Fig. 19 which shows two morphing sequences of the wings and the circle, each produced by a different mutual positions of the objects. The automatic decision of the mutual position remains a problem, as the maximal possible intersection is not always the best solution. But for this type of morphing problem, the user usually has a clear idea about which part of the object he/she wants to grow.

Figure 20 shows four morphing sequences of a circle and a curved shape to demonstrate the behavior of our algorithm when the overlapping area consists of more polygons. One of the polygons is chosen as a core (filled by blue color) and

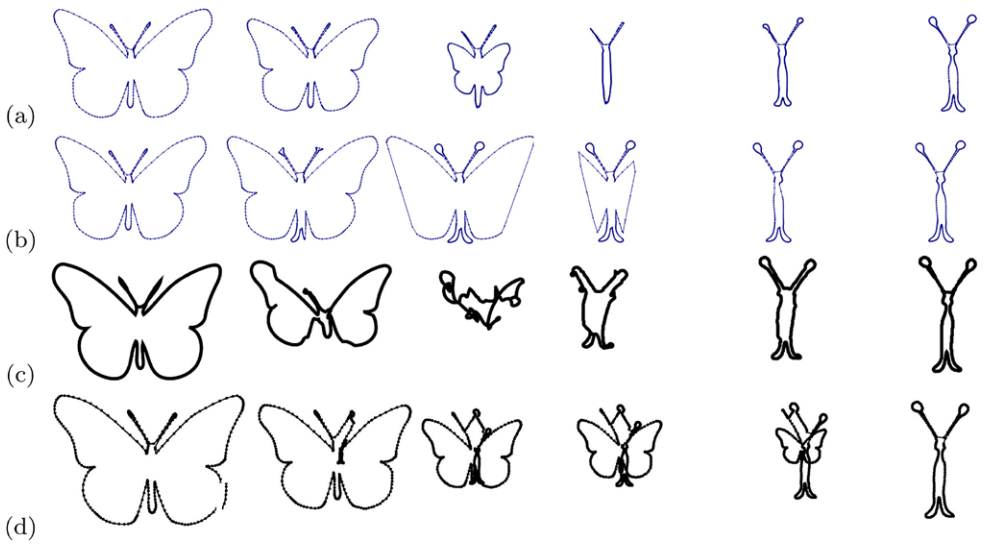

Fig. 16 Comparison of (**a**) the Projection growing method, (**b**) the Midpoint growing method, (**c**) the Carmel and Cohen-Or algorithm, and (**d**) the Sedeberg and Greenwood algorithm

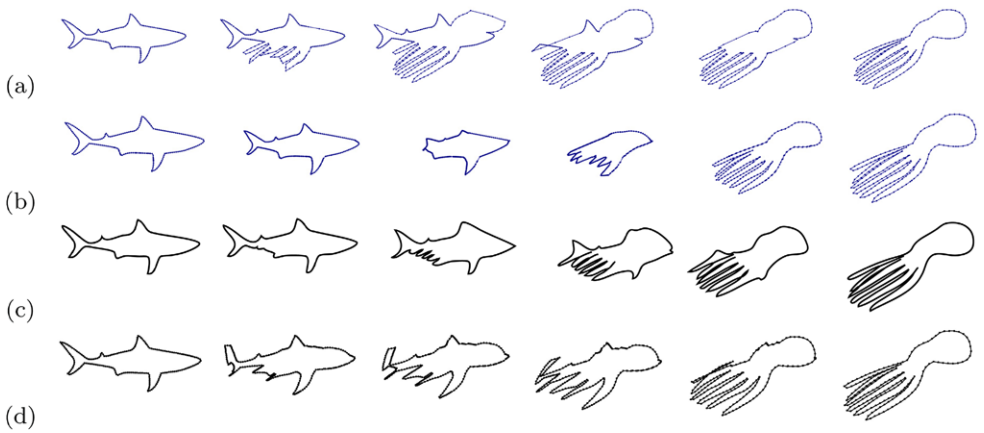

Fig. 17 Comparison of (**a**) the Projection growing method, (**b**) the Midpoint growing method, (**c**) the Carmel and Cohen-Or algorithm, and (**d**) the Sedeberg and Greenwood algorithm

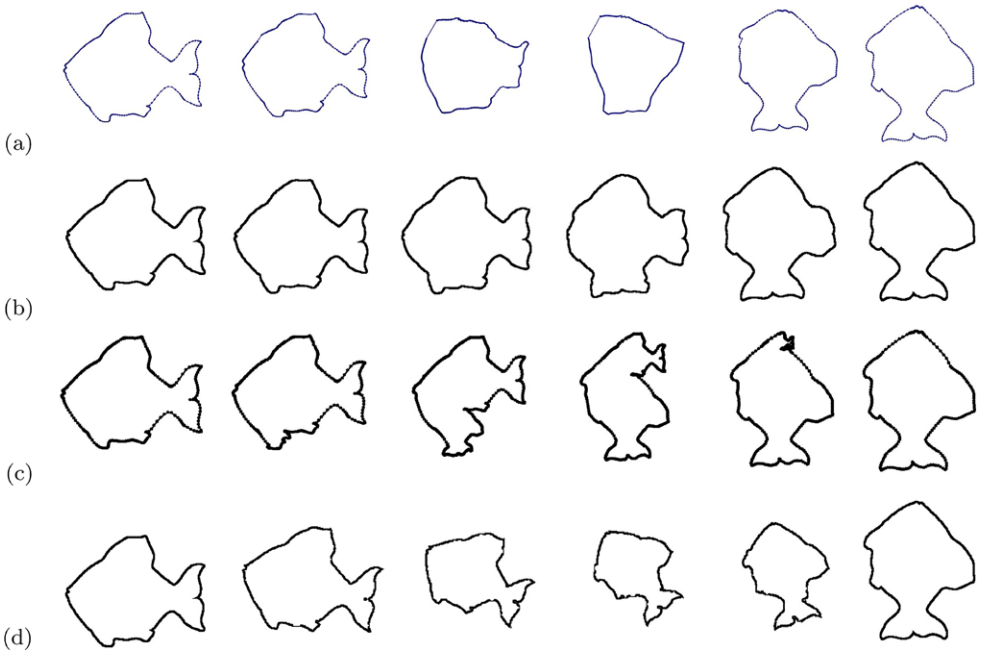

Fig. 18 Comparison of the methods for a case of a rotated object. (**a**) The Projection growing method, (**b**) the Carmel and Cohen-Or algorithm, (**c**) the Sedeberg and Greenwood algorithm, and (**d**) the Carmel and Cohen-Or algorithm with predefined four corresponding vertices

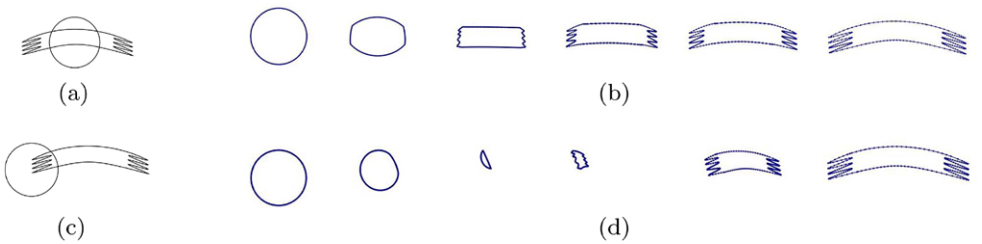

Fig. 19 The Projection growing method (different mutual positions of the polygons): (**a**, **c**) position of the polygons, (**b**, **d**) the resulting morphing sequence

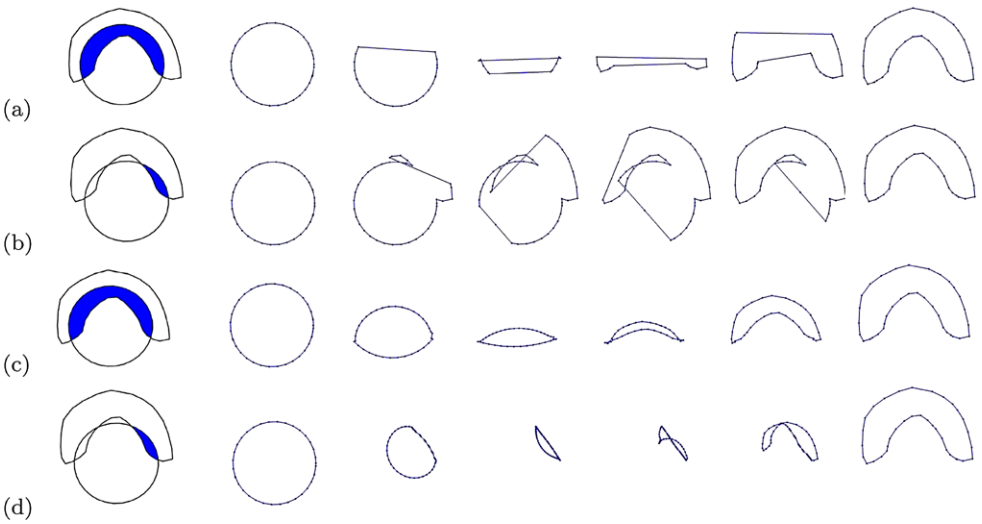

Fig. 20 The Perimeter (**a**, **b**) and the Projection (**c**, **d**) growing methods: the overlapping area consists of (**a**, **c**) one polygon, (**b**, **d**) more polygons—one is chosen.

the algorithm continues as if the overlapping area consisted of only one polygon. The core is chosen randomly or by the user. Such situation is not typical for growth-like morphing, because we would not expect growing behavior of a part that is connected with the core at several places.

5 Conclusions and future work

We have described a new algorithm for morphing of simple polygons which defines their correspondence by partially overlapping them. Our algorithm allows us to define

the correspondence in a more intuitive way than previously published methods.

We have shown three different techniques for the morph calculation, each of which is suitable for a different case of morphing, and we have demonstrated their use in various cases. We have also compared the results of our algorithm with two commonly used correspondence-based morphing methods. The experiments confirmed that our algorithm is suitable for instances of morphing, where the user expects some parts of the resulting polygon to grow out of or into the intersection. Our algorithm is not suitable for similar or nearly similar polygons with dissimilarities of the non-growth type (such as faces with different expressions, bending or straightening of fingers and polygons which are rotated, translated or scaled versions of each other), in which case the traditional algorithms should be used.

An obvious future work would be an extension of our approach to 3D (see [8] for preliminary results). Another future work lies in finding new strategies for the elementary growth problem together with some guidelines (or some automatic decision mechanism) as to which strategy to use in each distinct case of morphing. Last but not least, we would like to describe a framework that would integrate our growth-like morphing with the traditional approaches.

Acknowledgement We would like to thank Pavel Celba for providing implementation of the Sederberg algorithm.

References

1. Alexa, M.: Merging polyhedral shapes with scattered features. Vis. Comput. **16**(1), 26–37 (2000)
2. Alexa, M., Cohen-Or, D., Levin, D.: As-rigid-as-possible shape interpolation. In: Proceedings of SIGGRAPH 2000, pp. 157–164 (2000)
3. Carmel, E., Cohen-Or, D.: Warp-guided object-space morphing. Vis. Comput. **13**, 465–478 (1997)
4. Comninos, P.: Mathematical and Computer Programming Techniques for Computer Graphics. Springer, Berlin (2006)
5. Gomes, J., Darsa, L., Costa, B., Velho, L.: Warping and Morphing of Graphical Objects. Morgan Kaufmann, San Mateo (1999)
6. Johnstone, J.K., Wu, X.: Morphing two polygons into one. In: The 40th Annual Southeast ACM Conference (2002)
7. Kent, J.R., Carlson, R.E., Parent, W.E.: Shape transformation for polyhedral objects. Comput. Graph. **26**, 47–54 (1992)
8. Málková, M.: Core-based morphing algorithm for triangle meshes. In: SIGRAD 2008, vol. 34, pp. 39–46 (2008)
9. Sederberg, T.W., Gao, P., Mu, G., Wang, H.: 2-d shape blending: An intrinsic solution to the vertex path problem. Comput. Graph. **27**, 15–18 (1993)
10. Sederberg, T.W., Greenwood, E.: A physically based approach to 2-d shape blending. ACM SIGGRAPH **26**, 25–34 (1992)
11. Shapira, M., Rappoport, A.: Shape blending using the star-skeleton representation. IEEE Comput. Graph. Appl. **15**, 44–50 (1995)
12. Surazhsky, V., Gotsman, C.: Controllable morphing of compatible planar triangulations. ACM Trans. Graph. **20**, 203–231 (2001)
13. Surazhsky, V., Gotsman, C.: Intrinsic morphing of compatible triangulations. Int. J. Shape Model. **9**, 191–201 (2003)

Martina Málková received her Bachelor and Master's degree in Computer Science at the University of West Bohemia in Pilsen. Since 2008, she has been doing her postgraduate studies at the same university. So far, her research interests contain mainly morphing.

Jindřich Parus finished 5 years' M.Sc. studies at the University of West Bohemia in Pilsen in the field of Computer Graphics in 2003. He is a member of the Center of Computer Graphics and Data Visualization at the University of West Bohemia. He is interested in computer graphics, especially in the field of morphing. Nowadays he is a postgraduate student at the University of West Bohemia.

Ivana Kolingerová graduated in Computer Science in 1987, received her Ph.D. in Informatics and Computer Science in 1994 and habilitated in 2000. She works as Associate Professor in the Department of Computer Science and Engineering at the University of West Bohemia in Pilsen, Czech Republic. Her research interests include computer graphics and computational geometry, especially algorithms for triangulated models. Her international experience includes Denmark, USA, Slovenia and Austria.

Bedřich Beneš is Assistant Professor of Department of Computer Graphics Technology at Purdue University. His research interests include procedural modeling, artificial life, real-time rendering, and global illumination. Dr. Beneš received his Ph.D. from the Czech Technical University. He is a member of ACM SIGGRAPH, Eurographics, and IEEE.