**ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY**
Volume **14**, Number 2, 2011, 131–148

# Circular Interpolation for Morphing 3D Facial Animations

Mihai Daniel ILIE, Cristian NEGRESCU, Dumitru STANOMIR

Department of Telecommunications,
POLITEHNICA University of Bucharest, Romania

E-mail: mihai.iliedamaschin@yahoo.com
E-mail: {negrescu, dumitru.stanomir}@elcom.pub.ro

**Abstract.** In this paper, we present a novel shape interpolation method that gives very good results for mesh morphing 3D facial animations. Our method interpolates vertices using circle arcs, thus generating, in a preprocessing algorithm, a specific circle center in 3D cartesian space for each pair of vertices. The arc is more or less accentuated, depending on the coordinates which the algorithm chooses for the circle center. These are calculated depending on the relation between the two vertex normals that correspond to the initial vertex and target vertex, and thereby a novel vertex normal weighting method is presented. The efficiency of our algorithm was tested on several facial animation examples with most dissimilar geometries and with very different features, and the results verify the fact that our method gives undoubtedly better results than linear interpolation does, avoiding the undesired and unnatural shrinkage problem which occurs in the latter case. Moreover, our method strikes significantly better computational costs as compared to other non-linear shape interpolation methods developed so far, which is a major advantage.

**Key-words:** Circular Interpolation, Vertex Normals, 3D Facial Animations, Morphing, Blend Shapes.

## 1. Introduction

A problem that remains major in computer graphics is the construction and animation of 3D realistic human faces. For the past few decades, researchers have invested much interest in this field. Facial modeling and animation refers to means of visually representing virtual human or non-human faces on a computer and making them look and act natural. This task is a highly difficult one in the field of computer graphics animation, since the human eye is used to everyday interaction with other humans, and therefore is very trained to observe even the slightest irregularities in the facial movements of a character.

Mainly, there are two basic geometric ways of animating 3D faces. These work the same for both human and non-human face animations. The first one implies morphing the base head model between various morph targets. The 3D head model is stored as a set of triangles in 3D cartesian space, each triangle having a set of three vertices. For this method, the artist has to model or modify the basic head model in order to obtain several instances of the same head, for each needed facial expression. When animating, the vertices are interpolated between their initial position and the one that corresponds to the desired facial expression. Combining several such instances of the same model, at different moments of time, and at times even superimposing them, fluid and very natural facial animations could be obtained [1]. This method is often referred to as morph target animation, per-vertex animation, shape interpolation, or blend shapes animation. The other basic geometric way of animating 3D faces requires implementing muscle and skeletal systems to control areas of the face surface [2]. This method uses a model that virtually simulates the actions of real facial muscles. Unfortunately, the model does not work for any head structure and has to be manually defined in each case. As compared to the other method, it has the advantage that it requires less efforts from the artist. At all events, there are advantages to using morph target animation over muscle animation, because the artist modeler can individually create realistic facial movements over which he has absolute control, whereas in the other case he could be constrained by the limitations of the bone/muscle system, which could never model perfect and totally natural facial movements [2, 3, 4, 5]. Perfect results could not be obtained without the artist's intervention. For example, muscle systems can't simulate wrinkle animations or other detailed muscle animations, which are specific to some characters.

The morph target method aims to generate a sequence of intermediate 3D shapes between the two input 3D models (initial and target), giving the viewer the impression of the first object gradually transforming into the second one. This method is nowadays highly used in computer graphics. The shape interpolation method implies two steps [6]. The first one regards the problem of establishing a one-to-one correspondence between the vertices of the initial and target objects. The two objects must have the same connectivity. The second step takes care of the trajectory problem, i.e. determining the path each vertex should follow from its base position to its target one. The simplest and most often used way of performing the vertex interpolation is the linear interpolation, courtesy of its very pleasing computing time. Unfortunately, in the case of transformations that imply large scale rotations, linear interpolation doesn't strike good results, due to the shrinkage problem that appears. In such situations, other interpolation methods have to be addressed, so as for the metamorphosis to look natural and realistic. These methods have to be highly dependent on the object geometry. In this paper, we consider that the isomorphic correspondence problem has been taken care of, using one of the existing algorithms [7, 8] and we shall focus on the trajectory problem.

Our paper presents a novel interpolation method for mesh morphing in the case of 3D facial animations, by using arcs of different circles for each pair of vertices. For each such pair of vertices, a specific circle center is determined, and then the vertex follows the path of a circle arc between its initial and final position. The resulting trajectory could vary from a half-circle to something very close to a line, depending on the chosen position of the circle center. That is because, in some transformations, in order for the metamorphosis to look natural, some vertices of the mesh have to move linearly, whereas others have to move on a very obvious circle arc. To determine the circle center coordinates for each pair of vertices, our algorithm first determines the vertex normal which corresponds to the initial vertex and then calculates the vertex normal that corresponds to its position on the target mesh. The center position is determined depending on the relation between the two vertex normals. Our results and tests on various very different examples of 3D facial animations prove that the method presented in this paper does not bring about the eye soring effects that so often happen in the case of linear shape interpolation, such as the shrinkage problem or local self-interaction. The major advantage our algorithm brings is that it provides consistently better visual results than linear interpolation, whereas it also strikes significantly better computing times than other non-linear mesh interpolation methods that have so far been developed. Virtually, after the very short center-determination preprocessing part, the metamorphosis is performed in real-time.

## 2. Related Work

Researchers have invested much interest in the mesh morphing area of computer graphics, laying great emphasis on both the correspondence and trajectory problems [9]. As to shape interpolation, some of the first attempts were done by Sederberg and Gao [10]. Their method proposes a way of interpolating polygonal surfaces in 2D space by trying to preserve the values of the edges and dihedral angles, but it has the disadvantage that the results vary depending on the order in which the edges and angles are computed by the algorithm. Thereby, this method is regarded mostly as an edge interpolation method. It was also extrapolated to 3D cases, with similar results [11]. Surazhsky [12] proposes a method that adresses the use of mean value interpolation and barycentric coordinates to interpolate between isomorphic triangulations, while the mesh is regarded as a triangulated graph.

Kaneko [13] presents a method that morphs 3D models by first defining skeletal systems to describe the main features of the object's geometry. Nevertheless, it requires user interaction, since the patches in which the model has to be divided are manually selected. Lipman [14] proposes a shape interpolation scheme which minimizes elastic distortion by using an interactive editing mechanism that preserves local differential properties. The used differential coordinates are rigid-motion invariant.

Sheffer and Kraevoy [15] introduce a different way of describing local shape representation, by using pyramid coordinates. These coordinates describe the local proprieties of the mesh around each vertex, by taking into account the edge lengths and the dihedral angles. However, the algorithm requires user interaction, since a set of control vertices has to be defined by the user.

Alexa [16] proposes preserving the local features of a 3D shape by describing them in a differential way, and thus approaches the trajectory problem in 3D morphing by interpolating Laplacian coordinates. Xu [17] presents another interpolation method that is different in the fact that it interpolates vector fields in gradient domain instead of vertex positions directly. Each intermediary shape is computed by solving a set of Poisson equations defined on a domain mesh. Hu [18] approaches the problem of 3D morphing by using strain fields to interpolate between 3D shapes. *Strain* is a quantity used in mechanics to describe deformation, and therefore, if only a rigid body motion happens, the strain field is 0. The Lagrange strain field provides good results for 3D shape interpolation. This method is extrapolated from the 2D strain field interpolation method [19], proposed by the same authors.

The results these methods produce are very good, but the computational times are extremely long and therefore these methods could be hardly used in large-scale productions, since they are still far from real-time processing.

## 3. Framework of Circular Interpolation Morphing

In the case of similar shapes, where no major deformation or rotation occurs, linear shape interpolation produces acceptable results. Nonetheless, if the metamorphosis implies large scale rotations or partly bending the object, linear interpolation unavoidably leads to undesired shrinkage of the intermediate shapes during the morphing process. Therefore, we chose to create an algorithm that provides smooth variation of the object volume during the transformation. The object metamorphosis could be devided into two kinds of transformations: rigid body motion and deformation. The rigid body motion refers to translation and rotation transformations. If only a rigid body motion takes place, the resulting volume is equal to the initial one. For example, if we rotate a cube around its own center, its volume must remain unchanged during the rotation, but in this example linear shape interpolation fails. Deformation, though, refers to the way a vertex changes its position relative to its neighbor vertices, not to its initial position. Unlike other so far presented methods [17, 18], which tend to treat the two kinds of transformations separately, our method considers object deformation to be a sum of rotations applied on different parts of the 3D mesh. Using a different circle arc for each vertex to describe its transition, we could treat rotation and deformation together. For the vertices that are to be solely translated, the circle center is chosen in such a way that the resulting trajectory would be a very smooth arc, very close to a line, whereas for a rotation the arc would be very obvious. The deformation could be approximated with very good results if we consider each vertex on the deforming part of the object to move on a different arc, some on a higher-angle arc, some on a lower-angle arc. In other words, with our method, the object mesh is dissected into very small mesh parts, each one undergoing its own different rotation. As a result, the object suffers a very complex transformation, some of its parts being rotated, some deformed, some only translated, and others left unmoved. Since it is based on circle arc trajectories, we chose to call our method circular shape interpolation. We could now define the transformation plane incident to a shape metamorphosis as the plane in which most of the vertex rotations which approximate the deformation of a specific mesh part take place. This plane could be, in 3D Cartesian space, the XOY plane, the XOZ plane, the YOZ plane, or any other plane obtained by rotating one of these three planes with a varied angle. In the case of 3D facial animations, several experimental results have shown us that the metamorphosis could be very well represented by only one transformation plane for the whole 3D object. For other cases, the object is devided into a few mesh parts and the following algorithm is applied for each such part. In this paper, we shall consider the case of 3D facial animations. Thence, our method could be described by the following main steps:

**1.** First, define a total number of possible transformation planes, then perform for each such transformation plane the following sub-steps:

– Determine, for each vertex, the vertex normals that correspond to its positions on the initial and target mesh, using our novel vertex normal determination method that produces best results for our goal;

– Compute for each pair of vertices the coordinates of their corresponding circle center in 3D space, depending on the specific geometrical features of each such pair;

**2.** Develop a real-time morphing algorithm that moves each vertex on a circle arc, using the given initial and final vertex positions and their computed circle center.

**3.** Using the algorithm at step 2, process the shape of the resulting intermediary object at the exact middle of the transformation for the case of each transformation plane defined at step 1.

**4.** Compare the volume of each intermediary shape obtained at step 3 to the average between the initial object volume and the target object volume, and then choose the one which is the closest to the average, thus finding out which transformation plane is the right one.

**5.** Once the right transformation plane has been determined, the corresponding set of circle centers is passed on to the morphing algorithm described in step 2, which performs in real time the metamorphosis from the initial to the final shape.

Therefore, our method is composed of two major parts: the preprocessing algorithm (steps 1-4), with very short computational times, and the actual morphing (step 5), which virtually works in real time and interpolates using the computed set of circle centers.

In the following subsections, we shall present each of the algorithm steps in detail.

### 3.1. Computing the Signed Volume of a closed triangular mesh

In order for the algorithm to choose which transformation plane is best, we have to develop a method for determining the volume of a 3D triangle mesh object. To achieve that, we use a very elegant concept used in vector analysis for computer graphics, the "signed volume", which could be treated as a result of a triple scalar product [20].

The signed volume could have a positive value (if the three vectors form a left-handed set) or a negative value (if the three vectors form a right-handed set). To determine the volume of a closed triangular mesh, for all the triangles on the mesh we determine the volume of the tetrahedron that is formed from the current triangle and the origin (0,0,0).

To calculate this volume, we use the triple scalar product. If we algebrically sum up all these volumes, we should get the following formula for the volume of a triangular mesh of a closed 3D object:

$$V_{obj} = \sum_{i=0}^{N-1} \frac{\vec{a_i} \cdot (\vec{b_i} \times \vec{c_i})}{6} \tag{1}$$

where $N$ is the object's total number of triangles, and $\vec{a_i} = (x_{T_{1i}}\text{-}0,\ y_{T_{1i}}\text{-}0,\ z_{T_{1i}}\text{-}0)$ , $\vec{b_i} = (x_{T_{2i}}\text{-}0,\ y_{T_{2i}}\text{-}0,\ z_{T_{2i}}\text{-}0)$ , respectively $\vec{c_i} = (x_{T_{3i}}\text{-}0,\ y_{T_{3i}}\text{-}0,\ z_{T_{3i}}\text{-}0)$. Here, $T_{1i}$, $T_{2i}$ and $T_{3i}$ are the i indexed triangle's three vertices.

### 3.2. Our Vertex Normal determination method

Another concept that we use in this paper is the vertex normal concept. Vertex normal is an essential attribute for point based smoothing and mesh simplification in the field of computer graphics. It is also commonly used for illumination techniques, such as Phong shading [21]. The vertex normal which is proper to a specific vertex of a triangular mesh is obtained by averaging the normals of all the triangles that are incident in this vertex, by using different weights. Therefore, vertex normals express important local proprieties of a 3D surface. The vertex normal accuracy depends on the algorithm used for weighting the normals which are to be averaged. Several such algorithms have been developed [22], each of them focusing on different problems vertex normals may be useful for. The main influencing factors when weighting are the areas of the neighboring triangles and the incident angles. For example, in the case of global illumination and Phong shading, when interpolating the way light interacts with the polygonal surface, it is very important to preserve hard edges. Considering $T_{1i}$, $T_{2i}$ and $T_{3i}$ are the three vertices of the i indexed triangle on the mesh, the triangle's normal is calculated as:

$$\vec{n}_{ti} = \frac{\overrightarrow{(T_{2i} - T_{1i})} \times \overrightarrow{(T_{3i} - T_{1i})}}{\|\overrightarrow{(T_{2i} - T_{1i})} \times \overrightarrow{(T_{3i} - T_{1i})}\|} \tag{2}$$

Gouraud's method [22] simply averages the normals of the surrounding triangles:

$$\vec{n} = \frac{\sum\limits_{i} \vec{n}_{ti}}{\|\sum\limits_{i} \vec{n}_{ti}\|}, \tag{3}$$

where $\vec{n}_{ti}$ are the normals of the neighboring triangles incident to our vertex of interest. Other methods also need the $\alpha_i$ angle, which is the $i$ triangle's angle incident to the vertex of interest, and/or the $A_{ti}$ area of the triangle (Thurmer, Taubin and Max [22]):

$$\vec{n} = \frac{\sum_i \alpha_i \cdot \vec{n}_{ti}}{\|\sum_i \alpha_i \cdot \vec{n}_{ti}\|} \quad or \quad \vec{n} = \frac{\sum_i A_{ti} \cdot \vec{n}_{ti}}{\|\sum_i A_{ti} \cdot \vec{n}_{ti}\|} \tag{4}$$

We take the use of vertex normals a step forward, using this powerful mesh attribute for our shape interpolation method. For our algorithm, though, the vertex normal determination methods presented above are not satisfactory, since, in our case, we have to compute the resulting vertex normals by taking into account both initial and target meshes. So, we present a novel weighting method for vertex normal determination. The vertex normal on the initial mesh is also influenced by its corresponding mesh on the target object, whereas the vertex normal on the target mesh is also influenced by its corresponding mesh part on the initial object. After several experiments, we have decided over the following formulas, which provide very useful results for what we intend to use vertex normals for:

$$\vec{n} = \frac{\sum_i \left[\frac{A_{ti}}{A_{max1}} \cdot \vec{n}_{ti} \cdot \arccos\left(\frac{\vec{n}_{ti} \cdot \vec{n'}_{ti}}{\|\vec{n}_{ti}\| \cdot \|\vec{n'}_{ti}\|}\right) \cdot \frac{1}{\pi}\right]}{\left\|\sum_i \left[\frac{A_{ti}}{A_{max1}} \cdot \vec{n}_{ti} \cdot \arccos\left(\frac{\vec{n}_{ti} \cdot \vec{n'}_{ti}}{\|\vec{n}_{ti}\| \cdot \|\vec{n'}_{ti}\|}\right) \cdot \frac{1}{\pi}\right]\right\|} \tag{5}$$

for the vertex normal associated to the initial vertex, and

$$\vec{n'} = \frac{\sum_i \left[\frac{A'_{ti}}{A_{max2}} \cdot \vec{n}'_{ti} \cdot \arccos\left(\frac{\vec{n}_{ti} \cdot \vec{n'}_{ti}}{\|\vec{n}_{ti}\| \cdot \|\vec{n'}_{ti}\|}\right) \cdot \frac{1}{\pi}\right]}{\left\|\sum_i \left[\frac{A'_{ti}}{A_{max2}} \cdot \vec{n'}_{ti} \cdot \arccos\left(\frac{\vec{n}_{ti} \cdot \vec{n'}_{ti}}{\|\vec{n}_{ti}\| \cdot \|\vec{n'}_{ti}\|}\right) \cdot \frac{1}{\pi}\right]\right\|} \tag{6}$$

for the vertex normal associated to the target vertex.

In the above formulas, the $\vec{n}_{ti}$ vectors represent the normals of the neighboring triangles on the initial mesh, and the $\vec{n'}_{ti}$ vectors represent the normals of the neighboring triangles on the target mesh, all relative to the vertex of interest. $A_{max1}$ and $A_{max2}$ denote the area of the largest triangle out of all the neighboring triangles, in the case of the initial and the target mesh. The $A'_{ti}$ areas denote the areas of the corresponding triangles on the target mesh. As a result, the $\frac{A_{ti}}{A_{max1}}$ and $\frac{A'_{ti}}{A_{max2}}$ expressions vary between 0 and 1. Several experiments have shown us that, for our shape interpolation algorithm, it is compulsory not to take into account, when averaging, those triangle normals that are the same in the initial and target mesh (or differ just a little). Therefore, we added a new influence factor in the above formulas. Using the dot product, for each triangle, the angle between its initial and final normal is computed. The $\arccos\left(\frac{\vec{n_{ti}} \cdot \vec{n'}_{ti}}{\|\vec{n_{ti}}\| \cdot \|\vec{n'}_{ti}\|}\right) \cdot \frac{1}{\pi}$ expression returns 0 if the angle is 0 and 1 if the angle is $\pi$. The influence of the triangle normal increases with the result returned by this expression. Figure 1 demonstrates the efficiency of this method.

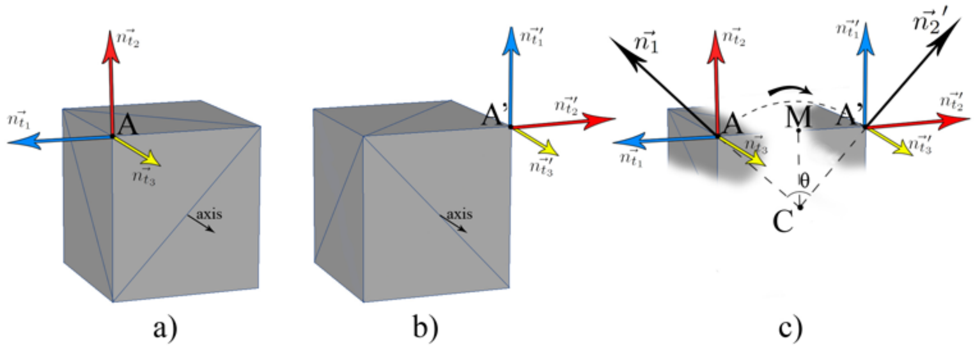

**Fig. 1.** Our Vertex Normal determination method applied on a simple object.

Figure 1 shows the example of a cube that, in its final shape (Fig. 1b), is rotated with a $\pi/2$ angle around its axis as compared to its initial instance (Fig. 1a). Its only transformation is this rotation. Our vertex of interest is the $A$ vertex on the figure, and its final position is denoted as $A'$. The normals associated to the three neighboring triangles are $\vec{n}_{t1}$, $\vec{n}_{t2}$ and $\vec{n}_{t3}$ for the initial shape and $\vec{n'}_{t1}$, $\vec{n'}_{t2}$ and $\vec{n'}_{t3}$ for the final one. As seen in Fig. 1c, the influence of the $\vec{n}_{t3}$ and $\vec{n'}_{t3}$ normals is most undesired, since the cube's rotation takes place solely in the plane defined by $\vec{n}_{t1}$ and $\vec{n}_{t2}$. When averaging, our algorithm resolves this issue by weighting every triangle normal with the angle formed between its initial and final states. In this case, the angle between $\vec{n}_{t3}$ and $\vec{n'}_{t3}$ is 0, so these normals have absolutely no influence when calculating the two resulting vertex normals. The vertex normals processed with our method are $\vec{n}_1$ and $\vec{n}_2$. Since the $\vec{n}_1$ and $\vec{n}_2$ vectors are not always coplanar, we project the $\vec{n}_2$ vector on the plane defined by the $A'$ point with the fixed vector $\vec{n}_1$ ($\vec{n}_1$ is fixed by the $A$ point). The result is denoted as $\vec{n'}_2$, this consisting in another step of our algorithm. In the particular case of Fig. 1, $\vec{n'}_2 = \vec{n}_2$. The $C$ point is the circle center which defines the resulting trajectory (here, it coincides with the intersection between $\vec{n}_1$ and $\vec{n'}_2$), while $\theta$ is the angle between $\vec{n}_1$ and $\vec{n'}_2$.

### 3.3. Computing the set of circle centers

Let $P$ and $P'$ be the initial and final positions for an arbitrary vertex of the object. We want to determine a specific circle center in 3D space, so as to trace the circle arc trajectory from $P$ to $P'$. To do that, we first need to find the intersection of the $\vec{n}_1$ and $\vec{n'}_2$ vertex normals. The fact that these two vectors are coplanar helps a lot. For this problem, we verify whether or not the two lines are parallel. If $\vec{n}_1 \times \vec{n'}_2 = 0$, then $\vec{n}_1$ and $\vec{n'}_2$ are parallel and the two lines do not intersect. In this case, the interpolation between $P$ and $P'$ is done linearly. This mainly happens when the only transformation for the mesh part that includes this vertex is a translation. If the lines do intersect, we denote the intersection point by $I$. $I$ is determined by equating the two lines and solving the system, knowing that both lines lie in the same plane:

$$\begin{cases} \dfrac{x - x_P}{x_{n_1}} = \dfrac{y - y_P}{y_{n_1}} = \dfrac{z - z_P}{z_{n_1}} \\[2ex] \dfrac{x - x_{P'}}{x_{n'_2}} = \dfrac{y - y_{P'}}{y_{n'_2}} = \dfrac{z - z_{P'}}{z_{n'_2}} \end{cases} \tag{7}$$

where $(x_P,\ y_P,\ z_P)$ are the coordinates of $P$, $(x_{P'},\ y_{P'},\ z_{P'})$ are the coordinates of $P'$, $\vec{n}_1 = (x_{n_1}, y_{n_1}, z_{n_1})$ and $\vec{n'}_2 = (x_{n'_2}, y_{n'_2}, z_{n'_2})$.

The $\theta$ angle between $\vec{n}_1$ and $\vec{n'}_2$ is also calculated, using the dot product:

$$\theta = \arccos\left(\frac{\vec{n}_1 \cdot \vec{n'}_2}{\|\vec{n}_1\| \cdot \|\vec{n'}_2\|}\right) \tag{8}$$

In the next step, we determine the coordinates of the $C$ circle center. We denote by $M$ the middle of the $PP'$ segment. The center should be situated somewhere on a line which is included in the $IPP'$ plane, is perpendicular to the $PP'$ line and also passes through $M$. Then we add another condition: the angle formed between the $CP$ and $CP'$ segments must be equal to $\theta$. This leads to two possible solutions, because $C$ could be located on either side of the $PP'$ line.

The free vector which describes the direction that is perpendicular to the $PP'$ line and is included in the $IPP'$ plane is denoted as $\vec{p} = (x_p, y_p, z_p)$ and we determine it by calculating two cross products:

$$\vec{p} = \overrightarrow{PP'} \times (\overrightarrow{PI} \times \overrightarrow{PP'}) \tag{9}$$

We formulate the equation of the $MC$ line as:

$$\frac{x - x_M}{x_p} = \frac{y - y_M}{y_p} = \frac{z - z_M}{z_p} \tag{10}$$

The other condition is:

$$\overrightarrow{CP} \cdot \overrightarrow{CP'} = \|\overrightarrow{CP}\| \cdot \|\overrightarrow{CP'}\| \cdot \cos(\theta), \tag{11}$$

The radius of the circle is:

$$r = CP = CP' = \frac{MP}{\sin\left(\dfrac{\theta}{2}\right)}, \tag{12}$$

which leads to:

$$CM = \sqrt{MP^2 - r^2} \tag{13}$$

Knowing the equation of the $CM$ line, the distance between $C$ and $M$, and the coordinates of the $M$ point, we solve the system and reach a second order equation that provides two possible solutions for the $C$ center. To find out on which side of the $PP'$ line $C$ is situated, we have to study the proprieties of the local geometry surrounding the $P$ and $P'$ vertices. For that, we propose the following approach:

We already know that $\overrightarrow{IP}$ and $\vec{n}_1$ share the same direction and that $\overrightarrow{IP'}$ and $\vec{n'}_2$ share the same direction. We want to find out whether or not they also share the same sense (positive/negative). If $\overrightarrow{IP}$ and $\vec{n}_1$ share the same sense and $\overrightarrow{IP'}$ and $\vec{n'}_2$ share the same sense, this means the $C$ point must be located on the same side of the $PP'$ line as $I$ is. If not, the $C$ point must be situated on the other side of the $PP'$ line than $I$. Out of the two solutions we reached for the $C$ center, the one which is located the closest to $I$ is on the same side of $PP'$ as $I$ is, whereas the one that is situated the farthest from $I$ is on the other side. This method produces the expected results for our algorithm to work properly. Using the same notations, Fig. 2 demonstrates how the $C$ circle center is determined in various examples, thus proving the efficiency of the center determination method presented so far.

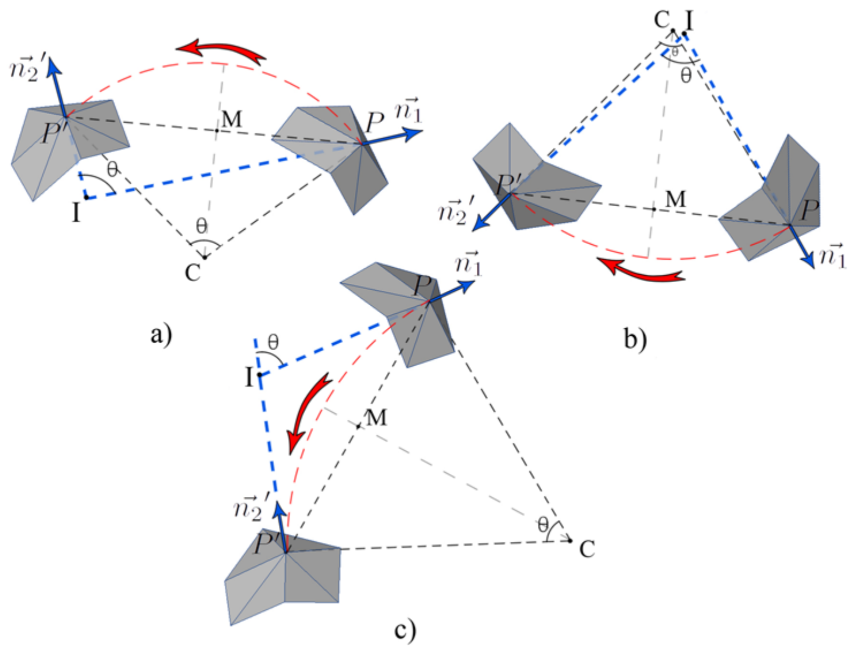

**Fig. 2.** Our center determination method applied for various situations.

In the first two examples shown in Fig. 2, the resulting $C$ point is located on the same side of $PP'$ as $I$, since both $\vec{IP}$ and $\vec{IP'}$ vectors share the same sense and direction with their corresponding vertex normals. In the case of the last example from Figure 2 though, our method places the center on the other side of the $PP'$ line. If it had been on the same side as $I$, the resulting trajectory would have produced a visually incorrect metamorphosis animation, which is most undesirable. In the figure, we chose to show only one very little mesh part of the object, while hiding the rest irrelevant mesh part, in order to efficiently describe the behaviour of the $(P, P')$ pair of vertices.

There is still one cardinal aspect to be discussed. Let $N_v$ be the number of vertices on the mesh and $i = \overline{1..N_v}$. So far, we have presented the method for one arbitrary $(P_i, P_i')$ pair of vertices on the mesh, for which we have determined the $\theta_i$ angle and the $C_i$ associated circle center. Now, we have to see how to establish the final set of circle centers. A key fact is that not all vertices on the mesh store in their vertex normals important information regarding the manner the interpolation has to be done. Considering $\alpha_i$ is the angle between the $\vec{n_{1i}}$ vertex normal and the transformation plane proper to our shape transformation, only those vertices with $\alpha_i \leqslant \frac{\pi}{180}$ have relevant vertex normals. We call these specific vertices directory vertices. All other vertices have to "steal" their $\theta$ value from the closest such directory vertex on the mesh. The algorithm that accomplishes this task goes as follows:

We run through all the vertices of the 3D object. For each vertex, we check if its $\alpha_i > \frac{\pi}{180}$, and if this happens we save a linked list of all its first-order neighboring vertices, then another linked list of all its second-order neighboring vertices, and so on and so forth. All these lists are also linked in a major linked list of lists that properly describes the neighboring proprieties of our vertex. For a short explanation of the concept used, the second-order neighboring vertices are all vertices on the mesh for which the shortest way to our vertex of interest passes through only one intermediary vertex (or through two edges), whereas the first-order neighboring vertices are connected to our vertex through only one edge. Running through all these neighbor lists, when we encounter a directory vertex, we stop and steal its $\theta$ value. Therefore, for a $P_i$ vertex with $\alpha_i > \frac{\pi}{180}$, if its closest directory vertex is denoted as $P_d$, then $\theta_i$ becomes $\theta_d$. In the particular case of $\theta_d = 0$, the interpolation is done linearly for both $P_d$ and $P_i$ vertices.

For each such non-directory vertex, once the new $\theta$ angle is established, a new circle center must be computed depending on the transformation plane. Our circular interpolation is always done, as discussed earlier, relative to a specific transformation plane, which comes as a known parameter to our interpolation function. Mainly, the transformation plane could be any of the three basic planes in the Cartesian system (XOY, XOZ, YOZ), or any plane obtained by rotating any of these planes with a certain angle. For planes that do not coincide with the three basic ones, we use the following method: we rotate the object till the transformation plane coincides with one of the three basic planes, we then retain the angle value and the axis we used for this rotation, we apply our interpolation algorithm using the basic plane we obtained, and eventually we rotate the modified object back to the way it was. Doing this, we only have to discuss here the case of the three basic transformation planes: XOY, XOZ and YOZ. Thereby, for each non-directory vertex we determine three points: $C_{i,XOY}$, $C_{i,XOZ}$ and $C_{i,YOZ}$, each corresponding to one of the three basic Cartesian planes. These three centers are determined in the exact same way we determined the $C_i$ circle center, only that instead of locating the resulting center in the $I_iP_iP_i'$ plane, we place it in a plane which includes the $P_i$ point and is parallel to the transformation plane (XOY, XOZ or YOZ). Also, instead of the $P_i'$ point we use the projection of the $P_i'$ point on this plane, which we denote as $P_i''$. If the transformation plane is YOZ, we get $P_i'' = (x_{P_i}, y_{P_i'}, z_{P_i'})$. If the transformation plane is XOY, we get $P_i'' = (x_{P_i'}, y_{P_i'}, z_{P_i})$. If the transformation plane is XOZ, we get $P_i'' = (x_{P_i'}, y_{P_i}, z_{P_i'})$. Another difference in the algorithm is that in the end, when choosing which one of the two resulting solutions is the right one, we select the one which is located the closest to the initial $C_i$. We also determine the following three radius parameters: $r_{i,YOZ} = \overline{P_iC_{i,YOZ}} = \overline{P_i''C_{i,YOZ}}$, $r_{i,XOY} = \overline{P_iC_{i,XOY}} = \overline{P_i''C_{i,XOY}}$ and respectively $r_{i,XOZ} = \overline{P_iC_{i,XOZ}} = \overline{P_i''C_{i,XOZ}}$.

### 3.4. Our Circle Arc interpolation method

Considering $(P_i, P_i')$ is one arbitrarly chosen pair of vertices on the mesh, we denote our interpolation function as $P_{i,circ}(t, P_i, P_i')$, where $t \in [0,1]$, $P_{i,circ}(0, P_i, P_i') = P_i$ and $P_{i,circ}(1, P_i, P_i') = P_i'$.

We now present our circular interpolation algorithm for the directory vertices. In a very first step, we determine linear interpolation point coordinates, as follows:

$$x_{lin}(t, x_{P_i}, x_{P_i'}) = \left\{ \begin{array}{ll} x_{P_i} + t \cdot (x_{P_i'} - x_{P_i}) & \text{if } x_{P_i} < x_{P_i'} \\ x_{P_i} - t \cdot (x_{P_i} - x_{P_i'}) & \text{otherwise} \end{array} \right\} \tag{14}$$

$$y_{lin}(t, y_{P_i}, y_{P_i'}) = \left\{ \begin{array}{ll} y_{P_i} + t \cdot (y_{P_i'} - y_{P_i}) & \text{if } y_{P_i} < y_{P_i'} \\ y_{P_i} - t \cdot (y_{P_i} - y_{P_i'}) & \text{otherwise} \end{array} \right\} \tag{15}$$

$$z_{lin}(t, z_{P_i}, z_{P_i'}) = \left\{ \begin{array}{ll} z_{P_i} + t \cdot (z_{P_i'} - z_{P_i}) & \text{if } z_{P_i} < z_{P_i'} \\ z_{P_i} - t \cdot (z_{P_i} - z_{P_i'}) & \text{otherwise} \end{array} \right\} \tag{16}$$

We introduce:

$$d_i(t) = r_i - \tag{17}$$

$$-\sqrt{(x_{C_i}^2 - x_{lin}(t, x_{P_i}, x_{P_i'})^2) + (y_{C_i}^2 - y_{lin}(t, y_{P_i}, y_{P_i'})^2) + (z_{C_i}^2 - z_{lin}(t, z_{P_i}, z_{P_i'})^2)}$$

Let $P_{i,lin}(t, P_i, P_i') = ((x_{lin}(t, x_{P_i}, x_{P_i'}), y_{lin}(t, y_{P_i}, y_{P_i'}), z_{lin}(t, z_{P_i}, z_{P_i'})))$. The line defined by $P_{i,lin}(t, P_i, P_i')$ and $C_i$ could be expressed as follows:

$$\frac{x - x_{lin}(t, x_{P_i}, x_{P_i'})}{x_{C_i} - x_{lin}(t, x_{P_i}, x_{P_i'})} = \frac{y - y_{lin}(t, y_{P_i}, y_{P_i'})}{y_{C_i} - y_{lin}(t, y_{P_i}, y_{P_i'})} = \frac{z - z_{lin}(t, z_{P_i}, z_{P_i'})}{z_{C_i} - z_{lin}(t, z_{P_i}, z_{P_i'})} \tag{18}$$

$P_{i,circ}(t, P_i, P_i')$ shall be found by determining the point which is located on this line, at the $d_i(t)$ distance from the $P_{i,lin}(t, P_i, P_i')$ point. The system is:

$$\left\{ \begin{array}{l} P_{i,circ}(t, P_i, P_i') \in C_iP_{i,lin}(t, P_i, P_i') \\ \\ d_i(t) = distance(P_{i,circ}(t, P_i, P_i'), P_{i,lin}(t, P_i, P_i')) \end{array} \right\} \tag{19}$$

The system has two solutions out of which we shall always choose the one that is located the farthest from $C_i$.

For the non-directory vertices, we apply the very same algorithm, only that in this case we use the $C_{i,XOY}$, $C_{i,XOZ}$ and $C_{i,YOZ}$ centers, depending on the transformation plane (XOY, XOZ or YOZ) and on the corresponding $P_i''$ projection point. If the transformation plane is XOY, the interpolation is done linearly on the OZ axis, therefore $z_{circ}(t, z_{P_i}, z_{P_i'}) = z_{lin}(t, z_{P_i}, z_{P_i'})$, while on the other two axes it is done circularly, solving a system similar to the one previously presented. This means $x_{circ}(t, x_{P_i}, x_{P_i'})$ and $y_{circ}(t, y_{P_i}, y_{P_i'})$ are determined using the $P_i$, $P_i''$ and $C_{i,XOY}$ points. Similarly, for the XOZ transformation plane the interpolation is linear on the OY axis and circular for the other two, and for the YOZ transformation plane the interpolation is linear on the OX axis.

### 3.5. Choosing the right transformation plane

Considering a finite set of test transformation planes (the three basic planes and some other ones obtained, as discussed before, by rotating the basic planes), we wish to verify which of these planes is best suited for each particular metamorphosis case. The finite set of transformation planes is denoted as $D$, and its elements as $D_k$, with $\overline{k = 1..N_D}$, and $N_D$ is the number of elements. Thereby, we take the following steps:

1. We calculate the volumes for the initial and final shapes and then determine the average volume as:

$$V_{Ref} = \frac{V_{initial} + V_{final}}{2} \tag{20}$$

2. We compute the intermediary 50% object shape in the case of each transformation plane, using the circular interpolation method presented in the previous subsection. For a 50% metamorphosis, we know that $t = 0.5$.

3. We determine the volumes of these 50% intermediary shapes in the case of each transformation plane. We denote them as $V_{D_k}(0.5)$, where $\overline{k = 1..N_D}$. If $D_k = XOY$, the volume is $V_{XOY}(0.5)$. Similarly, we define $V_{XOZ}(0.5)$ and $V_{YOZ}(0.5)$.

4. A primal fact used in all approaches regarding shape metamorphosis is that the volume should vary linearly during the metamorphosis. If we are dealing with a rigid body motion, the volume should stay unchanged during the transformation. Otherwise, its variation should be described by something very close to a line between its initial and final values. In this case, we determine:

$$V_{Res} = min(|V_{Ref} - V_{D_1}(0.5)|, |V_{Ref} - V_{D_2}(0.5)|, ..., |V_{Ref} - V_{D_{N_D}}(0.5)|) \tag{21}$$

as being the volume that is the closest to $V_{Ref}$. Thereby, the transformation plane which corresponds to this volume is the best suited transformation plane.

## 4. Results

We have tested the efficiency of our algorithm on several facial animation examples with most dissimilar geometries and with very different features, and the results verify the fact that our method gives undoubtedly better results than linear interpolation does, avoiding the undesired and unnatural shrinkage problem which occurs in the latter case. Figure 3 shows the example of an elephant raising its trump. The first row in the figure describes the transformation in the case of linear shape interpolation, whereas the second row shows our results. The third row shows the superimposed intermediary shapes for both methods, thus proving the advantages our method brings about.

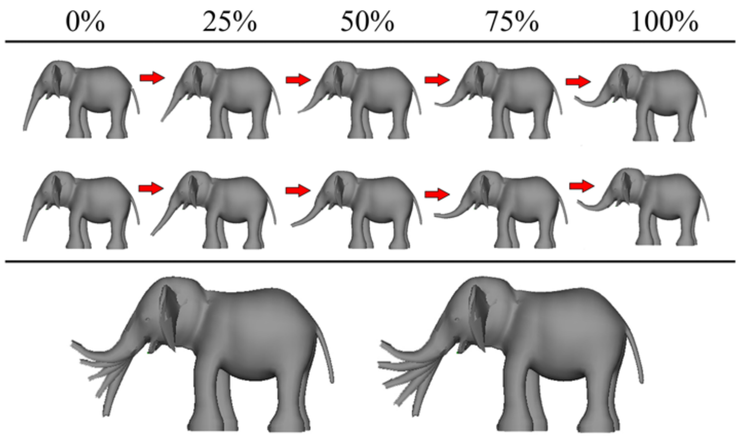

**Fig. 3.** Linear shape interpolation and our shape interpolation method applied on the elephant object.

The plot presented in Figure 4 describes the object volume variation during the transformation for both linear shape interpolation and our shape interpolation method, in the case of the elephant trump metamorphosis presented in Figure 3. Our resulting curve (the blue one) provides a volume variation graph which is very close to a line, whereas the shrinkage effect is very well described by the volume variation curve in the case of linear interpolation (the red curve).

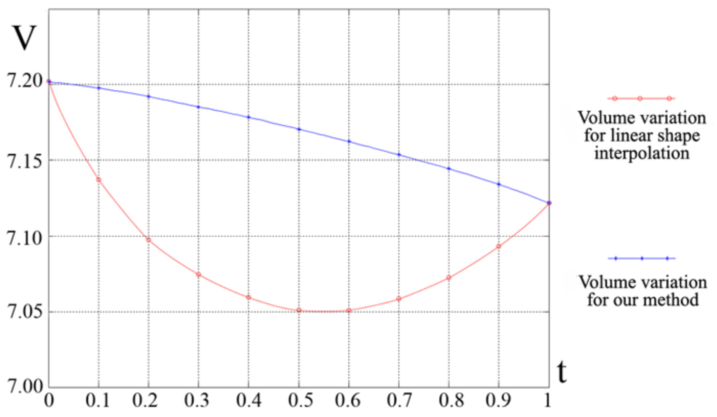

**Fig. 4.** Volume variation plot for both linear shape interpolation and our shape interpolation method applied on the elephant animation from Fig. 3.

Another self-explanatory deformation example is presented in Figure 5, for the case of a horse gradually changing its facial features into the ones of a giraffe-like animal. The result provided by our method shows a very natural shape metamorphosis between the two models (the bottom of the figure), as compared to the linear interpolation result (the top of the figure) which looks most unpleasant.

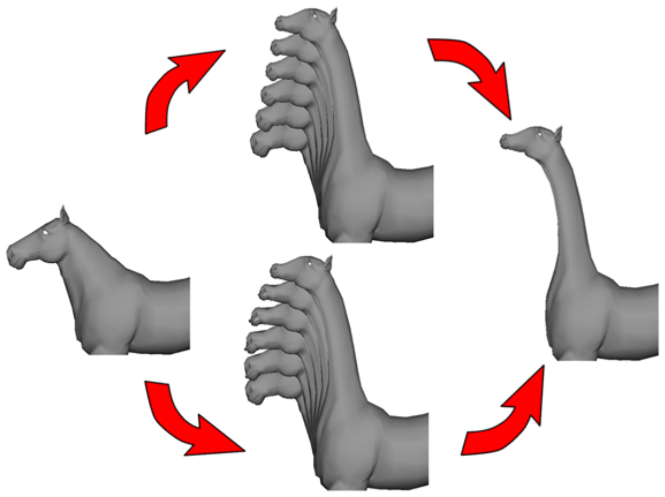

**Fig. 5.** Horse-giraffe metamorphosis.

Figure 6 shows the difference between our method (second row) and the linear interpolation one (first row) in the case of a low-polygon object: a twisting rod.

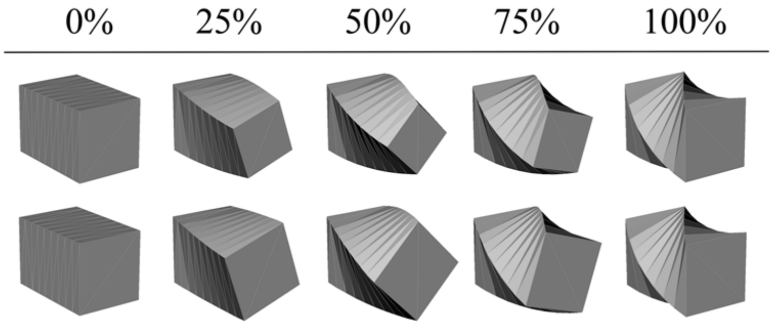

**Fig. 6.** Twisting a rod.

Figure 7 presents the example of a tongue animation, which is of uttermost importance in facial animations. In the case of our shape interpolation method (the second row in the figure), the tongue bends naturally and its muscles contract with a natural animation, whereas for the linear interpolation result the shrinkage effect again cannot be avoided (the first row in the figure). The third row shows the superimposed intermediary shapes for both methods.

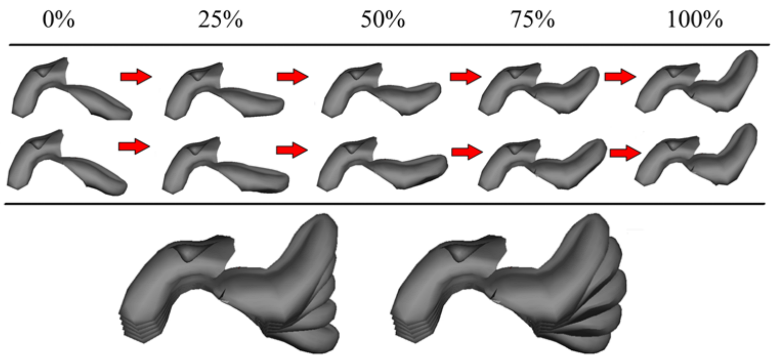

**Fig. 7.** Tongue animation results for both linear shape interpolation method and our shape interpolation method.

On the other hand, our method provides significantly better computational costs than other non-linear shape interpolation methods developed so far [16, 17, 18], where the computational times reach 22 seconds per each intermediary shape [17] and 55 minutes for generating the whole metamorphosis sequence [18]. Our shape interpolation method has two major steps: a preprocessing part, where the $C_i$, $C_{i,XOY}$, $C_{i,XOZ}$ and $C_{i,YOZ}$ circle centers are generated, and the actual circle arc morphing part, when the interpolation points are generated. The preprocessing algorithm usually requires only a few seconds for the entire metamorphosis sequence, no matter how many intermediary shapes are needed, while our morphing algorithm (the second part) takes less than 20 ms per intermediary shape and therefore is performed in real-time, which is a major advantage.

Table 1. Computational costs for different numbers of transformation planes

| Object | Triangle Number | Preprocessing Computational Costs | | | Morphing Computational Costs | Results |
|---|---|---|---|---|---|---|
| | | 3 transf. planes | 9 transf. planes | 12 transf. planes | | |
| Twisting a rod | 76 | $< 0.1$ s | $< 0.1$ s | $< 0.1$ s | Real-time | Figure 6 |
| Tongue bending | 1424 | $\approx 0.4$ s | $\approx 0.7$ s | $\approx 0.9$ s | Real-time | Figure 7 |
| Horse-Giraffe metamorphosis | 6902 | $\approx 1.6$ s | $\approx 2.2$ s | $\approx 2.7$ s | Real-time | Figure 5 |
| Elephant raising its trump | 9100 | $\approx 2.6$ s | $\approx 3.2$ s | $\approx 3.7$ s | Real-time | Figure 3 |

Table 1 shows the preprocessing algorithm computational costs our method provides in the case of the examples presented in this paper, for different numbers of test transformation planes. The computational times increase with the number of transformation planes tested, but so do the results. Also, computational times are higher for high-poly objects than for low-poly ones.

## 5. Conclusions and Future Work

In this paper, we have presented a novel shape interpolation method that gives very good results for mesh morphing 3D facial animations, which we called circular shape interpolation because it uses arcs of different circles for each pair of vertices.

The major advantage our method brings about is that, while producing considerably better results than the mostly used linear shape interpolation method does, it also strikes significantly better computational costs than other non-linear shape interpolation methods developed so far. Several experiments have proved the efficiency of our method. The method could also be adjusted for other kinds of animations by dividing the mesh in a few parts and then applying the algorithm on each such part in particular. Nevertheless, this should raise the computational time and therefore more research could be done in this area. Also, more research could be done to reduce the complexity of the algorithm and to improve its visual results.

The algorithm presented in this paper has been developed as part of a larger-scale project that aims to produce an application for helping people with disabilities learn how to speak Romanian correctly. For this purpose, the application should provide very natural and highly-detailed realistic lips and tongue animations, so as for the user to understand how different phonemes are pronounced in Romanian. For this result, as shown in Figure 7, the circular interpolation method proves its utility, thanks to its very pleasing computational cost. Also, due to the very short computational times, our facial animation method could very well find its place in any major commercial 3D animation software package, bringing about highly pleasing results for any 3D facial animation task one might require.

## References

[1] Terzopoulos D., Mones-Hattal B., Parke F., Waters K., *Facial Animation: Past, Present and Future*, *Proc. ACM SIGGRAPH 97 Conf.*, pp. 434–436.

[2] Radovan M., Pretorius L., *Facial animation in a nutshell: past, present and future*, *SAICSIT '06 Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, 2006, pp. 71–79.

[3] Parke F., Waters K., *Computer facial animation*, AK Peters, Wellesley, MA, 1996, ISBN 1-56881-014-8

[4] Ilie M. D., Negrescu C., Stanomir D., *Automatic Masked Morphing for 3D Facial Animations*, *Proceedings of the 2011 3rd Internatioanl Conference on Future Computer and Communication, (ICFCC 2011)*, ISBN 978-0-7918-5971-1, pp. 345–350, June 2011.

[5] Liu Chen: *An Analysis of the Current and Future State of 3D Facial Animation Techniques and Systems*, School of Interactive Arts and Technology, Fall 2009, Simon Fraser University, Canada.

[6] Lazarus F., Verroust A., *Three-dimensional metamorphosis: a survey*, Visual Computer, Vol. **14**, Nos. 8–9, pp. 373–389.

[7] Bennett J., Pascucci V., Joy K., *A genus oblivious approach to cross parameterization*, Computer Aided Geometric Design, **25** (2008), pp. 592–606, Elsevier Science Publishers B. V. Amsterdam.

[8] Kraevoy V., Sheffer A., *Cross-Parameterization and Compatible Remeshing of 3D Models*, SIGGRAPH '04 ACM SIGGRAPH 2004 Papers, pp. 861–869.

[9] Alexa M., *Recent advances in Mesh Morphing*, Computer Graphics Forum, 2002, Vol. **21**, No. 2, pp. 173–196.

[10] Sederberg T.W., Gao Peisheng, *2-D shape blending: an intrinsic solution to the vertex path problem*, SIGGRAPH ACM Special Interest Group on Computer Graphics and Interactive Techniques, 1993, pp. 15–18, ISBN: 0-89791-601-8.

[11] Liua Li-Gang, Wang Guo-Jin, *Three-dimensional shape blending: intrinsic solutions to spatial interpolation problems*, Computers & Graphics, Volume **23**, Issue 4, 1999, pp. 535–545.

[12] Surazhsky V., Gotsman C., *Intrinsic Morphing of Compatible Triangulations*, International Journal of Shape Modeling (IJSM), 2003, Volume **9**, Issue 2, pp. 191–201.

[13] Kaneko K., Okada Y., *Skeleton Based 3D Model Morphing Using Barycentric Map*, Computer Graphics, Imaging and Visualisation, 2008, pp. 132–137, ISBN: 978-0-7695-3359-9.

[14] Lipman Y., Sorkine O., Levin D., *Linear Rotation-invariant Coordinates for Meshes*, SIGGRAPH, ICCGIT, 2005, pp. 479–487.

[15] Sheffer A., Kraevoy V., *Pyramid Coordinates for Morphing and Deformation*, *3DPVT '04 Proceedings of the 3D Data Processing, Visualization and Transmission*, 2004, pp. 68–75, ISBN 0-7695-2223-8.

[16] Alexa M., *Local control for mesh morphing*, Shape Modeling and Applications, SMI 2001 International Conference, Genova 2001, pp. 209–215, ISBN 0-7695-0853-7.

[17] Xu D., Zhang H., Wang Q., Bao H., *Poisson shape interpolation*, *Proceedings of the 2005 ACM Symposium on Solid and physical modeling*, 2005, pp. 267–274, ISBN 1-59593-015-9.

[18] Yan Han-Bing, Hu Shi-Min, Martin R., *3D morphing using strain field interpolation*, Journal of Computer Science and Technology, Vol. **22**, Issue 1, 2007, pp. 147–155, ISSN 1000-9000.

[19] Yan Han-Bing, Hu Shi-Min, Martin R., *Morphing based on strain field interpolation*, Computer Animation and Virtual Worlds, Vol. **15**, Issue 3–4, 2004.

[20] Vince J., *Vector Analysis for Computer Graphics*, Springer; 1st Edition, 2010, ISBN-13: 978-1849966504.

[21] Van Overveld C. W. A. M., Wyvill B., *Phong normal interpolation revisited*, ACM Transactions on Graphics (TOG), Volume **16**, Issue 4, Oct. 1997, pp. 397–419 ISSN: 0730-0301.

[22] Peng Yuhui, Gao Chenghui, *Influencing Factors on Vertex Normal Accuracy for Triangular Meshes*, *IHMSC '09 Proceedings of the 2009 International Conference on Intelligent Human-Machine Systems and Cybernetics*, Vol. **1**, pp. 347–350, IEEE C. S. Washington, ISBN: 978-0-7695-3752-8.