University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitní 8
30614 Pilsen
Czech Republic

# Morphing of Meshes

The State of the Art and Concept of PhD. Thesis

**Jindřich Parus**

Technical Report No. DCSE/TR-2005-02
April, 2005

Distribution: public

Technical Report No. DCSE/TR-2005-02
April 2005

# Morphing of Meshes

**Jindřich Parus**

## Abstract

Morphing or metamorphosis is a technique for shape transformation between two objects. It smoothly transforms the shape of the source object to the shape of the target object. Morphing is usually used as an animation technique, for creation of some special effects, but it can be also used as a modeling tool, where some existing shapes are combined in order to obtain new shapes. In this technical report we will focus on morphing of objects given in boundary representation, i.e. objects described by a set of vertices and faces. Three basic steps of morphing boundary representation are investigated. The first step is an establishing of correspondence of vertices, which involves computing a parametrization of both input meshes. In the second step a supermesh is constructed by a process called topological merging. It results in a mesh which can be transformed to the shape of the source mesh as well as to the shape of the target mesh. The third step deals with interpolation of corresponding vertices, i.e. the morphing animation. It also involves interpolation of surface attributes as normals, colors, textures, etc. Possible improvements, ideas and suggestions for further research are also presented.

This work was supported by the Ministry of Education of the Czech Republic – project MSM 235200005, Microsoft research project 2003-187 and by the FRVŠ G1/1349 project.

# Acknowledgement

First of all I would like to thank to my supervisor doc. dr. ing. Ivana Kolingerová from the University of West Bohemia in Pilsen, Czech Republic, for a lot of inspiring ideas, useful comments, careful guiding during my PhD. study. Thanks also belong to prof. ing. Václav Skala, CSc. for providing good conditions under which this work was done. This work was supported by the Ministry of Education of the Czech Republic – project MSM 235200005, Microsoft research project 2003-187 and by the FRVŠ G1/1349 project.

# Contents

# 1 Introduction

It is a common feature of many things that they change their shape. Stones change their shape by erosion. Plants and animals are growing, getting strength and so changing the shape along with the surface color, internal structure, etc. And it is a common trend, maybe since the times when computers had been invented, that a certain group of people tries to simulate such phenomena in the computer. One of possible outputs of the simulation is a computer animation.

Computer animation is dynamically developing field of computer graphics. By means of computer animation we can simulate such effects as moving objects, mutual interaction, a change of shape, etc. Animation could be used in scientific visualization, education, entertainment industry, etc. Especially large field, where the computer animation is used, is the movie industry.

From one point of view, computer animation could be divided into two groups – rigid-body motion of objects (i.e. a translation and a rotation) and soft-body motion (i.e. a deformation of objects). In this technical report we will rather deal with the latter form of computer animation, i.e. the soft-body motion. More specifically we will focus on the animation technique for shape transformation, called *morphing* or *metamorphosis*. In general, this technique is used for transformation between two shapes. So in the terms of the first paragraph we can transform the shape of some animal's cub to the shape of the grown animal, simulating the process of growing in this way. The morphing technique is not only used in the computer animation; it can be used for an object synthesis also, where we create new shapes by combination of some existing shapes.

Morphing technique has been extensively studied in two dimensions for morphing of images and it has been successfully used in the movie industry. In image morphing settings there is usually a source image and a target image. The goal is to compute a smooth realistically looking transformation of the source image to the target image. But the 2D image morphing has some disadvantages. As the intermediate frame of the morphing transition is just a 2D image, it is not possible to easily change the position and orientation of the camera during the animation; it is not possible to compute exact lighting (specular reflections, shadows), etc.

In this technical report we will deal with morphing of 3D objects, namely meshes. The 3D morphing technique does not suffer from disadvantages of 2D image morphing described above, because in intermediate frames of the animation we have a 3D geometric representation which can be observed from different points of view, a lighting condition may vary during the animation, etc.

The goal of this technical report is to survey existing methods of 3D morphing, together with comments how much is a given method appropriate for different cases. Our contribution is also described, together with suggestions and ideas for the future work.

## 1.1 Motivation

The 3D morphing technique is also partially covered in professional animation tools such as 3ds max[1] or Lightwave[2]. But it has quite a strong limitation; it is possible to morph only models with the same number of vertices and the same topology. It is used mainly for facial animation where we create particular face expressions by modification of position of a few vertices. By modification of the position of vertices we do not change the topology but only geometry. Particular face expressions are called *morph targets*. The morphing is then just a simple interpolation between morph targets. The question, which is usually not solved in the animation software, is how to morph objects with different number of vertices and different topology. This question is also a motivation for the vivid research in this area as well as for our research.

## 1.2 Organization of the text and basic notation

The text of this technical report is organized as follows. Section 2 gives a basic problem overview and outlines various morphing approaches according to the input data representation. Section 3 describes the correspondence problem as the first step of the morphing computation. The idea of topology merging is described in Section 4. The interpolation problem is discussed in Section 5 together with description of our contribution, which deals with interpolation of surface attributes. Section 6 summarizes suggestions and ideas for a future work along with some test results. In Appendix A some details about spherical geometry are given. Appendix B contains some examples which were produced by our software, the examples of animations can be also found on the accompanying CD as well as on the Internet at http://herakles.zcu.cz/research/morph/tr02_2005_ex.html.

Throughout the text the *italics* will be used for introducing new terms, which will be further explained. Text also contains a number of color figures, in the morphing context we will usually depict two objects, so the figures related to the source object will be drawn in green and the figures related to the target object will be drawn in red. In mathematical relations the superscript will be used for referring which mesh (either the source mesh or the target mesh) the given element (vertex, face, edge, etc.) belongs to, the subscript will be usually used for denoting of index of the given element. For example, $v_i^0$ refers to the *i*-th vertex of the source mesh, $f_j^1$ refers to the *j*-th face of the target mesh. If the superscript is not explicitly specified, it usually refers to some general element, where it has no sense to distinguish between the source and the target mesh.

[1] 3ds max currently in version 7 is a product of Discreet (www.discreet.com)
[2] Lightwave currently in version 8 is a product of NewTek (www.newtek.com)

# 2 Morphing in general

In this section we will give a general overview of morphing. We will outline the influence of the object representation on the morphing algorithms and we will describe basic steps for the morphing of boundary representation, which is the main topic of this report.

## 2.1 Problem overview

The morphing task could be formalized as follows. Given two input models – the *source* model and the *target* model – the goal is to compute some transformation from the shape of the source model to the shape of the target model. According to terminology introduced in [Ken92] the term *object* refers to any 3D or 2D entity, the term *model* refers to some geometrical representation of 2D or 3D objects and the term *shape* refers to the set of all points which comprise the surface of the object. Note that different models may represent the same shape. The transformation between two shapes is not unique. Theoretically there is a big number of possible transformations, e.g., a degeneration of the source object into one single point followed by evolution of the target object or disintegration of the source object to individual faces and transformation of the individual faces into the shape of the target object. The problem is that that kind of transformation is usually not very visually plausible and so we are looking for some more attractive shape transformations. In [Gom99], there are given some principles for good morphing, e.g., topology preservation, feature preservation, rigidity preservation, smoothness, monotonicity, etc. Topology preservation means to preserve topology of the source and the target object, e.g., no holes should suddenly appear during the morphing transition when the source and the target objects are topologically equivalent. Feature preservation refers to the preservation of important features, which are present in the source as well as in the target object during the morphing transition, e.g., when morphing between two animals, legs, heads, tails, etc. should remain aligned during the transition. Rigidity preservation refers to the fact that sometimes a rigid transformation (rotation, translation) is preferred to a soft-body transformation (scaling, shearing, etc.). Smoothness means that the shape transformation should be smooth, avoiding discontinuities. Monotonicity refers to monotone change of some parameters, e.g., angles should change monotonically avoiding so a local self-intersection. It is important that these principles are strongly application dependent, e.g., in special-effects industry sometimes artificial shape transformation is more impressive that some physically correct transformation, which on the other side would be required in some technical applications.

The morphing technique is strongly dependent on the object representation, i.e. on the way we describe our models. In our work we will consider the boundary representation in the form of triangular meshes, i.e. the objects represented by the set of vertices, edges and faces. We choose the boundary representation because it is quite easy to store, modify, render and it is supported by hardware of graphics cards. We have also a lot of models available and, last but not least, a lot of professional animation tools, such as 3ds max or Lightwave, use this representation. On the other side, using triangular meshes, we have only a piecewise linear approximation of objects, which could sometimes cause problems during modification of the model.

As stated before, various approaches to morphing usually differ in object representation. A survey of various approaches to the morphing classified by object representation is given, e.g., in [Laz98]. In the following section we will briefly outline principles, advantages and disadvantages of morphing in the main types of object representation.

## 2.2 Volume representation

By the term volume representation we understand here a description of objects by an entity that can provide a value at each point in 3D space, e.g., evaluating of some analytically expressed function or computing the value from a discreet samples stored in a 3D grid. The simplest method is to linearly interpolate between the source and the target volume, which corresponds to image *cross-dissolving* in 2D case. The term cross-dissolving refers here to simple linear blending.

Hughes in [Hug92] suggest not to interpolate volumes directly, but in the frequency domain. It is based on the observation that high-frequency components of the volumetric model represent usually small details, while the low-frequency components represent a general shape of the object. The interpolation of details seems unimportant compared to the interpolation of the general shape. So Hughes suggests to gradually remove the high frequencies of the first model, interpolate over to low frequencies of the second model, and then blend in the high frequencies of the second model. The order in which the frequencies are added and removed is given by a *schedule*. The schedule is a function that describes how much of the information at a given frequency should remain from the model at time $t$. The morphing among two volumetric models given by functions $g(x, y, z)$ and $h(x, y, z)$ can be then described as:

$$K(t) = F^{-1}(S_1(t, f_x, f_y, f_z)F(g) + S_2(t, f_x, f_y, f_z)F(h))\,, \qquad (1)$$

where $K(t)$ is intermediate volume at time $t$, $F$ is the fast Fourier transform, $S_1$ is a schedule function, i.e. it describes how much information at frequency $(f_x, f_y, f_z)$ should remain from the first model at time $t$ and analogically $S_2$ describes how much information should remain from the second model. The schedule $S_2$ is usually set to $S_2(t, f_x, f_y, f_z)=1-S_1(t, f_x, f_y, f_z)$.

Another approach of Lerios et al. [Ler95] generalizes the idea of Feature-Based image metamorphosis [Bei92] in 3D. It works with volumetric models represented by a regular structured voxel grid. The morphing process is decomposed into two parts – warping and blending. In the first part, a user specifies corresponding features, i.e. features which remain aligned during the morphing transition. Corresponding features can be specified by pairs of feature elements (points, lines, rectangles and boxes[3]), which delineate features. During the morphing process the corresponding feature elements are interpolated from the source to the target orientation. The source and the target volume are warped towards the interpolated feature element. The result is that corresponding features are moved, turned and stretched to mach the position, orientation and size. In the second step, warped volumes are blended. The simplest form of blending is linear cross-dissolving. But the simple cross-dissolving causes errors when rendering an intermediate volume by the classical ray-casting method. This is

[3] Original 2D approach [Bei92] uses only pairs of corresponding lines.

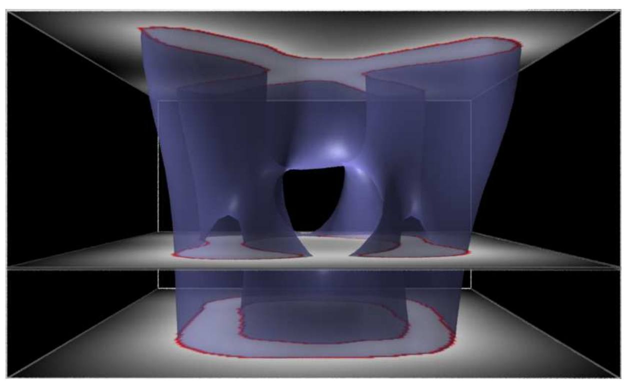
because of the exponential dependence of the color of a ray cast through the volume on the opacities of the voxels it encounters. In particular, the intermediate volume abruptly snaps into the source and target volume. So Lerios et al. suggest empirically established non-linear cross-dissolving.

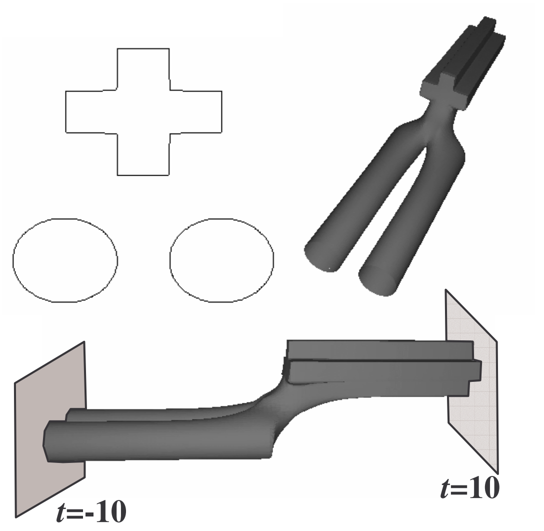
The common problem of morphing a volume representation is the data representation itself. It is quite memory consuming. Geometric models are usually voxelized, which brings problems with resolution, aliasing, etc. For rendering of volumetric models, some sort of direct volume rendering (DVR) or an isosurface extraction method have to be used. And last but not the least the volume representaion is not very common in professional animation tools, as they prefer to work with parametrical or polygonal objects. The advantage of volume approaches is that they quite easily overcome topological restrictions, e.g., a sphere and a torus can be easily morphed by the cross-dissolving, which is not so easy in the boundary representation based approaches.

## 2.3 Space-time morphing

Space-time morphing is usually connected with an implicit representation of objects, i.e. objects represented by function $f(x_1, x_2, \ldots, x_n)=0$. The basic idea is that the space in which the input objects (e.g., a 2D space for polygons or a 3D space for meshes) are defined is extended with one more dimension. The added dimension can be considered a time, thus the new space is called *space-time*. For example a 2D point $(x, y)$ is expressed in space-time as a triple $(x, y, t)$. The basic idea is to interpolate n-dimensional input objects by an (n+1)-dimensional smooth surface. An example of a surface interpolating the source and target object is in the Fig. 1a). In this case, the source object is an x-shaped planar polygon and the target object is o-shaped planar polygon. The input objects are 2D polygons, thus the space-time is a 3D space. The individual cross-sections of the interpolating 3D surface define an intermediate polygon, more specifically, the cross-section at the time $t=0$ is the source polygon and the cross-section at the time $t=t_{max}$ is the target polygon. The advantage of this idea is that it is extendible to 3D, with that difference that the interpolation runs in 4D, the cross-sections of the interpolated surface are intermediate 3D meshes.

Various approaches to the space-time morphing usually differ in the interpolation method. The approach of [Tur99] uses radial basis functions (RBF) in order to compute an implicit function which interpolates the input polygons. The vertices of the input polygons are so-called *boundary constraints*, i.e. the points in which the implicit function takes on the value of zero. Paired with the boundary constraints are *normal constraints*. The normal constraints are points in which the implicit function takes on the value of one and they are located towards the interior of the polygon. Both the boundary constraints and the normal constraints are passed along to the apparatus of RBF as scattered points. RBF interpolation provides a smooth surface which interpolates the source and the target polygon. By slicing the resulting surface by a plane $t$=const. an intermediate polygon is obtained. In 3D space the interpolation operates on 4D $(x, y, z, t)$ points and intermediate 3D meshes are obtained by slicing 4D implicit function. The advantage of this method is that is solves topology changes “for free” (note that in the Fig. 1a) the o-shape has one hole and the x-shape has no hole, so a change of topology is involved).


a)



b)

**Fig. 1: a) An interpolation between the o-shape and x-shape by the approach [Tur99]. b) Space-time blending by Pasko et. al [Pas04], the source polygons are two circles and the target polygon is a cross (top left), bounded space-time blending between two half-cylinders bounded by the intersection of half-spaces *t*≥-10 and *t*≤10 (two views – top right, bottom).**

Another approach by Pasko et. al [Pas04] operates on FRep, which is a generalization of implicit surfaces and CSG modeling[4]. Pasko et al. [Pas04] converts n-dimensional objects into half-cylinders in (n+1)-dimensional space-time. The half-cylinder is a cylinder bounded by a plane from one side. The idea is similar to [Tur99]. The input objects are placed along the time axis and an interpolation is applied. In this case, the interpolation method is a space-time bounded blending. The bounded blending blends two objects, more, the blend is bounded by an additional bounding solid. It is depicted in the Fig. 1b). The source polygons are two circles and the target polygon is a cross. Space-time blend of two half-cylinders is bounded by the intersection of two half-spaces *t*≥-10 and *t*≤10 (Fig. 1b) bottom). The advantage of this method is that it is able to morph between different topologies (e.g., to morph from two circles to one cross in the Fig. 1b) and it can also handle objects which do not reside the same place in the space.

## 2.4 Morphing polygons

We will also shortly describe the area of polygon morphing (i.e. the morphing of 2D vector representation) because it is quite closely related to our topic, just one dimension less. The morphing of polygons is usually divided into two parts – finding of correspondence and finding of trajectories for corresponding elements. Correspondence problem usually involves establishing correspondence between vertices of both polygons. Note that the source and the target polygon may not have the same number of vertices, so some new vertices have to be added.

The first problem is addressed in [Sed93a]. Sedeberg et al. incorporate a physical model. The polygon edges are modeled as wires with some material properties (modulus of elasticity, stretching stiffness constant). A shape transformation involves then some stretching and bending work. The goal is to establish such a correspondence, so that the work needed to

[4] CSG – Constructive Solid Geometry.

transform the source shape to the target shape is minimized. Sedeberg et al. also investigated cases which cause displeasing results during the shape interpolation. It is a self-intersection, i.e. the case when a part of the morphing polygon passes through another part of itself, and a non-monotonic angle change, e.g., angle first increases, reaches its extreme and again decreases. These cases are penalized in the physical model. It is important to mention that self-intersection penalization avoids only local self-intersections and does not guarantee a global self-intersection-free transformation. A problem is also cause by the linear vertex interpolation, in particular when morphing between highly dissimilar shapes.

In [Sed93b] the *intrinsic parameters* (e.g. edge lengths or internal angles) are interpolated, rather than vertex positions directly. The polygons are converted to the so-called *edge-angle representation* [Gom97]. In the edge-angle representation a polygon is described by one fixed vertex, one edge incident to the fixed vertex, the length of each edge and the internal angle of each vertex. The advantage of this representation is that except for the fixed vertex and edge it is invariant to rigid transformation[5]. The absolute vertex coordinates are extracted from interpolated intrinsic parameters. This interpolation scheme avoids edge collapsing and non-monotonic angle changes. This technique was used for generating in-betweens for the animation based on keyframes. The concept of interpolation of intrinsic parameters was also further used for morphing of planar triangulations in [Sur01, Sur04].

Another interesting approach to 2D morphing was introduced in [Sha95]. It first decomposes the source and the target polygon into *star-shaped* polygons. A star-shaped polygon is a polygon, where at least one interior vertex exists (so-called *star-point*), from which all other polygon vertices are *visible*. The term visibility means here, that the line segment connecting two vertices lies completely inside the polygon. Then the skeletons of the decompositions are constructed. The skeleton is a planar graph which joins star-points of neighboring star-shaped polygons, i.e. it is a dual graph to the star-shaped decomposition. Important is that skeletons of the source and the target polygon have to be isomorphic, which requires an isomorphic star-shaped decomposition. Then, the interior and the boundary of the polygon can be expressed relatively to the skeleton. The basic idea of morphing is that the intermediate shapes are reconstructed according to the interpolated skeletons. The difference between this approach and previous approaches [Sed93a, Sed93b] is that this approach takes into consideration also the interior of the polygon and not only the boundary.

Let us shortly mention how the concept of 2D polygon morphing can be used in 3D. A 3D model can be sliced to a set of planar contours and then morphed between corresponding contours. The intermediate object is then reconstructed from the interpolated contours. This approach was introduced, e.g., in [Kor97]. The problem of this method is the slicing itself, because a slice may in general contain multiple contours, so it brings a question how to morph between slices containing a different number of contours.

## 2.5 Morphing of meshes

In the following section we will shortly outline the basic steps in morphing of meshes which will be further described in detail in separate chapters.

[5] A rigid transformation is an affine transformation which preserves relative edge lengths and angles, i.e. rotation and translation.

### 2.5.1 Basic steps

Going back to the problem overview outlined before, we will first define the term *model* in the context of boundary representation. Objects in the boundary representation are described by vertices and faces, so the model $M$ is a pair ($V$, $F$), where $V$ is the set of vertices, each vertex $v_i$ is described by its coordinates in $R^3$. The set $F$ is the set of faces, where each face $f_i$ is described by a triple ($j$, $k$, $l$), where $j$, $k$, $l$ are indices of vertices which form the face $f_i$. In further text we will denote the model described by the pair ($V$, $F$) as a *mesh*.

In the classical setting of mesh morphing we have two input meshes. The source mesh $M^0 = (V^0, F^0)$ and the target mesh $M^1= (V^1, F^1)$. Note that unlike the approach in tools such as 3ds max and Lightwave, the source and the target mesh can have a different number of vertices and a different *topology*. The term topology refers here to the connectivity graph of vertices[6]. As the source and the target mesh may have a different number of vertices, the first step is *establishing of correspondence*. The most convenient element for establishing correspondence is a vertex – so we establish the correspondence of vertices of the source and the target mesh. Due to different number of vertices of the source and the target mesh, we cannot link to one vertex of the source mesh exactly one corresponding vertex of the target mesh. Instead, we assign to one vertex from the source mesh generally some place on the surface of the target mesh (i.e. not necessarily a vertex). This situation is illustrated in the Fig. 2. The green arrows represent the relation of correspondence between a few vertices of the source mesh and places on the surfaces of the target mesh. Note that in this way we established a correspondence only for the vertices of the source mesh, so the same process have to be repeated for the vertices of the target mesh, with that difference that we look for an appropriate place on the surface of the source mesh[7].

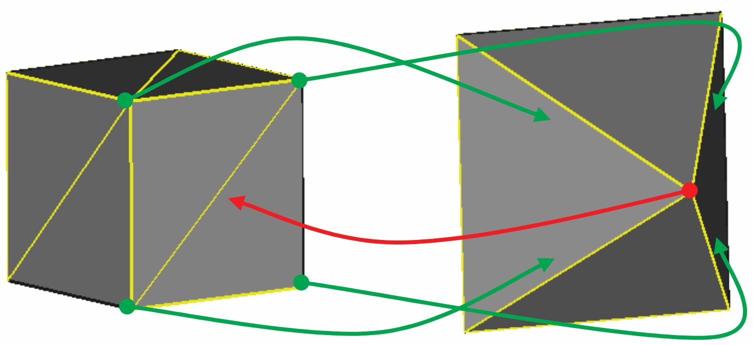


**Fig. 2: The correspondence of the vertices of the source mesh with the places of the surface of the target mesh (green arrows) and the correspondence of the vertex of the target mesh with the places on the surface of the source mesh (red arrow).**

In this step it is also possible to include some user interaction, usually in the form that user specifies some corresponding features of both input objects, which should remain aligned during the shape transformation. A typical example is morphing between two animals, where we usually want the common features (i.e. legs, head, tail, etc.) to remain aligned throughout

[6] The term topology could appear also in another meaning (i.e. when talking about the homeomorphism of objects), the reader will be warned of the change of the meaning.

[7] In the case that the topology of the source and the target mesh were the same, we would have the correspondence relation (vertex, vertex) and we would not need to establish the correspondence in reverse direction.

the transition. A detail description of establishing correspondence and aligning of corresponding features is given in Section 3.

Once we have established a correspondence, we are still not able to complete morphing by interpolation of corresponding elements, because the correspondence relation is a relation between the vertex of one mesh and the place on the surface of the other mesh. Therefore, the source and the target mesh have to be refined in order to have a vertex-vertex correspondence, i.e. on the positions of corresponding places new vertices have to be inserted. So the second step of the morphing task is a *construction of a new representation*, which is based on *merging* of the topologies of the source and the target mesh. This process is also denoted as the *topological merging*. The new representation is again a mesh, which we will call a *supermesh* in the further text, because it shares a topology of the source and the target mesh. Supermesh has one nice property[8] – it can be transformed to the shape of the source mesh as well as to the shape of the target mesh[9]. This situation is shown in the Fig. 3, there are the same objects as on Fig. 2 and the blue dotted lines represent how the topology of the source, and the target mesh respectively, has to be refined in order to be able to transform the source mesh to the shape of the target mesh and vice-versa. A detail description of the supermesh construction is given in Section 4.

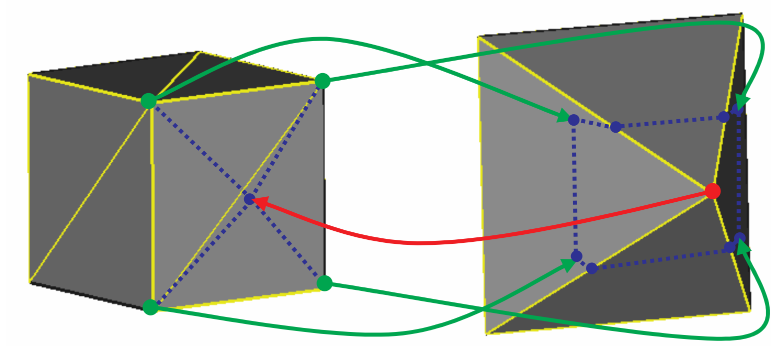


**Fig. 3: Refining the topology (blue dotted lines) of the source mesh, and the target mesh respectively, in order to obtain the supermesh, which can be transformed to the shape of the source mesh as well as to the shape of the target mesh.**

Having the supermesh, the last already advised step is to find trajectories for supermesh vertices. Note that we know the extreme positions of vertices, i.e. for the time $t$=0 we determine the vertices positions with respect to the source mesh and for time $t$=1 we determine the vertices positions with respect to the target mesh. In this step it is also suitable to interpolate the surface attributes, e.g., normals, textures, colors, etc. A detailed description of this step is given in Section 5.

Let us summarize the described steps:

1. Establishing of a correspondence of vertices, i.e., establishing of the relation $C_{0\to1}(v_i^0, p^1)$, where $v_i^0$ is the vertex of the source mesh and $p^1$ is corresponding place on the surface of the target model; and analogically the relation $C_{1\to0}(v_j^1, p^0)$ for

[8] This property follows from the fact that the supermesh shares topology of both the source and the target mesh.
[9] Note, altought the supermesh has the different topology (i.e. different model) than the source and the target mesh; it can be transformed to the shape of the source mesh as well as to the shape of the target mesh, as the shape is just a set of points comprising the surface of the object.

   the vertices of the target mesh. The correspondence should be then optionally adjusted by user.
2. Construction of the supermesh, which represents both the source and the target mesh.
3. Interpolation of the corresponding vertices along with surface attributes.

In the following sections we will describe the introduced steps in more detail.

# 3 Correspondence

In this chapter we will deal with establishing of the correspondence. We will summarize a current state of the art in this area. We will not describe approaches paper by paper as they were published in journals and conferences. Instead of this, we would like to identify common steps, describe them from a wider point of view and then introduce a concrete realization of particular steps. This section will be more varied as the establishing correspondence step is the step in which various morphing approaches differ.

The establishing of correspondence along with the vertex paths plays a key role in the quality (or plausibility) of the resulting animation. For a demonstration, see Fig. 4, where the top row sequence is an example badly established correspondence and the bottom row sequence is a result of better established correspondence.

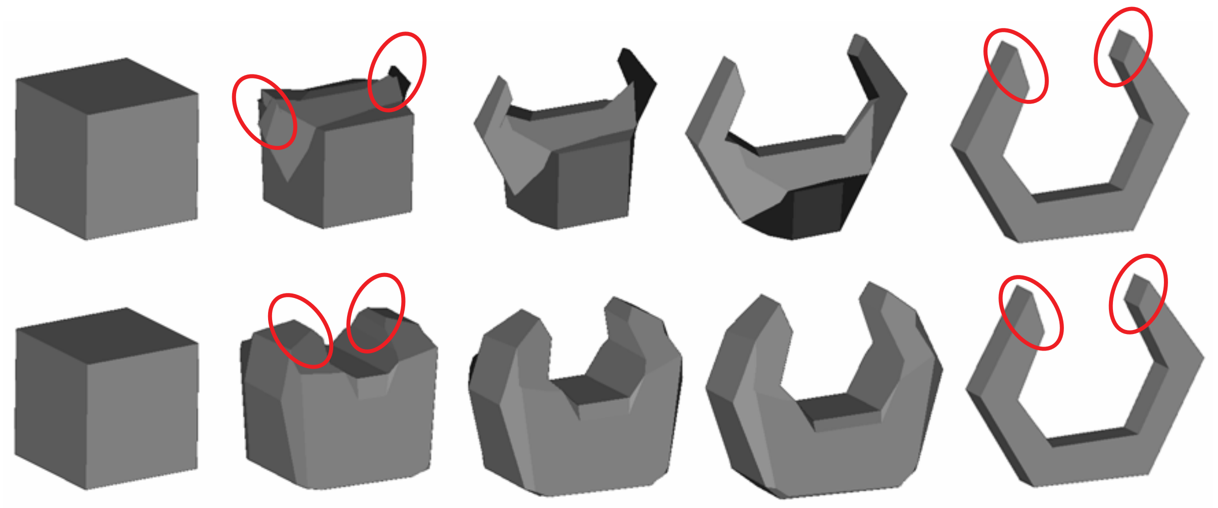


**Fig. 4: An example of the influence of the correspondence on the resulting animation. The top row shows an animation with a badly established correspondence, the bottom row shows an animation with better established correspondence. Note how parts of the target mesh (marked by ellipsis) grow out of the cube.**

So now is the question how to establish a correspondence. For this purpose we compute a *parametrization* of the source and the target mesh and the correspondence is established in the *parameter domain*. The parameter domain is a parameter space to which is the object mapped by the parametrization. So the parametrization is in our context a bijective mapping of the faces of the mesh to the parameter domain. The bijection is required because each face of the mesh has to have its unambiguous image in the parameter domain. In other words, the faces in the parameter domain must not overlap. Bijectivity yields also the existence of inverse mapping[10]. Due to introducing the parametrization, we have to distinguish two spaces – the *object space*, i.e. the space in which the mesh is defined[11], and the *parameter space*, i.e. the space to which the mesh is mapped by the parametrization. We will not introduce a special

[10] Inverse mapping (i.e. from parametrical domain to the object) is used in particular in the area of remeshing – see 4.3.

[11] In our case it is the Euclidean space.

notation for vertices in the parameter space and for vertices in the object space, because the whole finding correspondence process and the topological merging process run in the parameter space. The transition from the parameter space to the object space is described in Section 4.2.2 and the reader will be warned.

For morphing purposes the following parameter domains are used:

- A *unit sphere*, for the case that the mesh is *topologically equivalent* to the sphere[12], i.e. it is a closed unbounded mesh. The parametrization is then a mapping of the mesh to the surface of the unit sphere[13].
- A *unit disc*, for the case that the mesh is topologically equivalent to the disc, i.e. it is a mesh with a single closed polygonal boundary. The parametrization is then mapping of the mesh to the unit disc.

This is definitely not a complete list of possible parameter domains. For example, for objects with one hole a torus as a parameter domain has to be used. In this technical report we will limit our consideration on topological spheres, i.e. on closed unbounded meshes, which satisfy Euler's formula:

$$V - E + F = 2\,, \tag{2}$$

where $V$ is the number of vertices, $E$ is the number of edges and $F$ is the number of faces. Such objects are also called *genus-0* objects (as they have no "passages through"). An example of the mesh and its parametrization is in the Fig. 5.

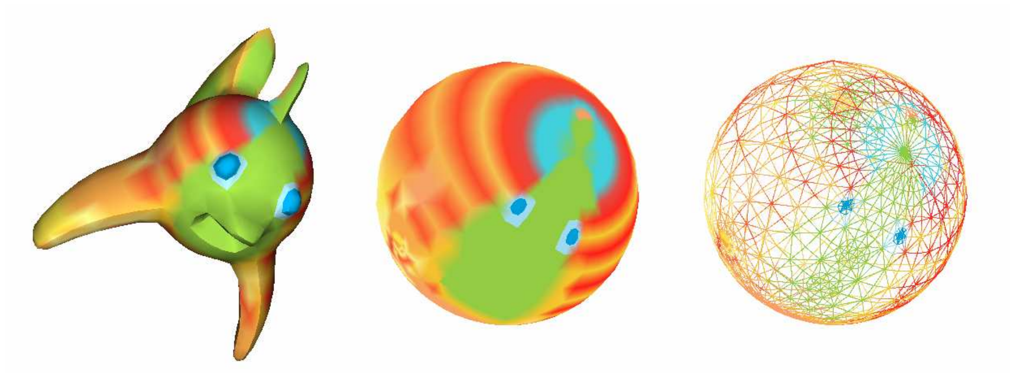


**Fig. 5: An example of the mesh (left) and its parametrization. The middle figure is the same parametrization as on the right but with faces filled according to the color of the mesh).**

The validity of the parametrization, i.e. whether no faces overlap in the parameter domain, for the case that the parameter domain is the unit sphere, can be checked by evaluating:

$$\operatorname{sgn}((v_0 \times v_1).v_2)\,, \tag{3}$$

[12] In simple terms, topological equivalence of two objects means that one can be deformed to the other by twisting and stretching (not tearing). Objects topologically equivalent to the sphere are also called topological spheres.
[13] Note that straight edges of the mesh became parts of great arcs on the surface of the sphere and so the faces of the mesh formed by edges became spherical triangles on the surface of the sphere.

where sgn is the signum function, $v_0$, $v_1$, $v_2$ are positions of the vertices of a triangle in the parameter domain. Formula (3) is evaluated for each face. The parametrization is valid iff all the faces are oriented in the same way[14], i.e., the signum of the result of the formula (3) is the same for each triangle.

From Steinitz's theorem follows [She04] that a graph may be embedded on the sphere if it is planar and 3-connected. Thus, according to [She04] a closed genus-0 triangulation can always be mapped to the sphere. Now suppose for a while that we have already "somehow" computed the parametrizations of the source and the target mesh (let us denote them the *source parametrization* and the *target parametrization*) and we want finally compute the correspondence. This is done by *overlaying of the parametrizations* of the source and the target mesh and checking for each vertex of the source parametrization in which face of the target parametrization it lies, and vice versa. The overlaying is described in the following section.

Important is that the both parameter domain of the source and the target mesh has to be of the same type, e.g., a unit sphere, because it would be impossible to overlay, e.g., a unit sphere and a torus (i.e. the topological domain for the objects with one hole). This implies that we are able to morph only objects which are topologically equivalent. The morphing of objects which are not topologically equivalent (i.e. with different genus) is usually solved by decomposition[15].

## 3.1 Overlaying of the parametrizations

By overlaying we mean here putting both parametrizations each over other (Fig. 6a)). Each vertex of the source parametrization then must lie inside some face or on the edge of the face of the target parametrization. So the corresponding place for the vertex $v_i^0$ of the source mesh is given by the *relative position* of the vertex $v_i^0$ with respect to the face $f_j^1$ of the target parametrization in which the vertex $v_i^0$ lies. Analogically, the corresponding place for the vertex $v_k^1$ of the target mesh is given by the relative position of the vertex $v_k^1$ with respect to the face $f_l^0$ of the source parametrization. So the goal is to establish mapping $\Omega_{0\to1} : v_i^0 \to f_j^1$, which maps vertices of the source mesh to the faces of the target mesh, and the mapping $\Omega_{1\to0} : v_k^1 \to f_l^0$, which maps vertices of the target mesh to the faces of the source mesh. This situation is depicted on Fig. 6b).

[14] Of course, it is assumed that the original input mesh has consistent orientation, i.e. clockwise or counter-clockwise.

[15] See Section 3.3.2.

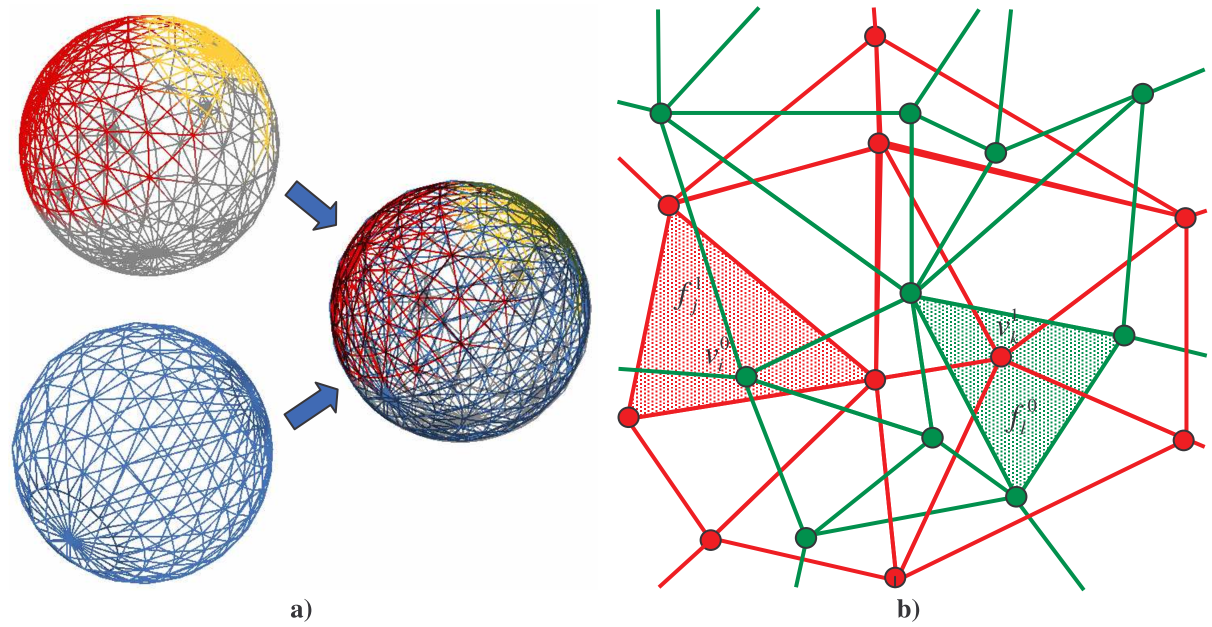


**Fig. 6: a) An overlying of the source parametrization (top) and the target parametrization (bottom). b) A detail of the overlaying (the figure is simplified because the triangles are drawn as planar and not as spherical). The red triangulation is the parametrization of the source mesh and the green triangulation is the parametrization of the target mesh.**

The establishing of mappings $\Omega_{0\to1}$ and $\Omega_{1\to0}$ is a point location problem, i.e. a location of the vertices of the source mesh in the target parametrization and vice versa. A brute force algorithm would be to test each vertex of one mesh against each face of the other mesh. This leads to O(*N.M*) algorithm, where *N* is the number of vertices and *M* is the number of faces. It is not a very convenient algorithm for complex meshes, instead of this we use a more efficient algorithm based on walking. This algorithm will be described in Section 4.2.1, because mapping of the vertices to the faces can be established together with the intersection computation.

Note that we are working in the parametrization, i.e. with the spherical triangulation, so it is necessary to modify standard point location test for the spherical case. The basic point-in-triangle test for the spherical case is described in Appendix A. For expressing the relative position of the vertex with respect to the face we use barycentric coordinates, also described in Appendix A.

## 3.2 Methods for parametrization

In this section we will deal with concrete methods for computing parametrization as a tool for establishing of correspondence. Parametrization is a huge area of computer graphics and a lot of researches deals with it. We will focus on methods which are related to morphing. In particular we will describe the methods for parametrizing topological discs and topological spheres. It is necessary to remind that the quality of the parametrization is a crucial aspect of the correspondence and it is then reflected in the quality or plausibility of the resulting animation (recall, e.g., Fig. 4). We classify the parametrization methods into two groups: local methods and global methods. The local methods usually take vertex by vertex and compute its parametrization. From local methods we will describe a relaxation (3.2.1) and a decimation-based method (3.2.2). The global methods usually express requirements on parametrization as a system of equations, which is solved in order to obtain a valid parametrization. From global methods we will describe a harmonic mapping (3.2.3) and robust spherical parametrization (3.2.4).

The parametrization is not useful only for the mesh morphing, but also in the area of texture mapping (i.e., assigning to each 3D vertex a 2D texture coordinate from the parameter space), remeshing, aproximation of meshes by spline surfaces, etc.

### 3.2.1 Relaxation

First we will introduce a method for parametrization of convex meshes, as it serves as a basis for the relaxation method. This method of parametrization was also used in the fundamental morphing paper [Ken92]. The parametrization of the convex meshes is very simple – we can choose any interior point of the mesh and project from this interior point all the vertices of the mesh to the surface of the unit sphere. We translate the mesh, so that the chosen interior point coincides with the origin, and then we normalize the position vector of each mesh vertex[16]. By normalization we push the vertex to the surface of the unit sphere. Thus, convex meshes can be parametrized by a simple spherical projection from some interior point.

In this way we can also compute the parametrizations of the so-called star-shaped objects. A star-shaped object is such an object, where we can find at least one interior point from which all the vertices of the mesh are *visible*. By term visible we understand here that the line segment connecting the interior point and the vertex of the mesh is completely *inside* the mesh. This interior point lies in the kernel of the star-shaped mesh, which is intersection of all halfspaces bounded by planes of the faces of the mesh. The parametrization is obtained by a projection of the star-shaped mesh through the interior point, which lies in the kernel.

The importance of the proper interior point is demonstrated in the Fig. 7, for simplicity in 2D. Imagine a counter-example, i.e. the where we have chosen an incorrect interior point for the projection. Then, two edges (denoted as $p(e_i)$ and $p(e_j)$ in the Fig. 7) overlap in the parameter domain (which is a unit circle in 2D), violating so the bijection of the mapping. Note that in 3D this is an overlapping of faces in the parameter domain (called *foldovers*).

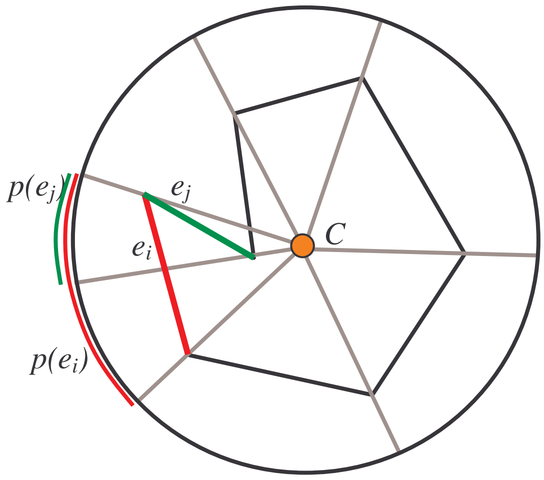


**Fig. 7: An example of an inappropriate choice of the central point. It causes that edges $e_i$ and $e_j$ overlap in the parameter domain, then the parametrization is not a bijective mapping.**

[16] Position vector (sometimes denoted as radius vector) is defined by a vertex and an origin of a coordinate system.

The idea of relaxation was introduced in [Ale99a]. Relaxation starts with the spherical projection. Of course, as we are working with general genus-0 meshes (neither necessarily convex nor star-shaped), there will be foldovers in the parameter domain. The goal of the relaxation procedure is to remove these foldovers. It is an iterative procedure, it moves in each step the vertex to the center of its neighbors:

$$v_i^{r+1} = \left\| \frac{1}{|N_i|} \sum_{j \in N_i} v_j^r \right\|, \tag{4}$$

where $v_i^{r+1}$ is the position of the vertex $v_i$ in the $(r+1)$-th iteration, $N_i$ is the set of adjacent vertices to the vertex $v_i$ and the notation $\|.\|$ stands for the normalization of the vector. The whole Eq. (4) is normalized in order to push the vertex to the surface of the unit sphere. This simple relation tends to converge a to global minimum, i.e. all vertices in the same place. In [Ale99a] it is suggested to fix some vertices. The fixed vertices are called *anchors*. The anchors avoid collapsing of the parametrization to one single point. But the anchors themselves bring problems, because as they are fixed, they cannot be moved to the center of their neighbors and so they may cause foldovers. This problem is solved by changing of the anchors after some number of iterations.

In [Ale00a] this scheme was improved and simplified. Instead of using the anchors (i.e. the fixed vertices), the main relation for the relaxation was extended with the use of weights, which penalize long edges and so prevent the vertices from collapsing. Eq. (4) was extended as follows:

$$v_i^{r+1} = \left\| v_i^r + \frac{c}{|N_i|} \sum_{j \in N_i} (v_i^r - v_j^r).\left|v_i^r - v_j^r\right| \right\|, \tag{5}$$

where $\left|v_i^r - v_j^r\right|$ is the weight, $c$ is the constant used to control the overall move length and $\|.\|$ stands for the normalization. With $c=1$, the relaxation runs robustly, but not very efficiently. Therefore, in [Ale00a] it is suggested to set the constant $c$ to the inverse of the longest edge incident to the relaxed vertex.

As well as in the previous approach the resulting vertex position has to be normalized in order to lie on the surface of the unit sphere. The progress of the relaxation is shown in the Fig. 8.

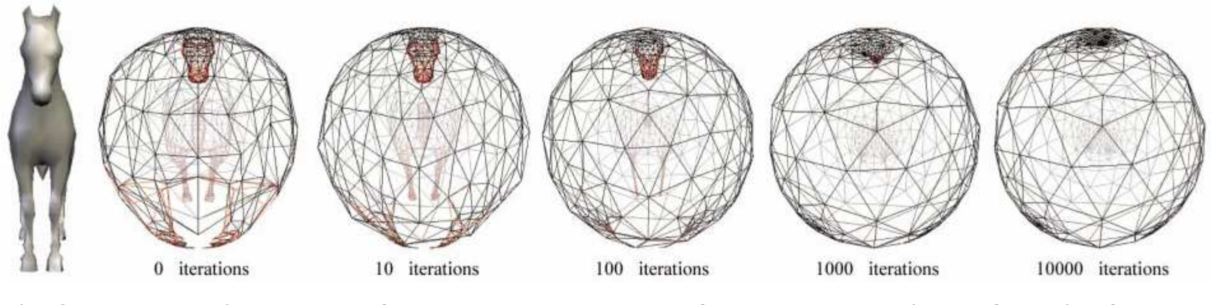


**Fig. 8: The relaxation process of the horse mesh. The red faces are badly oriented faces, i.e. foldovers (taken from [Ale00a]).**

The problem of this method is given by its local nature, it is not guarantied that it finds a valid parametrization. If the iterative procedure does not converge, it is necessary to set up new initial condition, i.e. the initial spherical projection.

### 3.2.2 Parametrization based on decimation

Basic idea of this method was introduced in [Sha98] and further extended in [Pra03, Bir04, Hor99]. The idea is that the mesh is decimated (e.g., by edge collapses) until tetrahedron remains. The tetrahedron is mapped to the unit sphere[17] and then the edges are inserted in the reverse order than they have been removed and new vertices are carefully mapped to the sphere in order to maintain validity of the parametrization.

The original approach [Sha98] introduces a so-called "polyhedron realization algorithm". This algorithm first simplifies the mesh by removing low-degree vertices until a tetrahedron remains. By removing a vertex a hole appears, which has to be retriangulated, reducing so the number of faces. By removing a vertex of degree three, a triangle remains. By removing a vertex of degree four, one diagonal has to be added in order to triangulate the resulting hole. By removing a vertex of degree five, two diagonals have to be added. According to [Sha98] a planar triangular graph must contain at least one vertex of degree smaller than six, thus no other vertex removal possibilities need to be discussed. In the second stage, removed vertices are re-attached in the reverse order maintaining the convexity, i.e. each time the vertex is inserted, its position is optimized so that its local neighborhood is convex. The basic idea of the optimization is as follows. Over the region where a new vertex will be re-attached a ray is constructed. The ray is given by the center of mass of the region and by the average normal of faces adjacent to the region. The ray then intersects planes supported by faces adjacent to the modified region. The intersection closest to the center of mass of the modified region gives a position of the re-attached vertex. The advantage of this approach is that it is able to realize general genus-0 mesh. Shapiro et al. use realized polyhedra further for morphing, i.e. the source and the target polyhedra are merged, in order to obtain a common connectivity graph[18]. It is from our point of view not very suitable to merge polyhedra, although convex, because the merging process is much simpler, when the meshes are mapped to a common parameter domain, i.e. when they are mapped to the same isosurface (the surface of the unit sphere). On the other side, a convex polyhedron can be mapped to the unit sphere easily.

The approach [Bir04] works similarly. Instead of removing vertices, it removes edges (i.e. edge-collapse). Edges are removed according to their lengths (shorter edges first), until a tetrahedron remains. Again, the order in which the edges were removed is recorded and in the re-attachment phase the edges inserted back. By inserting a new edge, a new vertex appears. Position of the new vertex is optimized, so that the resulting parametrization is *barycentric*. Barycentric parametrization for topological discs was described e.g. in [Flo97] and the basic idea is that each vertex in the parameter space is a convex combination of its equally weighted neighbors, i.e.:

$$p_i = \frac{1}{|N_i|} \sum_{j \in N_i} p_j \ , \tag{6}$$

[17] Note that it is simple, because the tetrahedron is a convex polyhedron. Strictly speaking it would be possible to stop the decimation process when some simple polyhedron remains, which we would be able to map to the unit sphere in some easy way.

[18] It is a concept of topological merging which is described in detail in Section 4.2.

where $N_i$ is a set of vertices adjacent to the vertex $p_i$, the weight in this case is the inverse of degree of the vertex $p_i$. In other words, barycentric parametrization maps each vertex to the center of its neighbors[19]. Having barycentric parametrization, [Bir04] further optimizes the position of vertices in the parameter space in order to obtain a *shape preserving* parametrization. Shape preserving parametrization was introduced in [Flo05] and it extends the equation (6) with the use of unequal weights. Weights are computed with respect to the geometric properties of the mesh, i.e. it takes into consideration edge lengths and angles. The shape preserving parametrization reflects geometric properties of the parametrizated mesh. The overall scheme of this approach is depicted in the Fig. 9.

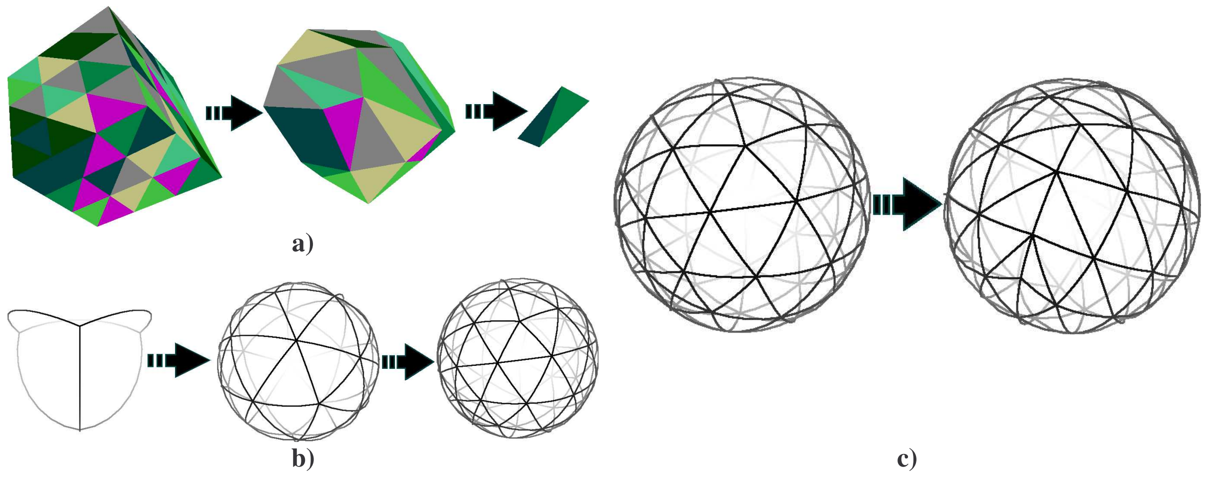


**Fig. 9: Parametrization based on decimation. a) Decimation until a tetrahedron remains. b) Parametrization of the tetrahedron and reattaching of the removed vertices, in this way generating barycentric parametrization c) An optimization of the barycentric parametrization to the shape-preserving parametrization (taken from [Bir04]).**

The approach [Pra03] differs from previous approaches mainly in the optimization step. When re-attaching a new vertex, its position is optimized in order to minimize a *stretch metric* on its adjacent faces. The stretch metric for genus-0 meshes was described in [Pra03] and it is a measure of distortion introduced by parametrization. If we draw a unit circle on the triangle from the parameter space, it will map to ellipse in the object space. The lengths of principal axes $\gamma$ and $\Gamma$of the ellipse are measures how much the distances in the parameter space get stretched in the object space. This situation is depicted in the Fig. 10.

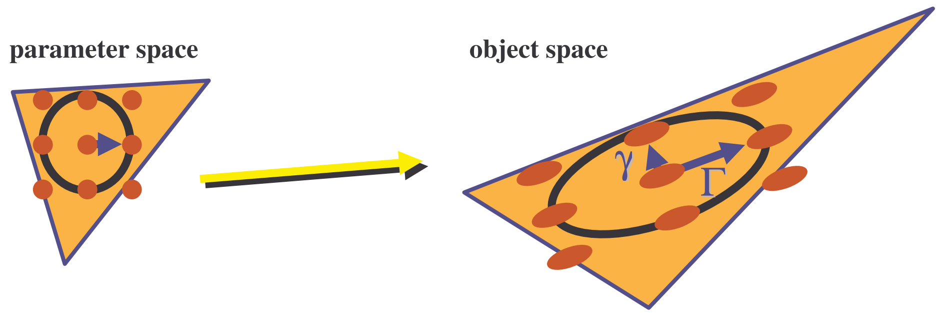


**Fig. 10: The stretch metric. A unit circle drawn on the triangle from the parameter space gets stretched to the ellipse in the object space.**

[19] Note that also Alexa [Ale00a] approaches the barycentric parametrization.

### 3.2.3 Harmonic mapping

Previously described methods worked more or less locally. In relaxation (3.2.1), vertex by vertex is taken and its position is optimized, in decimation-based methods (3.2.2), vertices or edges are collapsed until a tetrahedron remains and then in the reverse order the edges are inserted back. Unlike previous methods, the global methods express requirements on parametrization as a system of equations, which is solved in order to obtain a valid parametrization.

Harmonic mapping $h: M \rightarrow H$ was first used for morphing in [Kan97]. It was originally devised for parametrizing topological discs (i.e. objects topologically equivalent to the unit disc). Let us denote a *boundary vertex*; a vertex which lies on the boundary of the mesh, the non-boundary vertices will be denoted *interior vertices*. In this approach the boundary vertices are proportionally[20] mapped to the unit disc and the remaining (interior) vertices are mapped into the interior of the unit disc in order to minimize the total energy $E_{harm}$, which is:

$$E_{harm}(v_i) = \frac{1}{2}\sum_{j \in N_i} k_{i,j}\left|v_i - v_j\right|^2, \qquad (7)$$

where $N_i$ is the set of the vertices adjacent to the vertex $v_i$, $k_{i,j}$ is an analogy of the spring constant for edge connecting the vertices $v_i$ and $v_j$ and it is computed as:

$$k_{i,j} = \frac{l_{i,k1}^2 + l_{j,k1}^2 + l_{i,j}^2}{\left|\Delta v_i v_j v_{k1}\right|} + \frac{l_{i,k2}^2 + l_{j,k2}^2 + l_{i,j}^2}{\left|\Delta v_i v_j v_{k2}\right|}, \qquad (8)$$

where $l_{i,j}^2$ is the squared length of the edge connecting the vertices $v_i$ and $v_j$, $v_{k1}$ and $v_{k2}$ are the vertices of an adjacent face to the edge $v_iv_j$, $\left|\Delta v_i v_j v_{k1}\right|$ and $\left|\Delta v_i v_j v_{k2}\right|$ are the areas of the triangles formed by vertices $v_iv_jv_{k1}$ and $v_iv_jv_{k2}$. By composing equation (7) for every vertex we obtain a nonhomogeneous[21] linear sparse system. By solving the linear system we obtain a valid parametrization which minimizes a metric dispersion. Metric dispersion is a measure of the extent to which a map stretches regions of small diameter in the parametrical domain [Eck95]. An example of a parametrization by the harmonic mapping is in the Fig. 11. In the Fig. 11 the relatively dense regions of the polygon correspond to the ears and the nose of the cat. It can be seen that the aspect ratios of triangles tend to be preserved.

As stated before, this approach was originally devised for parametrizing topological discs, as the topological discs have a natural polygonal boundary, which is fixed on the circumference of the unit disc. [Kan97] adopted this method also for topological spheres, with that modification that the user has to manually specify the boundary loop which splits a closed object into two topological discs, then two parametrizations of respective parts are computed. Manual specification of the boundary loop (i.e. the cut) significantly influences the resulting correspondence; moreover, the specification of the boundary loop is quite far from an intuitive user control.

[20] Proportionally means that the distances between the consecutive vertices on the circumference of the unit disc are proportional to the angles formed by vertices $p_i$, $o$ and $p_j$, where $p_i$ and $p_j$ are two consecutive boundary vertices and the vertex $o$ is the origin of the unit disc.
[21] Note that the right side of the system is formed by the boundary vertices which are fixed on the circumference of the unit disc.

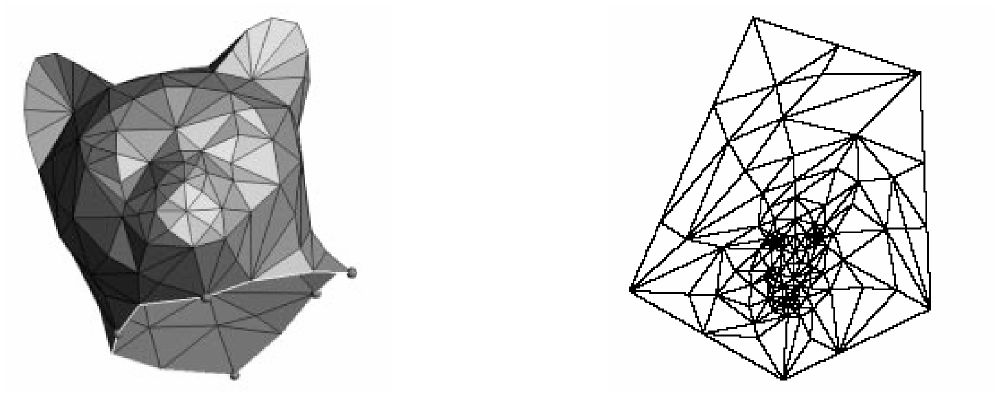

a) b)

**Fig. 11: An example of parametrization by the harmonic map (taken from [Eck95]).**

### 3.2.4 Robust spherical parametrization [She04]

This approach is based on the paper [She04]. In this approach it is operated rather on triangle angles than on the position of triangle vertices. Several conditions[22] for angles are formulated in order to have a valid spherical parametrization. Conditions come, e.g., from the spherical sine and cosine rule, from the fact that the sum of all spherical angles around any vertex in a spherical triangulation sums $2\pi$, etc. Conditions are necessary and sufficient to guarantee a valid spherical parametrization. Clearly there is a lot of valid spherical parametrizations fulfilling these conditions, so it is possible to add some additional constraints. Considering the parametrization as a mapping from the object space to the parameter space, following mapping can be distinguished – conformal (preserves angles), equiareal (preserves areas) and isometric (preserves lengths). So, if we want the parametrization to be conformal, we add an additional constraint in which we specify the so called *target values* of angles. The target value is a desired value which we want to achieve in the parameter space. For a conformal parametrization we set target values of angles equal to angles in the object space. Of course, not all angles will reach its target value in the parameter space, because parametrization always introduces some distortion. So we have a constrained minimization problem, where we minimize the difference between the actual values and the target values under already mentioned conditions. When we want to achieve an area preserving mapping, target values for areas are set equal to the areas of triangles in the object space. Isometric parametrization is possible only for developable surfaces (e.g., planes, cylindrical and conical surfaces)[23].

The result of the minimization problem is the determination of spherical angles, so it is necessary to reconstruct the vertex positions out of the angles by a recursive traversal of the triangulation. We start from an arbitrary triangle, fix one vertex and then according to the spherical sine and cosine rule determine the remaining triangle vertices, and then we determine the vertex positions for the neighboring triangles of the first chosen triangles, etc.

In the Fig. 12 there are examples of parametrizations of a cylindrical mesh (Fig. 12a) using a variety of target values for spherical angles and areas. Fig. 12b) shows a result of Alexa's method [Ale99a], which in fact aims for equal angles in the parametrization. Fig. 12c) is the result of Sheffer et al. [She04] which also aims for equal angles by according specification of the target values of angles. The result is quite similar to Fig. 12b). Fig. 12d) aims for the equal

[22] The conditions are formulated in the form of equalities and inequalities.
[23] It is clear that the isometric mapping implies preserving of angles and areas.

areas. Fig. 12e) aims for conformal mapping (note that the marked triangle stays right-angled) and Fig. 12f) aims for equiareal mapping, i.e. the target values for areas are set to match the areas of the input mesh.

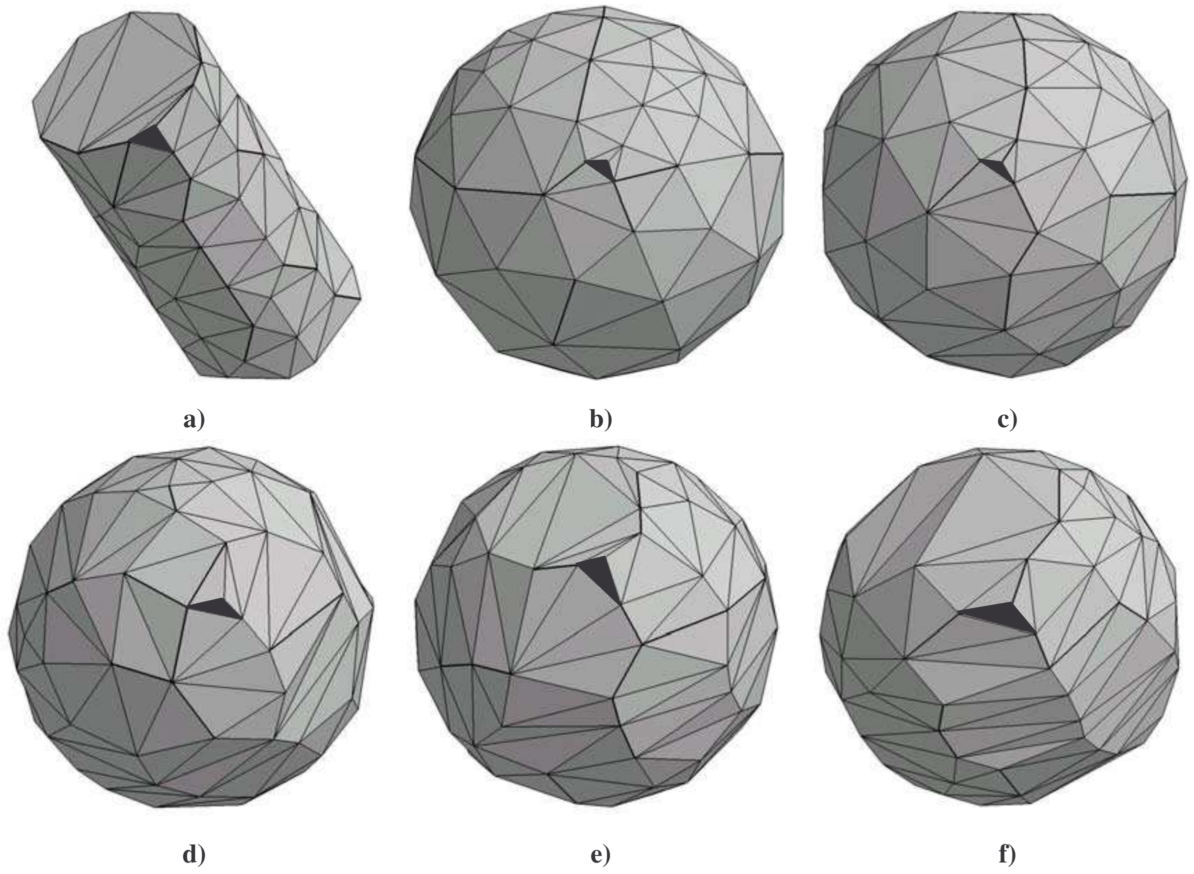


**Fig. 12: A spherical parametrization of a cylinder mesh. a) An input mesh. b) Parametrization by Alexa [Ale99a]. c) Parametrization aiming for equal angles. d) Parametrization aiming for equals areas. e) Parametrization aiming for conformal mapping, i.e. it tries to preserve original angles. f) Parametrization aiming for equiareal mapping, i.e. it tries to preserve original areas. The black triangles correspond in each of the mesh (taken from [She04]).**

### 3.2.5 Parametrization: a summary

As stated before, the parametrization always includes some sort of distortion, which then influences also the correspondence. Let us remind that the correspondence is established by overlaying the source parametrization and the target parametrization. So the distorted parametrization results in some sort of distorted correspondence. An example of the result of the distorted parametrization is in the Fig. 4 (top), the result of the parametrization which reflects[24] the shape of the mesh is in the Fig. 4 (bottom).

Alexa in [Ale01a] investigated methods for parametrization related to morphing. The resulting parametrizations are depicted in the Fig. 13. The Fig. 13b) shows the barycetric mapping. This mapping does not reflect the geometric properties of the mesh. Also the relaxation, i.e. the local iterative approach proposed in [Ale00a], tends to the barycentric mapping. The Fig. 13c) shows a shape preserving mapping [Flo05]. The Fig. 13d) shows a discretized harmonic mapping which was described in the section 3.2.3. In the Fig. 13e) is an

[24] This parametrization was prepared manually by adjusting of the distorted parametrization used for the morphing sequence shown in the Fig. 4 (top).

example of area preserving mapping computed by a recursive approach proposed in [Gre99]. It is visible that the general structure of the parametrizations is in cases b), c) and d) almost the same, as well as the resulting correspondence. The common problem of methods b), c) and d) is an area compression (also visible in the Fig. 11), i.e. the inner triangles have much less area than the outer triangles. This is not the problem of the area preserving parametrization (case e)) but it can be seen that the area preserving mapping leads to a distorted parametrization.

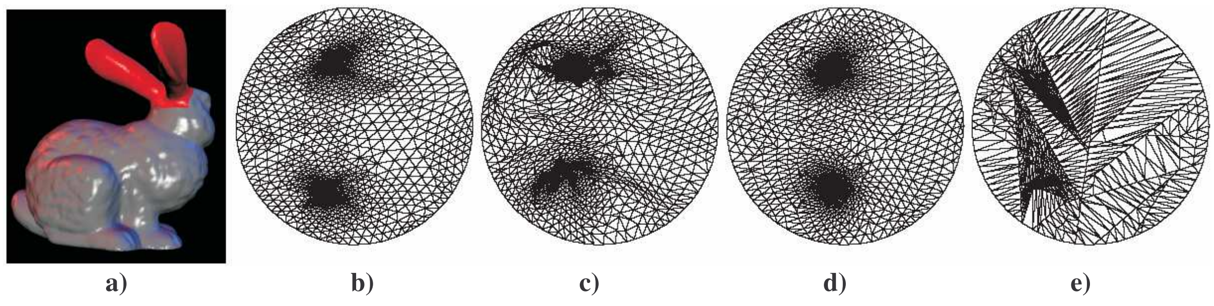


**Fig. 13: A part of the mesh (ears of the bunny) parametrized on the unit disc by various parametrization methods. a) A parametrized region. b) The barycentric parametrization. c) The shape preserving mapping. d) The discretized harmonic mapping. e) Area preserving mapping (images taken from [Ale01a]).**

We introduced a few out of a large number of parametrization methods, which are in our opinion related to the morphing. The local approaches such as relaxation and decimation based parametrization are usually easier to implement than the global methods, because we just apply some local rules. The global approach turns to the solution of some system of equations, which is usually more difficult to implement, but it offers to include some additional constraints, e.g., equal angles, preservation of areas, etc.

## 3.3 Aligning of corresponding features

An important step in establishing of the correspondence is incorporating of some additional user-supplied information[25]. This step is usually included for aligning of some corresponding features, because only the user knows the semantics of the transition. The user intervention usually appears in the following forms:

- establishing a few vertex-vertex correspondences, which are then exploited as much as possible (i.e. for remaining vertices),
- decomposition of the source and the target mesh into components and then establishing the correspondence of individual components.

The former form is much simpler, but do not offer as many possibilities as the latter form. It is based on warping of both the source and the target parametrization, in order to lie the corresponding vertices on the same place in the parameter domain (i.e. on the surface of the sphere). It is described in the following Section 3.3.1. The latter form requires a compatible decomposition and it is described in the Section 3.3.2. In the Fig. 14 is example of the morphing with (top sequence) and without (bottom sequence) feature alignment. The alignment of the legs and the heads leads clearly to a more plausible shape transformation.

[25] Note that till now the process was fully automatic.

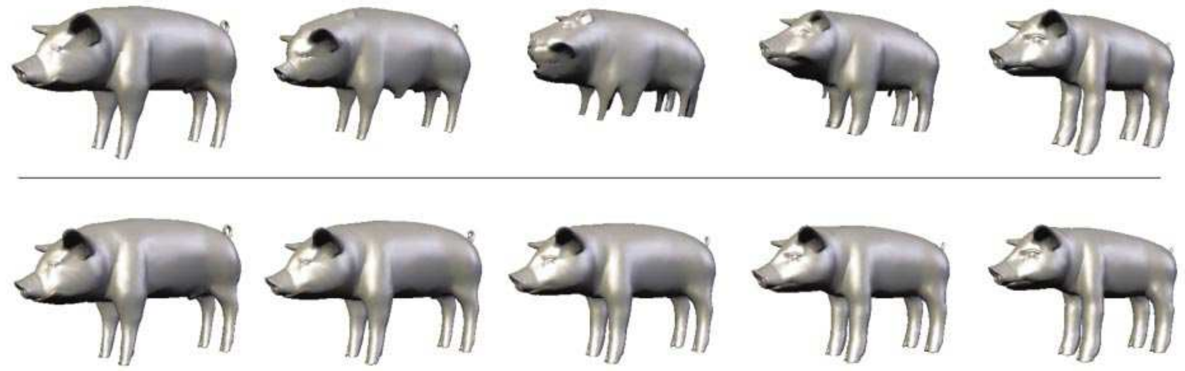


**Fig. 14: Morph between the models of a young pig and a grown-up pig. The top sequence shows a morphing transition without aligning of the corresponding features. The bottom row sequence is a transition with legs and heads aligned (taken from [Ale99a]).**

### 3.3.1 Parametrization warping

When establishing several vertex-vertex correspondences (*pairs of feature vertices*), we state that we want the given vertex to grow out of its corresponding counterpart. For this it is necessary to force both vertices to lie on the same place in the parameter domain. In a general case they lie at distinct places so it is required to move them, but the vertices cannot be moved arbitrarily, because the parametrization must remain valid (i.e. no foldovers must appear). The source and the target parametrization have to be carefully warped in such a way that the move of the vertices does not produce foldovers.

In [Ale99a] the following scheme for aligning of the corresponding features is suggested:

1. A rotation of one of the unit spheres (i.e. the parameter domain) so that the sum of squared distances is minimized.
2. A warping of both parametrizations to force the corresponding vertices to the same place in the parameter domain.

The warping algorithm is based on local improvements rules. The pairs of corresponding vertices ( $v_i^0, v_j^1$ ) vertices positions are optimized in order to achieve the global goal, i.e.:

$$v_i^0 = v_j^1, \tag{9}$$

where the vertex $v_i^0$ is the vertex of the source mesh, the vertex $v_j^1$ is the corresponding counterpart in the target mesh and together form a pair of corresponding vertices. In other words, by the Eq. (9) we say that we want the vertices $v_i^0$ and $v_j^1$ to be on the same place in the parametrical domain. Thus, the vertex $v_j^1$ has to be warped towards the vertex $v_i^0$. The warping function which moves the vertex at the position *v* (together with its neighborhood) to the position *w* is:

$$f(x) = \begin{cases} \left\| x + c(d - |x - v|)(w - v) \right\| & , |x - v| < d \\ x & , |x - v| \geq d \end{cases}, \; x \in V^1, \tag{10}$$

where (*w*-*v*) is the direction of the move, *d* is the radius of influence, i.e. the size of an area on the surface of the sphere which is influenced by moving of a particular vertex. The term $(d - |x - v|)$ is a linear falloff function which controls the amount of the move of the vertex *x* depending on the distance between the vertex *x* and *v*. The warping function (10) is applied on all the vertices of the target parametrization; when the vertex *x* is out of the area of influence (i.e. $|x - v| \geq d$ ), the vertex position remains unchanged. It means that not only the vertex *v* is moved towards its destination position *w* but also a neighborhood (specified by *d*) of the vertex *v* is moved. Note that this warping function does not avoid foldovers, so after an application of the warping function, the parametrization has to be checked and when foldovers occur, the constant *c* is decreased[26]. After warping, the new position of the vertex has to be normalized (the ||.|| notation), to push the vertex back to the surface of the sphere. The warping process is repeated for each pair of feature vertices.

This local approach is quite easy to implement but has also disadvantages, which are given by its local nature. By optimizing both parametrizations for one pair of feature vertices, we could damage the already established mapping. This problem becomes more evident when feature vertices of one mesh are mutually close.

### 3.3.2 Decomposition

Looking back to the warping of parametrization, we should consider it as some sort of postprocessing of the parametrization, i.e. given some initial parametrization (e.g. barycentric), deform it in order to fulfill some goal (e.g. Eq. (9)). The decomposition approach is in this meaning some sort of preprocessing, i.e. first decompose the mesh into individual components and then compute parametrization per individual component. An advantage of this approach is that by decomposition and establishing the correspondence between the components we naturally guarantee that the corresponding components remain aligned.

The methods based on decomposition work as follows. The source and the target mesh are decomposed into components and for each component of the source mesh a corresponding component in the target mesh is established. This yields that the decomposition has to be compatible, i.e. the source and the target mesh have to be decomposed to the same number of components and the components must have the same adjacency in both meshes. The decomposition process should be manual (e.g. [Kan00, Zoc99, Gre99]) or semi-automatic. Semi-automatic decomposition methods are based on finding a common coarse domain, which is obtained by mesh decimation. Individual faces of the coarse domain are handled as separate components.

In the Fig. 15 is an example of the decomposition based approach. In fact, it is a hybrid of both parametrization warping and decomposition approaches. A coarse correspondence is established by decomposition and in each component individual vertices can be aligned by a parametrization warping process. Coloring of input meshes corresponds to decomposition. Note that the decomposition is compatible and in this case it is a result of a manual interaction. Blue marked vertices are vertices which are further aligned in the context of one particular component using a parametrization warping process. As can be seen on Fig. 15c) and d) parameter domain is a unit discs (as the mesh is decomposed into individual patches) and a cylindrical surface (which is a natural parameter domain for cylinder-like surfaces).

[26] In [Ale99] the initial value of the constant *c* is *c*=0.5.

Parameter domains are colored with respect to coloring of components of input meshes. Fig. 15e) shows overlaid parametrizations and on Fig. 15e) are intermediate meshes which are constructed by a process described in Section 4.2.

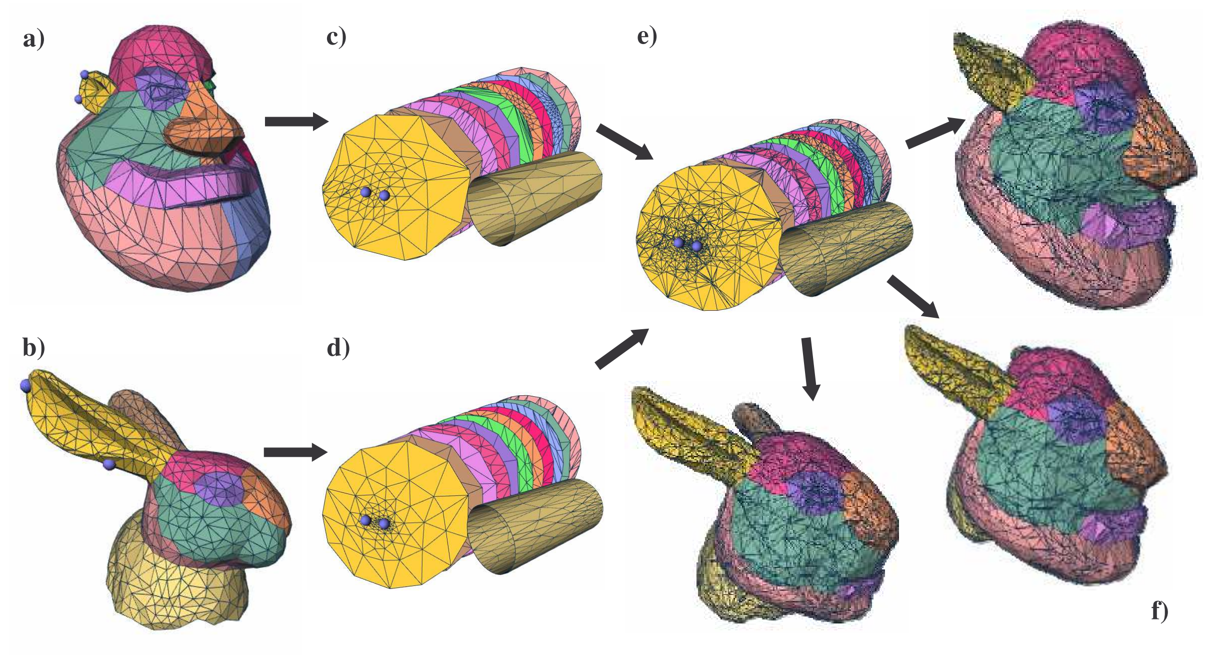


**Fig. 15: Different stages of the decomposition based morphing approach. a) and b) are the source and the target mesh, coloring corresponds to the decomposition, c) and d) are parametrizations of individual components. e) Shows overalying and merging of corresponding components. f) Intermediate morph models (taken from [Zoc99]).**

Zhao et. al [Zhao03] further extend the decomposition idea by incorporating a *null-component* concept. Null-component is a component which has no corresponding counterpart in the other mesh, so it will disappear (or grow) during the morphing sequence.

Next we will mention the approach by Gregory et al. [Gre99], which is an example of an extreme user interaction. The user has to specify a pair of feature vertices on each of the input mesh. Then the shortest paths along edges between the pair of features vertices are computed. Shortest paths of the source and the target mesh define corresponding chains. The problem is that the user has to a specify sufficient number of chains, so that it covers the whole mesh. It means that each feature vertex has to be adjacent to at least two chains and each chain must have a connected patch on each side. Moreover the structure of chains has to be isomorphic on both input meshes. Then the mesh is automatically decomposed to patches according to the specified chains and individual corresponding patches are morphed separately. A lot of user interaction is rewarded with very good results as depicted on Fig. 16. The author fairly admits that the time needed to manually construct the decomposition from Fig. 16 was approximately 6 hours. On the other side the input meshes were quite complex – 5660 triangles (human) and 17528 triangles (triceratops), and there was 86 patches needed.

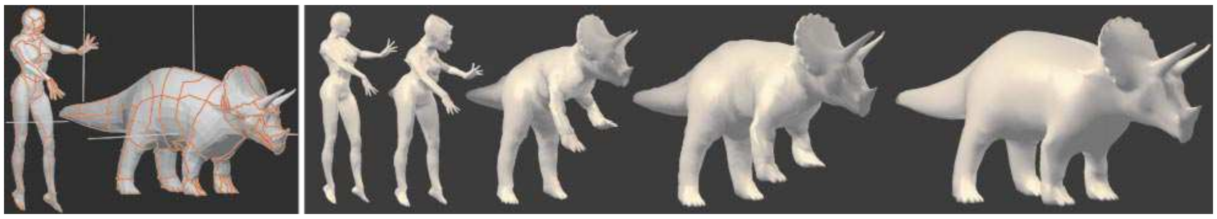


**Fig. 16: Decomposition approach Gregory et al. [Gre99]. The lines on the left show the isomorphic decomposition of the input meshes (image taken from [Gre99]).**

## 3.4 Correspondence: a summary

Let us summarize the establishing of the correspondence. The correspondence is the crucial step in the morphing process. The correspondence is established by mapping both the source and the target mesh to a common parameter domain. Both parametrizations are then overlaid and the correspondence problem turns to the point location problem of the vertices of the source mesh inside the faces of the target parametrization and vice-versa in parameter domain.

The methods for parametrization could be divided into local methods, which are based on local improvements rules applied on vertices, and global methods, usually based on the solution of some system of equations. During the establishing of correspondence it is possible to add some user-supplied information by that the resulting shape transition can be controlled. The user intervention could be divided into postprocessing (adjusting of an already established parametrization, e.g. parametrization warping) and preprocessing (decomposition into components which are parametrized individually).

# 4 Supermesh construction

In this section we will describe the construction of a new mesh, based on the previously established correspondence. The newly constructed mesh will be denoted as *supermesh* in the further text[27]. The supermesh is constructed by a process called *topology merging*, i.e. by refining of topology of the original meshes, and it has so-called *shared topology* of both the source and the target mesh. This section will not be so varied as the previous one, because the construction of supermesh is usually the same in the majority of morphing approaches.

## 4.1 Shared topology

The shared topology is a topology which results from merging of two existing topologies[28]. We will first discuss the shared topology in general, i.e. not specifying which topologies do we exactly merge in the context of morphing, we will merge just general triangulations[29]. The merging process consists of three steps:

1. overlaying of triangulations, i.e. putting the triangulations each over other (sometimes denoted as a graph overlay),
2. "welding" of topologies in *mutual intersections*, i.e. it is necessary to compute intersections between the overlaid triangulations,
3. local triangulation, i.e. inserting new edges in order to have the resulting shared topology triangular.

The Fig. 17 demonstrates the shared topology in 2D. In the Fig. 17a) there is a part of the triangulation $M^0$ (in green) and a part of the triangulation $M^1$ (in red) overlaid (the dashed line outline the continuation of the triangulation). In the Fig. 17b) mutual intersections are depicted and finally the Fig. 17c) shows both triangulations merged in the intersections. The orange marked region shows the edge of $M^0$ inserted into the triangulation $M^1$. As can be seen on Fig. 17 c), there arise new edges (dashed), which have to be inserted in order to have the resulting topology triangulated.

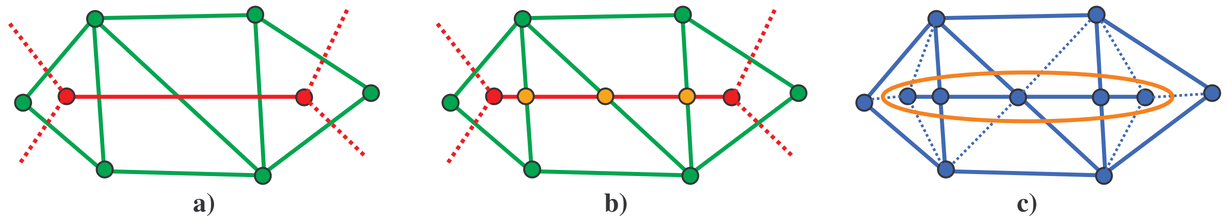


**Fig. 17: Computation of the shared topology.**

[27] In the literature the supermesh appears also under the terms metamesh [Lee99], combination mesh or interpolation mesh [Kan97, Kan99].
[28] Let us remind that the term topology appears here in the meaning of the vertex/edge/face structure.
[29] We suppose that the meshes which are we working with are triangulated, i.e. each face is a triangle.

As it can be seen on the figure the resulting shared topology contains the vertices of the triangulation M$^0$, the vertices of the triangulation M$^1$ and the new vertices which arise from the mutual intersections, i.e.:

$$V = V^0 + V^1 + I\,, \qquad (11)$$

where $V^0$ is the set of the vertices of the triangulation $M^0$, $V^1$ is the set of vertices of the triangulation of $M^1$ and $I$ is the set of intersections.

In the Fig. 17 we inserted the blue triangulation into the red triangulation. Note that this process can be inverted with the same result, i.e. it does not matter whether M$^0$ is merged with M$^1$ or M$^1$ with M$^0$, as the merging is a symmetric operation.

The concept of topology merging is not used only for morphing of meshes. It was used, e.g., by Alexa et al. in [Ale00b] for construction of *compatible triangulations* of polygons. Triangulations are compatible if their graphs are *isomorphic*, i.e. if they contain the same number of vertices which are connected in both graphs in the same way. A compatible triangulation is computed as follows. First, both polygons are encapsulated into a convex region, the interior and exterior (i.e. the region between the polygon and the convex boundary) are triangulated by an arbitrary method (e.g., Delaunay triangulation, some sort of ear clipping, etc.). Triangulations of both polygons are overlaid; the shared topology is computed and projected back on the input polygons.

Now we will describe the application of the topological merging for the case of morphing of meshes.

## 4.2 Topology merging for mesh morphing

The concept of topology merging for mesh morphing, as far as we know, was first introduced in [Ken92]. In the case of mesh morphing we do not merge input meshes directly, because the edges of input meshes are in a general case nonparallel and nonintersecting, so the mutual intersection cannot be computed. Instead of the direct mesh merging we merge the parametrizations of the source and the target mesh, where the edges lie on the surface of the sphere. The edges become parts of great circles in the parameter domain, so the intersection test for the spherical case has to be used. The calculation of intersection is described in Appendix A.

### 4.2.1 Finding intersections

An important procedure in the topological merging is finding intersections of the overlaid parametrizations. This can be achieved by a brute force algorithm by checking every edge of the first parametrization against every edge of the other parametrization. This algorithm has the worst case complexity $O(N^2)$. A brute force algorithm is not very suitable for complex meshes, because it is very time consuming in the case of complex meshes. In [Ken92] and [Ale99a] better algorithms based on walking were described. Both algorithms work basically in the same way; they differ only in the used data structure. The former approach by Kent et al. [Ken92] uses a variation of the winged-edge data structure, the latter approach uses the

DCEL[30] data structure. We will describe here the first approach which uses the winged edge data structure, because we use this structure also in our implementation.

The algorithm is based on the idea that for each edge of the source parametrization we can construct a *candidate list* (CL) of edges of the target parametrization. The candidate list contains edges of the target parametrization which can be eventually intersected by the edges of the source parametrization. For each vertex $v_i$ of one parametrization we know the mapping to the face $f_j$ of the other parametrization, so the edge originating from $v_i$ intersects just one of three edges of the face $f_i$. When the intersection $I$ is encountered, the algorithm "walks" to the face $f_k$, which is adjacent to the intersected edge, and the edges of $f_k$ are added into the candidate list. This situation is also depicted in the Fig. 18.

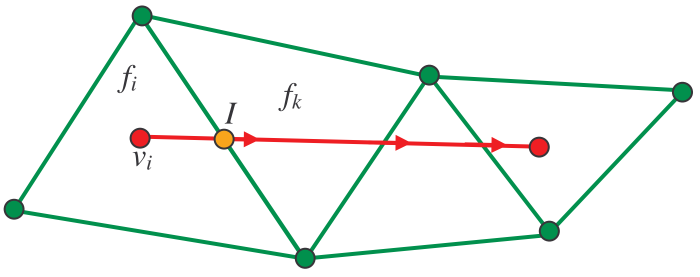


**Fig. 18: A demonstration of the walking algorithm. Edges of green triangles are successively added into the candidate list as they are encountered during the walking.**

The algorithm written in pseudocode is as follows:

```
Input: a parametrization $P_0$ of the source mesh, a parametrization $P_1$ of
the target mesh
Output: a list of mutual intersections of edges of the models $M_0$ and $M_1$
(IL)

// WL ... Work list, list of edges which are going to be processed
// CL ... Candidate list, list of edges which could possibly intersected

1:   $v_1 \leftarrow$ the first vertex of $P_0$
2:   $f \leftarrow$ face of $P_1$ to which $v_1$ maps
3:   add all edges incident to $v_1$ into WL
4:   while WL is not empty do
5:        $e_a \leftarrow$ the next edge from WL
6:        $v_1$, $v_2 \leftarrow$ endpoints of edge $e_a$
7:        add edges of the face $f$ into CL
8:        while CL is not empty do
9:             $e_b \leftarrow$ the next edge from CL
10:            if an intersection $I$ between $e_a$ and $e_b$ exists then
11:                 add the intersection $I$ into the IL
12:                 $f \leftarrow$ the second incident face to $e_b$
13:                 add remaining two edges of $f$ into CL
14:       add edges incident to $v_2$ into WL
```

It is clear that we must have information about faces incident to a particular edge and information about edges incident to a particular vertex in the data structure.

[30] DCEL – Doubly Connected Edge List.

In Section 3.1 it is described how mapping of vertices $\Omega_{0\to1}$ and $\Omega_{1\to0}$, i.e. mappings of vertices of one mesh to the faces of the other mesh and vice-versa, can be computed by brute force approach. The walking process described above can be used for establishing of mapping, too. The algorithm walks along an investigated edge in order to find all intersections. When the end of the edge is reached, i.e. no other intersections are encountered, we know in which face the end of the edge was reached. So we know the mapping of the vertex to a face. It only remains to find a mapping for the first processed vertex ($v_1$ in the pseudocode) and it can be established by a brute force search, by the walking algorithm or by some other point-in-triangle method. The mapping of the remaining vertices will be established by the walking algorithm together with the intersection computation.

### 4.2.2 Supermesh construction and transformation

In this step the overlaid parametrizations are merged in the intersections. This is again dependent on the used data structure and particular descriptions are given, e.g., in [Ken92] or [Ale99a]. Briefly the algorithm works as follows. Each edge of the source mesh has associated list of intersections with the edges of the target mesh. Assuming the intersection list is sorted, the intersection list is traversed along with splitting the edge in the intersection by inserting a new vertex. In a similar way the intersection lists of the edges of the former target mesh are processed with that difference that the edges of the former target mesh are not splitted but merged in vertices, which have been already inserted during splitting the edges of the former source mesh. By this method, both input models are “welded” in the intersections, creating so a new mesh – a supermesh. It is described by a set of vertices and edges and by the relation between them. For rendering purposes it is necessary to extract faces. Triangulation of the supermesh is described in the Section 4.2.3.

Until now the whole process (including establishing correspondence, overlaying, finding intersections and merging) worked in the parameter domain. So the topologically correct supermesh, which can be transformed to the shape of the source mesh as well as to the shape of the target mesh, was computed, but geometrically, all points of the supermesh lie on the surface of the unit sphere. So, now we will go over from the parameter space to the object space. Let us first denote $v_i(0)$ as the position of the *i*-th vertex in time $t=0$ and analogously $v_i(1)$ as the position of the *i*-th vertex in time $t=1$. So it remains for each vertex to establish its two extreme positions, i.e. the position of the vertex $v_i(0)$ (the supermesh transformed to the shape of the source mesh) and the position of the vertex $v_i(1)$ (the supermesh transformed to the shape of the target mesh).

Let us remind that the supermesh consists of three types of vertices, i.e.:

- *source vertices* – the vertices of the former source mesh,
- *target vertices* – the vertices of the former target mesh,
- *split vertices* – the intersection vertices originating from mutual intersections of edges of the source and the target mesh.

For each type of vertices the establishing of the extreme positions will be slightly different. First the case of the source vertices will be discussed. Let the source vertex $v_i$ of the supermesh is the vertex $v_i^0$ of the former source mesh. As the source vertices are the vertices of the former source mesh, we know the position $v_i(0)$. When establishing the position $v_i(1)$, the mapping $\Omega_{0\to1}$ of the vertex $v_i^0$ to the surface of the target mesh is used. The mapping gives us the face $f_j^1$ to which the vertex $v_i^0$ maps and its relative position in the face $f_j^1$ expressed by barycentric coordinates[31]. Note that the barycentric coordinates were established in the parameter space, but the trick is that we use these barycentric coordinates from the parameter space for computing $v_i(1)$ with respect to the face $f_j^1$ in the object space, i.e.:

$$v_i(1) = t.p_1 + u.p_2 + v.p_3\,, \qquad (12)$$

where $t$, $u$, $v \in$ <0;1> are barycentric coordinates established in the parameter space of the point $v_i^0$ with respect to the face $f_j^1$ comprised to the vertices $p_1, p_2, p_3$ in the object space. This implies that the vertex lies in the plane of the face formed by vertices $p_1, p_2, p_3$. This method is used in the majority of morphing approaches. Alexa in [Ale01a] suggests not to place vertices in the plane of the face to which a particular vertex maps, but to place a vertex so that it results in a smooth surface (e.g. to place a vertex according to the barycentric coordinates on the triangular patch defined by face vertices and their normals, etc.).

For the case of the target vertices, we know the position $v_i(1)$, so it remains to establish $v_i(0)$. It is similar as in the case of the source vertices with that difference that we use the mapping $\Omega_{1\to0}$ of the target vertex to the surface of the source mesh. So the vertex $v_i(0)$ lies in the face $f_k^1$ to which the vertex $v_i^1$ maps.

Each split vertex is formed by the intersection of the edge $e_s$ of the source mesh and the edge $e_t$ of the target mesh. So the position $v_i(0)$ of the split vertex is interpolated from the endpoints of the edge $e_s$ by the method proposed above. The position $v_i(1)$ of the split vertex is interpolated from the endpoints of the edge $e_t$.

### 4.2.3 Local triangulation

By the topology merging process described in the Section 4.2.2 we obtain a supermesh described by a set of vertices and associated *edge fans*, i.e. edges adjacent to a particular vertex. For rendering purposes it is suitable to extract faces, so each edge fan is traversed and locally triangulated by inserting some edges. In [Kan97] the following procedure is suggested. For each vertex of the supermesh a list of adjacent edges (i.e. the edge fan) is traversed by checking whether the endpoints of two successive edges are connected by an edge. If they are not connected, a new edge is inserted together with the creation of a new face. The suggested procedure implies that it is necessary to know for each vertex a list of adjacent edges, i.e. some sort of winged edge data structure is needed. Furthermore, the list of adjacent edges has to be sorted in order to connect adjacent edges. The algorithm is demonstrated in the Fig. 19. The list of adjacent edges of the central vertex is traversed and if necessary, the bold edges are inserted along with creation of a new face.

[31] See Fig. 6.

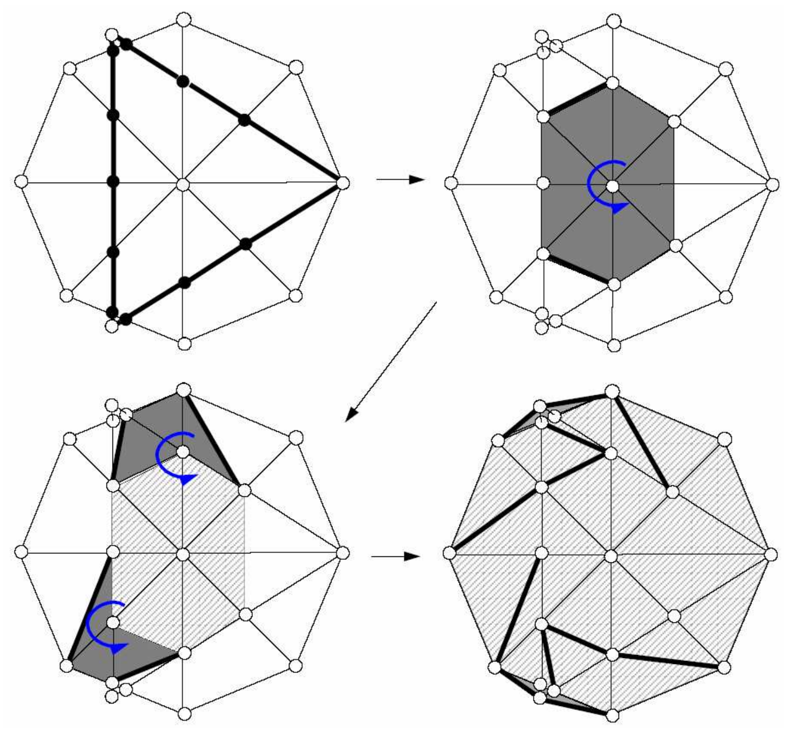


**Fig. 19: Demonstration of the triangulation procedure (taken from [Kan97]).**

The algorithm written in pseudocode is as follows:

```
Input: supermesh M, each vertex $v_i$ has an associated list of incident
edges $N_i$
Output: triangulated supermesh M

1: for each vertex $v_i$ of M do
2:   $N_i \leftarrow$ the list of edges incident to the vertex $v_i$
3:   sort the list $N_i$ angularly
4:   $e_j \leftarrow$ the first edge from $N_i$, $v_j \leftarrow$ endpoint of the edge $e_j$
5:   for each edge from the list $N_i$ do
6:        $e_k \leftarrow$ the next edge from the list $N_i$
7:        $v_k \leftarrow$ the endpoint of the edge $e_k$
8:        if edge connecting vertices $v_j$, $v_k$ does not exist in M then
9:             insert a new edge $e_{new}$ formed by vertices $v_j$, $v_k$ in M
10:            add $e_{new}$ into the lists $N_j$, $N_k$
11:            re-sort lists $N_j$ and $N_k$
12:       else $e_{new} \leftarrow$ the edge connecting vertices $v_j$, $v_k$
13:  construct a new face using edges $e_j$, $e_k$, $e_{new}$ and vertices $v_i$, $v_j$, $v_k$
```

The complexity of the algorithm is $O(N)$, where $N$ is the total number of vertices in the supermesh[32]. The number of faces of the resulting supermesh is strongly dependent on the mutual orientation of the source and the target mesh, i.e. on the number of mutual intersections. In [Par03] an analysis of the number of faces of the supermesh is given, it follows that:

$$|F| = |F^0| + |F^1| + |I| + |E_{tr}| - 2, \tag{13}$$

[32] $N=|V^0|+|V^1|+|I|$, where $|V^0|$ is the number of vertices of the source mesh, $|V^1|$ is the number of vertices of the target mesh and $|I|$ is the number of intersections.

where $|F|$ is the total number of faces of the supermesh, $|F^0|$, resp. $|F^1|$, is the number of faces of the source mesh, target mesh, respectively, $|I|$ is the number of mutual intersections and $|E_{tr}|$ is the number of edges inserted in the triangulation process. A little disadvantage of this approach is that the ordered list of edges adjacent to a vertex has to be maintained, so it requires some method of angular sorting in 3D.

## 4.3 Supermesh construction by remeshing

The advantage of topological merging is that we can simply obtain compatible triangulations of meshes with different topology. On the other side, the topology merging process brings following problems. It requires rather unstable computation of mutual intersections (during parametrization overlay) and it produces meshes with higher number of faces than the original input meshes.

These disadvantages were removed by the approach by Michikawa et al. in [Mic01]. The supermesh is constructed by subdivision fitting process. It works as follows. First, a base supermesh is manually constructed. The base supermesh is a coarse version of both the source and the target mesh. Then, the source and the target mesh are partitioned into several patches according to the faces of the base supermesh, i.e., each face of the source and the target mesh is associated with one face of the base supermesh. Each face of the base supermesh serves as a parameter domain for the corresponding patch. The individual patches are parametrized using Floater's shape preserving parametrization [Flo05]. Using parametrization, the base supermesh is remeshed by 4-to-1 splits[33]. Each time a new vertex is created, the inverse mapping (from parameter space to object space) is used to compute a position of the vertex on the surface of the source mesh and on the surface of the target mesh. This approach is schematically depicted in the Fig. 20

The advantage of this approach is a semiregular connectivity, i.e. the faces of the base supermesh are remeshed in a regular fashion. It is also easy to extend it to more then two meshes. For rendering of a coarser version of the supermesh the normal maps are used. On the other side, the remeshing could bring problems, because it is just sampling of the mesh in the parameter domain, so the remeshed mesh is only an approximation of the original object. Another problem lies in manual creation of the base supermesh, which requires a lot of user interaction. Let us remind that the base supermesh has to be common for both the source and the target mesh. So it requires finding an isomorphic structure of cuts on both input meshes.

We made some experiments with remeshing in 2D. The 2D analogy of parametrization is mapping of the vertices of a polygon to the circumference of the unit circle [Gom97]. An analogy of the remeshing in 2D is a sampling of the unit circle. For each sample point two vertex positions are recorded, i.e. the position of the vertex on the boundary of the source polygon (it is obtained from the parametrization of the source polygon) and the position of the vertex on the boundary of the target polygon (it is obtained from the parametrization of the target polygon). Of course, it depends on the density of sampling.

[33] 4-to-1 split is a subdivision of one face into four sub-faces. New vertices are inserted in the midpoints of the original face edges.

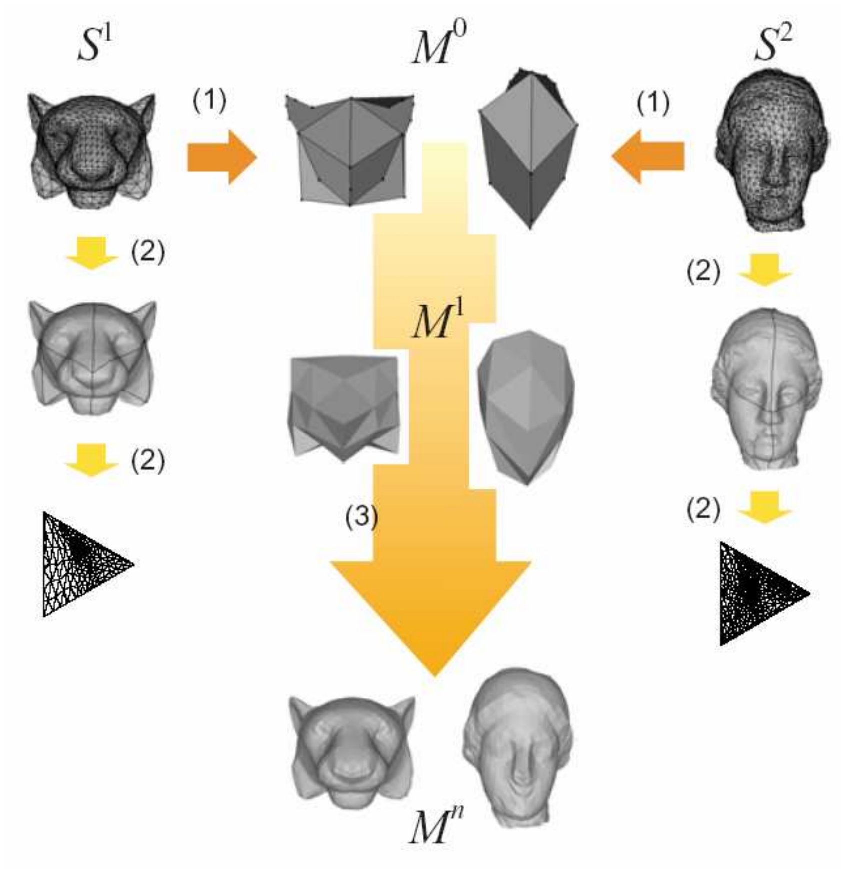


**Fig. 20: Construction of the supermesh by remeshing. (1) Creation of a base supermesh $M^0$ from the source mesh $S^1$ and the target mesh $S^2$. (2) Mesh partition according to the base supermesh and parametrization. (3) Subdivision of the base supermesh (taken from [Mic01])**

The results are depicted in the Fig. 21. On the left there is a source and the target polygon. The top row sequence shows a resampled polygon with 14 vertices. It is visible that the sampling is not sufficient because the shapes in time $t$=0 and $t$=1 does not correspond to the original source and the target polygons. The middle row sequence shows a resampled polygon with 30 vertices. The shapes in time $t$=0 and $t$=1 are more similar to the original shapes than the case with 14 vertices. The sampling in both cases was regular, i.e. in the first case it was with the constant step $2\pi/14$ and in the second case it was with the constant step $2\pi/30$. The bottom row sequence shows a superpolygon constructed by 2D analogy of topology merging. 2D analogy of topology merging is a simple ordering of vertices according to the angle. This superpolygon has only 14 vertices (i.e. the same number as the top row sequence using "remeshing" approach), and the shapes in time $t$=0 and $t$=1 match the original shapes exactly. So it seems that by topology merging we can achieve better results with the lower number of vertices than with the same number of vertices using remeshing approach. The lower number of vertices is paid by more complex computations.

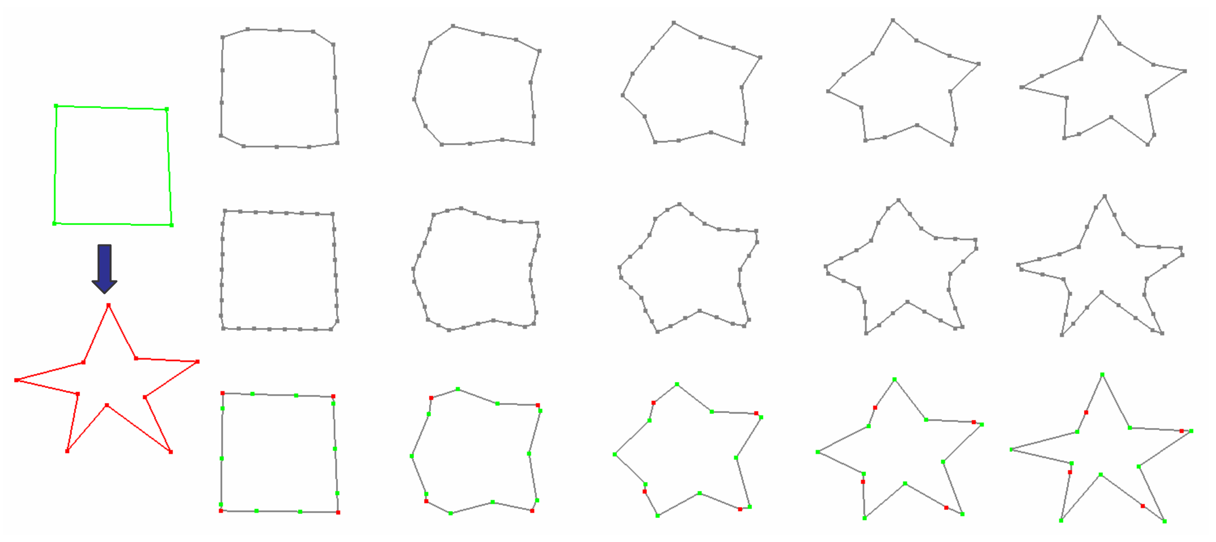


**Fig. 21: Comparison of resampling approach and topology merging. The top row sequence shows morphing with a resampled polygon with 14 vertices. The middle row sequence show morphing with a resampled polygon with 30 vertices. The bottom row sequence is the result of 2D topology merging process.**

# 5 Interpolation

In this section we will address the interpolation problem. We will discuss the interpolation of geometry and interpolation of surface attributes such as normals, colors, textures etc. The first problem appears in the literature (e.g., [Ale00a, Ale00b, Ale01a, Ken97]) also as a vertex-path problem. The latter problem is not very discussed in the literature and we will describe it here as a part of our contribution published in [Par04a, Par04b].

## 5.1 Geometry interpolation

### 5.1.1 Absolute vertex position interpolation

The simplest way how to interpolate between two vertex positions is a linear interpolation, i.e. the corresponding vertex travels along the line connecting two extreme vertex positions, i.e.:

$$v_i(t) = v_i(0) + t(v_i(1) - v_i(0)), \tag{14}$$

where $v_i(t)$ is the position of the i-th vertex of the supermesh in time $t$, $v_i(0)$, $v_i(1)$, resp. are the extreme vertex positions in time $t$=0, $t$=1, respectively. This simple approach is used in the majority of morphing approaches. As stated in [Ale01a], the linear interpolation works well for morphing objects which are rather similar and are oriented in a similar way. For objects with different shapes the linear vertex interpolation may introduce a self-intersection or some sort of collapsing, which is usually not a very plausible effect.

An interpolation of higher degree is also possible. It yields a smoother vertex path, but on the other side it requires to add some additional information, e.g. in the form of tangents for Hermite interpolation, control vertices for Bezier interpolation, etc. In [Mic01] it is, e.g., suggested to use vertex normals as tangents for Hermite interpolation. A good idea was introduced in [Gre99] where the user is allowed to specify tangent vectors for the vertex path. The modified trajectory is then spreaded with some falloff to the neighboring vertices. By a proper tangent vector specification some cases of a self-intersection can be avoided.

### 5.1.2 Interpolation of intrinsic parameters

Intrinsic parameters are, e.g., edge lengths or angles between adjacent edges or faces, face areas, etc. By interpolation of these parameters it is possible to force the angles and edges to change monotonically, i.e. they do not degenerate or flip inside out. An example of such an intrinsic representation is the edge-angle representation mentioned in Section 2.4. Approaches using the intrinsic parameter interpolation work generally as follows. First a forward transformation from the absolute vertex coordinates to representation by intrinsic parameters is computed. Then the intrinsic parameters are interpolated. And finally the backward transformation is computed (i.e. the computation of absolute vertex coordinates). In the following sections we will introduce three approaches of intrinsic parameters interpolation, where each of them uses a different set of intrinsic parameters.

### 5.1.3 As-rigid-as-possible shape interpolation [Ale00b]

In this approach the vertex positions are not interpolated directly, instead, individual transformation matrices for each face are interpolated, which then determine the vertex positions. The main idea of [Ale00b] is similar to [Sur01, Sur04]. First both input objects are isomorphically dissected, i.e. compatible triangulation of input objects is computed. Then the interior of the shapes rather than their boundaries is blended. In [Sur01, Sur04] the interior of the polygon is interpolated by interpolating barycentric coordinates. Alexa opposes not to interpolate barycentric coordinates as they are not related to physical or aesthetical principles. Instead of this it is suggested to interpolate the interior of the shape according to a *least-distorting morph*. Interesting is how the least-distorting morph is computed. First a case of only one transforming triangle is considered. In this case we have only the triangle orientation in time the $t$=0 and the orientation for the time $t$=1. Having three pairs of vertices we are able to establish the affine transformation[34] matrix $A$ and the translation vector $v$. The translation vector $v$ does not contribute to the shape transformation itself so we will consider only the affine transformation matrix $A$. Then a simple solution for the transition would be $A(t)=(1-t)I + tA$, where $I$ is the unit matrix. But by a simple linear interpolation of the transformation matrix, the following issues, which are often required, cannot be guaranteed:

- the transformation should be symmetric with respect to time,
- the angles and scale should change linearly and monotonically,
- the triangle should keep its orientation,
- the resulting vertex paths should be simple

Another possible solution is singular value decomposition [SVD] of the transformation matrix, which results in a decomposition of the matrix $A$ into an orthogonal, a diagonal and again an orthogonal matrices, i.e., $A=U.S.V^T$ where $U$, $V^T$ are orthogonal rotation matrices and $S$ is a diagonal matrix containing scale coefficient. However, through experimentation, Alexa found that the decomposition into one single rotation matrix and a symmetric matrix yields better result, i.e.:

$$A = R_\gamma S,\ S = \begin{bmatrix} s_x & s_h \\ s_h & s_y \end{bmatrix}, \tag{15}$$

where $R_\gamma$ is the rotation matrix, $s_x$, $s_y$ are the scale coefficients and $s_h$ is the shear coeffiecient. Then the matrix $A(t)$ is computed as:

$$A(t) = R_{\gamma t}((1-t)I + tS), \tag{16}$$

where $R_{\gamma t}$ is time dependent rotation matrix computed as $R_\gamma(t)=(1-t)I+tR_\gamma$, $S$ is the symmetric matrix from the Eq. (15). Now let us consider a complex triangulation; transforming one single triangle involves also the transformation of adjacent faces, so a desired transformation for one triangle is not usually the desired transformation for the adjacent faces. Then the solution turns to a minimization problem of a quadratic error between the desired transformation $A$ and the transformation induced by a simple linear interpolation. Comparison

[34] An affine transformation is any transformation that preserves collinearity (i.e., if two lines are parallel before the transformation, they remain parallel after the transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after the transformation).

of the linear vertex interpolation and Alexa's as-rigid-as-possible shape interpolation approach is shown in the Fig. 22.

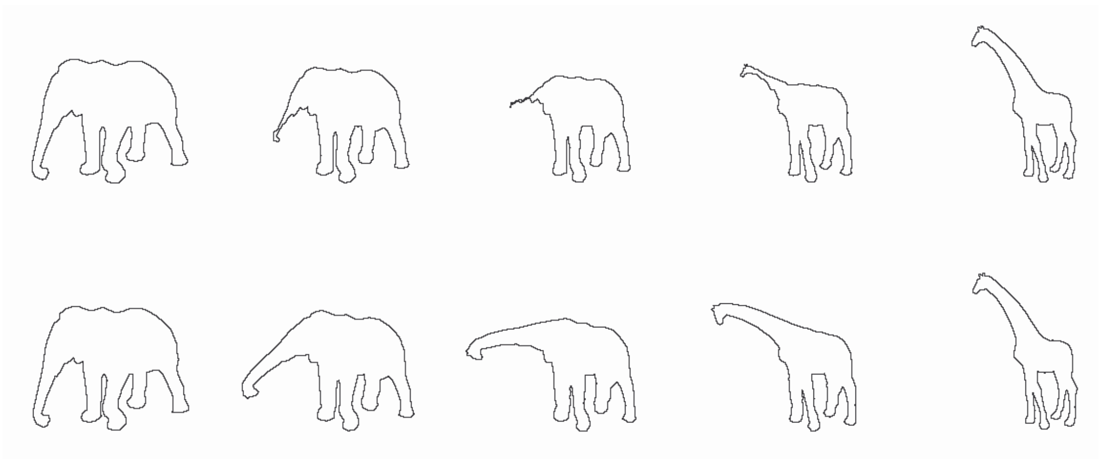


**Fig. 22: Morphs between an elephant and a giraffe. The top row sequence shows a simple linear vertex interpolation, the bottom row sequence shows Alexa's as-rigid-as-possible shape interpolation [Ale00b] (image taken from [Ale00b]).**

Extension of this approach into three dimensions is not easy. It is outlined in [Shu04] and a simple example is given in [Ale00b]. In 2D case this approach required an isomorphic dissection of polygons into triangles. In 3D it requires an isomorphic dissection of polyhedra into tetrahedra. An isomorphic dissection requires overlying of two tetrahedronizations and computing some sort of volume-based topology merging.

### 5.1.4 The approach by [Sun97]

The approach by Sun et al. [Sun97] tries to find an intrinsic representation for 3D meshes similar to edge-angle representation [Gom97]. Sut et al. formulates basic criteria for an intrinsic representation for polyhedral models. First let us introduce some notation. A morphing function $m(A, B, t)$, $t \in <0; 1>$ gives an intermediate mesh at time $t$ between the source mesh $A$ and the target mesh $B$, $T_x$ denotes a translation by a vector $x$ and $R_{N,\alpha}$ denotes a rotation about axis $N$ by an angle $\alpha$. Then the criteria are as follows:

- identity preservation, i.e. $m(A, A, t) = A$, $\forall\ t \in <0; 1>$, where $A$ is some input mesh,
- translation invariant, i.e. $m(T_x(A), T_y(B), t) = T_z(m(A, B, t))$, $\forall\ t \in <0; 1>$, where $z = (1 - t).x + t.y$ if the animation speed is uniform,
- rotation invariant, i.e. $m(R_{a,\alpha}(A), R_{b,\beta}(B), t) = R_{c,\gamma}(m(A, B, t)$, $\forall\ t \in <0; 1>$, for some axis $c$ and angle $\gamma$
- feature preservation, i.e. if there are feature common to both input objects, the features should be preserved during the metamorphosis.

According to the described criteria, first a *vertex adjacency graph* (a graph representing vertices of the mesh connected by edges, denoted as VAG) and its dual, a *face adjacency graph* (a graph representing adjacent faces of the mesh, denoted as FAG) for the source and the target mesh are constructed. Let us point out that this approach considers source and the target mesh with the same connectivity, thus FAG of the source mesh and FAG of the target mesh are isomorphic as well as VAG of the source mesh and VAG of the target mesh. A FAG contains face normals in its nodes and each edge of the graph contains a flag indicating whether two incident faces form a convex or a concave dihedral angle or whether they are coplanar. Note that face adjacency does not represent a mesh uniquely, so an additional geometric representation is needed to make the FAG a complete representation of a mesh. The interpolation then goes as follows. First the FAG is interpolated, which means an interpolation of face normals. It is started with two initial faces; the remaining normals are computed by propagation along edges of the FAG. The result of the FAG interpolation is establishing of an intermediate orientation of faces. Then the VAG is interpolated so that the vertices fit the already oriented faces. The comparison of the linear vertex interpolation and the interpolation of intrinsic parameters is given in the Fig. 23. Fig. 23 bottom shows linear vertex interpolation. Note that the intermediate human figures are unnaturally thin and the right arm is fairly distorted, which is not the case of the intrinsic parameter interpolation (Fig. 23 top).

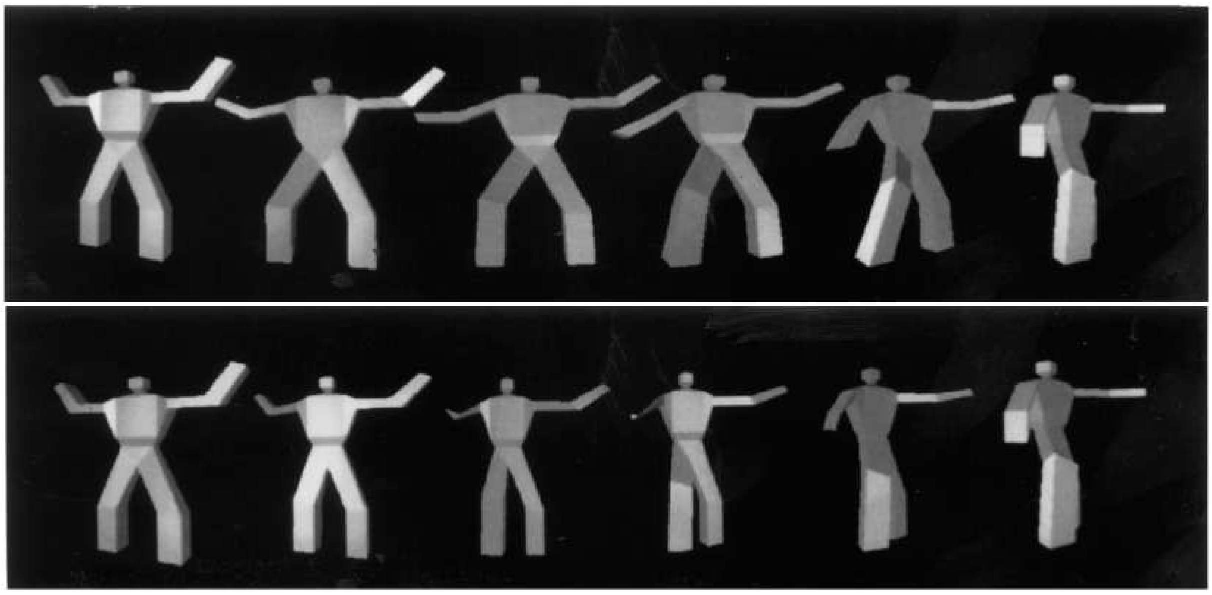


**Fig. 23: Comparison of the linear vertex interpolation (bottom) and the interpolation of intrinsic parameters using approach by [Sun97] (top) (image taken from [Sun97]).**

Let us highlight the difference between as-rigid-as-possible shape interpolation and the approach by [Sun97]. Alexa uses in his approach a *volume model*, i.e. he turns boundary representation into tetraheda. By interpolation of orientation of tetrahedra, he avoided collapsing or self-intersection. On the other side, Sun et. al [Sun97] interpolates the boundary representation directly without turning it into a volume model.

### 5.1.5 Laplacian representation

Alexa in [Ale01b] suggested an interpolation of Laplacian coordinates. It is another variant of an intrinsic parameters interpolation. Laplacian coordinates are computed as follows. First the center of mass of neighbors is computed, i.e.:

$$\bar{v}_i = \frac{1}{|N_i|} \sum_{j \in N_i} v_j \ , \tag{17}$$

where $\bar{v}_i$ is the center of mass of vertices adjacent to the vertex $v_i$, $N_i$ is a set of adjacent vertices to the vertex $v_i$. The Laplacian coordinates are then computed as a difference of the center of the mass and the original position, i.e.:

$$\tilde{v}_i = \bar{v}_i - v_i \,, \tag{18}$$

where $v_i$ is the original vertex position, $\bar{v}_i$ is the center of mass and $\tilde{v}_i$ are the Laplacian coordinates of the vertex $v_i$. So the Laplacian coordinates express the relative position of vertices with respect to the adjacent vertices. The interpolation goes as follows. First the forward transformation (i.e. from absolute vertex coordinates to relative Laplacian coordinates) is computed, then the Laplacian coordinates are lineary interpolated and the backward transformation is computed in order to obtain absolute vertex coordinates. The comparison of absolute vertex coordinates and Laplacian coordinates is shown in the Fig. 24. Note that not the whole mesh is morphed but only a part (nose). The displeasing result in c) is caused by different absolute positions of corresponding vertices. More pleasant result in d) is caused by the interpolation of relative vertex positions (i.e., Laplacian coordinates).

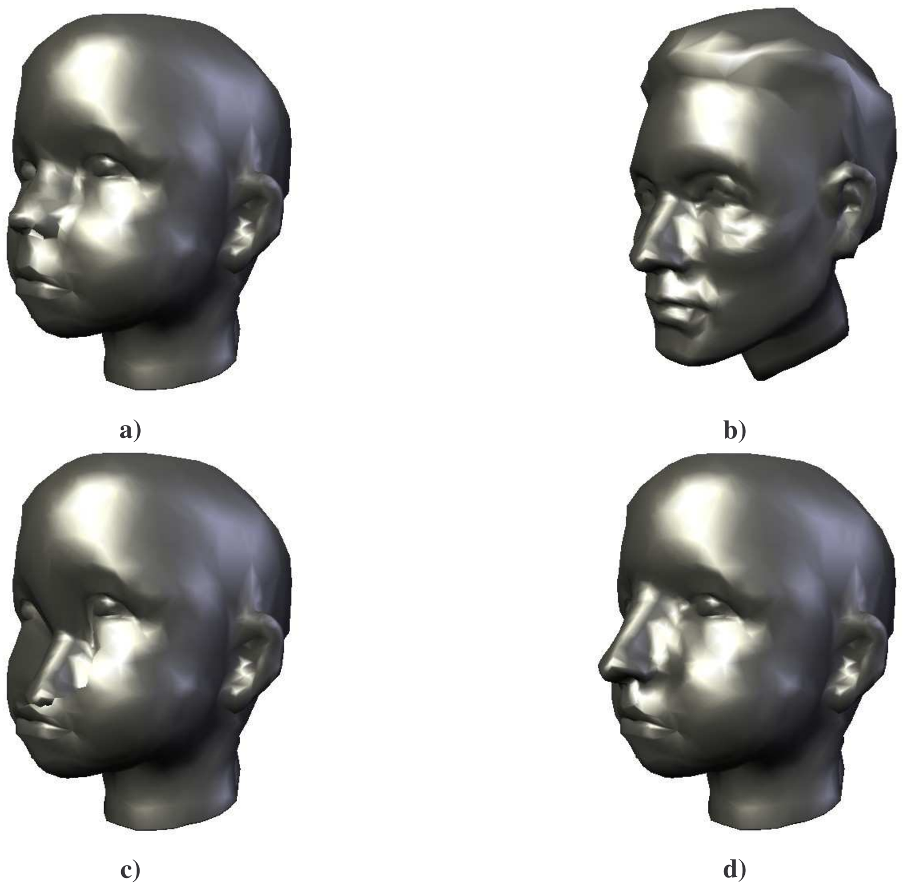


**Fig. 24: Comparison of morphing between the source mesh a) and the target mesh b). c) The result of absolute vertex interpolation. d) The result of Laplacian coordinates interpolation.**

In previous sections we described how the geometry interpolation is done. Basically there are two approaches, the interpolation of absolute vertex coordinates and the interpolation of intrinsic parameters. The latter approach usually provides better results then absolute vertex interpolation, on the other side, the absolute vertex interpolation is much easier to implement. The problem of intrinsic parameters interpolation is that we have to find some proper set of intrinsic parameters, i.e. such parameters which unambiguously represent the mesh. The problem is often the backward transformation, i.e. to extract absolute vertex coordinates from interpolated intrinsic parameters.

## 5.2 Morphing as a modeling technique

In [Kan99] the morphing technique was used also for modeling, i.e. for the creation of new objects by a combination of some existing objects. It is based on the concept of *local metamorposis*, which was also introduced in [Ale01b]. Local metamorphosis morphs only parts of meshes, not whole meshes. Local metamorphosis allows us to use classical cut and paste techniques, well known from 2D drawing and painting tools, which operate on meshes.

First, on both input meshes two boundary curves are specified by selecting the same number of corresponding vertices on each input mesh. The boundary curves determine tiles (regions surrounded by the boundary curve). The tiles are areas where the two input meshes will be fused. It is assumed that the tiles are topologically equivalent to the unit disc. Then, the tiles are parametrized by the harmonic mapping[35], overlaid and a supermesh is computed by a standard method proposed above. Then it remains to establish vertex positions of the supermesh, i.e., the vertex positions of the merged tiles. Let us remind that in classical morphing setting, we have two extreme vertex positions which are interpolated in order to obtain a morphing animation, the interpolation parameter is a time. In mesh fusion setting the time is replaced by a distance of a particular vertex of the supermesh to the fusion boundary, i.e. the boundary curve. It is expected that the vertices near the fusion boundary will lie more or less on the original surface and the vertices far from the boundary will lie on the surface of the pasted mesh. The dependency of the position of the vertex on the distance to the boundary loop is given by a *fusion control function*. An overview of mesh fusion process is depicted in the Fig. 25. Fig. 25a) shows the input meshes together with their boundary curves, the top mesh will be pasted in the bottom mesh, Fig. 25b) shows the parametrization of the tiles specified by the boundary curves and an overlay of parametized tiles. Fig. 25c) shows the resulting fusion according to the fusion control function depicted on Fig. 25d).

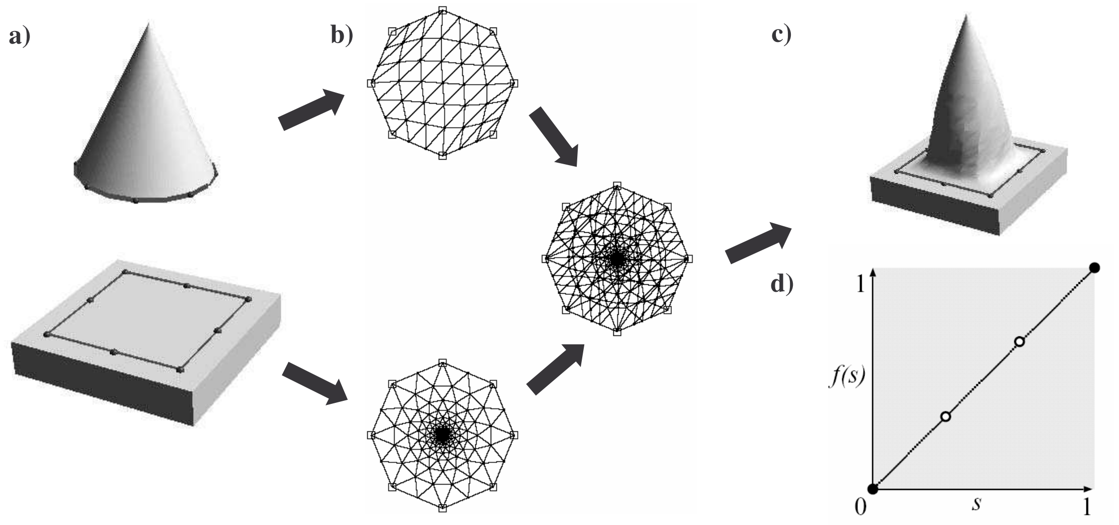


**Fig. 25: Overview of mesh fusion process (taken from [Kan99]).**

## 5.3 Interpolation of surface attributes

Meshes today are not only comprised by geometrical elements, such as vertices, edges and faces, but they have usually associated some surface attributes, e. g., colors, normals, opacity,

[35] See Section 3.2.3.

textures, etc. So along with the shape transformation it is also necessary to change surface attributes accordingly. We will first classify surface attributes into two groups; it allows us to generalize our ideas. Then we will describe the interpolation of normals. When morphing color surface, a problem of color interpolation appeared which will be discussed in Section 5.4. The interpolation of surface attributes is a part of our contribution and it was published in papers [Par04a, Par04b].

### 5.3.1 Attributes classification

We follow the classification by [Hop96], where two types of surface attributes are distinguished – discreet and scalar. Discreet attributes are usually associated with faces. A typical discreet attribute is, e.g., a material identifier, i.e., some property which is constant over the whole face. On the other side, the scalar attributes are associated with *corners*. The corner is a tuple (face, vertex) which allows us to assign some attribute to a particular vertex with respect to same face. For example a vertex normal is a typical scalar attribute, for one vertex we can have multiple normals depending on which face we are considering, e.g. a vertex of a cube has typically three different normals (one for each adjacent face). Other examples of scalar attributes are per vertex specified color, opacity, texture coordinates, etc.

### 5.3.2 Face mapping approach

As well as for the geometry interpolation, for attributes interpolation we have to specify extreme values of attributes, i.e. the values in time $t$=0 and time $t$=1. These values are established with respect to the source mesh and to the target mesh. For establishing of extreme attributes values we use a concept of *face mapping*. The basic idea is that the mapping of faces of the supermesh to the faces of the source mesh and the target mesh is established. From merging and triangulation process it is clear that each face of the supermesh maps to exactly one face of the target mesh and to exactly one face of the source mesh (i.e. no face of the supermesh can overlap faces of the source and the target mesh)[36]. Let us denote $\Phi_0 : f_i \to f_j^0$ for the mapping of the face $f_i$ of the supermesh to the face $f_j^0$ of the source mesh and analogously $\Phi_1 : f_i \to f_j^1$ for the mapping of the face $f_i$ of the supermesh to the face $f_j^1$ of the target mesh. This mapping is established in the parameter domain. Fig. 26 depicts a principle of the face mapping approach, on the right side there is a supermesh transformed to the shape of the source mesh (top) and to the shape of the target mesh (bottom), arrows represent the mapping of faces of the supermesh to the faces of the source and the target mesh.

Before we describe how to compute face map, we will first define mappings $\Omega_0$ and $\Omega_1$. The mapping $\Omega_0 : v_i \to f_j^1$ maps the vertices of the supermesh to the faces of the target mesh. It can be easily derived from the mapping $\Omega_{0\to1} : v_i^0 \to f_j^1$, which gives us the mapping of the vertices of the source mesh to the faces of the target mesh. We only have to know, which vertex $v_i$ of the supermesh is which vertex $v_i^0$ in the source mesh. It is clear that the mapping $\Omega_0$ has only sense for vertices of the source mesh. For vertices of the target mesh or split vertices[37] it has no sense and in the face mapping algorithm it is not needed. Similarly, the mapping $\Omega_1 : v_i \to f_j^0$ maps the vertices of the supermesh to the faces of the source mesh and

[36] So this mapping is "onto".
[37] See section 4.2.2.

it can be derived from the mapping $\Omega_{1\to 0}: v_i^1 \to f_j^0$. It has sense only for target vertices, not for source or split vertices.

Let us summarize already established mappings in the following table.

| **Notation** | **Meaning** |
|---|---|
| $\Omega_{0\to 1}: v_i^0 \to f_j^1$ | Mapping of the vertices of the source mesh to the faces of the target mesh established in the finding correspondence step (Section 3). |
| $\Omega_{1\to 0}: v_i^1 \to f_j^0$ | Mapping of the vertices of the target mesh to the faces of the source mesh |
| $\Phi_0: f_i \to f_j^0$ | Mapping of the faces of the supermesh to the faces of the source mesh |
| $\Phi_1: f_i \to f_j^1$ | Mapping of the faces of the supermesh to the faces of the target mesh |
| $\Omega_0: v_i \to f_j^1$ | Mapping of the vertices of the supermesh to the faces of the target mesh. |
| $\Omega_1: v_i \to f_j^0$ | Mapping of the vertices of the supermesh to the faces of the source mesh |

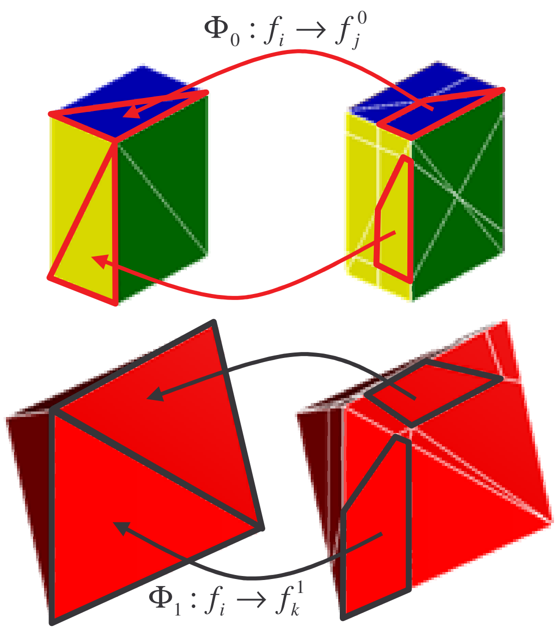


**Fig. 26: Mapping of faces of the supermesh to faces of the source (top) and the target (bottom) mesh.**

Now we will describe how to compute the face map. In general it is a location problem of a triangle is some coarse triangulation. Note that a brute force approach would require checking each face of the super mesh against each face of the source and the target mesh. The complexity of the brute force approach is then $O(|F|.|F^0| + |F|.|F^1|)$, where $|F|$ is the number of faces of the supermesh and $|F^0|$ and $|F^1|$ is the number of faces of the source mesh, target mesh, respectively. We further improved the algorithm described in [Par04a]. It reuses an already established mapping of vertices of one mesh to the surface of the other mesh. The supermesh is traversed vertex by vertex and *triangle fans* of each vertex are processed. A triangle fan is a set of triangles incident to a particular vertex. Each triangle fan is processed depending on which kind of vertex belongs to. Let us remind that there are three types of vertices in the supermesh – source vertices, target vertices and split vertices. Also note that each triangle belongs to more than one triangle fan, so once the triangle is processed, a flag *done* is set, which avoids multiple processing in the context of other triangle fans.

Let first consider the case of the source vertex $v_s$. From the establishing correspondence step we know the mapping of the source vertex $v_s$ to the target face $f_t$. So each face of the supermesh adjacent to the vertex $v_s$ maps to the face $f_t$, i.e.:

$$\Phi_1(f_i) = f_t\,, \tag{19}$$

where $f_i$ are faces adjacent to the vertex $v_s$, $f_t$ is the face of the target mesh to which vertex $v_s$ maps. Now the mapping $\Phi_1$ is solved. It remains the mapping $\Phi_0$, i.e., the mapping of the faces to the source mesh. It is easy, because faces of the supermesh adjacent to the source vertex map to the faces of the source mesh adjacent to the same vertex. So the faces adjacent to the vertex $v_s$ in the source mesh are candidates for mapping of faces of the supermesh adjacent to the vertex $v_s$ in the supermesh.

For the case of target vertices the mapping is computed analogously. For the target vertex $v_t$ the mapping to the face $f_s$ of the source mesh is known. So each face of the supermesh adjacent to the vertex $v_t$ maps to the face $f_s$, i.e.:

$$\Phi_0(f_j) = f_s\,, \tag{20}$$

where $f_j$ are adjacent faces to the vertex $v_t$, $f_s$ is the face of the source mesh to which vertex $v_t$ maps. The establishing of mapping $\Phi_1$ is again easy, because faces of the supermesh adjacent to the target vertex map to the faces of the target mesh adjacent to the same vertex.

For split vertex we know from which two edges of the source and the target mesh it arises. For each edge we also know the indices of adjacent faces. Adjacent faces are candidates for mapping of faces of the supermesh. So it remains to check to which of candidates a particular face of the supermesh maps.

The algorithm written in pseudocode is as follows:

```
Input: supermesh M, source mesh $M^0$, target mesh $M^0$, $\Omega_0$, $\Omega_1$
Output: mappings $\Phi_0$, $\Phi_1$

// $\Omega_0$ ... mapping of the vertices of M to the faces of $M^1$
// $\Omega_1$ ... mapping of the vertices of M to the faces of $M^0$
// $\Phi_0$ ... mapping of the faces of M to the faces of $M^0$
// $\Phi_1$ ... mapping of the faces of M to the faces of $M^1$

1: for each vertex $v_i$ of M do
2:   $N_i \leftarrow$ faces of M incident to the vertex $v_i$
3:   if $v_i$ is a source vertex then
4:        $v_s \leftarrow v_i$
5:        $N_S \leftarrow$ faces of $M^0$ incident to the vertex $v_s$
6:        for each face $f_i$ in $N_S$ do
7:             $\Phi_1(f_i) \leftarrow \Omega_0(v_S)$
8:             brute force search of $f_i$ in $N_S$ to obtain $\Phi_0(f_i)$
9:   if $v_i$ is a target vertex then
10:       $v_t \leftarrow v_i$
11:       $N_T \leftarrow$ faces of $M^1$ incident to the vertex $v_t$
12:       for each face $f_i$ in $N_T$ do
13:            $\Phi_0(f_i) \leftarrow \Omega_1(v_t)$
14:            brute force search of $f_i$ in $N_T$ to obtain $\Phi_1(f_i)$
15:  if $v_i$ is a split vertex then
16:       $e_a$, $e_b \leftarrow$ edges of $M^0$, $M^1$, which form the split vertex
17:       $f_{1S}$, $f_{2S} \leftarrow$ faces of $M^0$ adjacent to the edge $e_a$
18:       $f_{1T}$, $f_{2T} \leftarrow$ faces of $M^1$ adjacent to the edge $e_b$
19:       for each face $f_i$ in $N_i$ do
20:            if $f_i$ maps to $f_{1S}$ then $\Phi_0(f_i) \leftarrow f_{1S}$ else $\Phi_0(f_i) \leftarrow f_{2S}$
21:            if $f_i$ maps to $f_{1T}$ then $\Phi_1(f_i) \leftarrow f_{1T}$ else $\Phi_1(f_i) \leftarrow f_{2T}$
```

The algorithm is demonstrated in the Fig. 27. Fig. 27a) shows how the face mapping is established for the faces incident to the source vertex. Denote $v_s$ the vertex of the source mesh, $f_1^0, f_2^0, ..., f_6^0$ the faces of the source mesh incident to the vertex $v_s$, $f_j^1$ the face of the target mesh to which vertex $v_s$ maps. During the topology merging process, new faces around the vertex $v_s$ are created (i.e., the faces $f_1, f_2, ..., f_6$). As it can be seen in the Fig. 27a), the faces $f_1, f_2, ..., f_6$ of the supermesh map to the face $f_j^1$ of the target mesh, this corresponds to the line 7 in the pseudocode. When looking for mapping of the faces of the supermesh to the faces of the source mesh, the candidates are the faces $f_1^0, f_2^0, ..., f_6^0$. The search between candidates is done by brute force search (line 8 in the pseudocode). Fig. 27b) shows how the face mapping is established for the faces incident to the target vertex. In the Fig. 27b) $v_t$ is the vertex of the target mesh, $f_1^1, f_2^1, ..., f_6^1$ are the faces of the target mesh incident to the vertex $v_t$, $f_j^0$ is the face of the source mesh to which vertex $v_t$ maps. As well as in previous case the faces $f_1, f_2, ..., f_6$ of the supermesh map to the face $f_j^0$ of the source mesh (line 13) and to the faces $f_1^1, f_2^1, ..., f_6^1$ of the target mesh (line 14). Fig. 27c) shows the computation of the face mapping for the faces incident to the split vertex. In the Fig. 27c) there is $v_i$ the split vertex, $e_a$ and $e_b$ are the edges of the source mesh, target mesh, respectively, which create the split vertex $v_i$ (line 16), $f_{1S}, f_{2S}$ are the faces of the source mesh incident to the edge $e_a$ (line 17), $f_{1S}$, $f_{2S}$ are the faces of the target mesh incident to the edge $e_b$ (line 18). It can be seen that the faces $f_1$, $f_2$, ..., $f_4$ of the supermesh map either to $f_{1S}$ or $f_{2S}$ of the source mesh (line 20) and either to $f_{1T}$ or $f_{2T}$ of the target mesh (line 21).

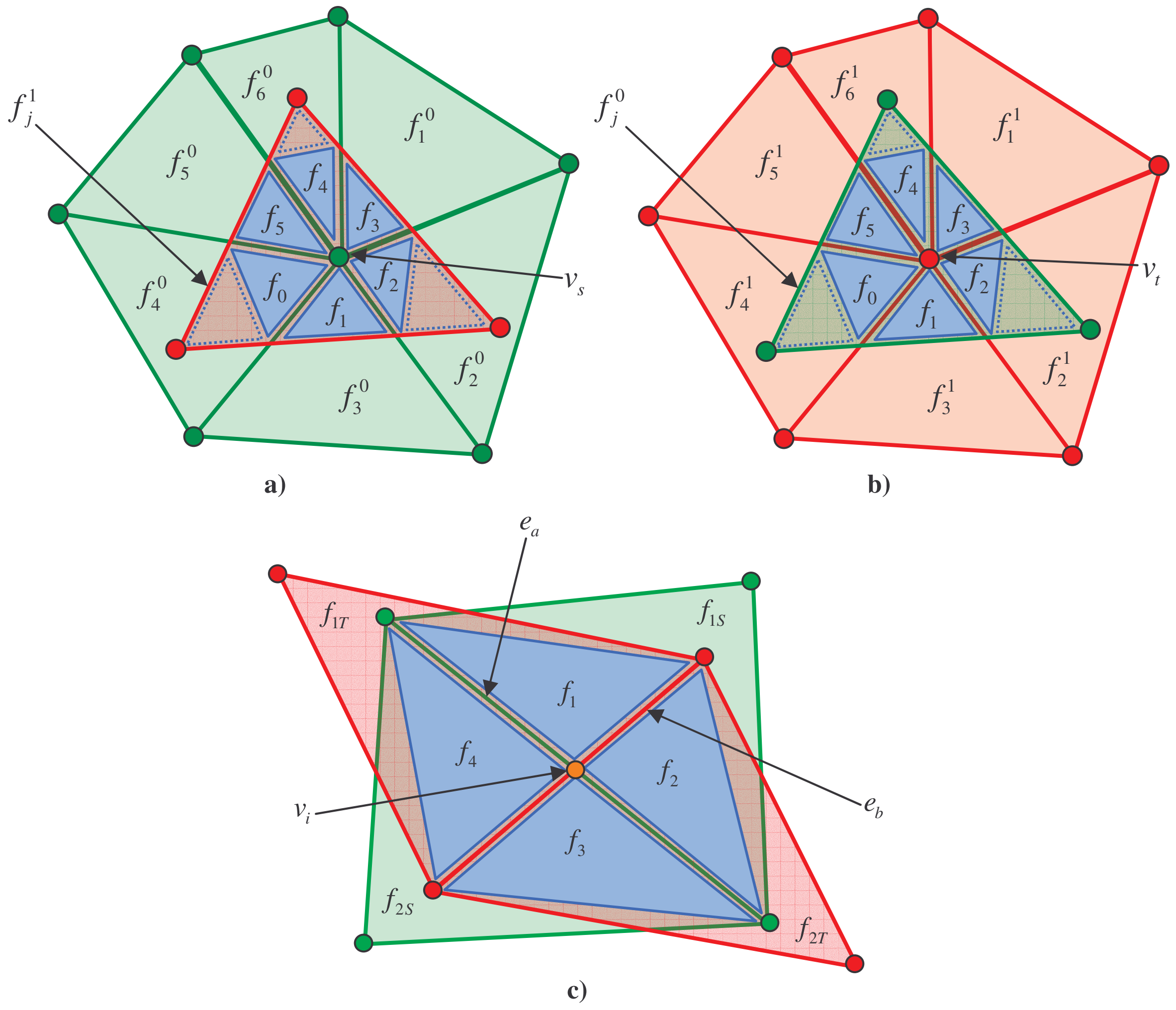


**Fig. 27: A demonstration of the face mapping algorithm. The green triangles are the faces of the source mesh, the red triangles are the faces of the target mesh, and the blue triangles are the faces of the supermesh, i.e., the faces which are created during the topology merging process.**

Complexity of this algorithm is $O(N)$, where $N$ is the number of supermesh vertices[32]. It is suitable to add a flag to each face which indicates whether the given face was already processed to avoid multiple processing in the context of more triangle fans.

### 5.3.3 Handling attributes

Having established mapping of faces of the supermesh to the faces of the source mesh and to the faces of the target mesh (i.e. face map), it is necessary to establish the extreme value of attributes.

For discreet attributes it is simple, because the face $f_i$ of the supermesh gets the value of the discreet attribute of the face to which it maps, i.e. for the time $t$=0 the face of the supermesh $f_i$ gets the value of the discreet attribute of the face of the source mesh $f_j^0$ to which the face $f_i$ maps; and for the time $t$=1 the face of the supermesh $f_i$ gets the values of the discreet attribute of the face of the target mesh $f_k^1$ to which the face $f_i$ maps. Let us denote $D(f_i)(t)$ a value of a discreet attribute of the face $f_i$ at the time $t$, then for $t$=0 we can write:

$$D(f_i)(0) = D(\Phi_0(f_i))\,, \tag{21}$$

where $\Phi_0(f_i)$ is the mapping of the face $f_i$ of the supermesh to the face of the source mesh. For $t$=1 we can analogously write:

$$D(f_i)(1) = D(\Phi_1(f_i))\,, \tag{22}$$

where $\Phi_1(f_i)$ is the mapping of the face $f_i$ of the supermesh to the face of the target mesh. A linear interpolation of a discreet attribute is then:

$$D(f_i)(t) = (1-t)D(\Phi_0(f_i)) + tD(\Phi_1(f_i))\,. \tag{23}$$

The values of scalar attributes are computed with respect to the relative position of the vertex inside the face to which the vertex maps. Let us denote $S(f_i, v_j)(t)$ as a value of a scalar attribute of the corner $(f_i, v_j)$ at time $t$. So for $t$=0 we can say that the value of scalar attribute $S(f_i, v_j)(0)$ is given by some linear combination of the values of the attributes in corners of the face $f_j^0$, where $f_j^0$ is the face of the source mesh to which the face $f_i$ of the supermesh maps. The coefficients of the linear combination are barycentric coordinates of the vertex $v_j$ with respect to the face $f_j^0$. This situation is depicted in the Fig. 28, where the blue triangle $f_i$ is the face of the supermesh and the green triangle $f_j^0$ is the face of the source mesh to which the face $f_i$ maps. The values of the scalar attribute $S(f_i, v_k)$ are given by the relative position of the vertex $v_k$ with respect to vertices of the triangle $f_j^0$.

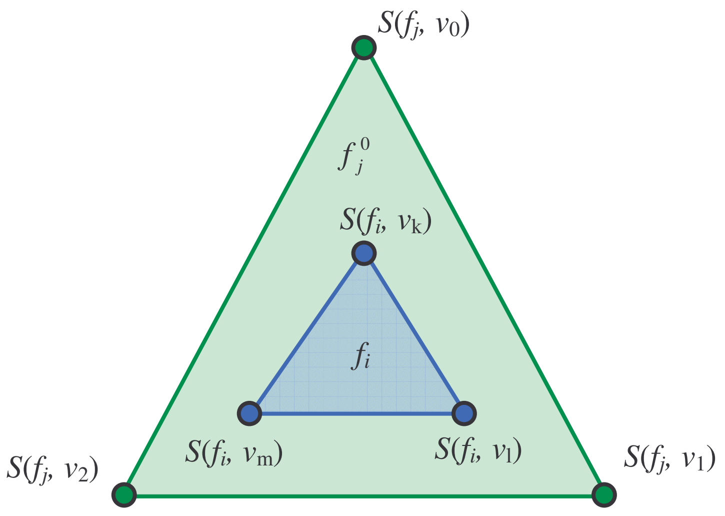


**Fig. 28: Mapping of the face $f_i$ of the supermesh (blue) to the face $f_j^0$ of the source mesh (green).**

It is similar for the time $t$=1, with the difference that the mapping of the faces of the supermesh to the faces of the target mesh is considered. During the morphing transition the values of scalar attributes are linearly interpolated.

## 5.4 Color interpolation

A per vertex specified color is classical scalar attribute. By the process described above it is possible to specify a vertex color for the time $t$=0 and for the time $t$=1. But the interpolation of color itself is not as straightforward as, e.g., a position interpolation. Note that color is in computers represented in a color system, so the color interpolation is dependent on particular color system. We will discuss here the color interpolation in the RGB and HLS/HSV color systems.

The RGB color system is a common color system in computer displays. In RGB (Fig. 29a)) color system, the color is represented by amounts of three basic colors, i.e. red, green and blue. In this color system most of common graphical interfaces work, thus it is easy to use and no additional color conversion is required.

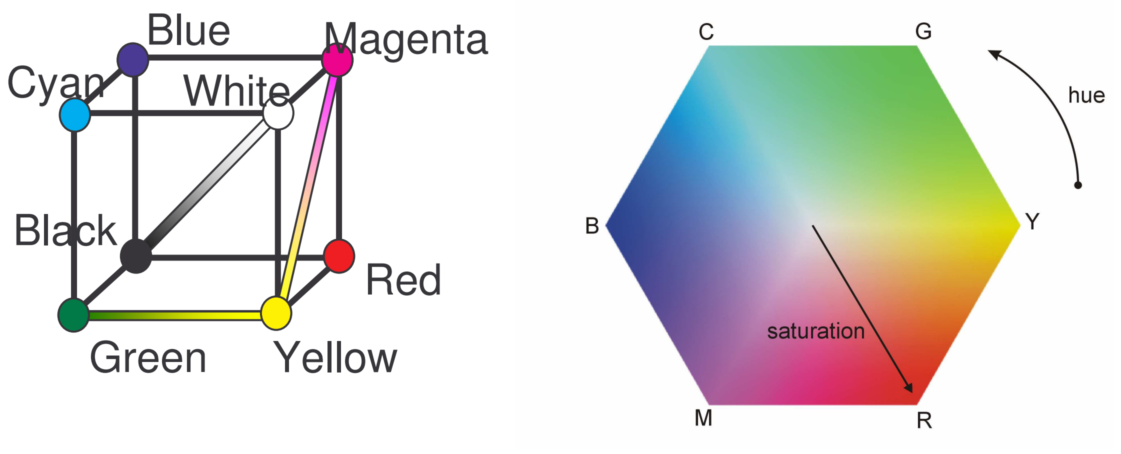


**Fig. 29: a)The RGB color cube b) HLS/HSV color system.**

In the RGB color system there are three basic types of color transition. First type is an interpolation of color along some edge of the color cube. This transition is probably the most pleasant, because no other color appears during the transition. The second type of color transition is an interpolation along some face diagonal. In this transition also some other color starts to appear, e.g. in the case of transition between cyan and yellow also green appears. This transition is less pleasant. The last basic type is interpolation along some cube diagonal. As the cube diagonal between the points (0, 0, 0) and (1, 1, 1) are grayscales colors, each other cube diagonal hits more or less this grayscale, so in this transition a gray color appears.

HLS and HSV are other common systems (Fig. 29b)). They represent a color in terms of hue, saturation and value or lightness. Representing a color in these systems is more natural for a human[38], but less convenient for interpolation, because the color transition in these systems looks sometimes strange. This is mainly because of interpolation of hue, which means changing of basic color tint during the interpolation. For example, when interpolating between red and green, yellow is crossed, or when interpolating between red and blue, even yellow and green is crossed. Note that it is possible to change hue in clockwise manner or counterclockwise manner. Usually the longer way means more colors to appear during the transition. Appearance of a new color tint sometimes produces disturbing effects. Also the

[38] Because it is the way we usually define the color, i.e. by its basic color tint (hue), saturation, lightness or value; not by weighted sum of three color primaries.

conversion between RGB and HLS/HSV, is a nonlinear transformation expressed as an algorithm, which carries an additional computational cost.

Example of morphing of colored meshes is in Appendix B. Fig. 40 shows an example of morphing between a cube and an octahedron. In this case the color is specified per face, so it is constant for the whole face. Fig. 41 presents morphing between two faces. In this case both input meshes are colored per vertex. Fig. 42 shows morphing between two meshes, where the first mesh is colored per face, the second mesh is colored per vertex. In the Fig. 43-38 it is possible to compare various strategies for color interpolation in HLS color system. In the Fig. 43 the hue component of HLS is interpolated in the clockwise direction. When comparing with the RGB color interpolation (Fig. 42) it can be seen that results are almost the same. On the other side, interpolation of hue in the counter-clockwise direction (Fig. 44) produces very unpleasant result. This can be observed also in the Fig. 29b), where the HLS color diagram is depicted. When interpolating between yellow (a petal of the flower) and pink (body of pig) the green, cyan, blue, magenta is crossed. This makes the animation quite wild.

## 5.5 Normal interpolation

In this section we will discuss the problem of normal interpolation. Normals are important for rendering, i.e. for evaluating of the lighting model. We usually distinguish between face normals, i.e. vectors perpendicular to a particular face, and vertex normals, i.e. vectors perpendicular to a particular vertex. Face normals are used for flat shading, vertex normals are used for some sort of smooth shading, e.g., Gouraud or Phong. Let us remind that vertex normals are usually computed on the basis of face normals, e.g., by averaging of normals of faces which are incident to a given vertex. According to the attribute classification described in Section 5.3.1 the face normal is a discreet attribute and the vertex normal is a scalar attribute. The problem is that normals cannot be handled as suggested in Section 5.3.3, because the face normal is a function of the position of vertices which form the face. The vertices of the faces are interpolated linearly, but each vertex of the face has different linear trajectory, so the normal cannot be linearly interpolated as for example color. The same holds for vertex normals, because they are computed on the basis of face normals.

Another problem is the existence of *sharp edges*. The sharp edge is an edge whose adjacent faces form an angle bigger than some predefined constant – called *crease angle*[39]. A cube is a typical object which has sharp edges. On the other side, a sphere has no sharp edges. When morphing between such two objects, the problem is a smooth transition between the region with a sharp edge and a region where the surface is smooth surface and vice-versa. Fig. 30a) shows an example of the transition where sharp edges are not considered at all, i.e., each vertex has associated only one normal. Note how the Gouraud shading smoothes the sharp edges of the cube.

[39] The term crease angle is used, e.g., in VRML or 3ds max.

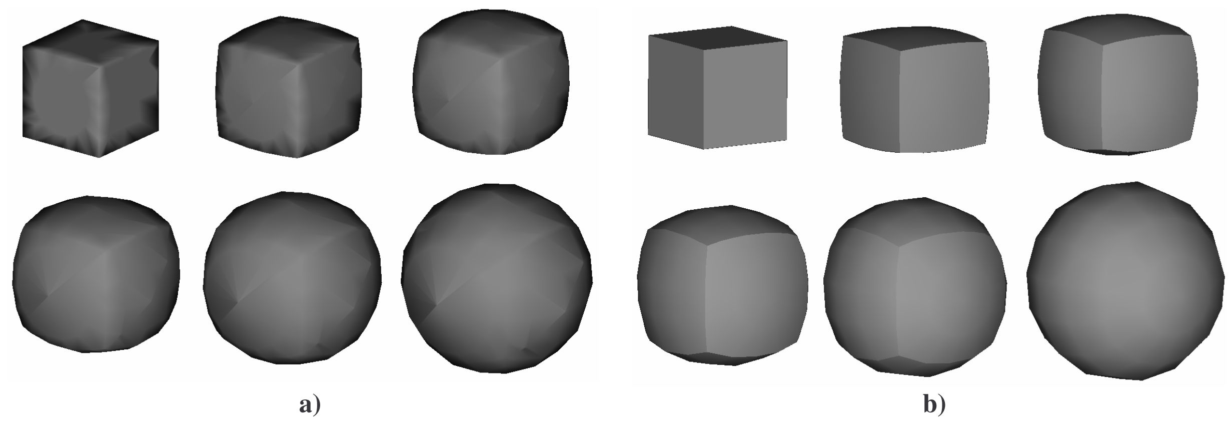


**Fig. 30: a) An example of transition where the sharp edges are not considered. b) An example of transition using face mapping approach.**

Regardless to the fact that the linear interpolation of normals is not correct, we tried to handle vertex normals as ordinary scalar attributes, i.e., the vertex normal is associated with the corner and during the animation the normal is interpolated linearly between the vertex normal in the time *t*=0 and the vertex normal in the time *t*=1. The result is depicted in the Fig. 30b). It can be seen clearly that the result is better than when the sharp edges are not considered at all. Another example of linear normal interpolation is in Appendix B, Fig. 46.

One solution to the normal interpolation would be to recompute normals in each frame of the morphing animation. The disadvantage of this approach is that it is computationally expensive and it also results in popping of sharp edges. Note that the computation of normals involves detection of crease curves and thus involves determining of angles of adjacent faces. The problem is that in the certain instant of time the angle of two adjacent faces exceeds the crease angle value, which results in abrupt appearance of the sharp edge. We rather expect the sharp edge to appear smoothly. An example of popping of sharp edges is in the Fig. 31. In the first line there are three frames of the morphing transition between a cube and a bended cylinder and in the second line there is a detail of the part where the sharp edge pops into a smooth edge.

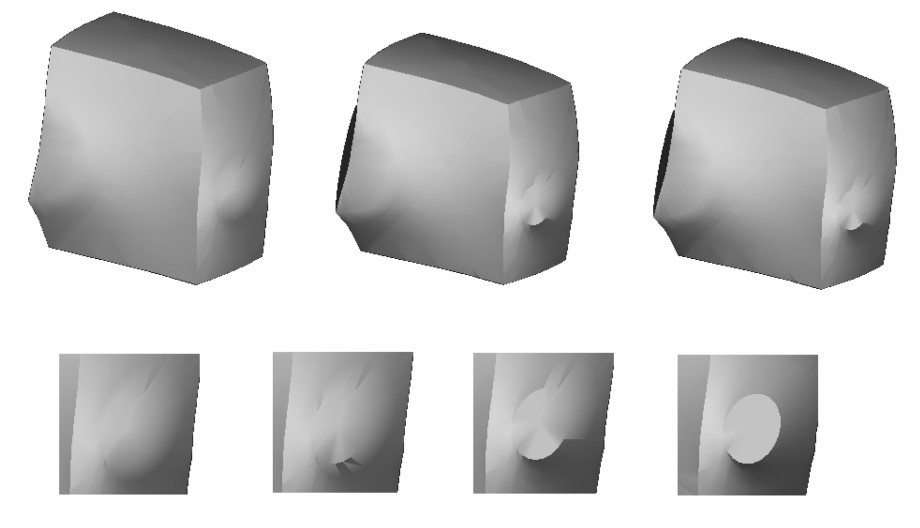


**Fig. 31:An example of popping of sharp edges, when recomputing the normals in each frame of the morphing transition. The top row sequence shows three frames of the morphing transition and the bottom row sequence is the detail of the part where the popping occurs.**

## 5.6 Kinetic control

Another interesting issue or degree of freedom is the *kinetic control*. The kinetic control was introduced, e.g., in [Ste85]. It allows us to change the timing of the animation, i.e. not the values of animated attributes themselves, but only a speed of the animation. By the kinetic control we can introduce effects such as a slowdown, a speedup, etc. The kinetic control usually appears in the animation systems in the following form. Two interpolation curves are given. One specifies the interpolation of a value of an attribute, e.g., a trajectory of a vertex, an interpolation curve in a color system, etc. The other curve specifies timing of the animation[40]. An example of both interpolation curves and their composition is shown in the Fig. 32.

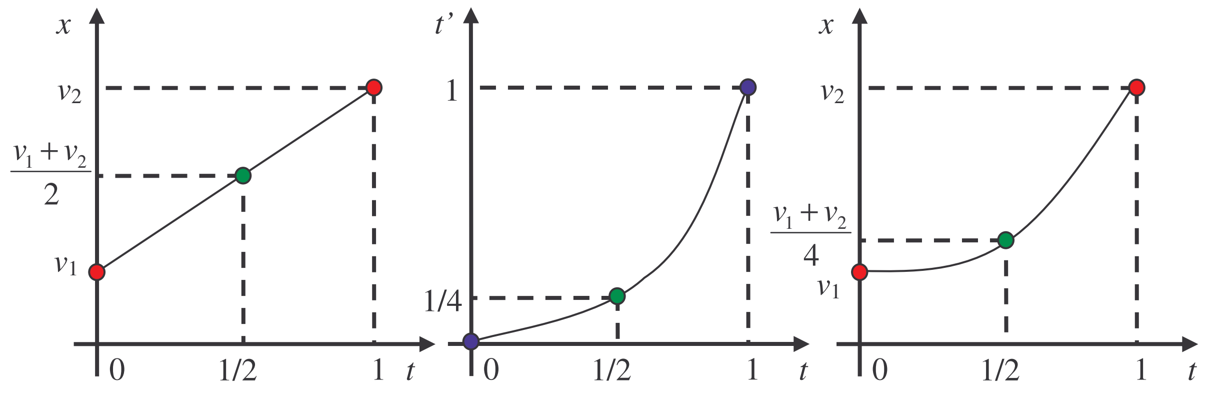


**Fig. 32: The kinetic control – a) a linear interpolation of some value, b) a kinetic interpolant, c) composition of both functions resulting in a slow change in the first part of animation and a fast change in the second part of the animation.**

The kinetic control for vertex trajectories is quite a powerful feature. By an appropriate slow-down and speed-up of some vertices (or even parts of meshes) self-intersection problems can be avoided. Unfortunately, this problem is not easy to solve, it was also included into the list of open problems [TOP, Dem04].

Another nice looking use of the kinetic control is a *constant speed* interpolation, i.e. taking into account the length of the trajectory, so that vertices with shorter trajectories are finished earlier than vertices with longer trajectory. An example is in the Fig. 48. Note how the legs of the pig slowly appear during the whole animation, it is because the vertices on the legs have the long trajectories. On the other side, vertices on the petal have a short trajectories, thus they are finished much earlier.

The kinetic control is not restricted only on vertex trajectories, we can apply this concept on the interpolation of whichever attribute. Especially in the case of color interpolation, by assigning different kinetic interpolants to different color channels, we can obtain interesting results. An example is in the Fig. 47 where the kinetic interpolant of geometry is set so that the shape interpolation is slow at the start and fast at the end and the kinetic interpolant for color interpolation is set so that the color change is fast at the start and slow at the end. It results in an interesting effect, where coloring of the target mesh appears very quickly on the source mesh.

[40] This curve is also denoted an *ease curve* for example in 3ds max.

# 6 Future work

In this section we will outline ideas for our future work. Some of ideas are supplemented with very preliminary tests or results, some are in the consideration stage and preliminary test would be needed. Before we describe ideas for the future work, let us briefly summarize what is already done and which is our point of departure.

For the test purposes we implemented a simple application, by which we produced results shown in Appendix B. For the parametrization we choose the relaxation method by Alexa described in the Section 3.2.1. We are also able to align a few corresponding features by parametrization warping (Section 3.3.1). The supermesh construction step consists of the intersection computation and the local triangulation. We compute the intersections by the walking algorithm described in the Section 4.2.1. For robust intersection computations we use Shewchuk's library [SHE]. The local triangulation is computed by the Kanai's method described in the Section 4.2.3. The implementation details of these two steps are described in [Par03]. For the interpolation of the geometry we use the linear vertex paths with the kinetic control (Section 5.6). For the interpolation of surface attributes we use our own method based on the face mapping (Section 5.3.2).

As for the future work, first of all, we would like to solve the problem of normal interpolation of the supermesh correctly. Then we would like to include also a morphing of textured surfaces, which seems only a small implementation improvement, but it has big practical use. Also the color interpolation is still a problem, which we would like to consider. These improvements touch the interpolation part of the metamorphosis process and consider two isomorphic meshes, i.e. meshes which have the same topology. Going back to the finding correspondence process, we would like to include some reasonable user input, by which it would be possible to control the morphing process. In the FRVŠ project "Use of strip representation for mesh morphing", we are cooperating with Ing. Petr Vaněček on stripification of supermeshes, so it is also a topic which will be explored. An interesting issue is also a morphing among more than two meshes. In the following subsections we will concretize the outlined ideas.

## 6.1 Normal interpolation

Normals are essential for shading of meshes. In case of shading animated sequence of a morphing object, the simplest solution is to recompute the normals in each animation frame. We do not want to use this approach because it is computationally expensive and during the animation it produces popping of sharp edges as shown in Section 5.5 and in [Par04a]. Instead of this we will establish a dependency of normal on individual vertex trajectories. The result is a time dependent formula. Evaluating the formula for time $t$ we obtain the normal of the face in the time $t$.

In [Par04a] we interpolate normals linearly, i.e., we linearly interpolated between normals in the time $t$=0 and time $t$=1. It produces quite good results in cases where the shape change is not very dramatic. In the case that the individual triangles dramatically change their orientation, a linear interpolation is not sufficient. Fig. 33a) shows the linear interpolation of vertex normals. The linearly interpolated normals used for shading cause unnatural results as depicted on the detail view in the Fig. 33b). For comparison, Fig. 33c) shows the same region which is shaded using normals recomputed in each frame of the animation.

For establishing a time dependent formula which gives us the correct face normal for the time *t*, imagine the example in the Fig. 34. Each vertex travels along its linear trajectory from the source position (green) to the target position (red).

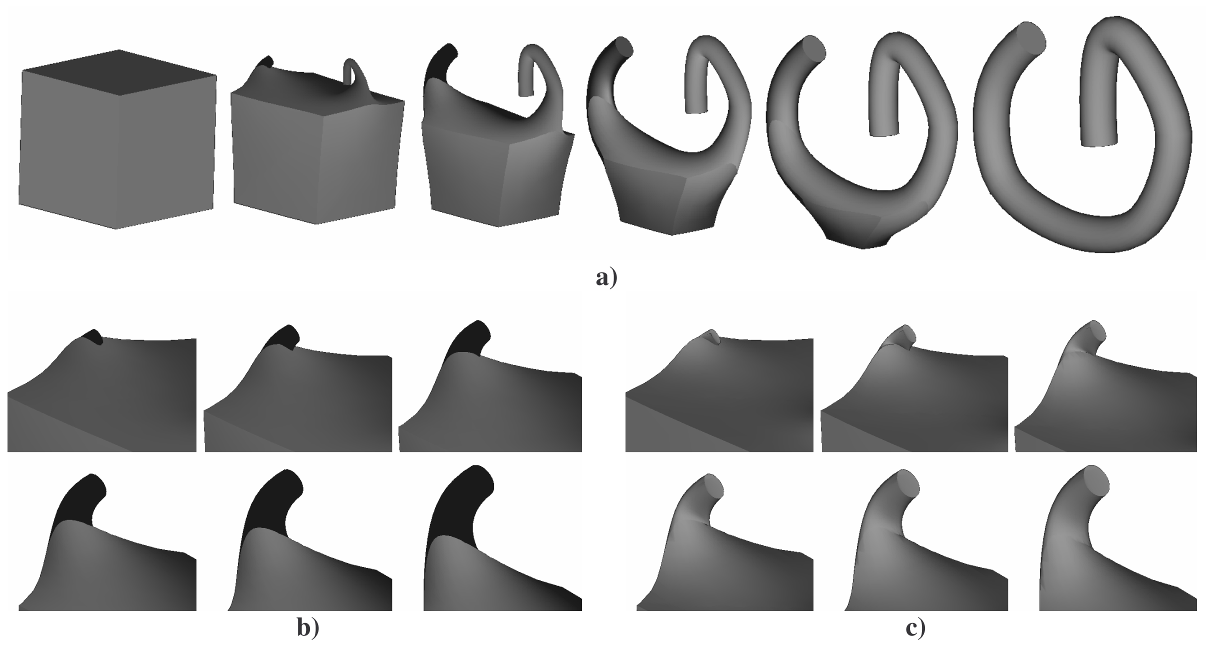


**Fig. 33: a) An example of the linear interpolation of the vertex normals, when the shape change is quite dramatic. b) The detail of the region where the linear interpolation of normals cause problems. c) The detail of the same part as in b) but with normals recomputed in each frame of the animation.**

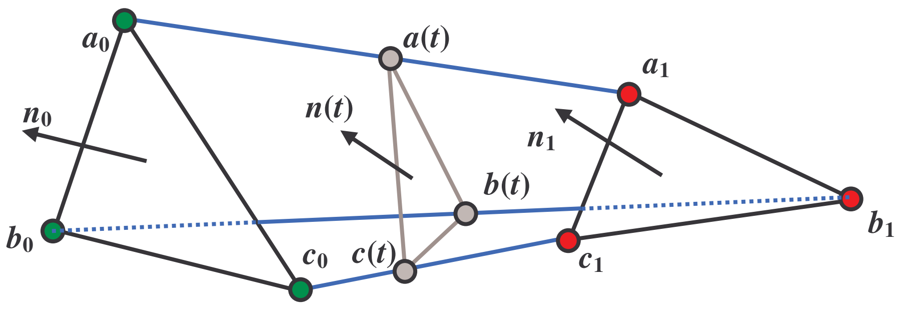


**Fig. 34: Normal interpolation**

As well as the face normal in the time $t$=0 is given by the cross product of vectors ($b_0$-$a_0$) and ($c_0$-$a_0$), the face normal in the time *t* is:

$$n(t) = (b(t) - a(t)) \times (c(t) - a(t)), \tag{24}$$

where $a(t)=a_0+t.(a_1-a_0)$, $b(t)=b_0+t.(b_1-b_0)$ and $c(t)=c_0+t.(c_1-c_0)$. The Eq. (24) results in a quadratic equation, which gives us the correct normal for the time *t*. By this process we can evaluate face normals, so it remains to compute vertex normals. Vertex normals are computed on the basis of normals of adjacent faces. It can be an average of normals of adjacent faces, an average weighted by the area of the face, an average weighted by the inverse area of the face, etc. We expect that we can obtain similar formula for the vertex normal as for the face normal, i.e., a time dependent formula which would give us the vertex normal in a particular time. Following problems need to be solved:

- the existence of sharp edges, i.e. the way how the vertex normal is computed will vary over time,
- the kinetic control, i.e. the linear vertex move will be expressed as $v(t)=v_0+f(t)(v_1-v_0)$, where $f(t)$ is a kinetic interpolant,
- complicated vertex paths, i.e. the cases where the vertex does not move along a straight line.

## 6.2 Morphing textured surfaces

Textures are applied on meshes using texture coordinates, which assign to a particular vertex a corresponding texel from the texture. A texture coordinate is a classical scalar attribute[41], so it seems that it can be interpolated as well as, e.g., color. It is also suggested by Alexa in [Ale99]. By practical experiments we found out that it is not suitable to interpolate texture coordinates, the problem is that any interpolation of texture coordinates yields moving of the texture which is a very unpleasant effect. It can be seen in the Fig. 35. There is the detail view of morphing transition between a textured cube and a textured sphere. Of course, texture of the cube and the sphere are cross-dissolved during the transition, which is not visible on the figure, because is shows only the first 12 frames of the animation. But is can be seen how the brick texture, i.e., the texture of the source mesh, moves.

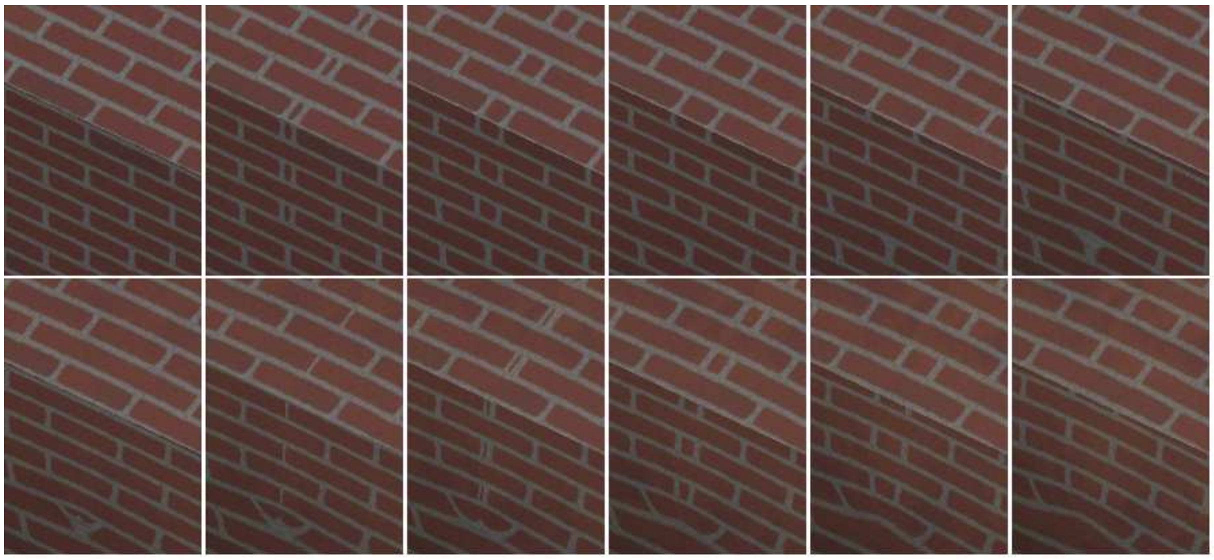


**Fig. 35: The detail view of "movig texture" effect caused by the interpolation of the texture coordinates. The consecutive images are ordered from the left to the right and from the top to the bottom.**

We rather expect a smooth cross-dissolving of the texture of the source mesh and the target. So the solution will be probably that the texture coordinates remain fixed, and two textures are applied on the supermesh. Of course, textures have to be blended according to the time.

[41] See Section 5.3.1.

This can be implemented by some multitexturing technique with a hardware support because a software texture blending will be probably slow.

## 6.3 Color interpolation

In Section 5.5 we dealt with a color interpolation. The color interpolation is strongly dependent on the color system used. We used only the RGB and HLS/HSV color systems. The problem of these color systems is that they are not perceptually linear, i.e., a change in the color system does not proportionally correspond to the perceived color change. So maybe some perceptually linear color space would yield better results. Another interesting issue is the interpolation curve in a particular color system. In general, the problem is as follows. The only way how to interpolate two colors is a simple linear interpolation, because only two points in the color system are given. By adding some additional points (e.g., some grey tone) more sophisticate curves would be achieved. An example is a desaturation strategy, which is demonstrated in the Fig. 45 in Appendix B. This approach is motivated by the fact that the hue interpolation sometimes produces a very unpleasant result (e.g. when interpolating from yellow to blue, cyan and green is crossed in the counter-clockwise direction or red and magenta is crossed in the clockwise direction). The idea is that the saturation and value of the color is decreased until the central axis of the color space is reached. Then the hue is flipped and the value and saturation is interpolated towards the destination color. In the Fig. 45 it can bee seen how the white color appears, i.e. the point on the central axis where the hue is flipped.

A general problem of color interpolation is that some objective criteria, i.e. some measure how much the given color transition is good or visually plausible, is missing. This leads us to the area of psychophysical experiments and study of human perception. This is the area where we do not want to go, because we do not feel competent enough. On the other side, we would like to provide a set of possible color transitions and give so to an eventual user a possibility to use a different color transition scheme. Together with a kinetic control this issue offers a lot of possibilities.

It is also important to mention that when talking about color interpolation in the context of morphing, we actually interpolate the color of vertices (considering per vertex specified color). Color is in common graphics libraries (OpenGL, DirectX) represented in the RGB color system. So for the color interpolation we usually convert a color from RGB to the required color system, interpolate and convert back to the RGB color system. The rasterization of triangles, guided by hardware, is done in the RGB color space. Thus, although we interpolate color of vertices in various color systems, the triangles itself are interpolated in RGB. The solution is to modify vertex and fragment shader, in order to have the whole interpolation process in one color system.

## 6.4 Self-intersection

An often discussed problem is the self-intersection of the mesh during the morphing animation. An example of the self-intersection is shown in Section 3. The self-intersection occurs usually when morphing highly dissimilar shapes. It can be partially avoided by an appropriate correspondence (or parametrization), but in many cases it will not help. It seems that an elementary test for self-intersection detection is a test of a moving face against a moving vertex. We are already able to detect such cases, so it remains to avoid the detected self-intersections, i.e. to find such a trajectory of the vertex, which avoids collision with a face. The problem of this approach is given by its local nature, because by avoiding one collision, another collision with different face may be caused. The solution would be not to adjust trajectory of individual vertices, but of complex parts of the mesh, which cause self-intersection during the animation. This leads us again to some reasonable partitioning of the mesh.

## 6.5 Generalized topology merging and multimorphing

Let us recall that topology merging is a process used for the construction of the supermesh. It merges topologies of two input meshes. The result is the mesh which can be transformed to the shape of the source mesh as well as to the shape of the target mesh. We would like to generalize this process for more that two input meshes, so that morphing among more than two meshes would be possible. So this issue contains two problems. The first problem is to compute the shared topology for more objects and the latter problem is to establish some reasonable mixture of base objects.

We will call the first problem a generalized topology merging. It involves merging topologies of individual meshes. We expect that following problems will have to be solved. First, as multiple meshes are merged, singular cases sooner or later appear. It will be necessary to suggest some general scheme how to cope with such cases. It seems that perturbation is not a good choice, because it can generate very small edges and faces which may cause problem later. So it seems that an appropriate vertex merging, i.e., a fusing of vertices which are nearby, would be better. Another problem is that due to the topology merging process, a supermesh has typically much more faces than both input meshes. Number of faces can be partially decreased by a suitable vertex merging as described above. As the complexity of the supermesh grows, a problematic point is also the rendering speed, so it seams reasonable to convert it to a strip representation in order to speed up the rendering process. This issue is also part of a FRVŠ G1 project which we are working on together with ing. Petr Vaněček from the University of West Bohemia, Pilsen, Czech Republic. The advantage is that we know how the supermesh is constructed, so we expect to use this information for construction of triangle strips.

The latter problem is as follows. For each vertex of the supermesh we know its source position, i.e. the position of the vertex in the mesh from which the given vertex originates, and several target positions, i.e., a position of the vertex on the surfaces of the other meshes. The input of this task will be probably weights which express how much a position of a given vertex is influenced by individual base meshes. This topic is partially covered in [Ale99b], where the so-called morphing space is introduced.

We already did very preliminary tests in 2D, i.e. morphing between multiple polygons. We expect that the situation in 3D will be very similar. We started in 2D because the topology merging process in 2D is very simple. As well as in 3D case, parametrizations are overlaid. Parametrization analogy in 2D is mapping of polygon vertices to the circumference of a unit circle. The advantage is that in 2D case the order of the vertices of the polygon is given, so it is easy to compute a parametrization, e.g., by a simple regular distribution of the polygon vertices along the circumference of a unit circle. The merging itself is just inserting of a given vertex to a particular edge. The resulting *superpolygon* (analogy of 3D supermesh) contains then vertices of the source polygon and the vertices of the target polygon and it can be transformed to the shape of the source polygon as well as to the shape of the target polygon. An example of merging three polygons is in the Fig. 36.

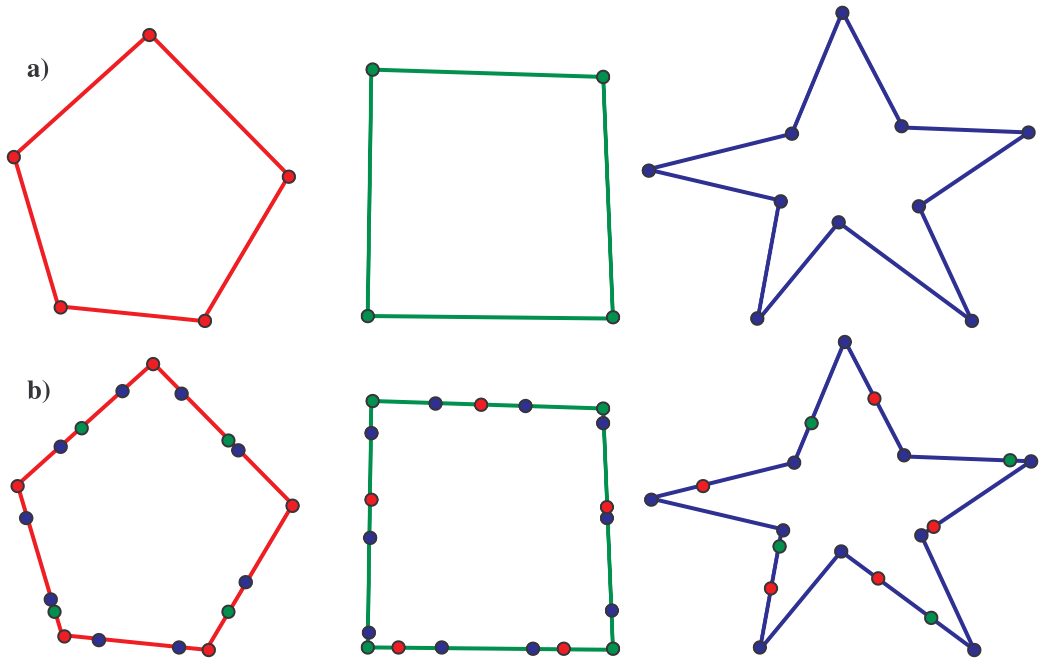


**Fig. 36: a) Input polygons b) Superpolygon transformed to the shape of source polygons. Vertices of superpolygon are colored according to source polygons.**

Having superpolygon, an open question is how to define the morph itself in a reasonable way. One idea is as follows. First let us denote the source polygons $P_0$, $P_1$, ..., $P_{N-1}$. We can transform superpolygon to the shape of $P_0$ and using $N$-1 weights we can control the position of a particular vertex as follows:

$$v_i(t) = v_i^0 + \sum_{j=1}^{N-1} w_j (v_i^j - v_i^0) \tag{25}$$

, where $v_i^0$ is the position of the i-th vertex of the superpolygon in the shape of $P_0$, $w_j$ is the weight which determines the influence of $P_j$ on the vertex position, $v_i^j$ is the position of the $i$-th vertex when transforming superpolygon to the shape $P_j$. Considering only three polygons, the situation according to Eq. (25) is depicted in the Fig. 37.

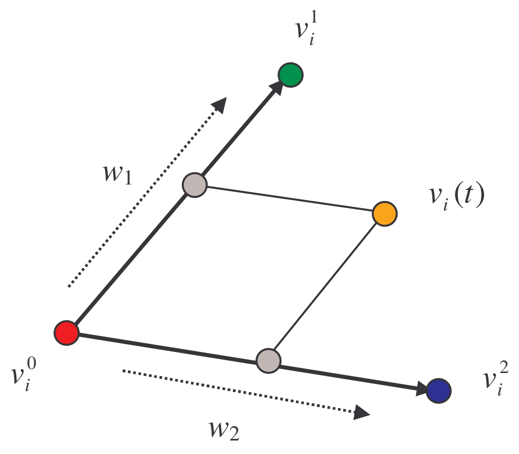


**Fig. 37: A multimorphing scheme according to Eq. (25). The orange point is the resulting position of the *i*-th vertex influenced by the vertices $v_i^1$ and $v_i^2$. The amount of influence is given by the weights $w_1$ and $w_2$.**

## 6.6 Core increment approach

Very recently a paper [Sem05] was published, dealing with morphing of volume representation based on core increment approach. A basic idea is as follows. First, the intersection of the source and the target volume is computed, i.e. voxels which are present in the source and in the target volume. The intersection volume is called a *core*. Then voxels surrounding the core are examined together with constructing two schedules – the *add schedule* and the *remove schedule*. In the add schedule the voxels which will be added during the morph are inserted, in the remove schedule the voxels which will be removed during the morph are inserted. The schedules are filled in the following way. Each voxel surrounding the core is examined – if the voxel is contained in the target volume, the voxel is added into the add schedule, if the voxel is contained in the source volume, it is added into the remove schedule. Then the examined voxel is added into the core. The process ends when the core cannot be extended any more. For generating a morph transition we start with a source volume, then schedules are traversed and the voxels from the add schedule are added to the volume, the voxels from the remove schedule are removed from the volume.

It is a very simple approach. It is fully automatic; the advantage is that it does not require any correspondence computation. We would like to adopt this approach for boundary representation. For simplicity let us consider a 2D case, 3D generalization seems to be possible. A core is computed by intersection of two polygons. Vertices adjacent to the core are examined, added to the appropriate schedules and the core is grown. Then schedules are traversed and vertices are added or removed. Adding and removing of vertices is not as simple as in the volume approach. When adding a vertex, we have to split some existing vertex and gradually move the new vertex to its final position. Analogously, when removing vertex we have to gradually shift it towards some existing vertex until they are coincident. A basic idea is depicted in the Fig. 38. The green polygon is the source polygon, the red polygon is the target polygon. The grey region is the initial core. Arrows shows how vertices of the source polygon will disappear in the core and how the vertices of the target polygon arise from the core in order to form the target polygon, respectively. Numbering shows the order in which vertices should arise.

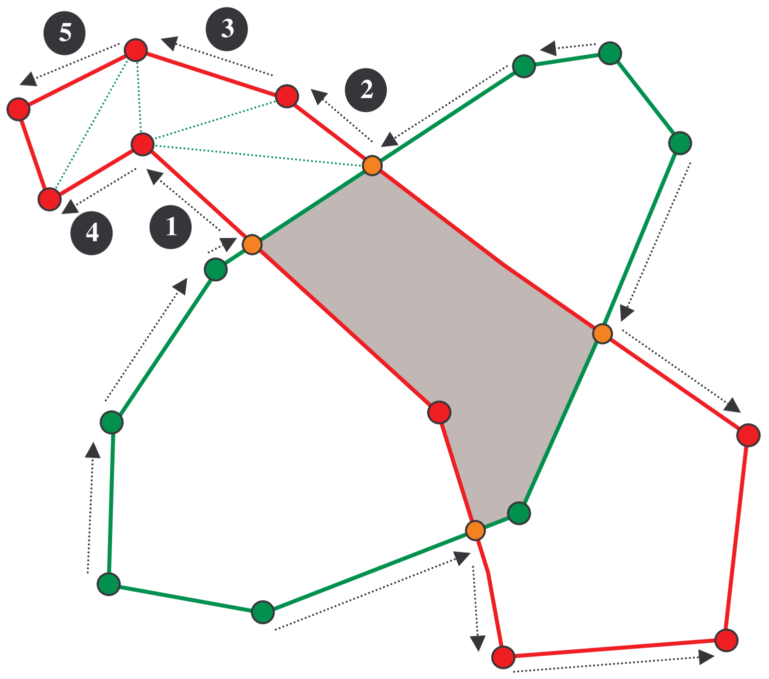


**Fig. 38: An example of the core increment approach.**

This approach is very simple. The main advantage is that it does not require any correspondence computation. Problems probably occur when the core will not be a single component. Also the order in which vertices will appear or disappear will be crucial. On the other side we expect, that this approach will be able to morph between a complicated objects without intersection, which follows from growing like process.

# 7 Conclusion

In this technical report we have introduced a technique for shape transformation called morphing or metamorphosis. It is usually used as an animation technique for creation of some special effects, but it can also be used in the area of design where two existing shapes are combined in order to obtain new shapes. A concrete method is strongly dependent on the object representation. Section 2 briefly outlines possible representations and their processing in the morphing tasks. This report is focused on morphing of the boundary representation. It is because the boundary representation is very widespread in professional animation tools, it is easy to store, render and edit. Another reason is given by the concentration of our team to this kind of representation and thus a possibility to cooperate with other team members. Three basic steps of morphing computation are investigated and described in separate sections 3, 4 and 5.

It is important to mention that morphing is not a well defined problem. Let us remind that usually required results are smooth, realistic looking, plausible, etc., shape transformation, which all are not well defined terms. Some approaches tried to incorporate some physical model. It produces in some cases very good results, but when looking on metamorphosis from the artistic point of view, we do not always want the objects to behave exactly in a physically correct way. It is often required to introduce some artificial behavior which in many cases is more impressive than odd physically correct behavior. Sometimes the animation simply does not need the behavior so perfect as people will not notice the difference anyway.

Another aspect is user workload when creating morphing sequences. In general it holds, the more user can specify various parameters (correspondence, vertex trajectories) the more the resulting animation can be adjusted according to the user's need. It is clear that the manual adjusting requires some time, generally, the better results we want to achieve, the more time is required. On the other side, there are methods which work fully automatically or with a very little user interaction. These methods alone usually provide worse results. But we can use the result of the automatic method as a starting point, which can be further adjusted only in areas where necessary. It can reduce user's workload. So it seems that it is still important to develop automatic methods, because the better results an automatic method gives, the less user's workload will be needed.

An important part of this report is Section 6, where ideas and suggestions for the future work are presented. Some of them are only small implementation improvements, but they have big practical use. Especially the area of interpolation of surface attributes is not very considered by researches, as they mainly focus on shape transformation itself. But it does not mean that the area of shape transformation is solved. As stated in the previous paragraph the plausible shape the transformation is not well defined term and very probably there will always appear new approaches to shape transformation with different effects, some of them more artificial, some of them more physically correct. Maybe some unifying concept of morphing scenarios would be needed, where a particular morphing approach would act as black box, which performs certain actions under certain conditions.

# Appendix A

## Barycentric coordinates

Barycentric coordinates are used for expressing a relative position of the point *p* with respect to a face comprised by vertices $v_1$, $v_2$, $v_3$. The position of the point *p* is expressed as a convex combination of the vertices $v_1$, $v_2$, $v_3$ of the face:

$$p = t.v_1 + u.v_2 + v.v_3\,, \tag{26}$$

where the weights of the convex combination *t, u, v* are barycentric coordinates. Convex combination is the liner combination where the coefficients of the linear combination sums to one, i.e., $t$+$u$+$v$=1. One way how to compute the barycentric coordinates is to evaluate ratios of the areas of the triangles formed by *p*, $v_1$, v$_2$ and $v_3$, i.e.:

$$t = \frac{|\Delta p v_2 v_3|}{|\Delta v_1 v_2 v_3|}, u = \frac{|\Delta p v_1 v_3|}{|\Delta v_1 v_2 v_3|}, v = \frac{|\Delta p v_1 v_2|}{|\Delta v_1 v_2 v_3|}\,. \tag{27}$$

When point *p* lies inside the triangle $v_{1,}$ $v_2$, $v_3$ then *t*, *u*, $v \in$ <0; 1>. In a singular case when the point *p* lies on some of vertices $v_1$, $v_2$, $v_3$ one barycentric coordinate is equal to one and the remaining are zero.

## Point in face

Point in face location test for a spherical case can be adapted as follows. An edge of a spherical triangle can be replaced by a plane formed by two edge endpoints and the center of the parameter domain. The point-in-triangle test then turns to checking whether the query point *q* is oriented in the same way with respect to all three planes formed by triangle edges and the center of the parameter domain, i.e.:

$$((v_0 \times v_1).q \ge 0) \wedge ((v_1 \times v_2).q \ge 0) \wedge ((v_2 \times v_0).q \ge 0)\,, \tag{28}$$

where $v_0$, $v_1$, $v_2$ are the vertices of the triangle and *q* is the query point. The term $(v_0 \times v_1).q$ is the orientation test of the query point *q* and the plane which is given by vertices $v_0$, $v_1$ and the center of the parameter domain. In fact it is a dot product between the normal of the plane and the vector formed by the query point *q* and the center of the parameter domain.

## Intersection of two arcs

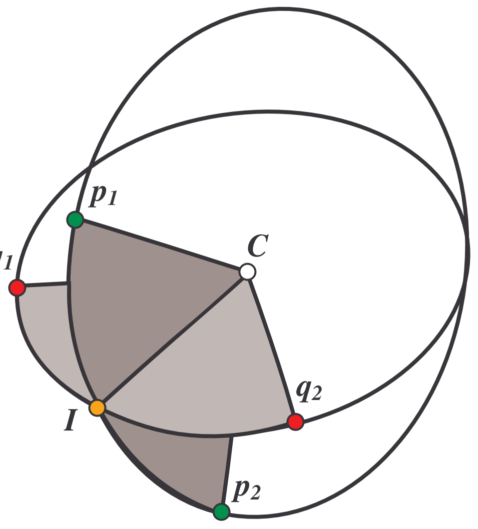
First, let us denote $p_1$, $p_2$ the endpoints of the first arc and $q_1$, $q_2$ the endpoints of the other arc. As the arcs are parts of great circles[42], we first investigate an intersection of great circles and then check whether the intersection is the common point of both arcs. As depicted in the Fig. 39a), the endpoints $p_1$, $p_2$ together with the central point $C$ form a plane. So the intersection of two great circles lies on the intersections of planes formed by $p_1$, $p_2$, $C$ and $q_1$, $q_2$, $C$. The intersection of two planes is a line given by a vector $\vec{r}$ :

$$\vec{r} = \pm(\vec{p}_1 \times \vec{p}_2) \times (\vec{q}_1 \times \vec{q}_2), \tag{29}$$

where $\vec{p}_1 \times \vec{p}_2$, $\vec{q}_1 \times \vec{q}_2$, respectively, gives the perpendicular vectors to the plane $p_1$, $p_2$, $C$ and $q_1$, $q_2$, $C$, respectively. By normalizing the vector $\vec{r}$ we have two intersections of great circles and one of them could be the desired intersection of arcs. To check whether the intersection lies on both arcs, we solve the system:

$$\begin{aligned} t_p.\vec{r} &= p_1 + s_p(p_2 - p_1) \\ t_q.\vec{r} &= q_1 + s_q(q_2 - q_1) \end{aligned}, \tag{30}$$

where $t_p$, $t_q$ and $s_p$, $s_q$ are unknowns, $p_1$, $p_2$ are endpoints of the first arc, $q_1$, $q_2$ are endpoints of the other arc. The values $t_p$, $t_q$ are parameters of the line $CI$ and parameters $s_p$, $s_q$ are parameters of the arcs $p_1$, $p_2$ and $q_1$, $q_2$ as depicted in the Fig. 39b). The intersection $I$ is a common point of two arcs iff $s_p$, $s_q \in$ <0; 1> and $t_p$, $t_q > 0$.



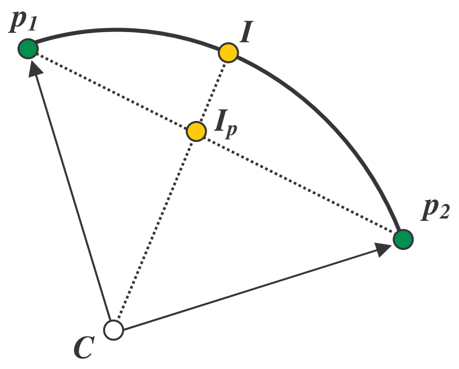


**Fig. 39: a) Intersection of two arcs, which are parts of great circles. b) Checking whether a point lies on the arc.**

[42] The great circle is a circle with the center coincident with the center of the sphere and the radius equal to the radius of the sphere.

# Appendix B

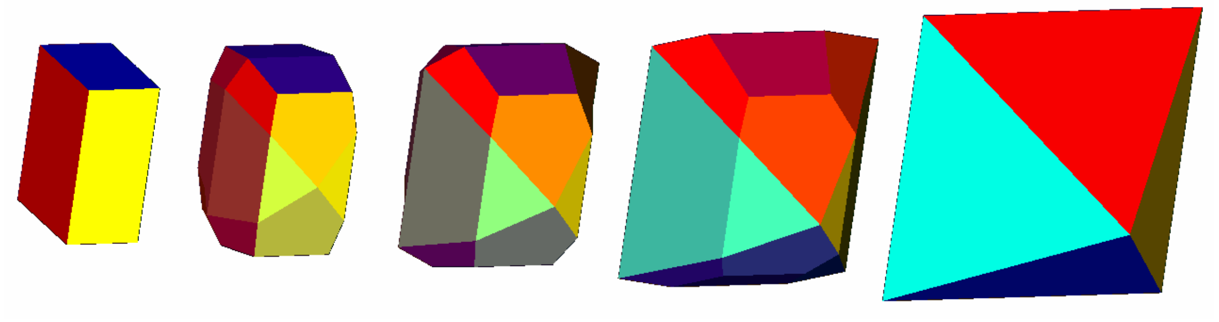


**Fig. 40: Morphing between per face colored meshes.**

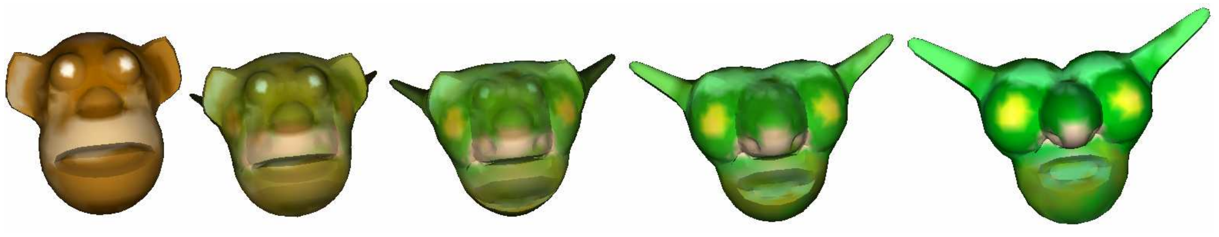


**Fig. 41: Morphing between per vertex colored meshes.**

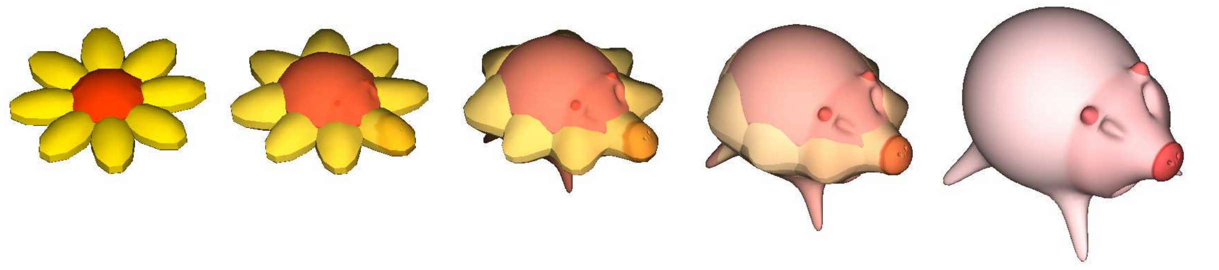


**Fig. 42: Morphing between per face (flower) and per vertex colored meshes (pig).**

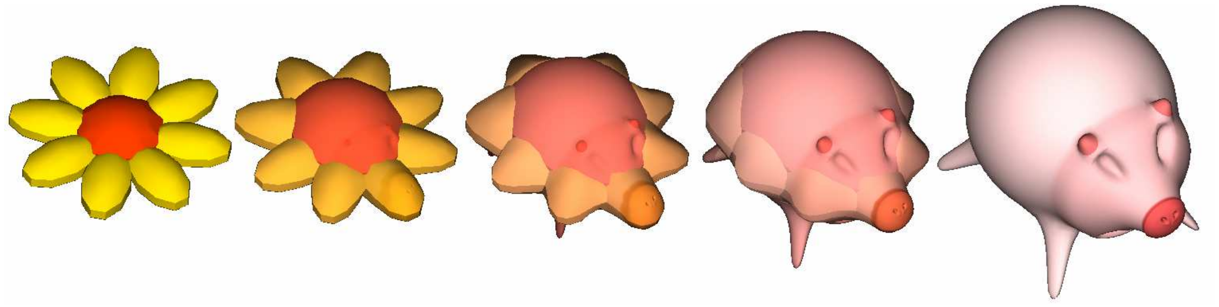


**Fig. 43: Color interpolation in HLS color space, where the hue is interpolated in clockwise direction.**

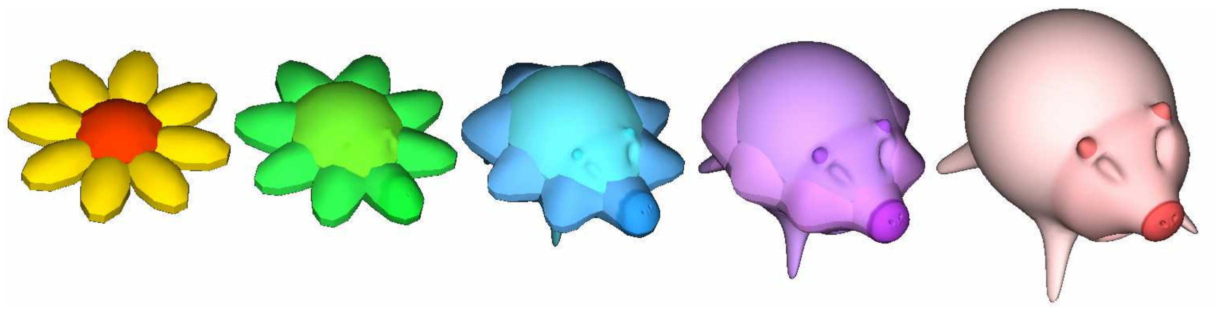


**Fig. 44: Color interpolation in the HLS color space, where hue is interpolated in the counter-clockwise direction.**

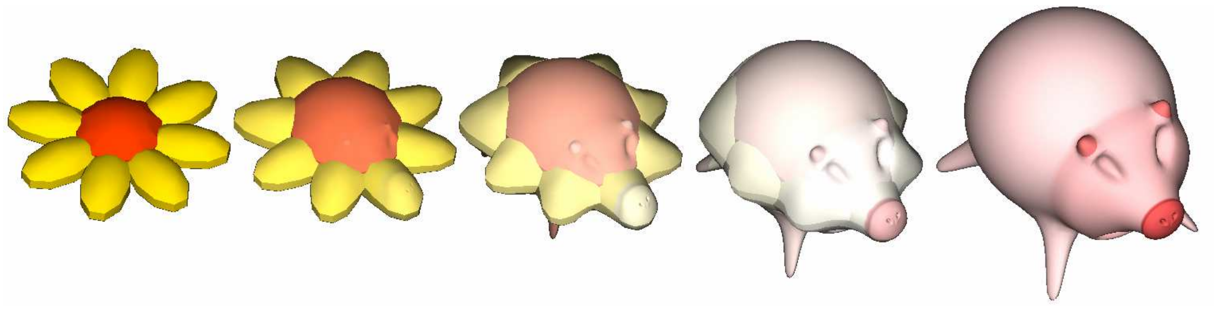


**Fig. 45: Color interpolation in the HLS color space using the desaturation approach.**

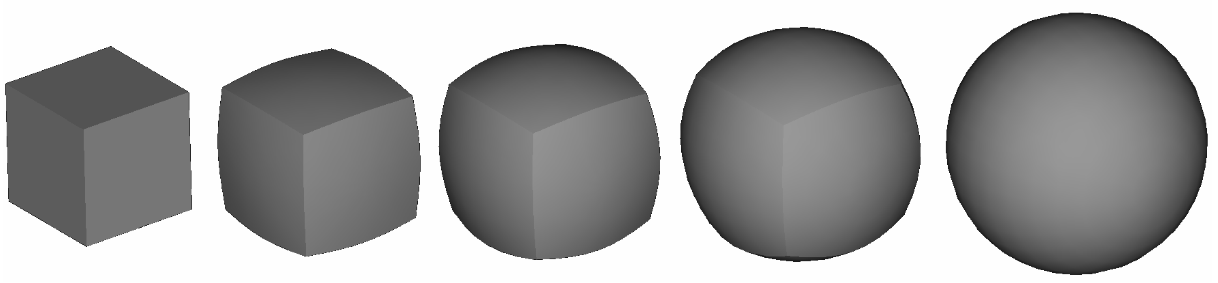


**Fig. 46: Morphing between a cube and a sphere.**

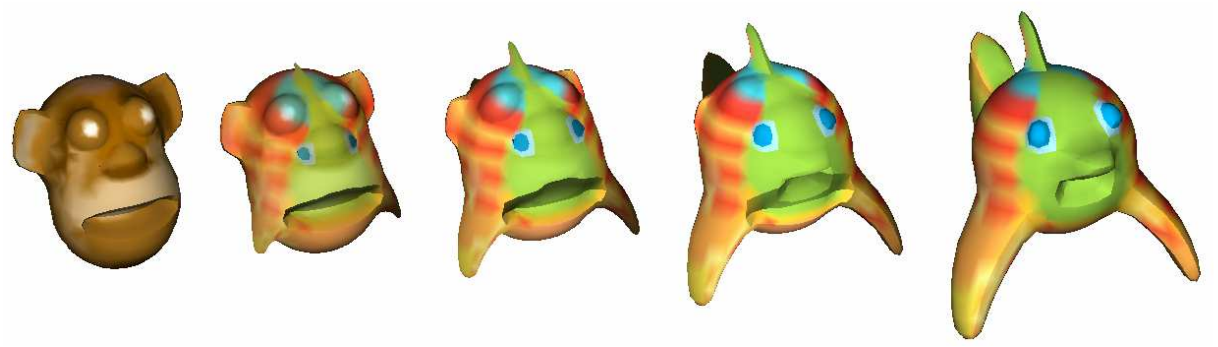


**Fig. 47: Morphing with a kinetic control. There are different kinetic interpolants for the geometry interpolation and for the color interpolation.**

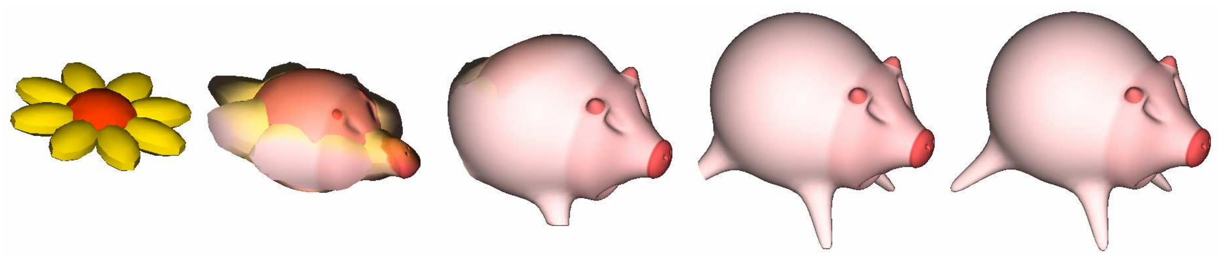


**Fig. 48: Morphing using constant speed.**

# Appendix C

## Publications

Not reviewed:

- Parus, J., Konligerová, I.: *Mesh Morphing*, Proceedings of the 7th Central European Seminar on Computer Graphics, Bratislava, 2003 (poster)
- Parus, J., Kolingerová, I.: *Mesh Morphing*, Proceedings of the International Conference and Competition Student EEICT, VUT Brno, Brno, 2003, pp. 298-302
- Parus, J: Morphing 3D geometrických objektů, diploma thesis, University of West Bohemia in Pilsen, 2003 (supervised by doc. dr. ing. I. Kolingerová)

Reviewed:

- Parus, J., Kolingerová, I.: *Morphing of Meshes with Attributes*, Proceedings of the Spring Conference on Computer Graphics (SCCG 2004), ACM Press, New York, USA, 2004, pp. 69 – 78
- Parus, J., Kolingerová, I.: *Morphing of Color Surfaces*, Proceedings of the Electronic Computers and Informatics (ECI 2004), Vienala Press, Košice, 2004, pp. 415 – 420
- Varnuška, M., Parus, J., Kolingerová, I.: Simple Holes Triangulation in Surface Reconstruction, Algoritmy 2005, Vydavatelstvo STU, Bratislava, 2005, pp. 280-289

## Related talks

- Parus, J.: Morphing trojrozměrných objektů, Centre of Computer Graphics and Data Visualization, University of West Bohemia, Pilsen, Czech Republic, May 2004
- Parus, J.: Morphing of Meshes, Technical University of Graz, Austria, September 2004
- Parus, J.: Morphing povrchových modelů, VŠB Ostrava, Czech Republic, December 2004
- Parus, J.: Morphing povrchových modelů, Centre of Computer Graphics and Data Visualization, University of West Bohemia, Pilsen, Czech Republic, March, 2005
- Parus, J.: Alternativní reprezentace 3D objektů, Centre of Computer Graphics and Data Visualization, University of West Bohemia, Pilsen, Czech Republic, April, 2005

## Stays abroad

- Technical university of Graz, Austria, September 2004

- University of Granada, Spain, February – July, 2002

## Projects

- The use of strip representation for morphing – FRVŠ G1/1349 project, responsible project leader, 2005
- Reconstruction of surfaces from scattered data – FRVŠ G1/1509 project, team member, 2004
- Bilateral Research Cooperation in the Geometric Models Construction and Visualization for Virtual Habitat and Virtual Archeology – project Aktion 36p9
- Algorithms of Applied Computational Geometry – project Kontakt 16-2003-04
- project MSM 235200005 Záměry
- Microsoft research project 2003-187

# References

[Ale00a] Alexa, M.: Merging Polyhedral Shapes with Scattered Features, The Visual Computer, Vol. 16, No. 1, 2000, pp. 26-37.

[Ale00b] Alexa, M., Cohen-Or, D., Levin, D.: As-Rigid-As-Possible Shape Interpolation, Proceedings of SIGGRAPH 2000, 2000, pp. 157 – 164.

[Ale01a] Alexa, M.: Recent Advances in Mesh Morphing, Computer Graphics Forum, Vol. 21, No. 2, 1992, pp. 173-196.

[Ale01b] Alexa, M.: Local Control for Mesh Morphing, Proceedings of Shape Modeling International 2001, 2001, pp. 209-215.

[Ale99a] Alexa, M.: Merging Polyhedral Shapes with Scattered Features, Proceedings of Shape Modeling International 1999, 1999, pp. 202-210.

[Ale99b] Alexa, M., Müller, W.: The Morphing Space, Proceedings of the WSCG '99, 1999, pp. 329-336.

[Bei92] Beier, T., Neely, S.: Feature-Based Image Metamorphosis, SIGGRAPH '92: Proceedings of the 19th annual conference on computer graphics and interactive techniques, 1992, pp. 35-42.

[Bir04] Birkholz, H.: Shape-Preserving Parametrization of Genus 0 Surfaces, WSCG 2004, 2004.

[Dem04] Demaine, E., D., O'Rourke, J.: Open Problems from CCCG 2003, CCCG 2004, 2004, pp. 211.

[Eck95] Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., Stuetzle, W.: Multiresolution analysis of arbitrary meshes, SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995, pp. 173-182.

[Flo05] Floater, S., M., Horman, K.: Surface Parametrization: a Tutorial and Survey, Advances in Multiresolution for Geometric Modelling, 2005, pp. 157-186

[Flo97] Floater, S., M.: Parametrization and Smooth Approximation of Surface Triangulations, Computer Aided Geometric Design, Vol. 14, No. 3, 1997, pp. 231-250.

[Gom99] Gomes, J., Darsa, L., Costa, B., Velho, L.: Morphing and Warping of Graphical Objects, Morgan Kaufmann, 1999

[Gre99] Gregory, A., State, A., Lin, M., C., Manocha, D., Livingston, M., A.: Interactive Surface Decomposition for Polyhedral Morphing, The Visual Computer, Vol. 15, 1999, pp. 453-470.

[Hop96] Hoppe, H.: Progressive Meshes, Computer Graphics, Vol. 30, 1996, pp. 99-108.

[Hor99] Hormann, K., Greiner, G., Campagna, S.: Hierarchical Parametrization of Triangulated Surfaces, Proceedings of Vision, Modeling, and Visualization 1999, 1999, pp. 219-226.

[Hug92] Hughes, J., F.: Scheduled Fourier Volume Morphing, SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques, 1992, pp. 43-46.

[Kan00] Kanai, T., Suzuki, H., Kimura, F.: Metamorphosis of Arbitrary Triangular Meshes with User-Specified Correspondence. IEEE Computer Graphics and Applications, Vol. 20, No. 2, 2000, pp. 62-75.

[Kan97] Kanai, T., Suzuki, H., Kimura, F.: 3D Geometric Metamorphosis Based on Harmonic Map, Proceedings of the 5th Pacific Conference on Computer Graphics and Applications, 1997, pp. 97-104.

[Kan99] Kanai, T., Suzuki, H., Mitani, J., Kimura, F.: Interactive Mesh Fusion Based on Local 3D Metamorphosis, Proc. Graphics Interface '99, 1999, pp. 148-156.

[Ken92] Kent, J., R., Carlson, W., E., Parent, R., E.: Shape Transformation for Polyhedral Objects, Computer Graphics, Vol. 26, Issue 2, 1992, pp. 47-54.

[Kor97] Korfiatis, I., Paker, Y.: The Three-dimensional object metamorphosis through energy minimization, Computers & Graphics, Vol. 22, No. 2-3, 1998, pp. 195-202.

[Laz98] Lazarus, F., Verroust, A.: Three-dimensional metamorphosis: a survey, The Visual Computer, Vol. 14, Issue 8 - 9, 1998, pp. 373 – 389.

[Lee99] Lee, A., W., F., Dobkin, D., Sweldens, W., Schröder, P.: Multiresolution Mesh Morphing, SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 1999, pp. 343-350.

[Ler95] Lerios, A., Garfinkle, C., D., Levoy, M.: Feature-Based Volume Metamorphosis, SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, 1995, pp. 449-456.

[Mic01] Michikawa, T., Kanai, T., Fujita, M., Chiyokura, H.: Multiresolution Interpolation Meshes, Proceedings of the 9th Pacific Conference on Computer Graphics and Applications, 2001, pp. 60-69.

[Par03] Parus, J.: Morphing 3D geometrických objektů, diploma thesis, University of West Bohemia in Pilsen, 2003 (supervised by doc. dr. ing. I. Kolingerová).

[Par04a] Parus, J., Kolingerova, I.: Morphing of Meshes with Attributes, Proceedings of Spring Conference on Computer Graphics 2004 (SCCG 2004), ACM Press, New York, USA, 2004, pp. 69 – 78.

[Par04b] Parus, J., Kolingerova, I.: Morphing of Color Surfaces, ECI 2004, 2004, pp. 415-420.

[Pas04] Pasko, G., Nieda, T., Pasko, A., Kunii, T., L.: Space-time modeling and analysis, SCCG '04: Proceedings of the 20th spring conference on Computer graphics, 2004, pp. 13-20.

[Pra03] Praun, E., Hoppe, H.: Spherical Parametrization and Remeshing, ACM Transactions on Graphics, Vol. 22, No. 3, 2003, pp. 340-349.

[Sed93a] Sedeberg, W., T., Greenwood, E.: A Physically Based Approach to 2-D Shape Blending, SIGGRAPH '92: Proceedings of the 19th annual conference on computer graphics and interactive techniques, 1992, pp. 25-34.

[Sed93b] Sedeberg, W., T., Gao, P., Wang, G., Mu, H.: 2-D Shape Blending: An Intrinsic Solution to the Vertex Path Problem, Computer Graphics, Vol. 27, 1993, pp. 15-18.

[Sem05] Semwal, S., K., Chandrashekhar, K.: Cellular Automata for 3D Morphing of Volume Data, WSCG 2005, University of West Bohemia, 2005, pp. 195-201.

[Sha95] Shapira, M., Rappoport, A.: Shape Blending Using the Star-Skeleton Representation, IEEE Computer Graphics and Applications, Vol. 15, No. 2, 1995, pp. 44-50.

[Sha98] Shapiro, A., Tal, A.: Polyhedron Realization for Shape Transformation, The Visual Computer, Vol. 14, No. 8-9, 1998, pp. 429-444.

[SHE] Adaptive Precision Floating-Point Arithmetic and Fast Robust Predicates for Computational Geometry, http://www-2.cs.cmu.edu/~quake/robust.html

[She04] Sheffer, A., Gotsman, C., Dyn, N.: Robust Spherical Parametrization of Triangular Meshes, Computing, Vol. 72, No. 1-2, 2004, pp. 185-193.

[Ste85] Steketee, S., N., Badler, N., I.: Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control, Proceedings of the 12th annual conference on Computer graphics and interactive techniques, 1985, pp. 255-262.

[Sun97] Sun, Y., M., Wang, W., Chin, F., Y, L.: Interpolation Polyhedral Models Using Intrinsic Shape Parameters, Journal of Visualization and Computer Animation, Vol. 8, 1997, 81-96.

[Sur01] Surazhsky, V., Gotsman, C.: Controllable morphing of compatible planar triangulations, ACM Transactions on Graphics, Vol. 20, No. 4, 2001, pp. 203-231.

[Sur04] Surazhsky, V., Gotsman, C.: Intrinsic Morphing of Compatible Triangulations, International Journal of Shape Modeling, Vol. 9, No. 2, 2003, pp. 191-201.

[SVD] Singular Value Decomposition, http://mathworld.wolfram.com/SingularValue Decomposition.html

[TOP] The Open Problems Project, http://maven.smith.edu/~orourke/TOPP/

[Tur99] Turk, G., O'Brien, J., F.: Shape transformation using variational implicit functions, SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, 1999, pp. 335-342.

[Zha03] Zhao, Y., Ong, H., Tan, T., Xiao, Y.: Interactive Control of Component-based Morphing, Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation, 2003, pp. 339-348.

[Zoc99] Zockler, M., Stalling, D., Hege, H.: Fast and Intuitive Generation of Geometric Shape Transitions, The Visual Computer, Vol. 16, No. 5, 2000, pp. 241-235.