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

Morphing of geometrical objects in boundary representation

The State of the Art and the Concept of Ph.D. Thesis

Martina Málková

Technical Report No. DCSE/TR-2010-02
April, 2010

Distribution: public

Technical Report No. DCSE/TR-2010-02
April 2010

Morphing of geometrical objects in boundary representation

Martina Málková

Abstract

Morphing is a technique for smooth transformation between two objects, however, it may be also used as a tool for modeling new objects from existing ones. In our work, we address both areas. We work with objects in boundary representation (simple polygons in 2D, triangle meshes in 3D). First, we present a new method for natural growth-like transformation, working both in 2D and 3D. Then, we focus on morphing faces, as modeling and animating faces is a very important task in computer graphics. We present a tool for computing average faces from scanned data, which we developed to create face models for psychological tests on face perception, done by psychologists from Charles University in Prague. Possible ideas for further research concerning face meshes are also presented.

This work was supported by The Grant Agency of the Czech Republic, project no. 201/09/0097.

Contents

Chapter 1

Introduction

From a general point of view, morphing can be described as a process when one object is continuously transformed into another. But morphing is not only the whole process (animation), it can be also a way to achieve new shapes or patterns from the old ones. When more than two objects are used as the input, we usually talk about *multimorphing*.

Morphing and multimorphing have wide practical use in computer graphics, animation, modeling, design and compression. Multimorphing is also often used in data visualization [36].

The existing morphing methods can be classified into the following groups - the traditional volume and image morphing and boundary-based morphing. In our work we focus on boundary-based morphing methods, namely methods for triangular meshes. Triangular meshes are widely used, since they are easy to store, modify and render.

In the first part of our work, we concentrate on a morphing problem for general triangular meshes (without further knowledge about the input). The algorithms designed in this area concentrate on morphing similar shapes, where some common features can be identified. However, in some cases it is not possible to align all the main features of the input shapes (e.g. a head with and without horns). A natural morph usually means growth of the non-aligned part from its aligned neighbor. However, the traditional algorithms usually produce something completely different in such a case. Therefore, we concentrated on this problem and designed a new approach for morphing with this "growth-like" nature. After designing the method for simple polygons, we extended it also to 3D for morphing triangle meshes.

In the following work, we decided to concentrate on a particular type of objects, since an algorithm can usually be enhanced if we know something about the input. As our area of interest, we chose human faces. Human faces play an important role in modeling figures for movies, computer games and applications with human-computer interaction. We started to cooperate with Věra Pivoňková from the Faculty of Humanities, Charles University in Prague, who works in the area of face perception.

Based on the needs of the researches from the Faculty of Humanities, we created a method for computing an average face from an arbitrary number of input facial scans (in the form of triangle meshes with texture). Average faces are important for the face perception research, because the average covers the differences between individual faces while the common traits remain unchanged.

1.1 Previous work

This work continues in the research first done by J. Parus [40]. Later we started to work together on the multimorphing problem. We tested several approaches of combining multiple objects together, along with providing the users intuitive interface capable of creating static combinations as well as animations. Based on this work and his own investigations, we published an article *Multimorphing: A tool for shape synthesis and analysis* [41].

After multimorphing, we continued in our cooperation on designing a new polygon morphing technique, which we called *core-based morphing.* This work was based on J. Parus idea that a growth-like morphing could be established by first computing the intersection of the two objects, and than letting the rest of the first object disappear in the intersection, while the rest of the second object is growing out. With his help, I designed and tested methods realizing the growth process. Based on this work, we published an article *An intuitive polygon morphing* [35].

During my work on diploma thesis, we extended the polygon morphing technique to 3D for triangle meshes. We later published this work on a conference as *Core-based morphing algorithm for triangle meshes* [33].

During my Ph.D. studies, I started to cooperate with V. Pivoňková[1], who proposed us a morphing problem interesting for the psychologists - creating a composite (morph) from several human faces for perception tests. We started to work together on this problem, where she provides the psychological part of the research, while we can concentrate on geometrical challenges. We have already discussed a possible future work in this area (see Section 8).

I also started to cooperate with B. Beneš[2]. He offered interesting ideas what to improve in our approach, and we worked together on the article [35]. During my ten-days stay at Purdue University[3] we discussed our possible future work concerning facial morphing (see Section 8).

1.2 Organization of the text

The text of this technical report is organized as follows. Chapter 2 contains definitions and description of common terms used in the text. Chapters 3-4 concentrate on morphing in general. The former describes the related work in both 2D and 3D, the latter presents our new core-based morphing algorithm.

Chapters 5-7 deal with morphing faces. The former offers an introduction to facial models, their creation, parametrization and morphing. The latter describes our work done in the cooperation with V. Pivoňková.

Chapter 8 summarizes the approaches presented and suggests a possible future work, concentrating on facial morphing. Appendix B shows the results from our core-based algorithm in 2D and 3D. Appendix C contains a table of sixteen personality factors defined by Catell. Appendix D shows the results of a remeshing algorithm used for multimorphing in my bachelor work.

[1]Faculty of Humanities, Charles University, Prague
[2]Purdue University,USA
[3]project Kontakt: ME09051

Chapter 2

Related terms

A *polygon* is an ordered set of vertices v_i, $i = 0, \ldots, n-1$. An edge e_i of polygon is a line segment with endpoints v_i, v_{i+1}. A *simple polygon* is a polygon whose consecutive edges e_i, e_{i+1} intersect only in the endpoint v_{i+1}. An unclosed sequence of edges is called a *polygon chain.* A *closed polygonal chain* is a polygonal chain, where also p_0 and p_{n-1} are connected by a line segment. δP denotes the border of a polygon P, $\delta P = \{v_i, e_i, i = 0, \ldots, n-1\}$.

A *polytope* is a generalization of polygon into two dimensions or polyhedron in three dimensions. A *convex polytope* is defined as the intersection of half-spaces.

A *topological distance* $d(v_i, v_j)$ between vertices v_i and v_j in nD is the number of edges in the shortest path from v_i to v_j.

An object O is *convex*, when a line segment connecting its two arbitrary vertices lies completely inside or on O. An object P is *star-shaped*, when at least one point p inside P exists such that a line segment connecting p with an arbitrary vertex of P lies completely inside or on P.

According to [4], a *topology* of an object refers to the vertex/edge/face network. An object is *Euler-valid* if its topology fulfills the formula $V - E + F = 2 - 2G$, where V are the vertices of the object, E edges, F faces and G is the number of passages through the object (*genus*).

2.1 2D Voronoi diagram

Voronoi diagram was first introduced in [51]. In the plane, the Voronoi diagram of a set of points S is the partition of the plane which associates a region $V(p)$ with each point p from S in such a way that all points in $V(p)$ are closer to p than to any other point in S.

The region $V(p)$ is the interior of a (in some cases unbounded) convex polytope called the Voronoi cell for p. The Voronoi diagram is then the set of such polytopes, which subdivides the whole plane. An example of a Voronoi diagram is in Figure 2.1.

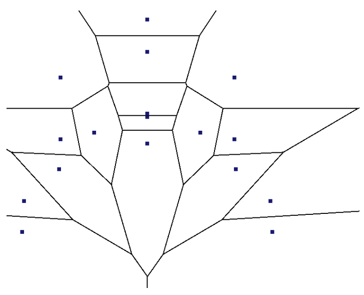


Figure 2.1: Voronoi diagram of a point set in 2D

Chapter 3

Morphing in general

Digital warping and morphing have a long tradition in computer graphics, and the full description is out of the scope of this report. We refer reader to [18] for a detailed description. Here we will describe the work related to polygon and mesh morphing. Most of the approaches described were already discussed in our previous work in this area [33, 35].

Morphing algorithms are usually *correspondence-based*, meaning they first find a correspondence between vertices of the source and the target polygon. They usually need to add vertices to both source and target polygon to make the best correspondence (according to their requirements). When the correspondence is established, the trajectories between corresponding vertices need to be found. This step can be very complicated, however, most solutions use a simple linear interpolation here. As is discussed in [18], this simple choice has some disadvantages when computing rotational morphing. The problem is shown in Figure 3.1 on morphing between two line segments, both of them of the same length, but one of them rotated. If we use the linear interpolation for computing the trajectory of corresponding vertices, the line segments shorten during the morphing process, which is something we do not expect.

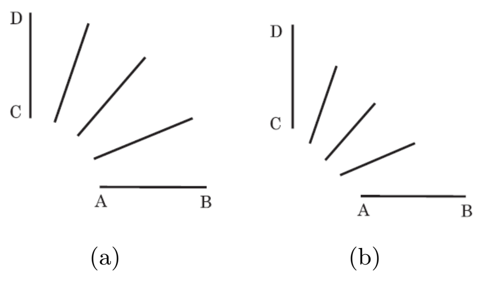


Figure 3.1: An expected morphing sequence (a) and (b) a morphing sequence using linear interpolation, from [18]

Unlike for morphing images, there are no rules how to measure the quality of the resulting morph for meshes. The criteria are dependent on the expectations of the user. However, there are some non-written criteria that the authors of the presented articles probably tried to follow. First, the in-between shapes should not contain self-intersecting edges. Second, the common features of the input objects (e.g. a head, legs, a tail) should not change during the morphing process and no other features should be introduced (no one expects a rib growing out of a dog's back when morphing it to a horse).

3.1 Polygon morphing

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

The problem of vertex to vertex correspondence was addressed by Sederberg et al. in [44] by using a physical model. The polygon edges are modeled as elastic connections and the shape transformation involves the calculation of a physical response by minimizing the energy of the system. This algorithm is efficient for similar input polygons and it can also handle cases when the initial shapes are rotated or translated. The algorithm does not address highly dissimilar shapes and self-intersections.

A computation of trajectories was described by Sederberg et al. in [43], where edge lengths or internal angles are interpolated instead of the vertex positions. The polygons are converted to the so-called edge-angle representation [18] that is invariant to rigid transformations. This interpolation scheme avoids edge collapsing and non-monotonic angle changes. This technique was used to generate in-betweens for animation based on keyframes. The concept of interpolation of intrinsic parameters was also further used for morphing of planar triangulations in [47, 48]. The intrinsic interpolation avoids local self-intersections. However, it does not solve the problem of global self-intersections which may occur for highly dissimilar and complicated shapes.

Shapira and Rappoport [45] introduced a method that first decomposes the source and the destination polygons into star-shaped polygons. The skeleton is a planar graph which joins star-points of neighboring star-shaped polygons. The skeletons are interpolated and the intermediate shapes are reconstructed from the interpolated skeletons. The difference between this approach and methods described in [43, 44] is that this approach also considers the interior of the polygon and not only the boundary. The problem with this approach is that it relies on isomorphic star-shaped decomposition which might be difficult to compute, especially in the case of dissimilar shapes.

Alexa et al. introduced *as-rigid-as-possible shape interpolation* in [5]. They compute a compatible triangulation of the input polygons. The compatible triangulation is a dissection of the source and the destination polygons so that the triangulations are isomorphic, i.e., there is one-to-one correspondence between triangles in the source and in the destination. Then, an affine transformation which transforms a source triangle to the destination triangle is computed. Interpolation of the affine transformation defines the morphing. Adjacent triangles are also considered in the interpolation. Similar approaches were also described by Surazhsky and Gotsman in [47, 48]. The principal problem of these methods is the computation of the isomorphic triangulation of the input shapes, as its quality influences the quality of the morphing.

Gomez et al. introduced *2D merging* [18], which is a 2D application of the algorithm that was originally developed for 3D meshes (see for example [3, 27]).First, the input polygons are mapped to a unit disc, then both mappings are merged. The vertices of the first polygon are mapped on the second polygon and vice versa using inverse mapping. This results in polygons with the same number of vertices. A linear interpolation is used to obtain the resulting morphing transition. This technique is suitable for convex, star-shaped or slightly non-convex polygons. This algorithm is not suitable for highly non-convex polygons, as it produces self-intersections during the morphing transition.

Carmel and Cohen-Or [9] showed an algorithm which combines a 2D merging and a polygon evolution. A user first specifies anchor points that define the correspondence between polygons. Using the anchor points, a warp function is computed that warps the source polygon to the destination polygon. Once the source polygon is warped a polygon evolution technique is used to evolve the source and the destination polygon to a convex shape.

Johnstone and Wu [23] described an alternative approach to merging polygons by morphing. The 2-to-1 morphing is a fundamental case in the morphing between different numbers of polygons. The basic idea is to merge the two polygons into one and then use the one-to-one polygon morphing technique to morph between the merged polygon and a destination polygon. During the merging the two polygons are morphed toward each other until they meet at a point. Then a curve evolution technique is used to morph the two polygons connected at some point into a more natural shape which is later morphed toward the destination shape.

3.2 Mesh morphing

In the terminology of Spanier [46], a mesh M is described by a pair (K, V), where $V = (v_1, ..., v_n)$ describes the geometric positions of the vertices in d-dimensional space (in our case $d = 3$), and K is an abstract complex that represents the connectivity of vertices, edges and faces.

The mesh morphing problem is described in [4]: Two input objects - meshes $M_0 = (K_0, V_0), M_1 = (K_1, V_1)$ are given, and the goal is to generate a family of meshes $M(t) = (K, V(t)), t \in \langle 0, 1 \rangle)$. Most of the morphing methods work in three following steps:

1. **Establish a correspondence between the meshes.** Decide which vertex of the mesh M_0 corresponds to which one of the mesh M_1. This is usually the crucial step of the whole process.
2. **Generating a *supermesh.*** A supermesh is a mesh that represents both M_0 and M_1.
3. **Creating paths $V(t), t \in \langle 0, 1 \rangle$ for the vertices.** Usually the algorithms use an interpolation of corresponding vertices, mostly a simple linear interpolation.

Kent et al. propose in ***Shape transformation for polyhedral objects*** [27] an algorithm that uses both the topology and the geometry of the input objects. First, both objects are projected onto a unit sphere. The vertex to vertex correspondence is established by merging the topologies of the input objects. The merging process is done by clipping the projected faces of one model to the projected faces of the other. The paths for the corresponding vertices are created by either a linear interpolation, or using a Hermite spline with its tangent vectors equal to the vertex normals.

The main problem discussed in their article is the projection onto the unit sphere. Several methods for the projection were proposed, depending on the type of the input objects. For the star-shaped objects, the center point (arbitrary point from the kernel of the object) of the object is found and then the vertices are moved to the surface of the sphere in the direction of a vector defined by their position and the position of the center point. The convex objects are projected in the same manner, only the center point is an arbitrary interior point of the object. Another type of objects are so-called *objects of revolution*. Such objects consist of a set of planar contours arranged at angular increments around an axis (axis of revolution), e.g., a glass constructed by rotating a curve around its axis. The contours are projected onto a longitudinal arc of the sphere by several methods, where among the best is the method of Ekoule [16]. Next type of objects, *extruded objects*, are created by moving a planar polygon along the straight line. The ends of the object are capped by two copies of the polygon. Projecting such object is done by mapping the two caps to its convex hull by Ekoule's method and then projecting the resulting object in the same way as were the convex objects.

Another approach for the projection was to treat the surface model as a flexible object, and inflate the object with air until it is convex. To ensure that the simulation will produce the convex model, the vertices already lying on the convex hull were fixed. The convex model was again projected to the sphere.

The methods of projection are discussed for the most of the genus-0 objects, the authors only suggest how to project other types of objects (replacing a sphere by a representative manifold, or cutting the objects).

Alexa uses the idea of Kent et al. and presents another correspondence-based algorithm for morphing polyhedra in his ***Merging polyhedral shapes with scattered features*** [3]. First, the polyhedron is triangulated. Then the approximation of the smallest enclosing sphere (*a circumsphere*) of the model is computed, the model is transformed such that the circumsphere is transformed to a unit sphere. Then the spherical projection is examined. Because the method is designed for all genus-0 meshes, there can be overlapping edges (*foldovers*) in the projected result. To remove the foldovers, the relaxation process is introduced. The relaxation works iteratively, moving in each step each vertex to the center of its neighbor's positions in the previous step. Some vertices on the sphere need to be fixed to avoid the vertices converge to one position. Such vertices are called *anchors*. At least four anchors are needed (in case of three anchors the vertices would converge into the triangle, as is shown in Figure 3.2a). Even with four anchors the embedding might collapse (see Figure 3.2b). Also, because the

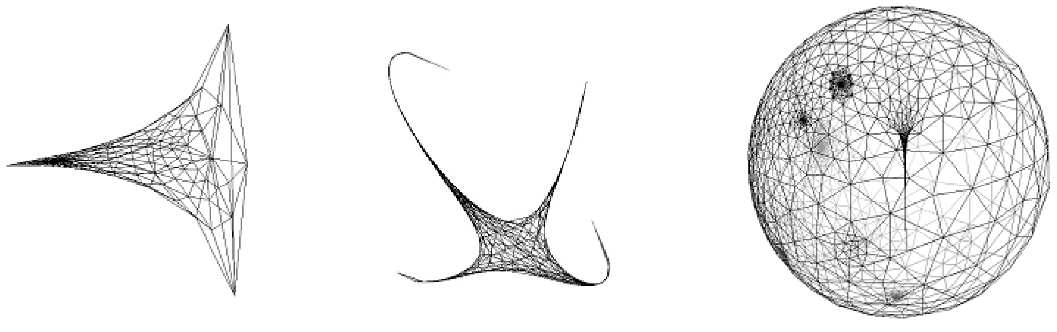

(a) (b) (c)

Figure 3.2: Problems of the sphere embedding: (a) collapsed embedding fixed with three vertices (b) collapsed embedding fixed with four vertices (c) foldovers (from [3])

anchors have fixed positions, they can cause the foldovers themselves. Their solution is simple: As anchor vertices, they choose a random regular tetrahedron with vertices on the unit sphere. Then they perform the relaxation until the largest movement of any vertex in one step is smaller than a predefined constant. If the relaxation collapsed, they move back to the original mesh and choose a different tetrahedron. If not, they fix the vertices diametric to the tetrahedron used and relax again to remove the possible foldovers made by the original tetrahedron's vertices. They switch those two tetrahedra and perform relaxation until there are no foldovers.

If the user has specified any vertex correspondence, the embeddings are deformed so that the corresponding vertices have the same position on the sphere.

The resulting embeddings are merged. All edge intersections are found (edges are here the shortest path between two points on the sphere), new vertices are inserted at their positions and the corresponding edges are cut. The result of this process is a merged mesh (a supermesh), which is not necessarily a triangle mesh, there can be non-triangle (but still convex) faces. Also, it is not guaranteed that more than three points lie on the same plane, so the supermesh needs to be triangulated after the merging.

The supermesh is deformed to have the shape of the source (and equivalently target) mesh by setting its vertex positions. The vertices of the source mesh remain the same, but as the vertices of the target mesh do not exist on the source mesh, their positions are computed by using the barycentric coordinates - the barycentric coordinates of the vertex v in the triangle v_1', v_2', v_3' in the supermesh are computed and used to find the position of v in the triangle v_1, v_2, v_3 in the original object. Also we need to compute the position of the vertices that were created due to the edge intersection. Each such vertex lied at the intersection of two edges - one of the source and one of the target mesh. When deforming the supermesh to have the shape of the source mesh, we use the source edge, and compute the barycentric coordinates with respect to the vertices defining this edge. And again, the barycentric coordinates are used to find the position of the vertex in the source mesh.

Ahn et al. try to enhance the correspondence-based morphing by decreasing the number of vertices of the supermesh in **Connectivity transformation for mesh metamorphosis** [2]. They use a spherical embedding from Alexa's approach [3] to find the correspondence between M_0 and M_1. both M_0 and M_1 are mapped onto the unit sphere. M_0' (and similarly M_1') is constructed by incrementally mapping each vertex v_1 of M_1 onto the surface of M_0: first, we find the face f_0 that contains the mapped position of v_1. Then, v_1 is added to M_0 and connected to the three vertices of f_0 (see Figure 3.3b). Then they swap some of the created edges (not the original meshes of M_0) to reduce the difference between M_0' and M_1 (Figure 3.3c).

Then, the sequence of connectivity transformations between M_0' and M_1' is computed by using an adaptation of Hanke and Ottmann's algorithm [19]: For each edge of M_0' and M_1', they check if there exist the corresponding edge on the opposite mesh, and if not, they compute an error (based on Euclidean distance) that occurs if they swap the edge to get the correct position. In such a way, they build the priority queue of edge swaps, sorted according to the computed errors. Because some edge swaps are dependent on the other ones, they construct the transformation dependency graph to perform all swaps in the shortest possible time. After the dependency graph construction, they compute the exact time portions for each swap. The resulting in-between meshes are constructed by transforming the vertices of the

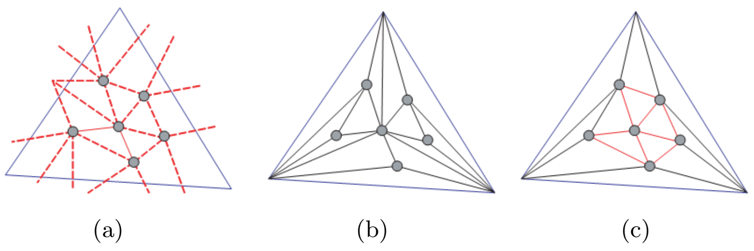


Figure 3.3: Mapping the target vertices onto the source mesh: (a) original configuration of target vertices mapped onto the source triangle (b) result of simple embedding (c) enhanced result after edge swaps (from [2])

supermesh according to the vertex-to-vertex correspondence established at the beginning, and incrementally swapping the edges according to the plan established by the dependency graph. Each swap is realized by a *geomorph* [20] to make it smooth: A vertex is inserted at the intersection of the two edge positions, and it moves toward one of the vertices of the target edges. When it reaches the vertex's position, it is removed.

Because of the used spherical embedding, the approach can be used only for genus-0 objects. Another disadvantage is the need of computations during the creation of the in-between meshes, which slows down the resulting animation. On the other side, the resulting meshes contain much smaller number of vertices in comparison to other approaches using a fixed connectivity. The visual results are claimed to be similar to Alexa's approach.

Cohen-Or et al. describe a non-correspondence based approach in their ***Three-dimensional distance field metamorphosis*** [12]. Their method needs the user to define corresponding control points (anchor points) on the input objects first (their number depends on their complexity). Then they use the corresponding points to define such warp function $\{W_t\}_{t=[0,1]}$ that $W_1(M_0)$ approximates M_1 as well as possible. The warp function consists of a rigid (rotation, translation) and elastic transformation of M_0. Then it generates a signed 3D distance field by rasterizing the warped object into a binary discrete volumetric representation and converting it into a distance field by a method presented in [31]. Both M_0 and M_1 are represented as discrete distance field (DF) volumes, and the intermediate object (supermesh) is constructed by generating its DF-volume and extracting its surface. The quality of the resulting morph highly depends on a proper warp - if the corresponding points of the two objects are correctly aligned by the warp, it produces the expected results. Otherwise, it may produce results that are far away from the expected ones, sometimes containing parts that unexpectedly disappear and reappear. Also the creation of volumetric representation can consume a large storage space for meshes with a large number of triangles. Then the method requires the object to be simplified before it is converted into a distance field. The main benefit of this method is that it does not require the input objects to be of the same topological genus.

Chapter 4

Core increment morphing

In this chapter we will describe our new morphing approach. The approach in 2D and 3D was already published in [33, 34, 35]. This chapter will provide a basic overview over the algorithm, we refer the reader to [34] for a complete description.

The main motivation for designing the algorithm were results from testing available methods [44, 9]. It showed up that the methods concentrate on morphing similar shapes, where some common features can be identified. They try to preserve the common features and morph between them, which is also a behavior expected by the user. However, in some cases it is not possible to align all main features of the input shapes (e.g. a head with and without horns). In such a case, the methods usually fail, mostly when the shape of the non-aligned part is highly curved. The resulting morphs contain lots of self-intersections (an example of such morph is on Figure 4.1). In this case, the non-aligned part should probably continuously grow from or disappear in the rest. We decided to create a new method dealing with this problem.

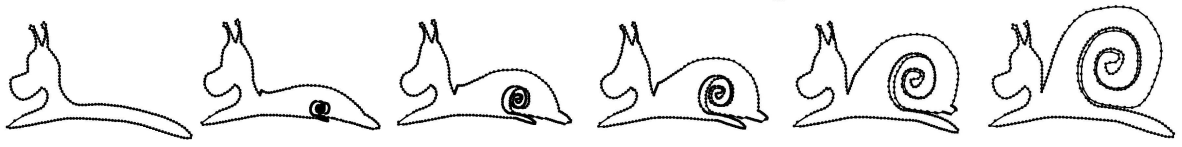


Figure 4.1: Morphing methods have problems with curved, non-aligned parts (Sedergerg and Greenwood algorithm)

During our research, we first concentrated on solving the problem in two dimensions - between arbitrary polygons. After designing and testing the algorithm for polygons in 2D, we extended the methods for triangle meshes in 3D.

In Section 4.1, we will describe the general structure of our algorithm independently on the dimension. Sections 4.2 and 4.3 contain short description of the 2D and 3D versions of the algorithm.

4.1 The algorithm in general

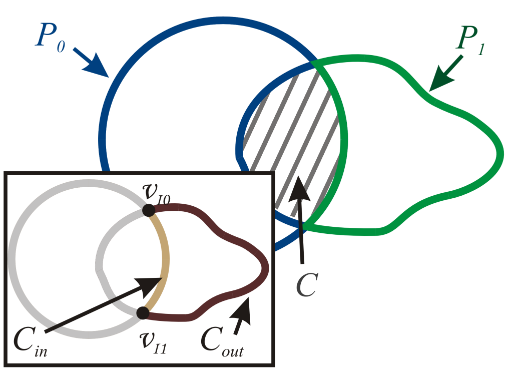
To stay independent on space, let us denote a simple polygon in 2D and a triangle mesh in 3D as an *object*.

The algorithm takes as input two simple objects (see Fig. 4.2), the source object A and the destination object B which must spatially overlap, $A \bigcap B \neq \emptyset$. No further condition on the objects is imposed. However, it is beneficial if their vertices are distributed equidistantly. Therefore, an optional preprocessing step is to resample A, B so that their vertices are equidistantly distributed.

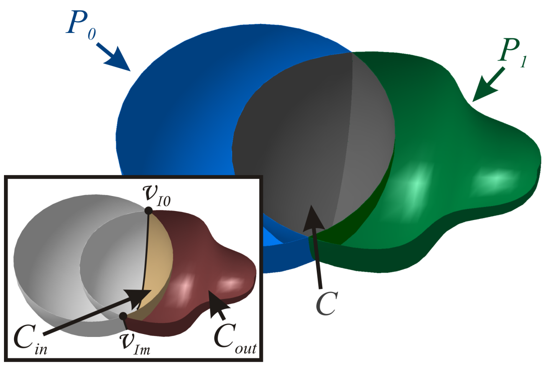
The overlapping area of A and B is called *the core* of the morph and it is denoted by $C = A \cap B$. Without loss of generality, we suppose that the core C consists of a single object.

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

Only the boundary of the core and the parts are computed: $C = (V_C, E_C)$, $P_i = (V_{Pi}, E_{Pi})$. We can divide the vertices and edges of P_i into two sets C_{in}, C_{out}, where C_{in} consists of vertices and edges common for the core C and the part P_i ($V_{Cin} = V_C \cap V_{Pi}$, $E_{Cin} = E_C \cap E_{Pi}$), C_{out} is the part of P_i which remains after removing C_{in} from P_i ($V_{Cout} = V_{Pi} \backslash V_{Cin}$, $E_{Cout} = E_{Pi} \backslash E_{Cin}$). Intersection vertices v_{Ii} are such vertices of C_{in} that lie at its "edge", meaning at least one of their neighbors belongs to C_{out}. By morphing C_{out} to C_{in} we achieve the effect of disappearing of the part P_i in the core C. Figure 4.2 shows the discussed terms, note that the fill is used to distinguish between the terms, not to denote volumetric objects. Also notice that in 2D, there are two intersection vertices, but in 3D, there are n intersection vertices. Therefore we introduce an *intersection chain* $I = (V_I, E_I)$, which is a closed polygonal chain, whose vertices are the intersection vertices and its edges are such edges of C_{in} that connect the intersection vertices. Part of the intersection chain is sketched in Figure 4.2b by a solid black line between v_{I0} and v_{Im}.



(a) 2D



(b) 3D

Figure 4.2: General terms: Core C, part $P_i = C_{out} \cup C_{in}$, intersection vertices v_{Ii}.

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

The computation of a vertex path depends on the specific method of our algorithm. Most of the methods use a concept of a topological distance, which is computed with respect to the intersection vertices. The concrete computation of the topological distance depends on the space, so it will be discussed later.

As already told, the vertex paths are computed only for the vertices of C_{out}. That is because the final object (superpolygon/supermesh) will contain only the vertices of C_{out} of each part, the intersection vertices and some parts of the core. Formally, the superpolygon/supermesh can be denoted as $S = (V_S, E_S)$, $S = (\bigcup C_{out} \cup (C \setminus \bigcup C_{in}) \cup \bigcup I)$, where $\bigcup C_{out}$ denotes the union of all C_{out} from P, Q, $\bigcup C_{in}$ and $\bigcup I$ analogically.

The reason why we cannot just take the whole parts and the core and produce the final morphing sequence is that the vertices of C_{in} rest at their positions during the time, while the vertices of C_{out} change their positions (travel toward their corresponding vertices in C_{in}). During this change, some vertices of C_{out} can cross an edge of C_{in} - and at that time, the vertices and edges that were inside and so were not visible, are now visible producing self-intersections (see Figure 4.3).

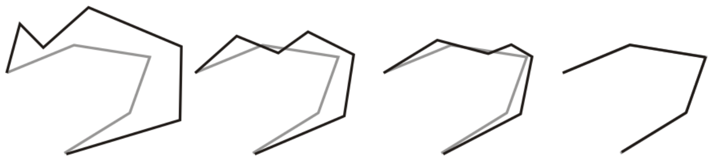


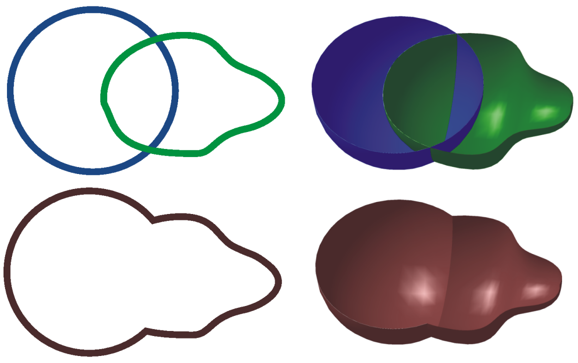
Figure 4.3: Not removing C_{in} may result in unwanted self-intersections: C_{in} (gray), C_{out} (black)

The merging process is shown in Figure 4.4. The final object consists of vertices of only C_{out} from each part and the intersection vertices. Sometimes it can contain parts of the core - it happens when the input objects A and B share some edges and vertices. The merging algorithm is also dependent on the space dimension, so it will be described in the corresponding sections.

The pseudocode of the algorithm can be seen in Fig. 4.5. In step 4 of the algorithm, we use three different methods for computing the vertex paths to morph C_{in} and C_{out} and to simulate a process of absorption of P_i; the Perimeter growing, the Midpoint growing and the Projection growing. These methods will be described in the further text.

4.2 2D solution

In 2D, the input objects A, B are simple polygons. A part P_i consists of two polygon chains C_{in}, C_{out} (described in the previous section). The polygon chains C_{in}, C_{out} are separated by intersection vertices v_{I0}, v_{I1} (Figure 4.2a). An intersection vertex lies in the intersection of the input polygons A and B. If $P_i \neq A$ and $P_i \neq B$ then P_i has two intersection vertices. By morphing the polygon chain C_{out} to the polygon chain C_{in} we achieve the effect of disappearing



(a) 2D (b) 3D

Figure 4.4: Merging (top: the input objects, bottom: the merged result)

Input: Two partially overlapping polygons/meshes A,B (*Optional:* Resample the input objects so that their vertices are equidistantly distributed).

Output: Polygon/mesh $R = A \cup B$, where each vertex of δR either contains a vertex path determining its behavior over time, or belongs to $A \cap B$.

The algorithm:

```
1. Compute the core $C = A \cap B$.
2. Compute the polygon/mesh sets $P = A - B = \bigcup_i P_i$, $Q = B - A = \bigcup_i Q_i$.
3. $\forall v_i \in \delta P$ and $\forall v_i \in \delta Q$: compute its topological distance $d_i$ .
4. Using a user-selected method (Perimeter, Midpoint, or Projection growing), compute
   the vertex path of each vertex $v_i$.
5. Merge the polygon/mesh $C$ and the polygon/mesh sets $P, Q$ to get the resulting poly-
   gon/mesh $R$ .
```

Figure 4.5: Pseudocode of the algorithm.

of the part P_i in the core C. Hereby, we decompose the polygon morphing problem into several polygon chain morphing problems.

The morphing between polygonal chains C_{in} and C_{out} is realized by computing a vertex path for each vertex of C_{out} (see previous section). We designed three different methods for the vertex path computation: the Perimeter growing, the Midpoint growing and the Projection growing. These methods will be described below.

4.2.1 Vertex path computation

We designed three different methods for the vertex path computation: the Perimeter growing, the Midpoint growing and the Projection growing. The methods for the vertex path computation use the concept of *topological distance*. From the definition from Section 2, the topological distance should be positive only, but we use also negative sign to distinguish the vertices lying on the polygonal chain C_{in}. d_{min}, d_{max} are then minimal and maximal topological distances of the vertices from δP_i.

Perimeter growing The vertices $v_i \in C_{out}$ travel along the perimeter of the polygon P_i, meaning that their vertex paths contain only positions of the vertices of δP_i. The vertex path of a vertex v_i with the topological distance d_i contains vertices with topological distances $(d_{i-1}, d_{i-2}, ..., d_0, d_{-1}, ..., d_j)$ The vertex path of the vertex with d_{max-i} should end at the vertex with d_{min+i} (see Fig. 4.6).

Midpoint growing This method uses the midpoints of the line segments defined by the vertices of the same polygonal chain C_{out} or C_{in} with the same topological distance (Fig. 4.7) as the vertices in the vertex paths. If m_i is the midpoint of the line segment $v_i v_j$ where $d_i = d_j$, then the vertex path of a vertex v_i with the topological distance d_i contains vertices $(m_{i-1}, m_{i-2}, \ldots, m_0, \ldots, m_{j-1}, v_j)$. As for the Perimeter growing method, the vertex path of the vertex with d_{max-i} should end at the vertex with d_{min+i}.

Projection growing Here, all vertices of C_{out} have the same number of elements in their vertex paths. First, the vertices of C_{out} are mapped (projected) onto the line segment l_i defined by the intersection vertices (i.e., the vertices with a zero topological distance), using an equidistant mapping. The line segment l_i is divided into $n_{out} + 1$ parts, where n_{out} is the number of the vertices of C_{out}. We assign the vertices of C_{out} sequentially to the new vertices on the line segment (Fig. 4.8a). The next step is to map the vertices of C_{in} in a similar way (Fig. 4.8b). The last step (Fig. 4.8c) is to sort the projected vertices of C_{in} and C_{out} into an ordered list in the order in which they appear on l_i. We denote the list $v_{I0} = a_0, a_1, a_2, \ldots, a_{n-1} = v_{I1}$, where a_i is a mapped vertex that originally belongs either to C_{in} or to C_{out}. Then we traverse this list as follows:

1. Start at v_{I0}. Go through the list until a vertex of C_{in} is reached. Add it into the vertex paths of all the vertices of C_{out} that are located before this vertex.
2. Continue traversing the list. Each time the vertex C_{out} is reached, add the recent vertex of C_{in} to its vertex path.
3. The traversal is completed when v_{I1} is reached.

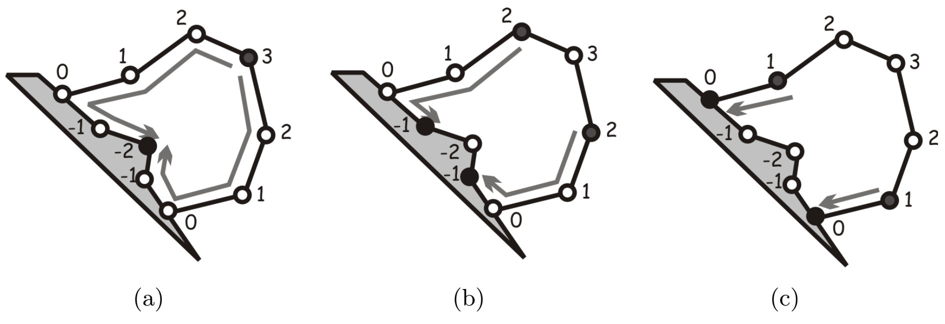


Figure 4.6: Vertex paths ($d_{max} = 3, d_{min} = -2$): (a) the vertex path for the vertex with $d = d_{max}$ ends at the vertex with $d = d_{min}$ (b) for the vertex with $d = d_{max-1}$ it ends at the vertex with $d = d_{min+1}$ (c) and so on

Merging

As has been already figured in the general description, after we handle separately each part, we want to merge them, so that the result is one polygon with a vertex path for each vertex.

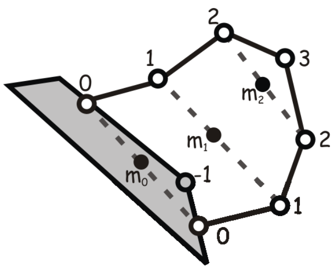


Figure 4.7: Midpoints m_0, m_1, m_2 connecting the vertices with the same topological distance (and of the same polygonal chain)

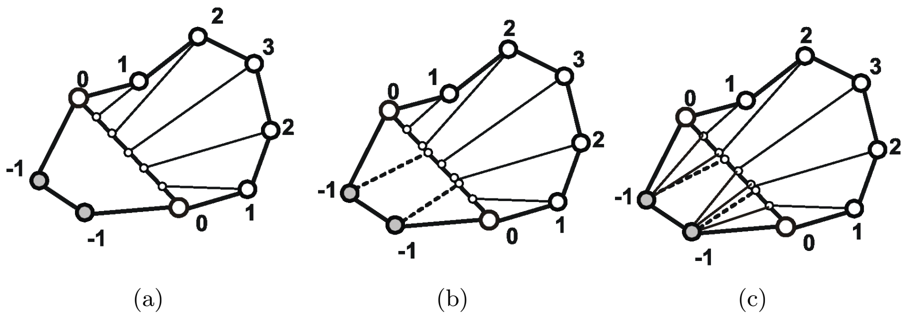


Figure 4.8: Computing the vertex paths in the Projection growing method (a) mapping the vertices of C_{out} onto the line segment between the intersection vertices (b) the vertices of C_{in} onto the same line segment (c) choosing the vertices of C_{in} for the vertex paths of the vertices of C_{out}(the mapping is marked by a dashed line, the vertex paths are outlined by thin lines)

Also remember that vertex paths of the growing parts must be reverted because we considered only the disappearing.

The merging process in 2D is motivated by Weiler-Atherton algorithm for polygons intersection [54]. The merging algorithm sequentially processes all parts from P, Q, copying vertices with positive topological distance to the new list of vertices. It skips between the adjacent parts at the intersection vertices. The new list of vertices forms the new polygon. The details of the merging algorithm are shown in Figure 4.9.

4.3 3D solution

In 3D, the input objects A, B are triangle meshes. A part P_i consists of two surfaces C_{in}, C_{out} (shown in Figure 4.2). The surfaces C_{in}, C_{out} are separated by polylines, called *intersection chains*. Unlike in 2D, there are not always two intersection chains - the number of the intersection chains can vary from one to infinity (see Figure 4.10). By morphing the surface C_{out} to the surface C_{in} we achieve the effect of disappearing of the part P_i in the core C.

In 3D, we compute the topological distance with respect to the vertices lying on the intersection chains and separately for C_{in} and C_{out}. We also use the negative topological distances for C_{in} as we did in 2D. The topological distance of a vertex v_i is computed by the *Breadth-first search* algorithm, where the search begins at the intersection chain(s). The topological

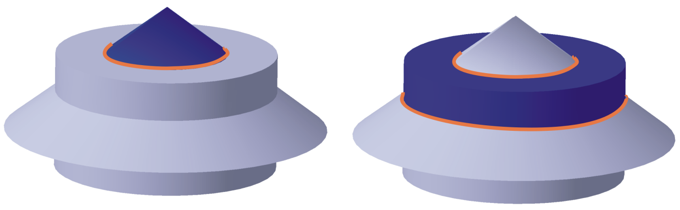
Input: List of parts $R = P, Q = \bigcup R_i$, lists of vertices of each part $l_i = (v_0, \dots, v_{n-1})$, list of vertices of the core $C = (c_0, \dots, c_{m-1})$. The lists l_i and C are circular (so the next vertex to v_{n-1} is v_0 and the previous vertex to v_0 is v_{n-1}). Each part has a different number of vertices in its list, but each part shares exactly two vertices with two other parts (the intersection vertices) or with the core.

Output: One list of vertices containing such vertices v_j from the lists l_i that have $d_j \geq 0$.

The merging algorithm:

1. Choose an arbitrary part R_i from the list of input parts (for example the first one). Start from the first vertex in R_i. Go through l_i until the first intersection vertex v_j is found. Add v_j to the resulting list (which now contains only v_j).
2. Check the vertex v_{j+1} if its topological distance is positive. If so, continue forward, otherwise backward, in l_i. Add each visited vertex to the resulting list until the next intersection vertex v_k is added. Delete the part R_i from the list of parts.
3. Because v_k was the intersection vertex, there are three possibilities:
   - One of the parts in the list contains it - in such a case use this part and continue by 2.
   - The list of parts is empty (v_k is the intersection vertex from step 1). In such a case, the algorithm is finished.
   - The input polygons A and B shared some vertices and edges, and therefore no part in the list contains v_k. In such a case, we find v_k in the core list C - let us denote the found v_k as c_j to know that it is in the list C. We check the vertex c_{j+1} if it belongs to l_i of the part where v_k was. If so, go backward, otherwise forward, in the list C. Add each visited vertex to the resulting list until the next intersection vertex c_k is added. Denote c_k as v_k and continue by step 3.

Figure 4.9: The merging algorithm.



(a) one intersection chain (b) two intersection chains

Figure 4.10: There can be an arbitrary number of the part's intersection chains: a cone and a cylinder (light blue represents the input objects, dark blue indicates the current part being processed, intersection chains are marked by orange lines).

distances are computed separately for C_{in} and C_{out} of each part (we could search the whole part together, but we want to assign negative topological distances to C_{in}). An example of computed topological distances can be seen in Figure 4.11.

4.3.1 Vertex path computation

From the 2D methods for the vertex path computation, we decided to extend only the Perimeter and the Projection growing methods, as the Midpoint growing method produced almost similar results to the Perimeter growing. The methods are shortly described below (full

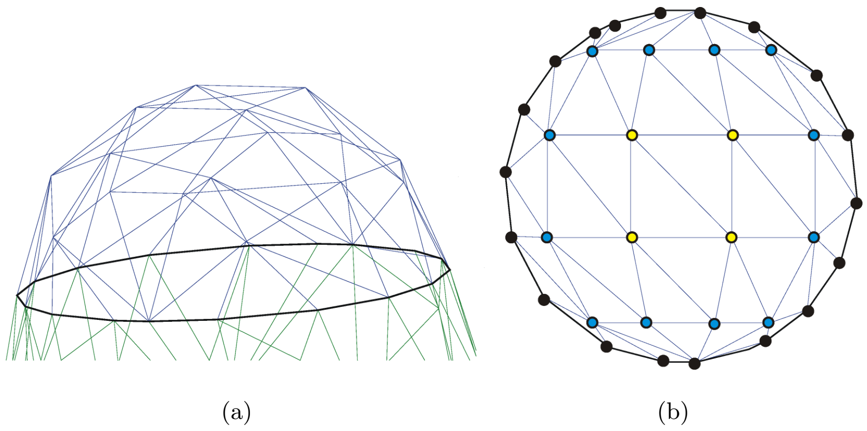


Figure 4.11: An example of topological distances in 3D (a) side view of the part (blue), the intersection chain (black) (b) top view - computed topological distances: - intersection points ($d_i = 0$, black), $d_i = 1$ (blue), $d_i = 2$ (yellow).

description can be found in [34]).

Perimeter growing The vertex paths are set during the computation of the topological distance. When a vertex v_i assigns the topological distance $d_i + 1$ to a vertex v_j, its vertex path and the vertex itself is copied to the vertex path of the vertex v_j. The correspondence problem cannot be solved as easily as in 2D, which resulted in the following extension. The vertex paths are computed for both vertices from C_{in} and C_{out} and the final morphing animation is done as follows. In the first part, $(0, t-\delta)$, the outside of the part (vertices with $d > 0$) is morphing toward the intersection vertices. At $t - \delta$, the resulting morph appears to contain only the intersection vertices (but there are also the vertices of the outside part, which are at the same positions as the intersection vertices), connected by edges defined by the outside part. Let us call it S_0. In the last part, $(t + \delta, 0)$ the inside of the part (vertices with $d < 0$) is morphing from the intersection vertices toward their positions. At $t + \delta$, the resulting morph appears to have only the intersection vertices, connected by edges defined by the inside part. Let us call it S_1. During the time $(t - \delta, t + \delta)$, we need to morph from S_0 to S_1, which are two triangular meshes with the same vertices, but a different connectivity. This can be done using a method from [2]. The process is shown in Figure 4.12.

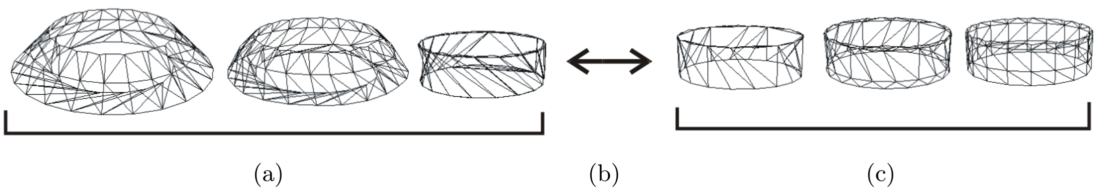


Figure 4.12: The resulting morphing sequence consists of three parts: (a) the outside part morphs toward the intersection vertices, (b) connectivity change, (c) the inside part morphs from the intersection vertices.

Projection growing The vertices from C_{out} and C_{in} are projected (using orthogonal projection) on a plane p defined by three vertices from the intersection chain. The correspondence problem is solved using Voronoi diagram on p. The Voronoi diagram is set for the vertices from C_{out}, and each vertex travels toward such a vertex from C_{in} that lies in its cell. In the case of more vertices, the vertex is duplicated, and in the case of no vertices in the cell, the neighborhood cells are searched to find the nearest vertex from C_{in} to travel to. Figure 4.13 shows several examples of the projected parts and resulting Voronoi diagrams. Bottom left corner shows the original part (blue) with its projection on the plane (red). Note that the plane's orientation depends on the intersection chain. On the plane, a 2D Voronoi diagram is constructed (top), defined by the vertices from C_{out} (blue).

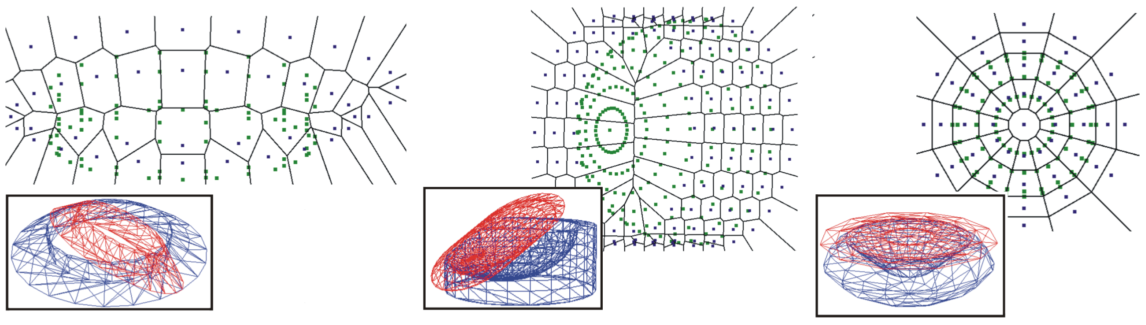


Figure 4.13: Voronoi diagrams of the projected parts
(bottom left corner: a part (blue), projected part (red); top - diagram of the part: vertices from C_{out}, forming the diagram (blue), vertices from C_{in} (green))

Figure 4.14 shows an example of the complete computation together with selected frames from the resulting animation.

4.4 Summary

Results from the methods are shown in B. It can be seen, that we succeeded in creating growth-like animation, which resulted in better results for the non-common features between the shapes. However, our methods are not built to align the common features of the shapes. The user should decide which method to use according to the concrete input shapes.

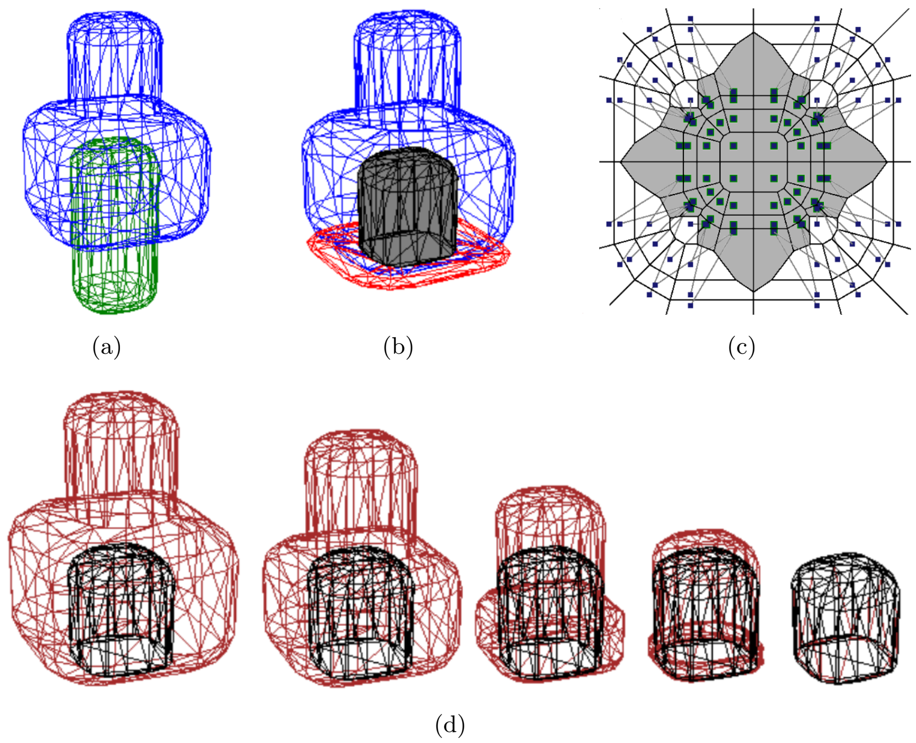


Figure 4.14: An example of the vertex path computation by the Projection growing method (a) input objects, (b) core (black filled with gray) and one part (blue) projected onto the plane (red), (c) Voronoi diagram of the part: vertices outside the core (blue), vertices on the core (green), corresponding vertices connected by gray lines, (d) examples of the resulting morph the part (red), the core (black)

Chapter 5

Facial models and face morphing

The rest of this report concentrates on facial models and face morphing. This chapter poses as an introduction to this area, showing several methods, how to obtain a face model, and some best-known parameterizations used for model animation.

5.1 Capturing and creation of a face model

Real face data can be captured by laser scanners (Section 5.1.1), which capture the geometry of the model as well as the color information. However, the laser scans capture only the "visible" parts of the face, the other parts have to be approximated.

Digitizers (Section 5.1.2) may be a solution for capturing the whole surface, however, the process takes quite a long time and the subject has to remain still the whole time. Therefore, digitizers are used mainly for capturing faces that were first sculpted from clay.

A cheap, but not very precise way of capturing the data is generating a model from photographs (Section 5.1.3). Much work has been done in this area, as it does not require any expensive equipment. With less or more manual effort, contemporary methods can produce models from one, two or more photographs.

When we do not require a model of a particular face, we only want to generate a model with specific characteristic or only any plausible face model, we can either use methods generating models according to the data measured on real faces (Section 5.1.4), or methods using a set of real faces and creating new faces as a combination of the real ones (Section 5.1.5).

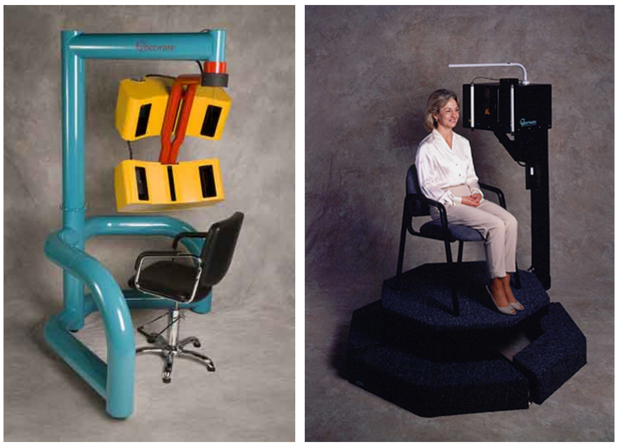
5.1.1 Laser Scanning Systems

The laser scanners are able to capture static surface data. The surface is sampled at regular intervals, producing an unorganized set of surface points in cylindrical coordinate system, which need to be processed to create a usable face model. Some scanners are also able to capture color information, in other case the photograph of the face is taken and needs to be mapped onto the model.

For capturing the whole head, a scanning apparatus is moved around the object (see Figure 5.1). Regular mesh in a cylindrical coordinate system is created.

The main problem in creating models from laser scanned data is that some points may be missing - for example in the eye pupils, hair, under the chin or in the nose. Such data needs to be computed, for example by iteratively using nearest neighbor values [30], or by using methods processing images [56].

Also, the data from the laser scanners may be noisy, so some data smoothing needs to be applied [56]. Also, the surface normals computed for the polygonal meshes should be smoothed, as the common methods for the computation are too local.



(a) (b)

Figure 5.1: Examples of head scanners from the Cyberware Laboratory [22]

5.1.2 Digitizers

Digitizers measure the location of each surface points, where the surface points may be the polygonal vertices or control points for parametric surfaces. The positions are measured sequentially in a specific order. The types of digitizers differ in their physical measurement technique.

Mechanical digitizers have a mechanical stylus attached to a mechanical arm or an orthogonal set of mechanical tracks. A disadvantage of such a solution is that some points on the surface may be unreachable. Acoustic digitizers measure the time of flight of sound pulses to multiple sound sensors. The sound source is usually located at the measuring stylus. Electromagnetic digitizers generate orthogonal electromagnetic fields. Field sensors are attached to the stylus and provide signals which can be converted to the stylus location and orientation. The principle of this method excludes the use of materials that can block or distort electromagnetic fields.

Examples of digitized models, created by fitting a smooth parametric surface to the digitized data points are Billy from a short Pixar film *Tin Toy* and Gollum from The Lord of the Rings. Both models were first sculpted from clay, on which identifying marks were positioned.

A process of digitizing a human head by a MicroScribe digitizer is shown in Figure 5.2.

5.1.3 Photogrammetric Measurement

Generating model from photographs is the cheapest way of obtaining the data, as no expensive equipment is needed. The head is photographed from several views. To ensure that the head

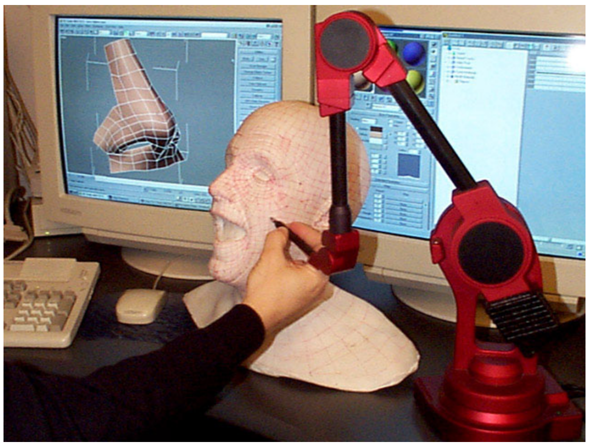


Figure 5.2: Digitizing a head using MicroScribe digitizer (from [1])

is at exactly the same position, mirrors are often used. Also, the methods usually assume a face symmetry, so they construct only one half of the face.

The photogrammetric methods usually need several points marked to set up the correspondence between the photographs [21, 52].

An impressive approach creating a model from a single photograph was proposed by Blanz and Vetter in [7]. Their approach uses a PCA-based statistical head model, whose parameters can be adjusted to resemble the input image. The method adjusts the parameters automatically, however, the initial camera parameters must be supplied with the image. Figure 5.3 shows an example of the process: in the input image (top left), the user adjusts the provided 3D model, so that the gaze and size of the object corresponds to the photograph (top right). Then an automatic, iterative comparison and object's parameter change is done.

5.1.4 Creating the models from general measurements

Physical anthropologists have been measuring various aspects of human heads and faces. The data from a specific age and gender have been collected and analyzed, and average values and ranges have been computed (the main anthropometric landmarks are shown in Figure 5.4). There exist three ways of using the measured values: designing a method that automatically generates plausible faces, as a clue in designing new models manually, or evaluating models generated by other methods. The general guidelines are nicely summarized in [39], and describe the plausible shape of the head, distance between the eyes, position of the nose, mouth, ears and the shape of the eyes, nose, lips and ears. Differences between male, female and a child are also discussed.

The approach using anthropometric measurements from [17] to automatically generate plausible face geometries is described in [14]. Their face geometry is represented as a B-spline surface, the mesh is a tube with openings at the mouth and neck.

Their algorithm works as follows. First, a random set of values is generated according to the face anthropometry. These values are then treated as constraints on a parameterized surface. The best surface satisfying the constraints is constructed using variational modeling [55].

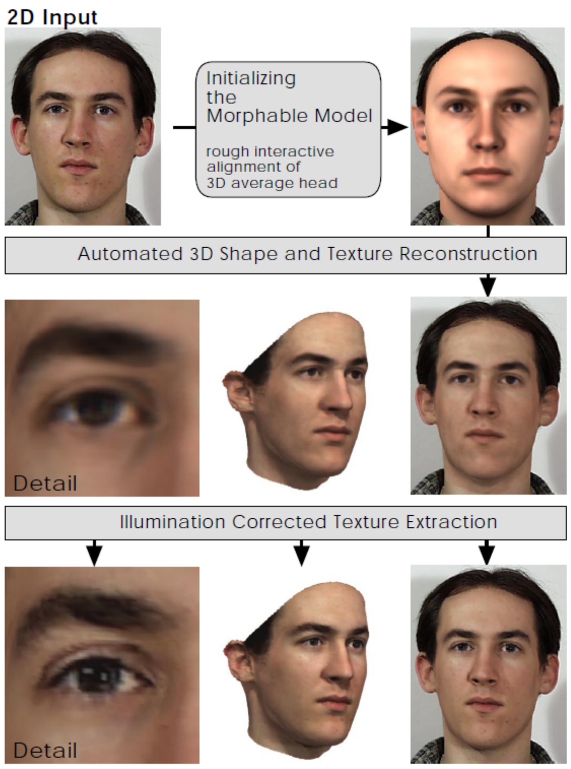

2D Input
Initializing the Morphable Model
rough interactive alignment of 3D average head
Automated 3D Shape and Texture Reconstruction
Detail
Illumination Corrected Texture Extraction
Detail

Figure 5.3: Steps for reconstruction 3D shape and texture from a single image (from [7])

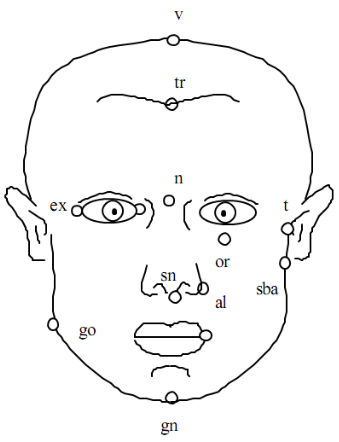


Figure 5.4: The main anthropometric landmarks (from [17])

Along with the anthropometric measures, they use a measure of *fairness*, which formalizes how much the surface bends and stretches away from the prototype face shape. The prototype had to be manually constructed and it is shown in Figure 5.5. Two examples of faces, one male and one female, generated by the algorithm are shown in Figure 5.6.

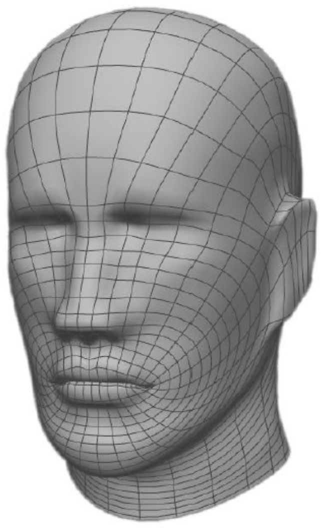


Figure 5.5: The prototype face model from [14]

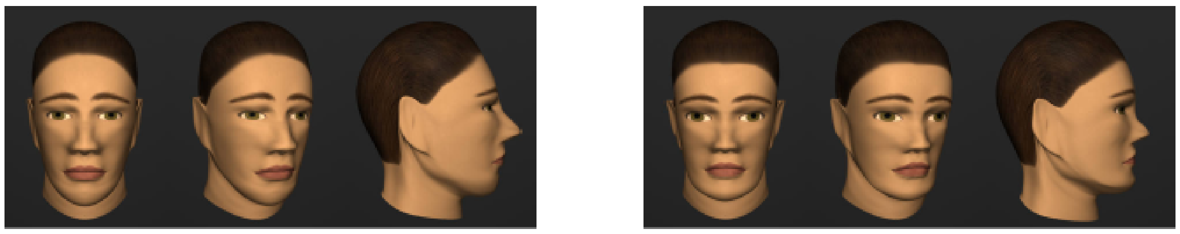


Figure 5.6: An automatically generated male and female face from [14]

5.1.5 Creating new faces from the existing ones

We can also create new faces from the existing ones by using interpolation or morphing techniques, or by using deformation.

In the simplest face, when all input faces have the same topology and the same number of vertices, we can use interpolation techniques to create new faces. For the parametric surfaces, we can use the same method to process the data with the same type of surface, the same number of patches and the same number of control points.

If the input faces do not have the same topology, we may use some of the morphing techniques. These techniques usually create so-called *supermesh*, which shares the topology of all the input objects, and may be deformed to form any of the objects. Each vertex of such supermesh then contains information about its position for each mesh, and the resulting object is created by interpolation between these positions.

PCA (Principal Component Analysis) [24] can be also used for the generation of new models. PCA describes a face model as a weighted sum of an orthogonal basis of 3D shapes (principal components). The construction of the basis requires a huge amount of models that are placed in mutual correspondence. Each basis weights produce different models, however, it is not defined, which weights produce plausible models and which not. Also, the resulting models are highly dependent on the models from which the principal components were constructed. Some models that are outside the example set may not be approximated well.

Based on PCA is an algorithm developed by Blanz and Vetter [7]. They started with laser scans of 200 heads of young adults. Then they established the correspondence between the scans using a gradient-based optic flow algorithm [6], modified for the 3D scans [50]. This way, they created m corresponding shapes S_i and textures T_i, from which a new model S with a texture T could be created as a linear combination of the input models:

$$S = \sum_{i=1}^{m} a_i S_i$$

$$T = \sum_{i=1}^{m} b_i T_i, \sum_{i=1}^{m} a_i \sum_{i=1}^{m} b_i = 1$$

The parameters a and b need to be generated in a specific way to create only plausible faces. The average shape S and texture T are computed, along with covariance matrices computed over the shape and texture differences. PCA is used to transform the data to an orthogonal coordinate system formed by the eigenvectors s_i and t_i of the covariance matrices (in descending order according to their eigenvalues).

This way, the new face was always a combination of the whole base faces. They decided to get more freedom in producing new shapes by segmenting the morphable model and creating the new model independently from the segments. The independent segments were eyes, nose, mouth and a surrounding region (see Figure 5.7). After the new segments were created, they blended them at the borders as it is done in the algorithm for images [8].

5.2 Facial animation

Creating a facial animation deals with two problems: defining a control parameterization and developing a technique and model based on this parameterization.

Face parameterization should be done according to facial primitive actions. As the animator controls the animation only by a manipulation of the parameters, the set of the parameters should be as small as possible, with independent effects, and the effect of each parameter should be intuitive. Also, the animator should be able to generate common, important or interesting motions by manipulating only one or few parameters.

Parameterizing faces is useful also for encoding and decoding face animation for teleconferencing. On one end, the camera captures images of a face and face parameters need to be extracted. Those parameters are then compressed and transmitted, so that on the other end, a visual surrogate can be displayed.

Three main schemes of parameterization for facial animation will be discussed in the following text: interpolation (Section 5.2.1), Facial Action Coding System (FACS) (Section 5.2.2) and Facial Animation Parameters (FAP) (Section 5.2.3).

There are many facial deformation models, varying from the simplest working only with the geometry to complex models considering physical and structural laws. Those models will be discussed along with the parameterizations that they use.

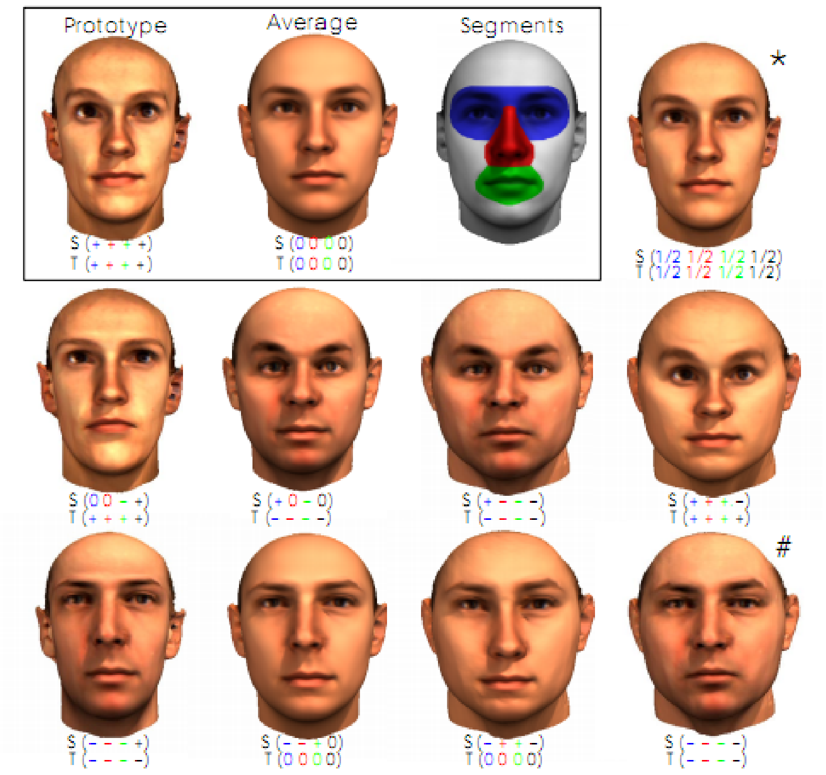


Figure 5.7: Dividing a morphable model into segments and creating a new face from a prototype and an average model (from [7])

5.2.1 Shape Interpolation

The simplest approach of face animation is using interpolation techniques to smoothly animate between two key poses. At each time step from a normalized time interval, interpolation function updates the positions of the face mesh vertices. Usually, a linear interpolation is used for its simplicity.

Interpolation technique can be used when the vertex correspondence of the input models has been already computed. This can be done using one of the morphing techniques. After the correspondence is set up, we can either interpolate between the positions of corresponding vertices between the pairs of key poses, or we can use multimorphing to generate facial poses not directly represented by the keys. In multimorphing, we describe key positions in the animation not by the key poses themselves, but by a set of their weights. The result is then computed by the weighted sum of the key poses, and the animation is done by interpolation between the pairs of weight vectors.

The shape interpolation approach is quite restrictive, as the animator cannot control parts of the face separately and the visual impact of changing the contribution of a key pose might be difficult to predict for the animator. Also, the animation is restricted to poses that can be represented as a linear combination of the key poses. To generate a significant range of expressions, there need to be a huge amount of key poses in the database.

The approach of Joshi et al. [25] presents a way to reduce the negatives of shape interpolation by (automatically) segmenting the face into regions. To create the expression, the user manipulates the regions separately - changing one region does not effect anything else. Segmenting the face has a positive impact on the number of expressions as well - a wider range of expressions can be expressed by the same number of key poses. The automatic segmentation is done for a specified key poses. A deformation map is computed for the model, containing the maximum amount of local deformation across all expressions for each vertex. Using a selected threshold, the model is split into regions with low and high deformation. The last step is to clean the regions by absorbing isolated regions into larger ones and minimizing the concavity of the regions. Example of the process is shown in Figure 5.8. The input models for the process must have the same number of vertices and connectivity, with known vertex-to-vertex correspondence. The created regions overlap by exactly one vertex all around their boundary. The texture of the resulting model is computed by blending all input textures, however, the parts of the texture that should not vary among expressions, such as hair, neck and ears, are not blended, but used from the model with neutral expression.

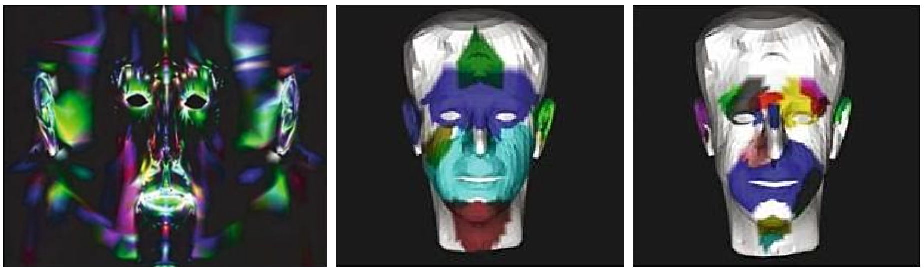

(a) (b) (c)

Figure 5.8: Automatically generated regions on the face: (a) deformation map (b) segmentation for a low threshold, (c) segmentation for a high threshold (From [25]).

5.2.2 Facial Action Coding System (FACS)

Facial Action Coding System (FACS) is a system for describing facial expressions, based on a research of facial anatomy done by the psychologists Paul Ekman and Wallace Friesen in 1976 ([15]). Their goal was to describe all possible visually distinguishable facial movements. The expressions are defined as a set of basic facial movements, which they called Action Units (AU). Their system contains 46 AUs, where each AU represent a contraction or relaxation of one or more muscles. Among AUs are for example brow raiser, cheek raiser, lid tightener, chin raiser, mouth stretch, eyes closed or eyelid slit (see Tables 5.1,5.2). Two different action units can be associated with one muscle, i.e., an inner brow and an outer brow. Some AUs even do not involve any of the facial muscles, i.e. tongue out, neck tightener, lip bite or cheek puff.

A facial movement may involve only one AU, but sometimes more AUs need to be combined. Not all AUs can be combined together, as some involve opposite actions. Also some actions can conceal the presence of others, which are no longer needed to be involved in the action.

FACs can reliably distinguish actions of the upper part of the face, however, it does not include all visible actions of the lower part of the face, especially movements for forming

AU	FACS name
1	Inner Brow Raiser
2	Outer Bow Raiser
4	Brow Lower
6	Check Raiser
9	Nose Wrinkler
12	Lid Corner Puller
14	Dimpler
15	Lip Corner Depressor
17	Chin Raiser
23	Lip Tightener

Table 5.1: Sample Action Units

Expression	Involved AUs
Happiness	AU1, 6, 12, 14
Sadness	AU1, 4, 15, 23
Disgust	AU2, 4, 9, 15, 17

Table 5.2: Simple Expressions in FACS

phonemes. In addition, it does not include the head movements.

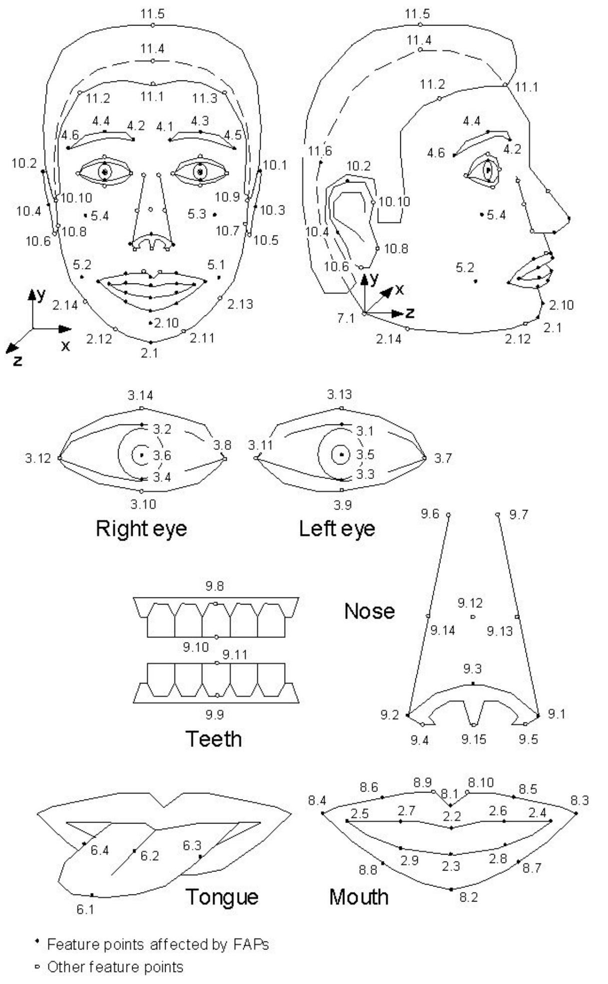
Although FACs was not intended to be used in computer animation, it has been widely used as a basis for expression control in many facial animation systems. Originally, FACs was designed only to describe facial movements, but in the animation systems, it is used to control facial movement by specifying the muscle actions needed to achieve the desired expression changes. Also, most of muscle models for animating face use FACS to relate expressions to muscle activation.

5.2.3 Facial Animation Parameters

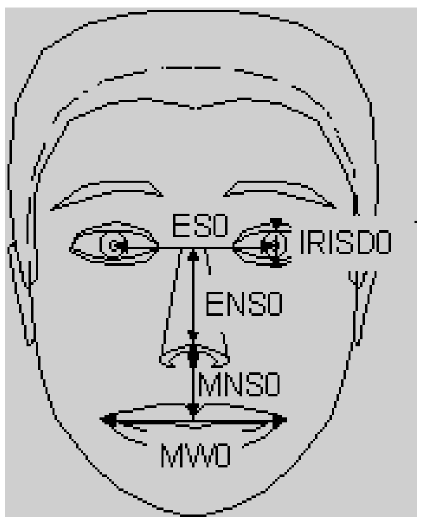
SNHC (Synthetic Natural Hybrid Coding), a subgroup of MPEG-4, designed a coding method for facial models and compressed transmission of their animation parameters ([37]) for purposes of video teleconferencing.

The standard defines 84 Feature Points (FPs). The model defined by FPs can be animated by an associated set of Facial Animation Parameters (FAPs). There are 68 FAPs, categorized into 10 groups related to parts of the face. The parameter set contains two high-level parameters, which represent visemes (visual part of phonemes in speech) and facial expression. Every low-level FAP corresponds to a FP and defines deformations (on a vertex level) applicable to it. (Figure 5.9(a)) shows MPEG-4 feature points. All displayed feature points can be used for calibrating the face, but only the ones filled by black color can be used for animation. Feature points are numbered according to a group they belong to.

As FAPs are universal parameters (independent on the geometry of the model), they have to be calibrated before using them for an animation on a particular model. For this purpose, there exist Facial Animation Parameter Units (FAPUs) and the FAP values are defined in this units. FAPUs are computed as fractions of distances between key features on the face in its neutral state. As the animation is defined by FAPs in FAPUs, we can easily replace the model in the animation by replacing FAPUs.
Figure 5.9(b) shows FAPUs specified by the standard: IRISDO (iris diameter - distance between upper and lower eyelid), ESO (eye separation), ENSO (eye - nose separation), MNSO (mouth - nose separation) and MWO (mouth-width).



(a)



(b)

Figure 5.9: MPEG-4 Feature Points (a) and FAPUs (b) (from [37])

The neutral state, needed for the calibration, is defined as follows:

- the coordinate system is right handed; head axes are parallel to the world axes
- gaze is in direction of Z axis
- eyelids are tangent to the iris
- the pupil diameter is one third of IRISDO
- lips are in contact; the line of the inner lips is horizontal and at the same height of lip corners
- the tongue is flat, horizontal, with the tip of the tongue touching the boundary between upper and lower teeth

The standard does not specify any particular way of achieving facial mesh deformation for a given FAP. It only specifies a Facial Animation Table (FAT) to determine which vertices are affected by each FAP and how.

[Pelachaud et al., 2001, Dalong et al., 2002, Pelachaud, 2002, Kshirsagar et al., 2000].

5.3 Parke facial model

Parke facial model [39] is a model for animating symmetrical, polygonal face mesh, where edges of the polygons are aligned to facial feature lines. The movement of the groups of vertices is defined by five basic operations:

Procedural construction According to given parameters, such as iris size or eye position, the eyes are defined.

Deformation Each part undergoing deformation (such as forehead, cheek, neck or neck) has two extreme shapes defined and the resulting position is computed as an interpolation between these positions according to a given parameter.

Rotation Rotation is used for moving the jaw.

Scale Scaling modifies the size of basic face parts, such as nose, mouth or jaw.

Translation Translation modifies the length of the nose, width of the mouth or elevation of the upper lip.

The parameters for face animation can be divided into two groups: expression parameters, i.e., eye position, iris size, jaw rotation, width of the mouth, head orientation, and conformational parameters, i.e., shape (relative position of assigned vertices) of the neck, chin, cheek, forehead and cheekbones, position of the eyes, size of eyelids, jaw width, length of the nose, color of the skin, eyebrows, lips and iris. The conformational parameter set is to distinguish between different individuals, however, the expression parameters provide the animation itself. The parameter set is shown in Figure 5.10.

5.3.1 Muscle models

Parameterizations defining the movement of the vertices directly may sometimes lead into creating an expression that is physiologically impossible. Muscle models try to eliminate these results by incorporating a muscle layer into the model. The muscle layer included is usually based on FACs (see Section 5.2.2), which describes which muscles have a visual effect on the face and what type of effect it is. To create more realistic face, the model usually consists of three layers: a surface layer, a muscle layer and a bone layer. Each layer is represented as a triangle mesh, and all layers are connected by edges. Each vertex contains information about its position, velocity, weight, and force affecting it. Each edge contains information about its elasticity.

Muscle models also include forces to avoid penetrating the scull during the muscle contraction, and include volume preservation in their calculations.

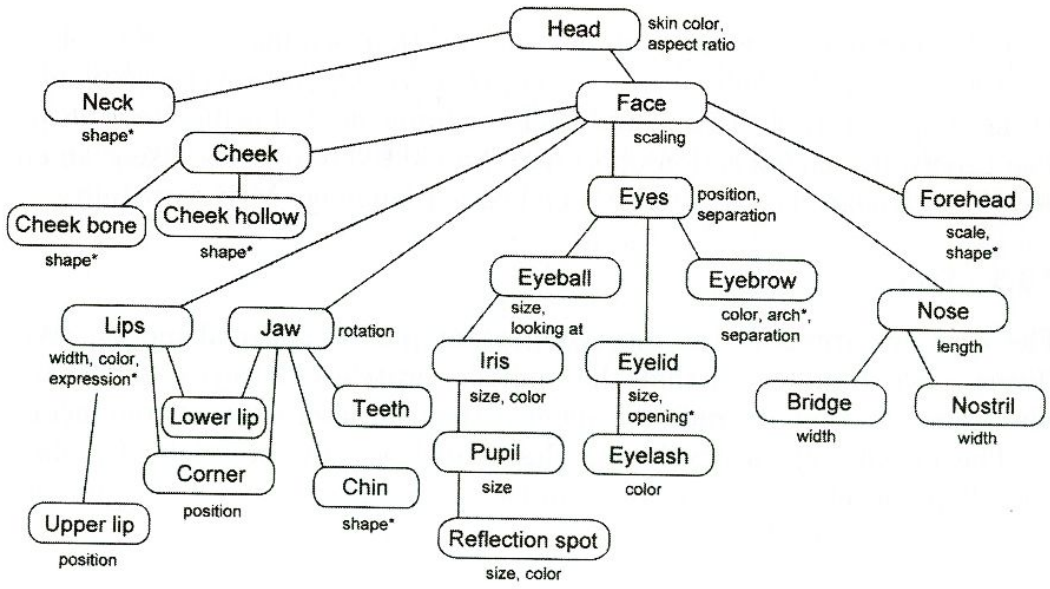
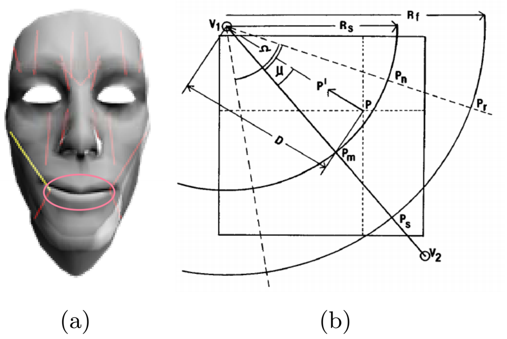


Figure 5.10: Structure of the parametrized model (from [38])

Waters muscle model

Waters [53] presents a muscle facial model based on FACs , where muscles are represented as *muscle vectors*, which describe the effect of muscle contraction on the geometry of the skin surface - which vertices will be affected by the muscle, and how it will affect them. The model approximates the work of linear muscles, one sheet muscle (frontalis - forehead muscle) and one sphincter (orbicularis oris - muscle around the mouth). The muscles are scatched in Figure 5.11(a).

Linear muscles are composed of a point of attachment and direction (Figure 5.11(b), sheet muscle from a line of attachment and direction and sphincter is composed of a center point and two semi-axes defining an ellipse. Each muscle affects the adjacent tissue (geometry) in a predefined radius (R_f), attracting the nodes toward its origin.



(a) (b)

Figure 5.11: Waters muscle model: (a) used muscles on the face (b) muscle vector model for the linear muscle (2D version) (from [53])

The approach needs the mesh to be as regular as possible, otherwise polygonal intersections may occur during the animation.

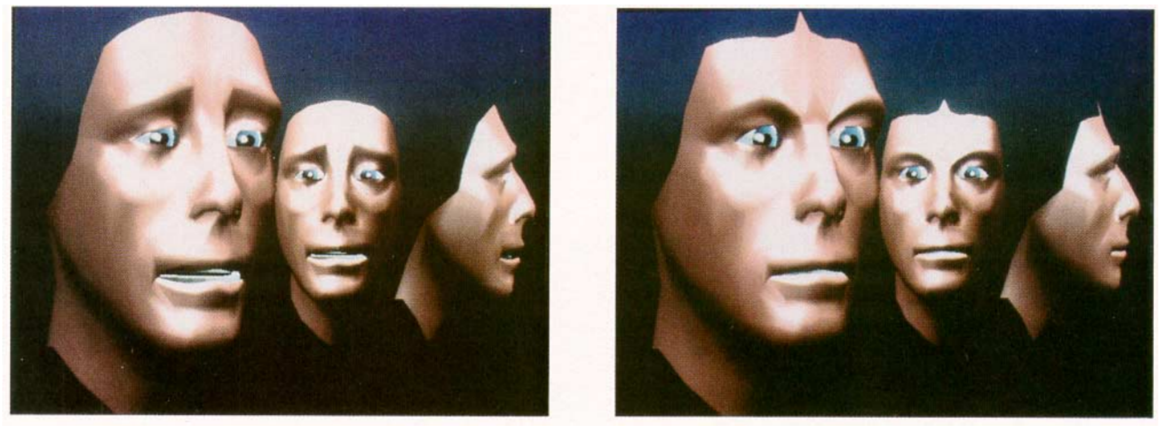
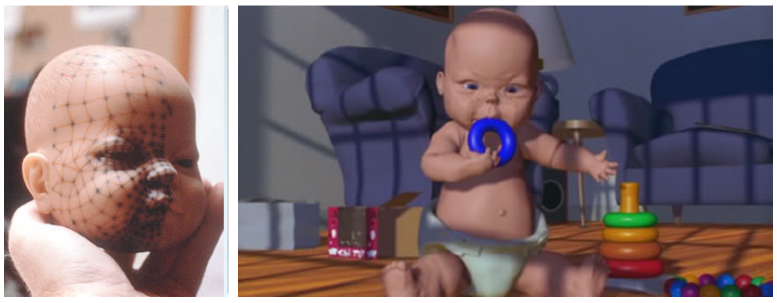


Figure 5.12: Fear (left) and anger (right) emotions produced using the Waters muscle model(from [53])

The child's face in the Pixar's "Tin Toy" (1998) was animated using the Waters model (see Figure 5.13). Tin Toy was the first character that was animated digitally.

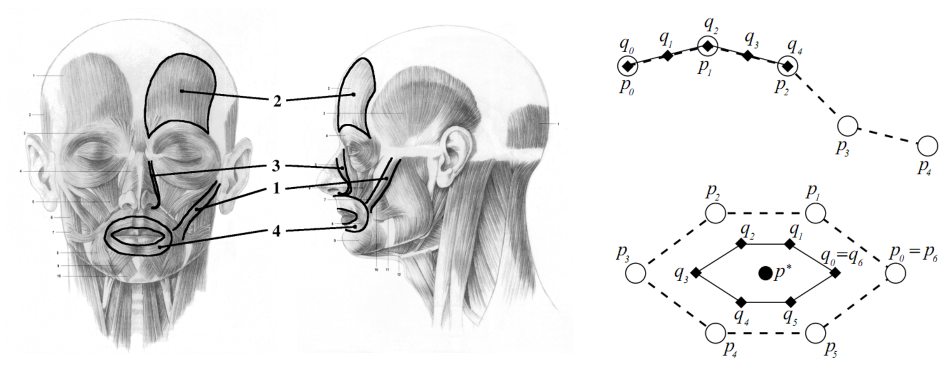


(a) (b)

Figure 5.13: Pixar's Tin Toy: (a) clay model for scanning purposes (b) example expression

Kähler's muscle model

Kähler et al. [26] presents a muscle model using three layers: a skin, muscle and bone layer. Their muscle model contains linear, sheet, curved and sphincter muscles (see Figure 5.14(a)). The muscles are represented by control polygons $P = \{p_i\}$, where each segment is assigned an ellipsoidal shape. The contraction of a muscle is defined by a contraction value $c \in [0, 1]$, where $c = 0$ means no contraction and $c = 1$ full contraction. The contraction of a linear muscle is computed by shrinking the segments and mapping them to the original polygon, the segments in sphincter muscles are contracted toward a center point (see Figure 5.14(b)). During the contraction, the real muscles get thicker - this behavior is achieved by scaling the height of each muscle segment according to its position in the control polygon: the center parts get thicker than the edge parts.


2
3
1
4
q_0 q_1 q_2 q_3 q_4
p_0 p_1 p_2 p_3 p_4
p_2 p_1
q_2 q_1
p_3 q_3 p^* $q_0=q_6$ $p_0=p_6$
q_4 q_5
p_4 p_5

(a) (b)

Figure 5.14: (a) types of muscles used in Kähler's muscle model and a muscle fiber with control polygon $P = \{p_i\}$ and (b) muscle contraction of a linear and sphincter muscle fiber (from [26]).

The muscles can cross each other, and also be attached to each other. Attached muscles are stored in so-called *constraint groups*. After the computation of muscle contractions, the control points of attached muscles are moved to maintain the original distances.

No muscle model is used to define the muscles, the definition is done manually by the user. The user specifies at least four points on the grid, and a center point for sphincter muscles. The muscle grid is automatically refined to fit the geometry of the model.

The skin and tissue simulation is done using a spring mesh, with a surface nodes attached either to the bone layer or to the muscle fiber.

5.3.2 Data-driven parametrization

Data-driven parametrization does not define the parameters explicitly or manually (such as the shape interpolation), but derive the parametrization using statistical methods applied to some large amount of input data. The most often statistical method is principal component analysis (PCA). The input data for PCA may be for example motion capture data. PCA extracts parameters common to the input set of faces (i.e., one face with different expressions). The principle components become the "conformational" parameters that specify a concrete face expression. This type of parametrization is used in Blanz and Vetter work [7] that has already been discussed in 5.1.5.

Chapter 6

Personality factors

Since our work concentrates on modeling faces for personality perception research, this chapter provides a brief introduction to the psychological part of the work.

There has been a lot of psychological research concerning human personality, its measuring and description. The goal of this chapter is not to give a complete overview in this area, however, several interesting topics concerning our work will be mentioned.

In 1940s, R. Cattell's research [10] showed that human personality can be described by sixteen major factors. The presence or lack of each factor is expressed by certain personality features, called *high* and *low range descriptors* (table listing the factors along with high and low range descriptors can be found in Appendix C). To evaluate a person's personality factors, Catell developed so-called *16PF Personality Questionnaire* (16PF stands for "16 personality factors"). Catell's questionnaire is still internationally used in schools and colleges, clinical and counseling settings, in career counseling and employee selection and development, as well as in basic personality research.

The test was revised several times (mostly to simplify and update the language), its most recent edition is from 1993. Its fifth edition contains 185 multiple-choice items which are written at a reading level of an average fifth-grader. Each item has a three-choice answer format with the middle choice being a question mark. Its content asks simple questions about daily behavior, interests, and opinions, asks question on concrete situations, does not want the test-taker to self-assess himself. An example questions from the questionnaire are shown below (from [11]):

- When I find myself in a boring situation, I usually "tune out" and daydream about other things. *(a. true; b. ?; c. false)*
- In talking to a friend, I tend to: *(a. let my feelings show; b. ?; c. keep my feelings to myself)*

The test provides scores on 16 primary personality scales and five global personality scales, all of which are bi-polar (meaning that both ends of each scale have a distinct, meaningful definition). The 16PF test was designed for adults of at least age 16 and older, but there are also parallel tests for various younger age ranges (e.g., the 16PF Adolescent Personality Questionnaire [42]).

In 1961, E. Tupes and R. Christal [49] discovered five global (second-order) personality factors by analyzing the factors described by Catell. These factors were later called the *Big Five factors*. The Catell's questionnaire can be used to find out the primary and the secondary personality factors of the given person as well.

The Big Five personality factors are:

Openness An open person is inventive and curious, emotional, has appreciation for art and sense of adventure. A person lacking openness is cautious and conservative.

Conscientiousness A conscientious person is self-disciplined, organized and has a planned behavior. A conscienceless person is easy-going and careless.

Extroversion An extrovert is outgoing, full of energy, positive emotions, with a tendency to seek stimulation in the company of others. An introvert is shy and withdrawn.

Agreeableness An agreeable person is friendly and compassionate, however an averse person is competitive and suspicious.

Neuroticism A neurotic is sensitive and nervous, lack of neuroticism leads to being secure and confident.

Chapter 7

Morphing for face averaging

This section describes our work done in cooperation with V. Pivoňková from the Faculty of Humanities, Charles University in Prague, on a project concerning human faces. The aim of the project is to study human perception of 3D face composites. Our task in the project was to compute average faces of given sets of scanned data.

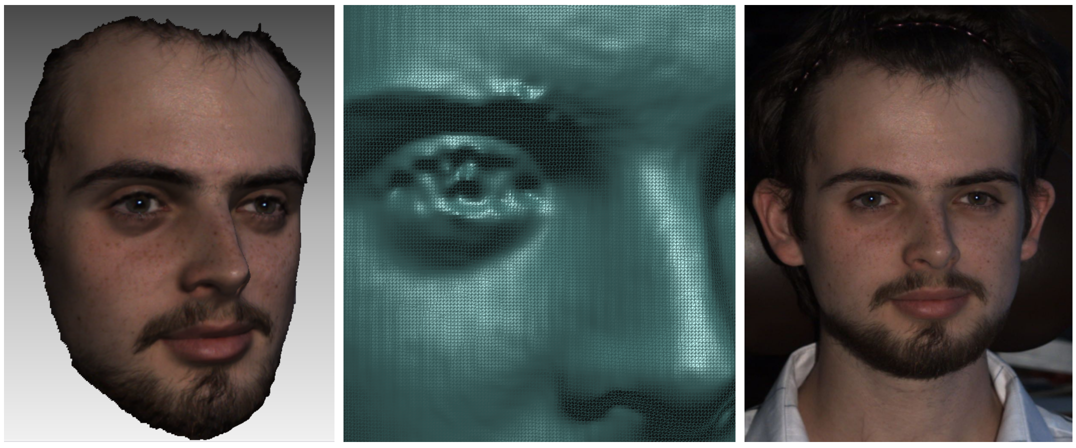
The background of the project will be described in Section 7.1, and our solution for the face average computation will be described in Section 7.2.

7.1 Composite faces for face perception research

In the first part of their research [28], Věra Pivoňková and her team worked with photographs of human faces. They concentrated on face perception in neutral state, i.e., they did not include expressions in their research. In their test, they selected only several factors from the Catell's 16 personality factors: warmth, reasoning, emotional stability, dominance, liveliness, social boldness, abstractedness and privateness.

In the first step, they took photographs of 138 females and 80 males aged 19-29. They used Catell's 16PF questionnaire (see Section 6) to determine the personality factors of the persons. Images of 15 men and women scoring highest and lowest on each factor were selected and morphed into one, composite image. Composite images were used to study if the personality ratings in the composites were more accurate then in the original photographs. The results were positive, confirming the presumptions that the composite images contain all the physical traits common to the individuals, however, the individual facial characteristics are minimized. Therefore, they can be used to separate the physical traits common to the personality factor being tested.

In the next part of the research, they wanted to repeat the process with 3D models. About the same number of the same age group was scanned, producing an amount of dense triangle meshes (each mesh had about 40,000 vertices). The faces were scanned only from one direction, which resulted in open meshes, containing only the face (without hair and ears). The texture was in the form of a simple photograph. They continued with the same process as with the photographs. First, they evaluated the perception of the original data and chosed 15 men and women scoring highest and lowest on each factor. They asked us to create a composite from each group. How we handled this task is described in the following section.



(a) (b) (c)

Figure 7.1: Input data for computation: (a) a face mesh with texture (b) detail of the face mesh (c) texture is in the form of a photograph.

7.2 Average face computation

As was described in the previous section, the input data for the face average computation are 15 open meshes of about 40,000 vertices, with a photograph as a texture (see Figure 7.1). The goal is to compute an average face mesh with an average texture. Computing an average of two faces can be defined as computing a face morph at a time $\frac{t}{2}$. Therefore, traditional morphing techniques may be used for such a problem. Computation of an average face R of n faces $M = \bigcup_i M_i, i = \{0, \ldots, n-1\}$ is in fact a multimorphing problem, where the weights of the base meshes the same:

$$R = \sum_{i=0}^{n-1} w_i M_i \qquad \sum_{i=0}^{n-1} w_i = 1 \qquad w_i = w_j \; \forall (i, j)$$

Therefore, traditional multimorphing techniques may be used for our problem. However, since we have some knowledge about the input data, we can simplify the techniques used or adjust them to perform better for our type of data.

Solving a multimorphing problem usually means to:

- compute the vertex-to-vertex correspondence between the meshes
- compute the target mesh by computed the weighted sum of the corresponding vertices.

The crucial part in the process is the correspondence computation. To be able to compute a vertex-to-vertex correspondence, we need to refine input meshes so that they have the same number of vertices and the same connectivity. We already tested two techniques for such a refinement in my bachelor thesis [32]: Alexa topology merging technique [3] and a remeshing technique of Kraewoy and Scheffer [29]. Both methods not only refine meshes to have the same topology, but also compute the vertex-to-vertex correspondence.

The topology merging method showed unsuitable for a larger number of input objects. To discuss the reasons, let us recall how the method works (see Section 3.2. The method establishes the correspondence between two meshes by inserting vertices and edges into both of them. As a result, the two meshes have the same topology. To ensure this property for all meshes, this process has to be done incrementally by twos. After merging two objects with n_1 and n_2 vertices, the resulting object has $n_1 + n_2$ vertices. Merging m objects, where the i-th object has n_i vertices, the resulting object has $\sum_{i=0}^{m} n_i$ vertices. It can be seen that the method as it is has a severe problem with increasing number of vertices, and so it is not very suitable for a larger number of input objects. Some kind of mesh decimation could be included in intermediate steps, however, there would be problems with preserving the topology among the objects.

The remeshing method might seem as a solution here. The number of vertices and edges of the resulting objects does not depend on the input objects, but on a so-called *supermesh*. The idea is to use an arbitrary object as a supermesh, and remesh all input objects so that they have the same topology as the supermesh. The process can be imagined as deforming the supermesh so that its shape approximates the input object. As all the input objects are approximated by the same object (supermesh), the mutual correspondence is given by the supermesh vertices: the corresponding vertices have the same origin in the supermesh. For a detailed description of the process and a discussion about the quality of the approximation, see Section 7.2.1.

The whole algorithm is sketched in Figure 7.2. In the first step, the meshes are projected onto the common parametric domain, in this case a plane (step 1). Since the data are only of a height map character (2D data + height information at each vertex), the projection is realized by simply omitting the height information. A simple feature correspondence is done by using user-selected correspondence points on each face (step 2) and the warping method on the projection plane (step 3, more in Section 7.2.2). After the warping, we use the remeshing method to refine the meshes to have the same topology and to compute the vertex-to-vertex correspondence among the meshes (step 4, more in 7.2.1). When the corresponding vertices are known, a new mesh is computed, having the same topology as the others, and its vertices are computed as an average positions of the corresponding vertices of the input meshes (step 5). The texture is handled separately, using warping with the same user-selected correspondence points and blending (step 6-7, more in Section 7.2.2).

```
Input: input meshes $M = \bigcup_i M_i, i = \{0, \dots, n-1\}$ , $n$ photographs (textures) $m = \bigcup_i m_i$
Output: the average mesh $S$ and its average texture $s$
The algorithm:
    1. Project each mesh from $M$ on the plane - obtain $M^{2D} = \bigcup_i M_i^{2D}$.
    2. Obtain the correspondence of feature points (by the user/from the file).
    3. Warp $M_i^{2D}, i = \{1, \dots, n-1\}$ using the feature point correspondence information.
    4. Remesh each mesh $M_i$ using a position information from $M_i^{2D}$ (more in Section 7.2.1)
       - obtain remeshed meshes $R = \bigcup_i R_i$.
    5. Compute $S$, sharing the topology with $R_i, \forall i$. Compute each vertex of $S$ as an average
       of corresponding vertices of all meshes from $R$.
    6. Warp $m$ using the feature point correspondence information.
    7. Compute $s$ as a blend (average) of $m$.
```

Figure 7.2: Pseudocode of the algorithm.

7.2.1 Remeshing

Remeshing method is a way to refine arbitrary number of input meshes so that they have the same topology, and to set up the vertex-to-vertex correspondence between the meshes. This section describes the method in detail along with its particular use for our application.

The main idea of remeshing is to create a mesh (called *supermesh*) and deform it to the shape of the input mesh. The deformed supermesh is then considered as the remeshed input. If we use the same supermesh to remesh more input meshes, they will have the same number of vertices and the same connectivity. What more, the vertex-to-vertex correspondence will be established, since we know the exact location of each vertex of the supermesh in all remeshed meshes.

The remeshing algorithm is sketched in Figure 7.3. The process consists of several steps. In the first step, the supermesh S is created. The supermesh may be an arbitrary mesh, however, it should be chosen to have its shape as close as possible to the source mesh M. In the second step, the source mesh M and the supermesh S are projected to a common parametric domain. From [4], a typical parametric domain is a unit sphere for closed meshes and a unit disk for open meshes (see Figure 7.4). In our application, we used projection to a plane along with warping method (see Section 7.2.2) to include user-defined feature correspondence.

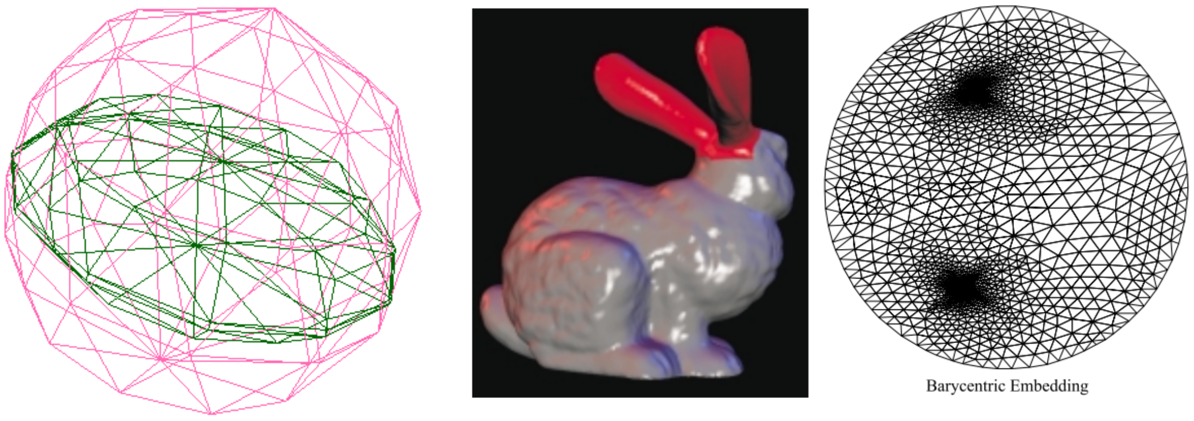
As the next step, for each vertex from S, we locate the triangle from M in which the vertex is located, and using barycentric coordinates (Section 2) compute its position on the boundary of M. This way, we deform the whole shape of S to the shape of M, while the connectivity remains the same. An example process is shown in Figure 7.5, where a sphere is taken as a supermesh and a cone as an input mesh.

```
Input: input mesh $M$ , supermesh $S$
Output: mesh $S$ deformed to the shape of $M$
The algorithm:
  1. Project $M, S$ to the common parametric domain $\rightarrow M', S'$.
  2. For each vertex $v$ of $S'$:
       – Find in which (spheric) triangle $t'$ of $M'$ it lies.
       – Compute barycentric coordinates $\alpha, \beta, \gamma$ of $v$ in $t'$.
       – Using $\alpha, \beta, \gamma$, compute the position of $v$ in the original triangle $t$.
```

Figure 7.3: Remeshing algorithm.

Generally, the remeshing method only approximates the source mesh by the supermesh. The quality of the approximation depends on the shape and the number of the vertices of the supermesh. Obviously, the best object for remeshing would be the one with the same shape and topology. The more differences in shape or topology are between the two objects, the worse the approximation is. Appendix D shows several remeshing outputs when an approximated unit sphere is taken as a supermesh. It can be seen that remeshing has problems with sharp edges, such as the bottom edge of the cone (Figure D.2). More information about remeshing can be found in our previous work [32].

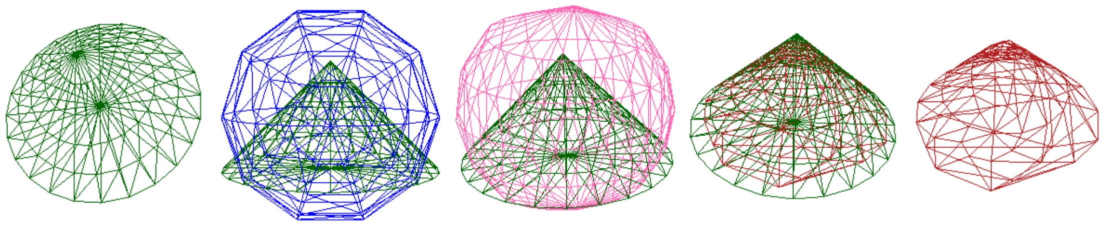
Despite its bad results for sharp edges, remeshing can be used in our application dealing with scanned faces without any remarkable defects, since the faces do not contain any sharp edge.



(a) sphere for closed meshes: input mesh (green), projected mesh (pink)

(b) unit disk for open meshes (from [4])

Figure 7.4: Typical parametric domains.



(a) source mesh

(b) supermesh (blue)

(c) source mesh parametrization (pink)

(d) resulting mesh along with the original

(e) resulting mesh

Figure 7.5: The remeshing process.

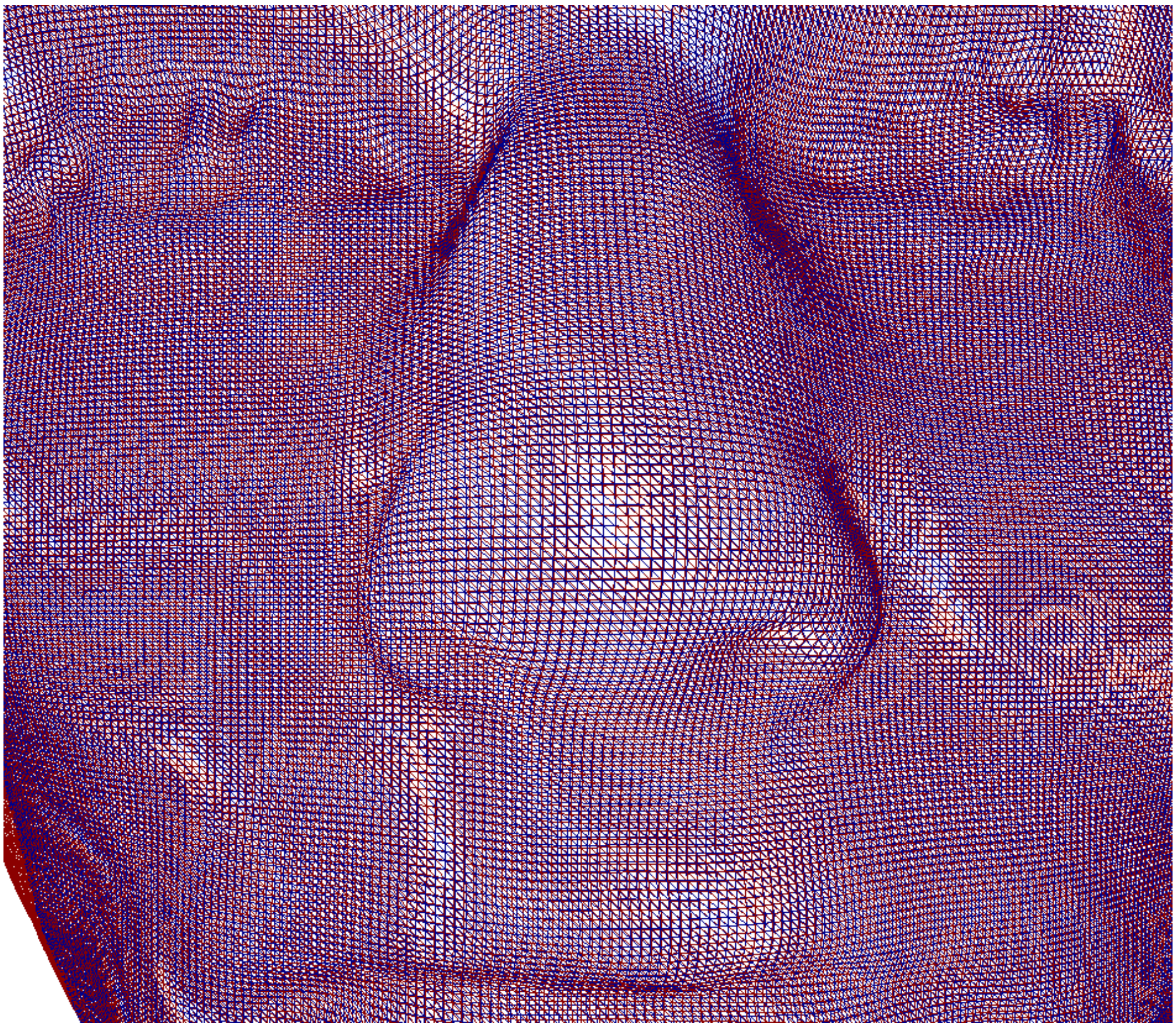


Figure 7.6: Remeshing a face mesh by another face mesh (detail): original (red), remeshed (blue).

Moreover, as the faces were scanned with the same density, they have very similar topology and number of vertices. Therefore, the best supermesh for remeshing is one of the faces themselves. Figure 7.6 shows an example of remeshing of one face by another one.

In the context of our application, the remeshing method has one drawback. As the meshes are projected on the plane and not on a unit disk, there are vertices of S which are not in any triangle of M. To have the same number of vertices for all meshes that will be remeshed by S, we cannot simply delete such vertices from S. We solved this problem by computing the position of such vertex v from its neighbors. For each neighboring vertex of v, its translation vector is computed from its previous and current position. An average vector is computed and used to move v.

Figure 7.7 shows a remeshed face (blue) along with those vertices, whose positions had to be computed from their neighbors. The original mesh (red) is not approximated completely, there are several triangles which are "deleted", because they do not contain any vertex from S.

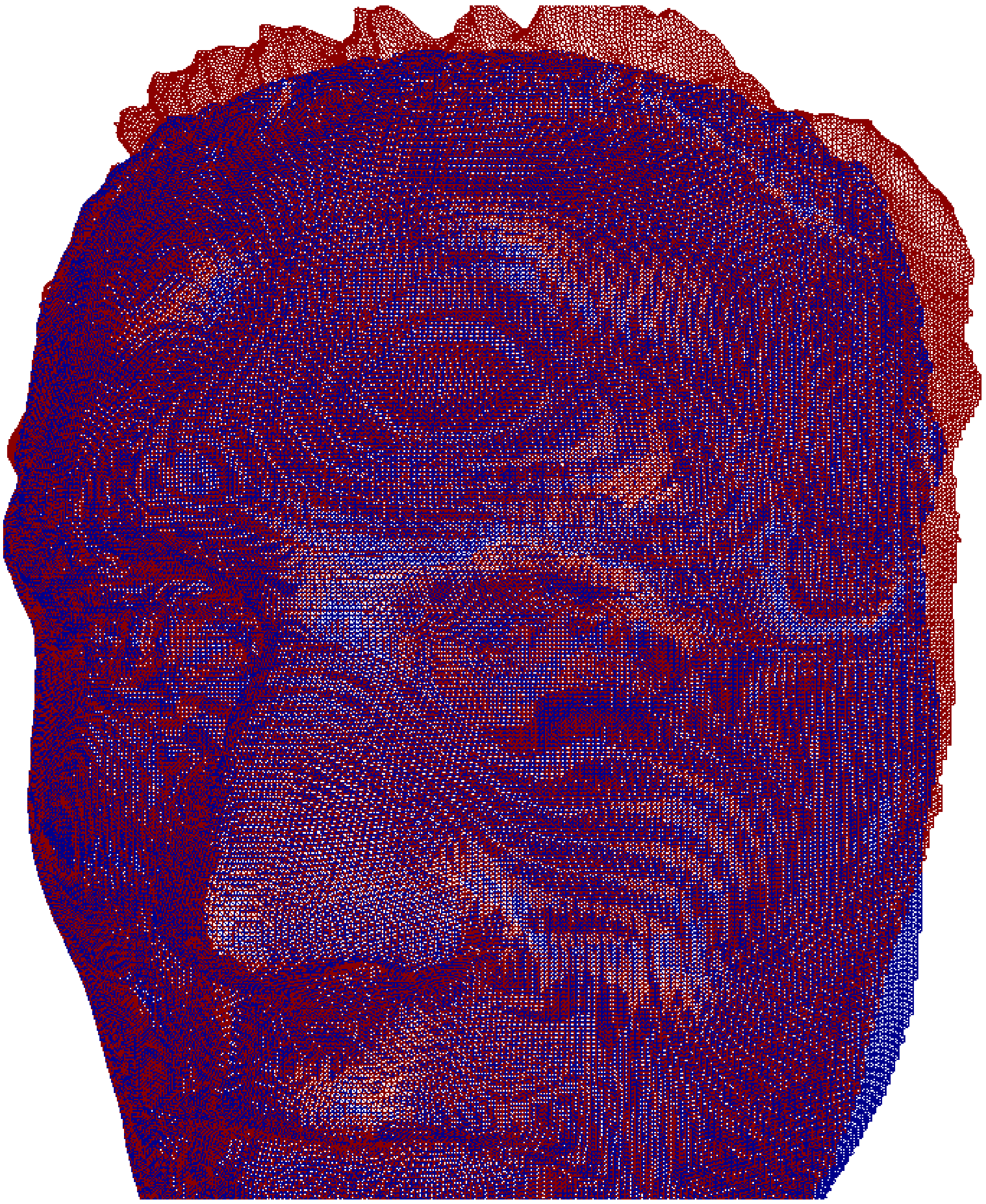


Figure 7.7: Remeshing a face mesh by another face mesh: problem with the borders; original (red), remeshed (blue).

7.2.2 Warping

For aligning the predefined feature correspondences on the texture and also on the mesh, we used the morphing technique based on image warping and color interpolation. Image warping uses geometric transformations to align predefined features, while color interpolation blends the colors at each pixel.

Wolberg in his book [57] shows two basic warping methods used for morphing images: mesh warping and feature-based warping. Mesh warping uses control mesh laid over the image. The user defines the positions of the mesh nodes in each image, and the result is computed as an interpolation between the nodes positions. Feature-based warping uses line segments instead of the control mesh. User defines corresponding line segments in each image, and the result is computed as if the line segments were magnets attracting the surrounding points.

Since feature based-warping needs only a definition of key features, not of the whole feature mesh as the mesh warping, it is more intuitive for the user. Therefore, we decided to use the feature-based warping in our approach. The following text describes this technique in detail.

There are two ways of computing a warped image: a forward mapping and a reverse mapping. The former scans through the source image pixels and copies them to the appropriate place in the destination image, the latter does the opposite: for each pixel in the target (warped) image, it finds a correct pixel in the source image. An important feature of the reverse mapping is that every pixel of the warped image is set to something appropriate, however, after a forward mapping, interpolation methods have to be used for the unset pixels.

The feature-based warping uses the reverse mapping technique. The next sections will show how the method computes the appropriate origin of a pixel in the target image. First, we will consider only one pair of corresponding line segments, and then an extension for an arbitrary number of corresponding line segments will be shown.

Feature-based warping for a pair of corresponding line segments

Let us denote (see Figure 7.8):

A	...	the source image
B	...	the target (warped) image
$P'Q'$	...	the user-defined line segment in A
PQ	...	the user-defined line segment in B
X_i	...	a pixel in B
X'_i	...	a pixel in A (origin of X)

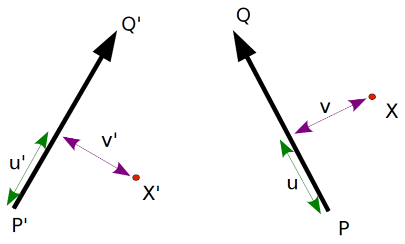
The following is done for each pixel X_i from B. Its position u along PQ and its distance from PQ is computed:

$$u = \frac{(X_i - P)(Q - P)}{||Q - P||^2}$$

$$v = \frac{(X_i - P)(Q - P)_\perp}{||Q - P||}$$

The position X'_i of the vertex X_i in the original image A is computed as:

$$X'_i = P' + u(Q' - P') + \frac{v(Q' - P')_\perp}{||Q' - P'||}$$



(a) A (b) B

Figure 7.8: Line segments for feature-based image warping (from [57]).

The color from this position is set as the color of the vertex X_i.

Feature-based warping for a multiple corresponding line segments

In the case of n predefined line segments, we compute u_i, v_i, X_i' for each of them, and according to the values computed, we get n translation vectors:

$$D_i^{\rightarrow} = X_i' - X_i$$

The result is computed as a weighted average of $D_i^{\rightarrow}$:

$$X_i' = X_i + \frac{\sum_{i=1}^{n} D_i^{\rightarrow} w_i}{\sum_{i=1}^{n} w_i}$$

where

$$w_i = \left(\frac{|Q_i' - P_i'|^p}{(a + dist)} \right)^b$$

The values a, b, p are constants which were according to [57] obtained from a number of performance tests.

Use of feature-based warping in our approach

In our approach, we use five line segments defined by the user, aligning eyebrows, eyes, nose and mouth (see Figure 7.9(a),7.9(b)). Since we have n images as the input, we use one of them as a reference image and warp the others to align all predefined line segments. The resulting average image is computed by simply averaging all images pixel by pixel. An example of the whole process for two images is shown in Figure 7.9.

We use image warping also for aligning the corresponding feature points in the projected meshes. The use is analogical, with a simple difference. The image warping used reverse mapping to avoid unset pixels in the warped image. However, such a mapping is inconvenient for meshes, where we need to modify the position of the vertices, and not the color information. Therefore, we use forward mapping for the meshes. The new position of each vertex is computed in the same way as was described in Section 7.2.2.

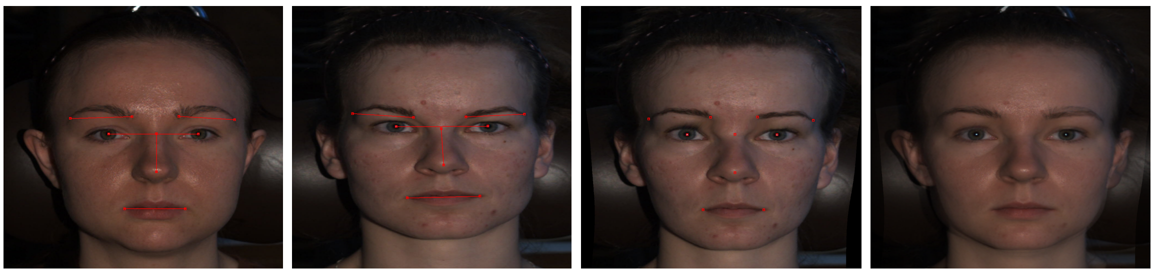
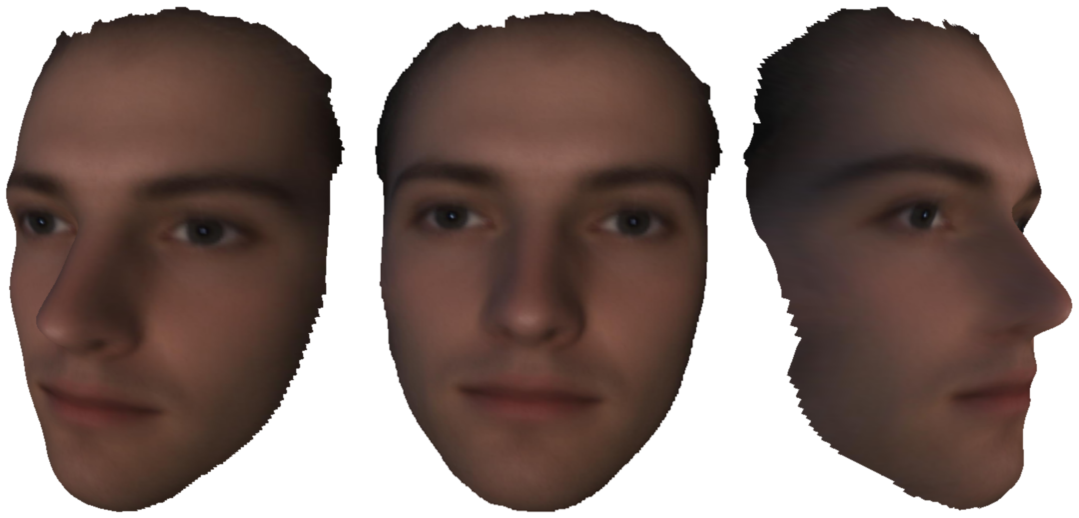


Figure 7.9: Averaging face images using feature-based warping (red dots mark corresponding points defined by the user, red line segments mark the resulting line segments used by warping).

7.2.3 Summary

This section showed a typical use of a multimorphing technique - creating a new model from several existing ones. In our case the new model (a face) was computed as an average of fifteen input faces, selected by V. Pivoňková from the scanned data, which they provided us for research purposes. Several face averages from the scanned data were created, an example of average man and woman (both made from 15 scans) is shown in Figure 7.10. V. Pivoňková was satisfied with the output, however, in our opinion, the method has one significant drawback. It does not deal with the geometry of the face in a way distinguishing the face features. It uses the corresponding points defined by the user, but this way, it only aligns the 2D position of the corresponding parts. A better way would have been to use some method using PCA (i.e. [7]).


(a) an average man

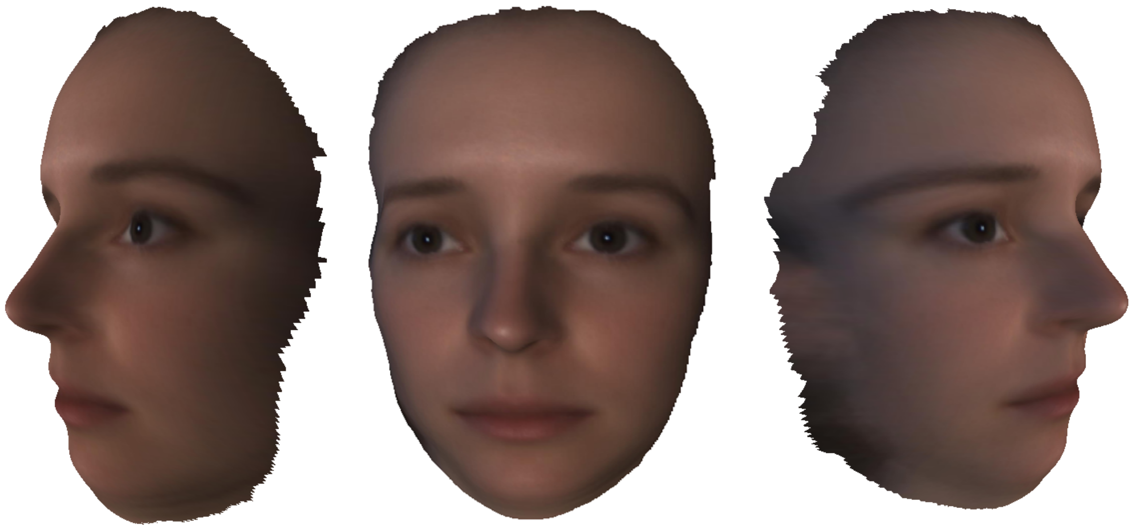

(b) an average woman

Figure 7.10: Average man and woman computed by our algorithm.

Chapter 8

Conclusion and Future work

In the first part of our work, we created a new polygon morphing algorithm with a growth-like behavior. This algorithm is simple, easy to implement and intuitive to use by an animator. The algorithm is suitable for objects, where the animator expects some growth-like animation. It is not appropriate for animating translation, rotation or a scale of an object, or for morphing similar objects, where some features need to be preserved during the animation.

In the following work, we decided to concentrate on morphing faces. We started to cooperate with V. Pivoňková, from the Faculty of Humanities, Charles University in Prague, who provided us a number of face scans for research purposes. We designed an algorithm computing an average face from an arbitrary number of input objects.

During our ten-days stay at Purdue University, we discussed the possibilities of our future cooperation with B. Beneš. Together, we decided to concentrate on morphing and deformation of facial models and came up with several ideas, which will be discussed in the following text.

Although there is lot of research done in the area of facial model animation and new models creation using multimorphing, there are still some challenges. The first challenge is as follows. In Section 5, we discussed Blanz and Vetter approach [7] for modeling new faces from the old ones, where PCA was used to define parameters intuitive for the user, such as width of the nose, skin color or age. The new model was created as a weighted average of the basis input models, where the parameter values posed as weights. However, we have not found any work trying to parametrize the model according to the human personality factors and not only the physical appearance.

As was discussed in Section 6, one's personality can be fully described by fifteen distinct personality factors. What is more, people can distinguish the personality traits from the person's face. Therefore, with our approach, we can offer the users to create a new models perceived as they were having the desired personality.

However, such an approach would lead to a parameterized model, where each parameter would represent a weight of some input model(s). In Section 5.2.1, we discussed that creating a new model by weighting the input models as a whole does not allow the user to simply modify only selected features of the face, since rising the weight of a model having this feature can influence other features which the user wanted to remain consistent. To obtain the desired behavior, the model was split into several regions. We decided to follow this idea also in our work, to bring more flexibility to the user, allowing him to create more complex, but still easily perceivable personality.

When the face is cut into pieces, the parts can be perceived differently than they were as a whole. Therefore, new questionnaires need to be created to evaluate the perception of the parts. To preserve the personality information, the parts need to be selected carefully, which will be realized by the psychologists from V. Pivoňková's team.

The part selection and the questionnaires will lead into the creation of a set of parts, where each personality factor will have two parts describing both extremes (the presence and the lack of the factor). In our parameter set, each parameter will influence the proportion of the two extremes of its assigned personality factor.

The parts will partially overlap, so the creation of a morph from weighted, overlapping parts will be a challenge as well. Also, we would like to create a model, where the user has the possibility to adjust the parameters of the parts, having an immediate response. So morphing of a part overlapping with a static surroundings will need to be solved.

Our work is interesting for the psychologists, since the research on facial parts perception is at its beginning, and such a method would allow various tests to be done in this area. Therefore, we expect to cooperate with the psychologists during our research.

Chapter A

Professional activities

Publications

Reviewed publications

[1] M Málková, I Kolingerová, and J Parus, *Core-based morphing algorithm for triangle meshes*, SIGRAD, 34:39-46, 2008, Stockholm.

[2] J Parus, I Kolingerová, and M Málková, *Multimorphing: A tool for shape synthesis and analysis*, Advances in Engineering Software, Elsevier, 40(5):323-33, 2009. Impact factor for the year 2008: 1,188.

[3] M Málková, J Parus, I Kolingerová, and B Beneš, *An intuitive polygon morphing*, The Visual Computer, Springer Verlag, 26(3):205-215, 2010. Impact factor for the year 2008: 1,061.

Student publications

[1] M Málková. *Morphing of geometrical objects in boundary representation.* Bachelor thesis, University of West Bohemia in Pilsen, 2006.

[2] M Málková. *Morphing of geometrical objects in boundary representation.* Diploma thesis, University of West Bohemia in Pilsen, 2008.

[3] M Málková. *A new core-based morphing algorithm for polygons.* Proceedings of the 11th Central European Seminar on Computer Graphics, pp. 39-46, 2007, Budměrice.

Stays abroad

- University of Bath, UK, February-June 2006
- University of Maribor, Slovenia, November 9-16, 2008
- Purdue University, Indiana, USA, October 19-29, 2009

Related talks

- M Málková: A new approach to morphing objects in boundary representation, University of Maribor, Slovenia, November 2008
- M Málková: Using Morphing for Averaging of Faces, University of West Bohemia in Pilsen, Czech Republic, April 2009
- M Málková: Dva pohledy do problematiky morphingu, VŠB Ostrava, Czech Republic, November 2009
- M Málková: Face Models, University of West Bohemia in Pilsen, Czech Republic, April 2010

Participation on scientific projects

- Triangulated Models for Haptic and Virtual Reality. Project leader Ivana Kolingerová. Funded by The Czech Science Foundation (GACR), project code 201/09/0097.
- Bilateral Cooperation in Computational Geometry Research for Visualization. Project leader Josef Kohout. Funded by The Ministry of Education, Youth and Sports (MSMT), project code KONTAKT 5/2005-06.
- CPG Center of Computer Graphics National Network of Fundamental Research Centers. Project leader Václav Skala. Funded by The Ministry of Education, Youth and Sports (MSMT), project code LC 06008.
- Modeling Natural Phenomena Using Computational Geometry. Project leader Ivana Kolingerová. Funded by The Ministry of Education, Youth and Sports (MSMT), project code KONTAKT 5076/2009-32.

Other scientific or academic activities

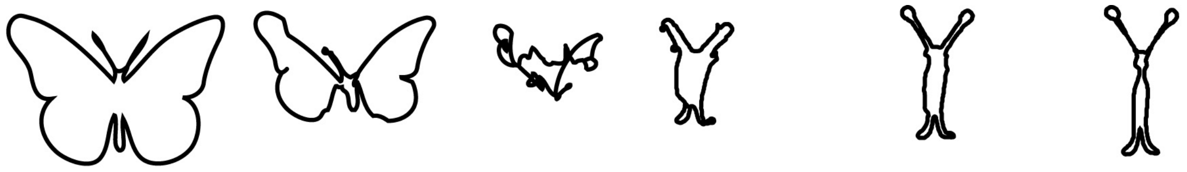
- Tool for computing average faces. Authorized software made freely available to the public. `http://www.kiv.zcu.cz/vyzkum/software/detail.html?id=58`

Chapter B

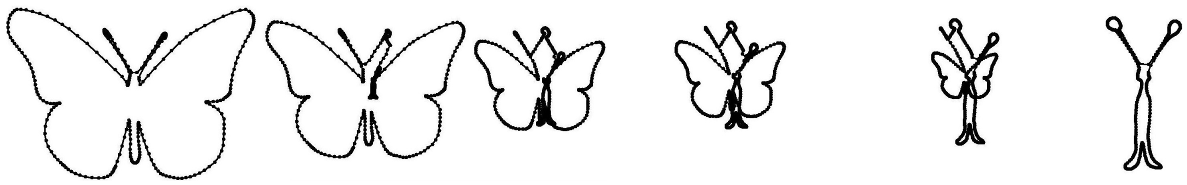
Core increment morphing - results

B.1 2D methods

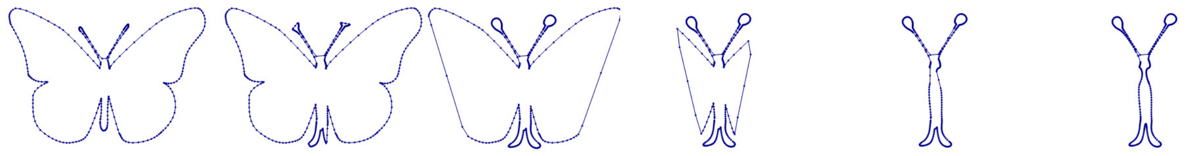
An animation showing the behavior of our algorithm is available at `http://graphics.zcu.cz/media/videos/malkova/2D-core.avi`.



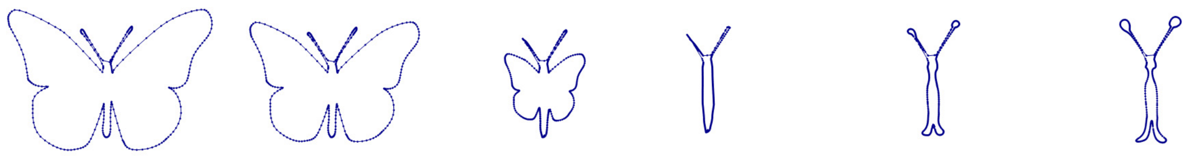
(a) Carmel and Cohen-Or algorithm
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/alien-cohen.avi`)



(b) Sederberg and Greenwood algorithm
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/alien-sed.avi`)

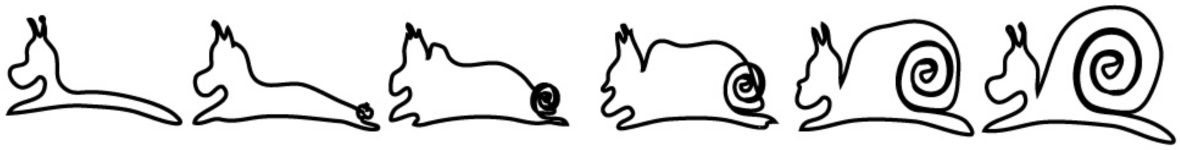


(c) Midpoint growing method
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/alien-mid.avi`)

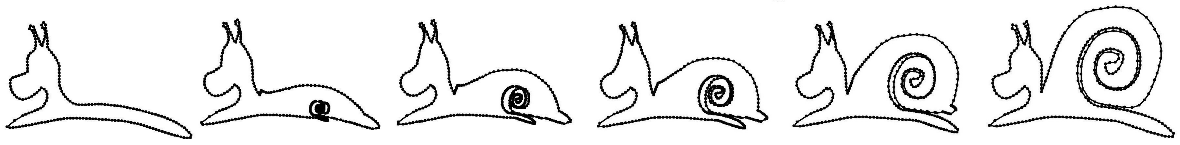


(d) Projection growing method
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/alien-proj.avi`)

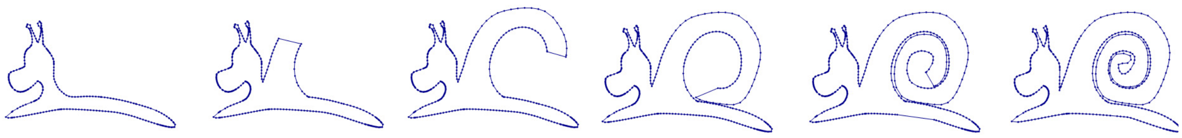
Figure B.1: Morphing between a butterfly and an alien



(a) Carmel and Cohen-Or algorithm
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/snail-cohen.avi`)



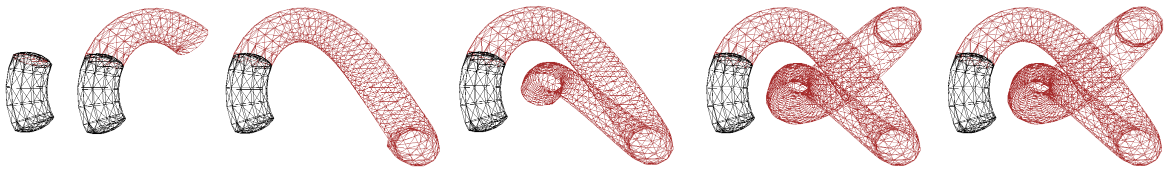
(b) Sederberg and Greenwood algorithm
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/snail-sed.avi`)



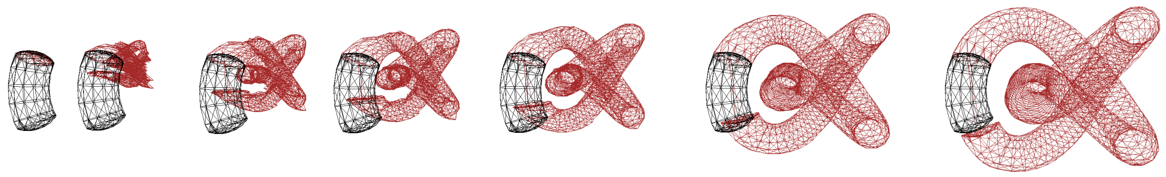
(c) Perimeter growing method
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/snail-per.avi`)

Figure B.2: Morphing between a snail and a slug

B.2 3D methods



(a) Perimeter growing method
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/knot-perimeter.avi`)



(b) Projection growing method
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/knot-proj.avi`)

Figure B.3: Morphing - growing knot

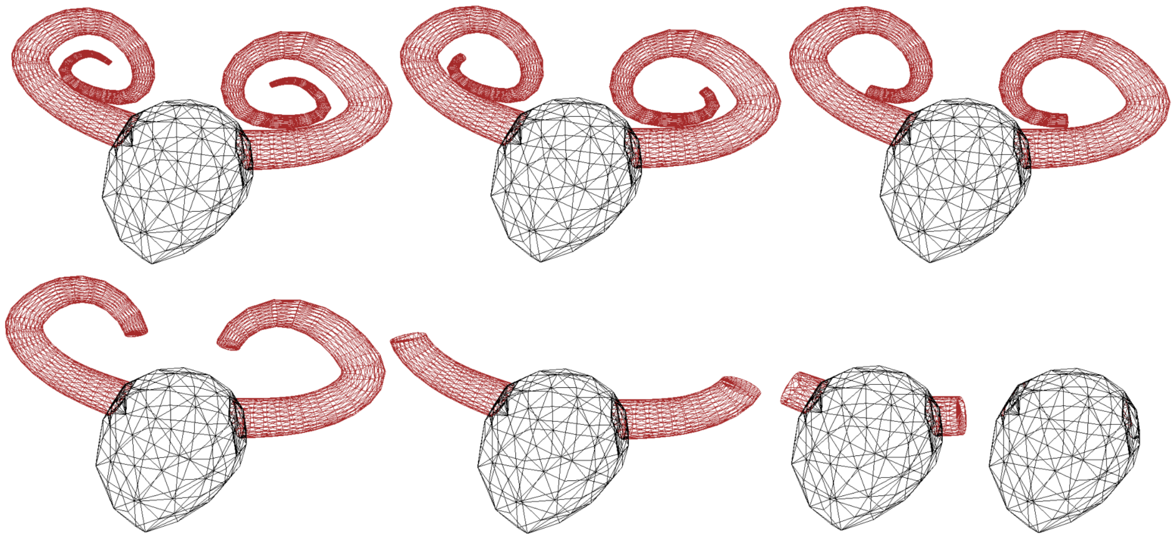


Figure B.4: Morphing - a head with and without horns (Perimeter growing method)
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/moufflon-perimeter.avi`)

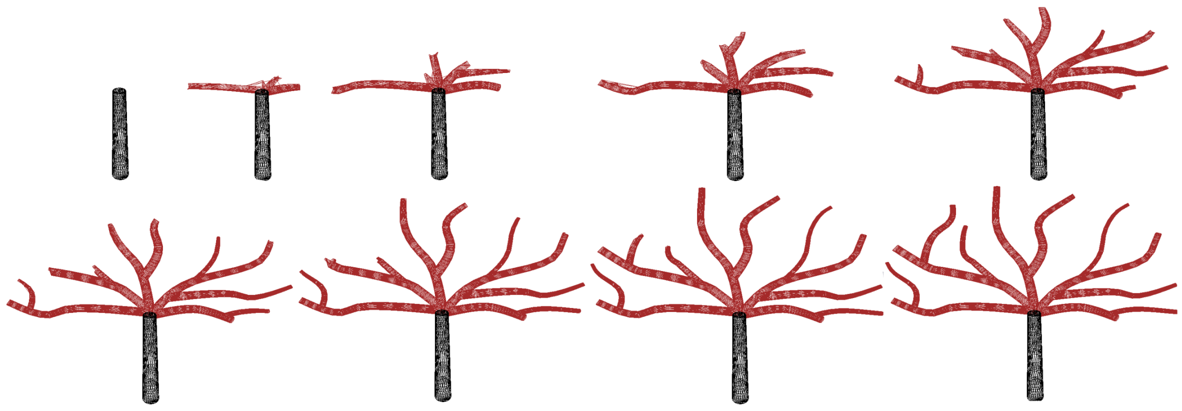


Figure B.5: Growing tree (Perimeter growing method)
(animation is available at `http://graphics.zcu.cz/media/videos/malkova/tree-perimeter.avi`)

Chapter C

Catell's 16 Personality Factors

Descriptors of Low Range	**Primary Factor**	**Descriptors of High Range**
Impersonal, distant, cool, reserved, detached, formal, aloof	Warmth	Warm, outgoing, attentive to others, kindly, easy-going, participating, likes people
Concrete thinking, lower general mental capacity, less intelligent, unable to handle abstract problems	Reasoning	Abstract-thinking, more intelligent, bright, higher general mental capacity, fast learner
Reactive emotionally, changeable, affected by feelings, emotionally less stable, easily upset	Emotional Stability	Emotionally stable, adaptive, mature, faces reality calmly
Deferential, cooperative, avoids conflict, submissive, humble, obedient, easily led, docile, accommodating	Dominance	Dominant, forceful, assertive, aggressive, competitive, stubborn, bossy
Serious, restrained, prudent, taciturn, introspective, silent	Liveliness	Lively, animated, spontaneous, enthusiastic, happy go lucky, cheerful, expressive, impulsive
Expedient, nonconforming, disregards rules, self indulgent	Rule-Consciousness	Rule-conscious, dutiful, conscientious, conforming, moralistic, staid, rule bound
Shy, threat-sensitive, timid, hesitant, intimidated	Social Boldness	Socially bold, venturesome, thick skinned, uninhibited
Utilitarian, objective, unsentimental, tough minded, self-reliant, no-nonsense, rough	Sensitivity	Sensitive, aesthetic, sentimental, tender minded, intuitive, refined
Trusting, unsuspecting, accepting, unconditional, easy	Vigilance	Vigilant, suspicious, skeptical, distrustful, oppositional
Grounded, practical, prosaic, solution oriented, steady, conventional	Abstractedness	Abstract, imaginative, absent minded, impractical, absorbed in ideas
Forthright, genuine, artless, open, guileless, naive, unpretentious, involved	Privateness	Private, discreet, nondisclosing, shrewd, polished, worldly, astute, diplomatic
Self-Assured, unworried, complacent, secure, free of guilt, confident, self satisfied	Apprehension	Apprehensive, self doubting, worried, guilt prone, insecure, worrying, self blaming
Traditional, attached to familiar, conservative, respecting traditional ideas	Openness to Change	Open to change, experimental, liberal, analytical, critical, free thinking, flexibility
Group-oriented, affiliative, a joiner and follower dependent	Self-Reliance	Self-reliant, solitary, resourceful, individualistic, self sufficient
Tolerates disorder, unexacting, flexible, undisciplined, lax, self-conflict, impulsive, careless of social rules, uncontrolled	Perfectionism	Perfectionistic, organized, compulsive, self-disciplined, socially precise, exacting will power, control, self-sentimental
Relaxed, placid, tranquil, torpid, patient, composed low drive	Tension	Tense, high energy, impatient, driven, frustrated, over wrought, time driven.

Table C.1: Primary factors and descriptors in Cattell's 16 Personality Factor Model (from [13])

Chapter D

Remeshing - results

The results from the remeshing method described in Section 7.2.1. The supermesh is a sphere approximated with a selected number of vertices. Figures D.1, D.2 are from our previous work [32].

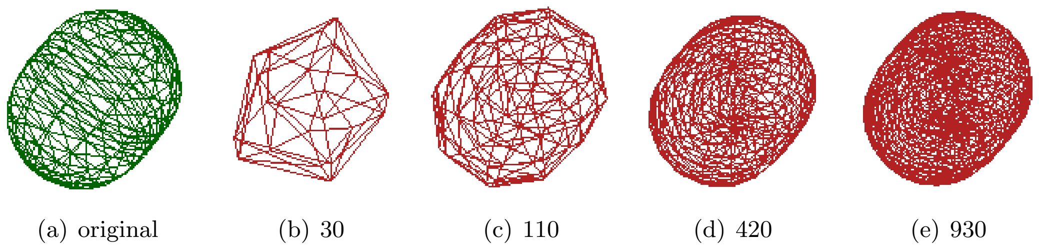


Figure D.1: Remeshing a capsule with a sphere with a given number of vertices: original (green), remeshed object (red).

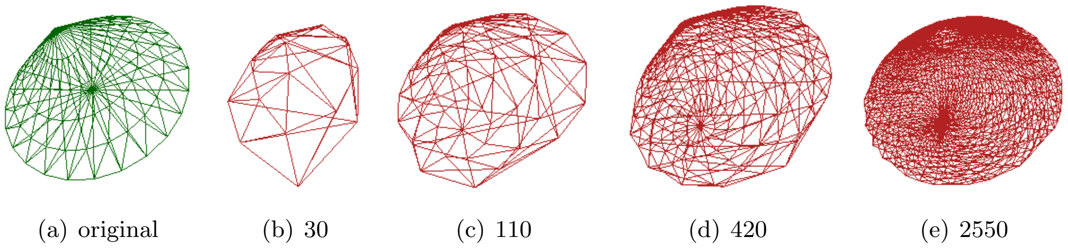


Figure D.2: Remeshing a cone with a sphere with a given number of vertices: original (green), remeshed object (red). Note that the method has problems with sharp edges.

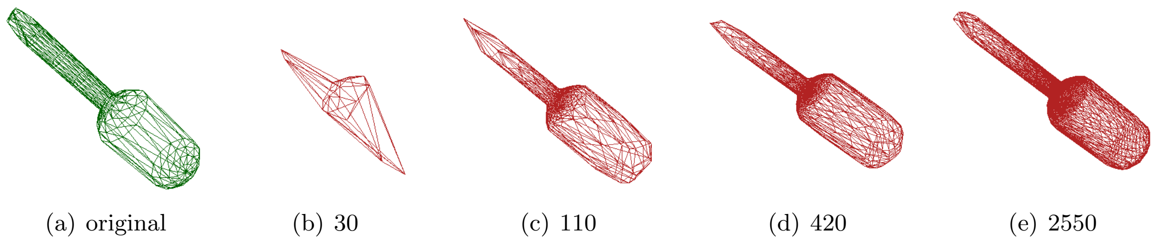


Figure D.3: Remeshing a screw driver with a sphere with a given number of vertices: original (green), remeshed object (red).

Bibliography

[1] http://www.ghost3d.com/g3d_mg_tutorial_digitize_head_1.htm. [16.3.2010].

[2] M. Ahn, S. Lee, and H. Seidel. Connectivity transformation for mesh metamorphosis. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 75–82, New York, NY, USA, 2004. ACM.

[3] M. Alexa. Merging polyhedral shapes with scattered features. *The Visual Computer*, 16(1):26–37, 2000.

[4] M. Alexa. Recent advances in mesh morphing. *Computer Graphics Forum*, 21(2):173–197, 2002. citeseer.ist.psu.edu/alexa02recent.html.

[5] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 157–164. ACM Press/Addison-Wesley Publishing Co., 2000.

[6] J. R. Bergen and R. Hingorani. Hierarchical motion-based frame rate conversion. Technical report, David Sarnoff Research Center Princeton NJ 08540, 1990.

[7] V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH'99 Conference Proceedings*, pages 187–194, 1999.

[8] P. J. Burt and E. H. Adelson. Merging images through pattern decomposition. In *Applications of Digital Image Processing VIII, number 575*, pages 173–181, 1985.

[9] E. Carmel and D. Cohen-Or. Warp-guided object-space morphing. *The Visual Computer*, 13:465–478, 1997.

[10] R. B. Catell. *The description and measurement of personality.* New York: Harcourt, Brace, & World, 1946.

[11] H. E. P. Cattell and J. M. Schuerger. *Essentials of 16PF Assessment.* John Wiley & Sons, Inc., 2003.

[12] D. Cohen-Or, A. Solomovic, and D. Levin. Three-dimensional distance field metamorphosis. *ACM Transactions on Graphics*, 17:116–141, 1998.

[13] S.R. Conn and M.L. Rieke. The 16pf fifth edition technical manual. Technical report, Champaign, IL: Institute for Personality and Ability Testing, Inc., 1994.

[14] D. DeCarlo, D. Metaxas, and M. Stone. An anthropometric face model using variational techniques. In *SIGGRAPH '98, Computer Graphics Proceedings*, pages 67–74, 1998.

[15] P. Ekman and W. Friesen. Facial action coding system. *Consulting Psychologists Press*, 1978.

[16] A. B. Ekoule, F. C. Peyrin, and C. L. Odet. A triangulation algorithm from arbitrary shaped multiple planar contours. *ACM Trans. Graph.*, 10(2):182–199, 1991.

[17] L. Farkas. *Anthropometry of the Head and Face.* Raven Press, 1994.

[18] J. Gomes, L. Darsa, B. Costa, and L. Velho. *Warping and morphing of graphical objects.* Morgan Kaufmann Publishers, Inc., 1999.

[19] S. Hanke, T. Ottmann, and S. Schuierer. The edge-flipping distance of triangulations. *j-jucs*, 2(8):570–579, aug 1996.

[20] H. Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, New York, NY, USA, 1996. ACM.

[21] Yuxiao Hu, Dalong Jiang, Shuicheng Yan, Lei Zhang, and Hongjiang Zhang. Automatic 3d reconstruction for face recognition. In *In Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, pages 843–848, 2004.

[22] Cyberware Laboratory Inc. 4020/rgb 3d scanner with color digitizer, 1990. Monterey, CA.

[23] J. K. Johnstone and X. Wu. Morphing two polygons into one. *40th Annual Southeast ACM Conference*, 2002.

[24] I. T. Jollife. *Principal Component Analysis.* Springer-Verlag, New York, 1986.

[25] P Joshi, W Tien, M Desbrun, and F Pighin. Learning controls for blendshape-based realistic facial animation. *Data-Driven 3D Facial Animation*, 2007.

[26] Kolja Kähler, Jörg Haber, and Hans-Peter Seidel. Geometry-based muscle modeling for facial animation. In *GRIN'01: No description on Graphics interface 2001*, pages 37–46, Toronto, Ont., Canada, Canada, 2001. Canadian Information Processing Society.

[27] J. R. Kent and R. E. Carlson, W. E.and Parent. Shape transformation for polyhedral objects. *Computer Graphics*, 26(2):47–54, 1992.

[28] A. Kotrcova, V. Pivonkova, J. Lindova, J. Flegr, and J. Havlicek. Accuracy of personality judgments made on the basis of composite pictures. In *XIX Biennial Conference of the International Society for Human Ethology*, 2008.

[29] V. Kraevoy and A. Sheffer. Cross-parameterization and compatible remeshing of 3d models. *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, 23(3):861–869, 2004.

[30] Y. Lee, D. Terzopoulos, and K. Waters. Realistic modeling for facial animation. *Computer Graphics*, pages 55–62, 1995.

[31] D. Levin. Multidimensional reconstruction by set-valued approximation. *Clarendon Press Institute Of Mathematics And Its Applications Conference Series, Algorithms for approximation*, pages 421–431, 1987.

[32] M. Málková. Morphing of geometrical objects in boundary representation. Bachelor thesis, University of West Bohemia in Pilsen, 2006.

[33] M. Málková. Core-based morphing algorithm for triangle meshes. *SIGRAD 2008*, 34:39–46, 2008.

[34] M. Málková. Morphing of geometrical objects in boundary representation. Master's thesis, University of West Bohemia in Pilsen, 2008.

[35] M Málková, J Parus, I Kolingerová, and B Beneš. An intuitive polygon morphing. *Visual Computer*, 26(3):205–215, 2010.

[36] Wolfgang Müller and Marc Alexa. Using morphing for information visualization. In *NPIV '98: Proceedings of the 1998 workshop on New paradigms in information visualization and manipulation*, pages 49–52, New York, NY, USA, 1998. ACM.

[37] I. Pandzic and R. Forchheimer. *MPEG-4 Facial Animation: The Standard, Implementation and Applications.* Wiley, 2002.

[38] F. Parke. *A Parametric Model for Human Faces.* PhD thesis, University of Utah, 1974.

[39] F. I. Parke and K. Waters. *Computer Facial Animation.* AK Peters Ltd, 2008.

[40] J. Parus. *Morphing 3D geometrical objects in boundary representation.* PhD thesis, University of West Bohemia in Pilsen, 2009.

[41] J Parus, I Kolingerová, and M Málková. Multimorphing: A tool for shape synthesis and analysis. *Advances in Engineering Software*, 40(5):323–333, 2009.

[42] J.M. Schuerger. 16pf adolescent personality questionnaire. *Champaign, IL: Institute for Personality and Ability Testing*, 2001.

[43] T. W. Sederberg, P. Gao, and G. Mu H. Wang. 2-d shape blending: An intrinsic solution to the vertex path problem. *Computer Graphics*, 27:15–18, 1993.

[44] T. W. Sederberg and E. Greenwood. A physically based approach to 2-d shape blending. *ACM SIGGRAPH*, 26:25–34, 1992.

[45] M. Shapira and A. Rappoport. Shape blending using the star-skeleton representation. *IEEE Computer Graphics and Applications*, 15:44–50, 1995.

[46] E. H. Spanier. *Algebraic Topology.* McGraw-Hill, New York, 1966.

[47] V. Surazhsky and C. Gotsman. Controllable morphing of compatible planar triangulations. *ACM Transactions on Graphics*, 20:203–231, 2001.

[48] V. Surazhsky and C. Gotsman. Intrinsic morphing of compatible triangulations. *International Journal of Shape Modeling*, 9:191–201, 2003.

[49] E. C. Tupes and R. E. Christal. Recurrent personality factors based on trait ratings. Technical report, USAF ASD Tech. Rep. No. 61-97, Lackland Airforce Base, TX: U. S. Air Force, 1961.

[50] T. Vetter and V. Blanz. Estimating coloured 3d face models from single images: An example based approach. In *Computer Vision - ECCV'98, volume 11*, 1998.

[51] G. Voronoi. Nouvelles applications des paramétres continus á la théorie des formes quadratiques. *Journal fur die Reine and Angewandte Mathematic*, 133:97–178, 1907.

[52] Yushun Wang and Yueting Zhuang. 3d facial modeling for animation: A nonlinear approach. In *Advances in Multimedia Modeling*, pages 64–73. 2006.

[53] Keith Waters. A muscle model for animation three-dimensional facial expression. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 17–24, New York, NY, USA, 1987. ACM.

[54] K. Weiler and P. Atherton. Hidden surface removal using polygon area sorting. In *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 214–222. ACM, 1977.

[55] W. Welch and A. Witkin. Variational surface modeling. In *SIGGRAPH '92, volume 26*, pages 157–166, 1992.

[56] L. Williams. Performance driven facial animation. *SIGGRAPH '90, Computer Graphics*, pages 235–242, 1990.

[57] George Wolberg. *Digital Image Warping.* IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.