

# Facial Motion Cloning<sup>1</sup>

IGOR S. PANDZIC

Department of Electrical Engineering  
Linköping University, SE-581 83 Linköping  
igor@isy.liu.se

## Abstract

*We propose a method for automatically copying facial motion from one 3D face model to another, while preserving the compliance of the motion to the MPEG-4 FBA standard. Despite the enormous progress in the field of Facial Animation, producing a new animatable face from scratch is still a tremendous task for an artist. Although many methods exist to animate a face automatically based on procedural methods, these methods still need to be initialized by defining facial regions or similar, and they lack flexibility because the artist can only obtain the facial motion that a particular algorithm offers. Therefore a very common approach is interpolation between key facial expressions, usually called morph targets, containing either speech elements (visemes) or emotional expressions. Following the same approach, the MPEG-4 Facial Animation specification offers a method for interpolation of facial motion from key positions, called Facial Animation Tables. Our method automatically copies the motion of vertices, as well as geometry transforms, from source face to target face while maintaining the regional correspondences and the correct scale of motion. It requires the user only to identify a subset of the MPEG-4 Feature Points in the source and target faces. The scale of the movement is normalized with respect to MPEG-4 normalization units (FAPUs), meaning that the MPEG-4 FBA compliance of the copied motion is preserved. Our method is therefore suitable not only for cloning of free facial expressions, but also of MPEG-4 compatible facial motion, in particular the Facial Animation Tables. We believe that Facial Motion Cloning offers dramatic time saving to artists producing morph targets for facial animation or MPEG-4 Facial Animation Tables.*

**Keywords:** facial animation, morph targets, MPEG-4, FBA, VRML, text-to-speech, virtual characters, virtual humans

## Introduction

The term *animatable face model* would hardly win an elegant-wording contest, but it is often used for a 3D face model with all additional information necessary to animate it using a particular facial animation system. An animatable face model is capable of being animated by a stream of parameters producing facial animation on a particular animation system. We will use another non-elegant term, *dumb face model*, to describe a face model that does not include the necessary animation-related information and therefore can not be animated directly. A dumb face model is essentially a plain polygon mesh looking like a face.

Producing an animatable model from a dumb model is not a trivial task. The complexity of the task depends on the facial animation system used. Early systems like [Parke74], as well as a surprising

---

<sup>1</sup> US Patent Pending

Submitted to Graphical Models journal, 18.09.2001.

number of much more recent systems, are almost hard-coded to a particular face model that they are built upon, and it would be very hard to adapt a new face model to such systems.

The procedural face animation systems like [Magenat-Thalmann88, Chadwick89, Kalra92, Escher98] usually allow more flexibility. For example, [Kalra92] needs a definition of facial regions in order to make a face model animatable. This is certainly a good approach as it allows any facial model to be animated; however the definition of regions requires a fair amount of work. The more complex muscle based models like [Platt81, Waters87, Terzopoulos90] typically need even more adaptation on the face model.

The MPEG-4 Facial Animation standard (see next section for an overview) specifies the motion of a small number of points on the face, leaving the rest to the particular implementations. Therefore, most of the previously mentioned methods can be made MPEG-4 compatible and indeed, many facial animation algorithms created specifically for MPEG-4 are direct descendants of the previous procedural approaches. As an example, [Lavagetto99] requires a definition of facial feature points and their regions of influence to make a face model animatable.

A very popular approach to making animatable face models is the use of *morph targets*. This is a method widely used in the computer animation community and its use is by no means limited to faces. In the context of facial animation, it means defining a number of key positions of the face by displacing its vertices. These key positions are called morph targets. The animation system then interpolates the vertex positions between the morph targets. Traditionally, morph targets are used for high-level facial motions like visemes or expressions, like in [Arai96]. However, they have been successfully extended to low-level facial motions, in particular the MPEG-4 Facial Animation Parameters [Pandzic01]. It can be remarked that the MPEG-4 Facial Animation Tables are in fact morph targets, although the MPEG-4 specification does not use such terminology.

Producing morph targets is obviously a tedious manual task for the animator. Even just 10 basic mouth positions and a few expressions require a tremendous amount of work. If we extend this to the full set of high- and low-level MPEG-4 FAPs, 88 morph targets need to be manually constructed. Obviously, this is a much higher volume of work than defining facial regions or points for procedural approaches. On the other hand, morph targets have a tremendous advantage: they let the artist completely control the final appearance of animations, whereas the procedural models do not leave such liberty and may leave the animator unsatisfied. For example, if a particular procedural system does not produce wrinkles, the face will never have any. With morph targets, the animator designs wrinkles exactly where he/she wants them to appear.

Can we keep this advantage of the morph target approach, but reduce the amount of work needed? Yes. That is the point of this article. We propose Facial Motion Cloning (FMC), a method to copy a whole set of morph targets from one face to another. Thus one carefully produced animatable face can serve as a template for a very fast production of another animatable face, starting with a dumb model. Conceivably, libraries of animatable models may become available, letting the artist choose the kind of animation desired (e.g. wrinkles or no wrinkles).

A very similar idea has recently been proposed in [Noh01] and termed Expression Cloning (EC). The major differences between FMC and EC are the MPEG-4 compatibility of our approach, support of eye, teeth and tongue movements, solution to problems appearing when cloning between models of different mesh density, as well as numerous implementation differences. We analyze these differences in the Discussion section.

In the next section we briefly introduce the MPEG-4 Facial Animation, as necessary for the understanding of this article. The main part of the article explains in several subsections the steps of the Facial Motion Cloning method. We finish with Results and Discussion sections.

## Introduction to MPEG-4 Facial Animation

Beside the standard itself [ISO14496], there are other excellent references [Tekalp00] [Escher98] covering the subject of MPEG-4 Facial Animation. The purpose of this short section is therefore not to offer in-depth coverage, but to provide just enough background for understanding the rest of the article. Readers familiar with MPEG-4 FA may wish to skip this section, or use it only as a quick reference.

The MPEG-4 specification defines 64 low-level Facial Animation Parameters (FAPs) and two high-level FAPs. The low-level FAPs are based on the study of minimal facial actions and are closely related to muscle actions. They represent a complete set of basic facial actions, and therefore allow the representation of most natural facial expressions. Exaggerated values permit the definition of actions that are normally not possible for humans, but could be desirable for cartoon-like characters.

All low-level FAPs are expressed in terms of the *Facial Animation Parameter Units (FAPUs)*, as illustrated in Figure 3. These units are defined in order to allow interpretation of the FAPs on any facial model in a consistent way, producing reasonable results in terms of expression and speech pronunciation. They correspond to distances between key facial features and are defined in terms of distances between the MPEG-4 facial Feature Points (FPs, see Figure 4). For each FAP it is defined on which FP it acts, in which direction it moves, and which FAPU is used as the unit for its movement. For example, FAP no. 3, `open_jaw`, moves the Feature Point 2.1 (bottom of the chin) downwards and is expressed in MNS (mouth-nose separation) units. The MNS unit is defined as the distance between the nose and the mouth (see Figure 3) divided by 1024. Therefore, in this example, a value of 512 for the FAP no. 3 means that the bottom of the chin moves down by half of the mouth-nose separation. The division by 1024 is introduced in order to have the units sufficiently small that FAPs can be represented in integer numbers.

The specification includes two high-level FAPs: `expression` and `viseme`. Expression can contain two out of a predefined list of six basic expressions. Intensity values allow to blend the two expressions. Similarly, the Viseme parameter can contain two out of a predefined list of 14 visemes, and a blending factor to blend between them.

The specification also defines the Facial Animation Tables (FATs). The FATs allow to specify, for each FAP (high- and low-level), the exact motion of the vertices and/or transforms in the 3D model. This means that the expressions, visemes and low-level FAPs are described in a way that is essentially equivalent to the mentioned morph-targets. Animation systems then interpolate and blend between the values from the FAT.

## The Facial Motion Cloning method

Facial Motion Cloning can be schematically represented by Figure 1. The inputs to the method are the source and target face. The source face is available in neutral position (*source face*) as well as in a position containing some motion we want to copy (*animated source face*). The target face exists only as neutral (*target face*). The goal is to obtain the target face with the motion copied from the source face – the *animated target face*.

To reach this goal we first obtain *facial motion* as the difference of 3D vertex positions between the animated source face and the neutral source face. The facial motion is then added to the vertex positions of the target face, resulting in the animated target face.

In order for this to work, the facial motion must be normalized, which insures that the scale of the motion is correct. In the *normalized facial space*, we compute facial motion by subtracting vertex positions of the animated and the neutral face. To map the facial motion correctly from one face to another, the faces need to be aligned with respect to the facial features. This is done in the *alignment space*. Once the faces have been aligned, we use interpolation to obtain facial motion vectors for vertices of the target face. The obtained facial motion vectors are applied by adding them to vertex positions, which is possible because we are working in the normalized facial space. Finally, the target face is denormalized. The whole process is described in the next five subsections.

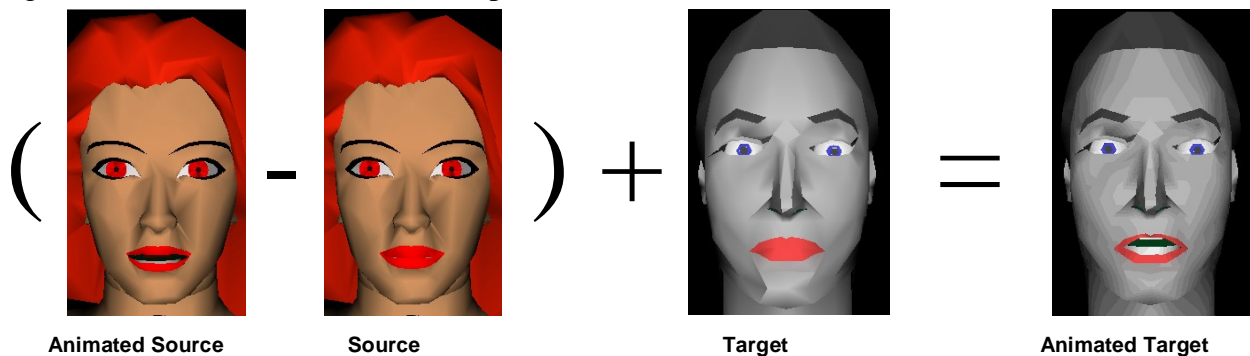


Figure 1: Overview of Facial Motion Cloning

When applying mathematical algorithms to real world problems, we often have to treat exceptions. With the human face this seems to be the case even more than usual. In the last three subsections of this section we deal with the aliasing problem when mapping between meshes of different densities, and with the parts of the face that can not be fully treated by the basic algorithm: the lip region, eyes, teeth and tongue.

### *Interpolation from 2D triangle mesh*

As it will be shown further, two steps of our method, alignment and motion vector mapping, can be reduced to the problem of interpolating function values from a number of points in a 2D triangular mesh with known function values. We therefore begin by explaining how this interpolation method works.

As illustrated in Figure 2, the inputs to the method are:

- $N$  input points  $P_i(x_i, y_i)$  in 2D space.
- $N$  function values at input points  $F_i = f(P_i)$ ; note that  $F_i$  can be vectors.
- Triangulation of input points defined as  $M$  triangles  $T_j(a_j, b_j, c_j)$ ,  $1 < a_j, b_j, c_j < N$ ,  $a_j, b_j, c_j$  defining the vertices of triangles as pointers into the array of input points.

From these inputs, we want to interpolate the value of the function  $f(P)$  for any point  $P(x, y)$  in the 2D space.

We first determine if  $P$  is within any of the triangles  $T_j$ . To do this, we test each triangle first with a simple bounding box test, then by computing barycentric coordinates [Watson92] of the point  $P$  with respect to the triangle  $T_j$ . If  $P_k(x_k, y_k)$ , are the coordinates of the vertices of the triangle, barycentric coordinates  $b_k$  are computed by solving the following system of equations:

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (1)$$

Because of the properties of barycentric coordinates, if  $0 < b_k < 1$ ,  $P$  is inside  $T_j$ . If this is the case, as for point  $P_A$  in Figure 2, we can compute the interpolated value as follows:

$$f(P) = \sum_{k=1}^3 b_k \cdot f(P_k) \quad (2)$$

If point  $P$  is lying within more than one triangle  $T_j$ , this means that the triangles are overlapping. In this case the interpolation method needs some application-specific way of resolving the conflict and choosing one of the triangles. The function value is then interpolated from the chosen triangle using (2).

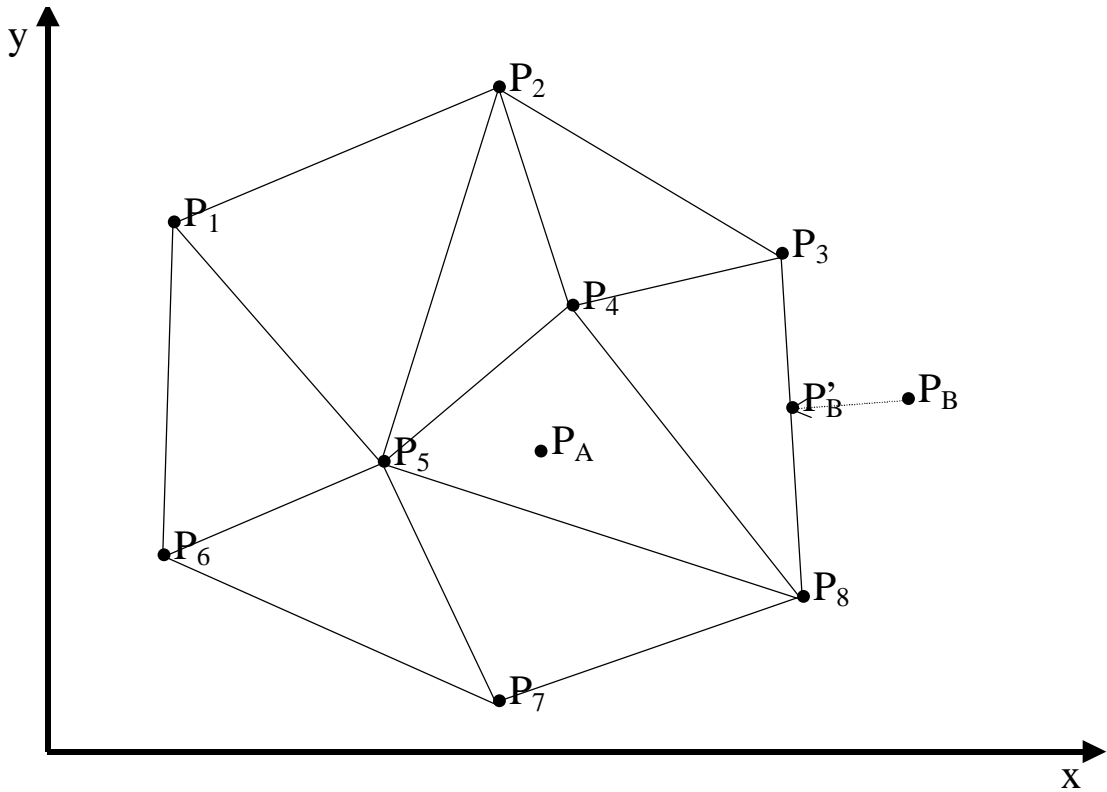


Figure 2: Interpolation from 2D triangle mesh

Finally, if  $P$  is lying outside all triangles, like point  $P_B$  in Figure 2, we use the interpolation value from the closest point on the closest triangle to  $P$ . As the measure of distance from  $P$  to a triangle, we use the sum of the excess barycentric coordinates, i.e. any amount of  $b_k$  above 1 or below 0. In pseudo code, it looks like this:

```

distance = 0
for k = 1..3
    if( $b_k < 0$ ) than distance += ( $-b_k$ )
    if( $b_k > 1$ ) than distance += ( $b_k - 1$ )
end for
    
```

After identifying the triangle  $T$  with the minimum distance, we find the barycentric coordinates  $b'_k$  of the point  $P'$ , which is the closest point to  $P$  on the triangle  $T$ . This is done by truncating each  $b_k$  to

values between 0 and 1, then dividing them by their sum to make sure that the sum of the new coordinates equals 1. In pseudo code it looks like this.

```

sum = 0
for k = 1..3
  if(bk < 0) than bk = 0
  if(bk > 1) than bk = 1
  sum += bk
end for
for k = 1..3
  b'k = bk / sum
end for

```

Finally we apply (2) to  $b'_k$  and obtain the interpolated value.

### Normalizing the face

The goal of normalization is to transform a face into a new 3D space, the normalized facial space, in which all faces have the same key proportions. This means that in the normalized space, all faces have the same mouth width, same distance between the eyes etc. The nice consequence of this is that the magnitude of motion in the normalized space is the same for different faces and we can therefore apply motion from one face to another by simple addition. Figure 5B shows examples of normalized faces.

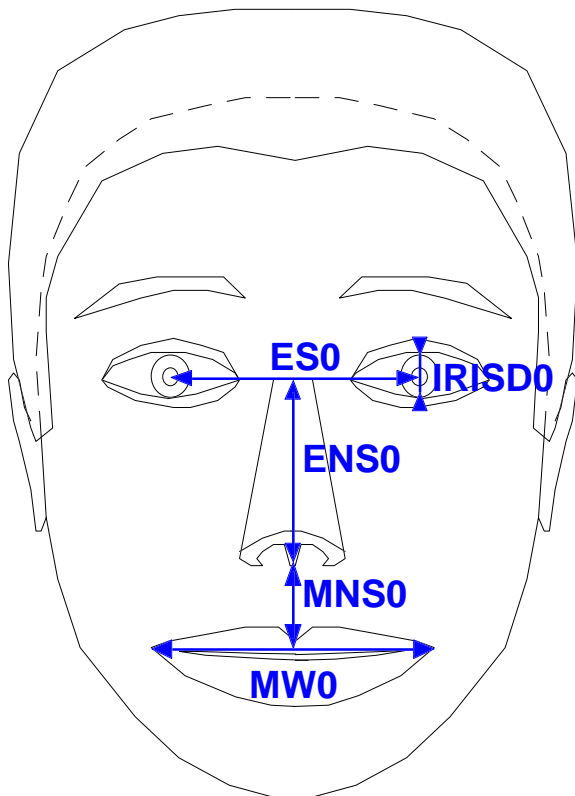


Figure 3: Facial Animation Parameter Units (FAPU)

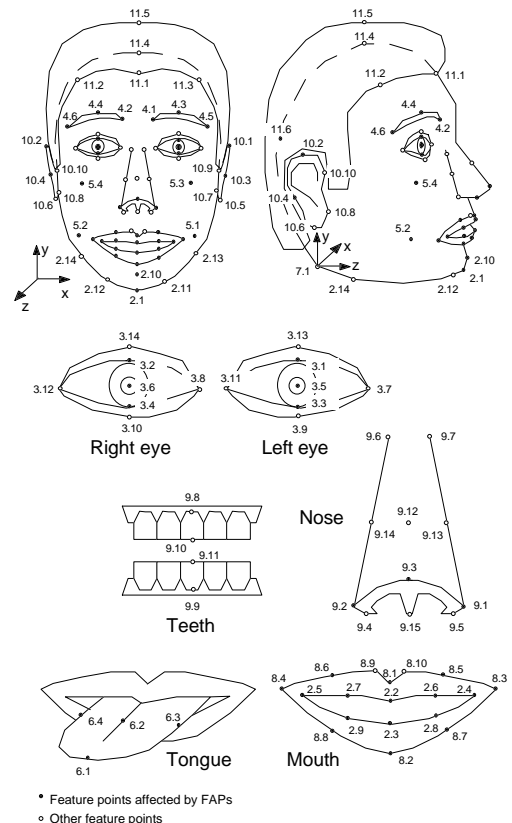


Figure 4: Facial Feature Points (FP)

To define the proportions of the face we use the Facial Animation Parameter Units (FAPU, Figure 3), as specified in the MPEG-4 standard, and the MPEG-4 Feature Points (FP, Figure 4). Our method requires that the coordinates of key FPs be known for both source and target face. The required FPs are indicated in Figure 7. The regions of the face are determined from the proximity to

the FPs. The appropriate FAPUs are then used as normalization factors for each vertex. This amounts to scaling the face model with a regionally changing scaling factor. As the center of scaling we chose the center of the face. The normalization process can thus be represented as follows:

```
for each vertex V
  find closest Feature Point FP
  V = (V - C) / N(FP)  // this is a component-wise division
end for
```

C is the center of the face. N(FP) is a vector containing the appropriate normalization factors for the region of the feature point FP. For example, the lip region is normalized with factors {MW, MNS, MNS} (see Figure 3). The normalization factors are FAPUs chosen to obtain MPEG-4 compatible facial motion in the normalized space. The MPEG-4 FAPs are normalized with respect to FAPUs. By normalizing with the same FAPUs we make sure that the motion of the feature points in the normalized space corresponds to the MPEG-4 FAPs. At the same time, by normalizing whole regions, we insure consistent mapping of the general facial motion.

The process of denormalization is the simple inverse: we scale vertex coordinates by the inverse normalization factors.

### ***Computing facial motion***

The facial motion is defined as the difference in vertex positions between the animated and neutral face. It is expressed by an array of *facial motion vectors*, each vector corresponding to one vertex of the face. The facial motion is always computed in the normalized facial space, thus the facial motion vectors are correctly normalized and can be transferred from one face to another. To compute the facial motion vectors we simply subtract the 3D position of each vertex in the neutral face from the position of the corresponding vertex in the animated face.

### ***Aligning source and target face***

When transferring facial motion from source to target face, we need to transfer the motion to the corresponding facial region. This means that the motion of the left lip corner in the source face has to be mapped to the left lip corner in the target face; the vertices near this lip corner also have to get the motion from vertices in the corresponding region of the source face. In order to achieve this, we compute the motion mappings using the *alignment space*. This is a 2D space in which the source and target face are aligned with respect to the feature points. So, in the previous example, the left lip corners of the source and target face are in the same point in the alignment space. We can therefore use the alignment space to map the facial motion from source to target, as will be explained in detail in the next subsection.

Figure 5 shows the process of projecting source and target face into the 2D space. The initial faces are shown in Figure 5A. After normalization, as explained in the previous subsection, we obtain the normalized faces in Figure 5B.

The normalized faces are then mapped into 2D space using cylindrical projection around Y-axis. The center of projection is computed in such a way that the angular width of the mouth remains constant. The zero angle is always aligned with the tip of the nose. These two rules, together with previous normalization, ensure that the projected faces, shown in Figure 5C, are already roughly aligned.

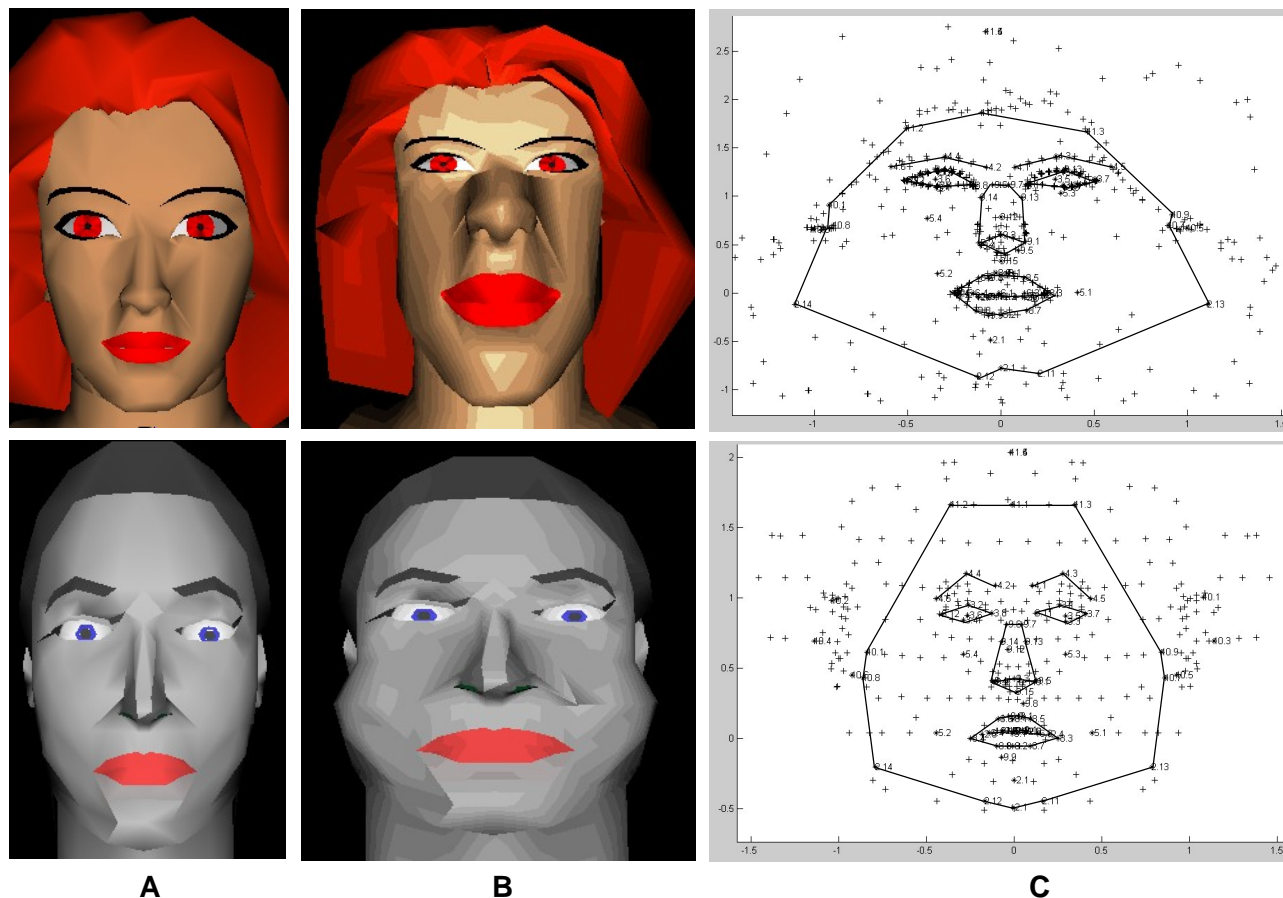


Figure 5: Aligning source and target face. Upper row: source face. Lower row: target face. A: faces in initial state. B: normalized faces. C: faces projected into 2D. Facial features are outlined for better visibility.

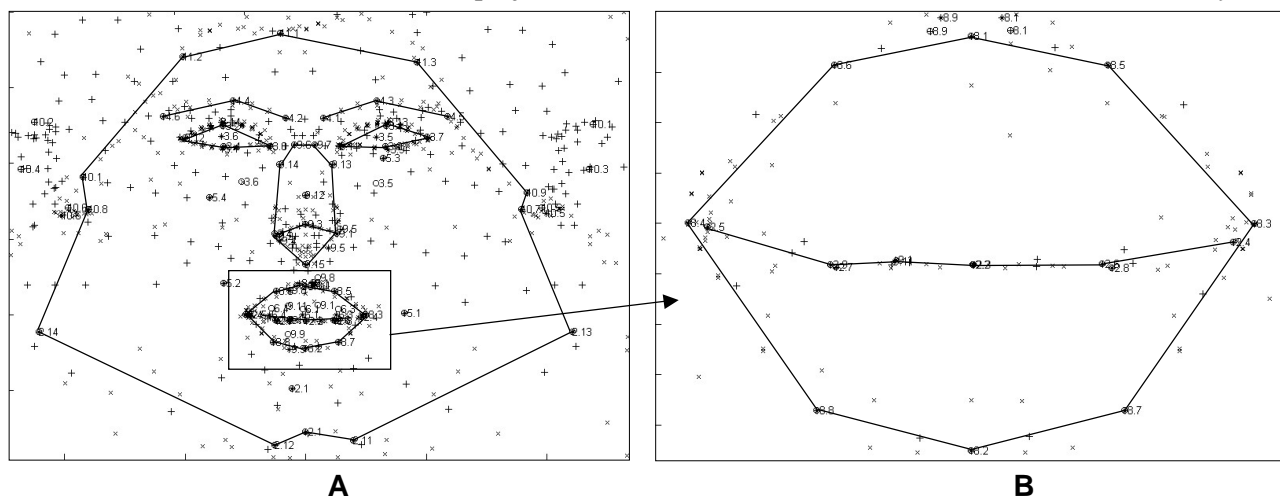


Figure 6: Aligned source and target faces; source face shown with 'x', Feature Points marked with '\*', target face shown with '+', Feature Points marked with a circle. Facial features are outlined for visibility.

In the final alignment, we move the feature points of the target face onto the corresponding feature points of the source face. The non-feature points are *pulled* by the feature points so that facial regions remain intact. Figure 6A shows the source and target face aligned, and Figure 6B shows a zoom-in detail of the same.



The pulling of non-feature points into their regional position is achieved by the previously described triangle mesh interpolation method. The input points for interpolation are the feature points. The interpolated function is the movement vector of the feature point obtained by subtracting its original position from its new position (i.e. the position of the feature point in the source face). The triangulation, necessary for the interpolation algorithm, is produced manually, as illustrated in Figure 7. We have also experimented with Delaunay triangulation. However, the manual triangulation gave us flexibility to experiment with different triangulations and choose the optimal one. As this is done only once, and is fairly simple, there is no advantage in using automatic triangulation.

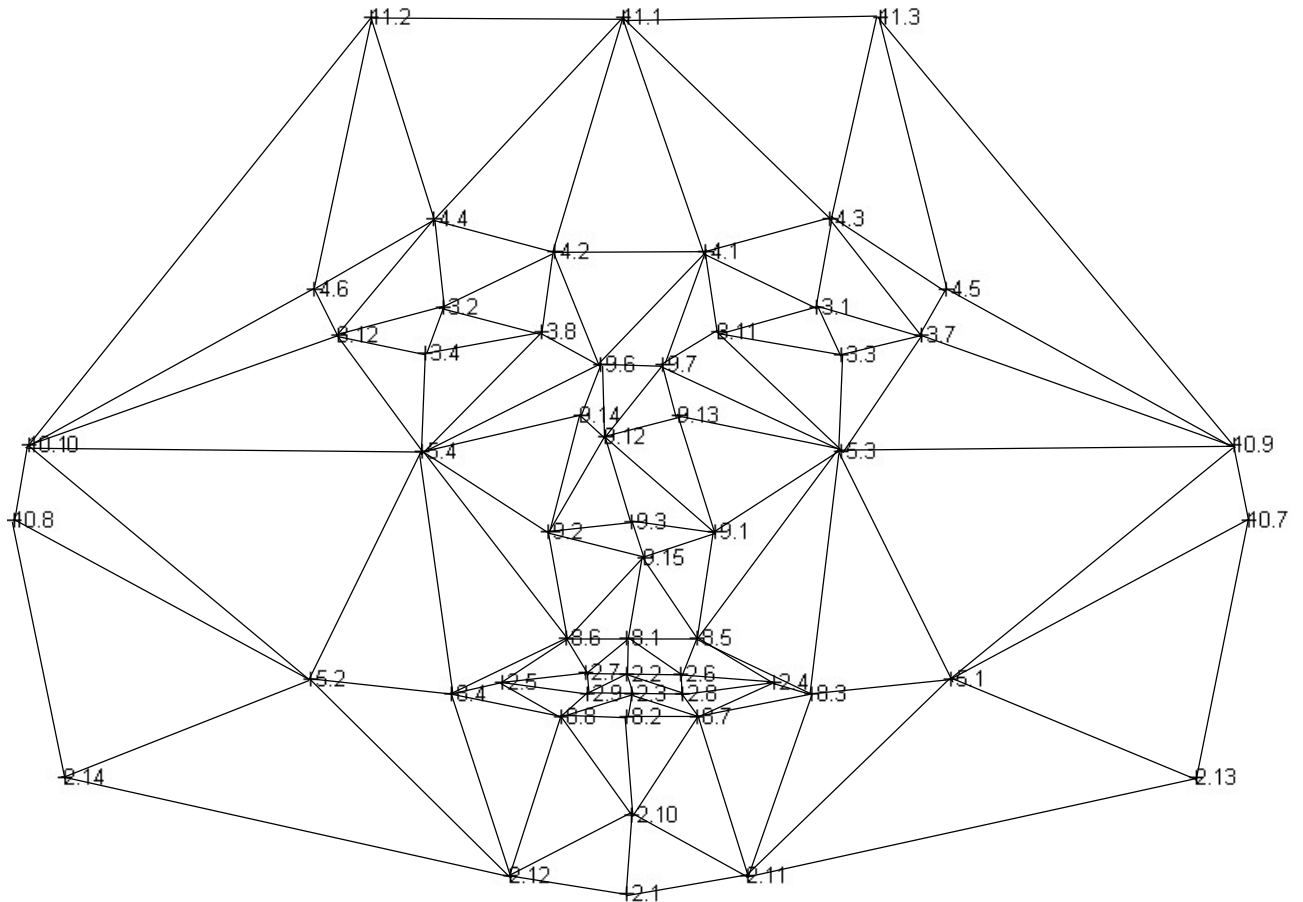


Figure 7: Triangulation of the feature points used for interpolation of non-feature-points movements for alignment. This figure also illustrates which Feature Points are required for the Facial Motion Cloning. In addition to the ones illustrated in the figure, Feature points 7.2 (neck base), 3.5 and 3.6 (left and right iris centers) are also required.

### *Mapping facial motion*

We know the facial motion vector for each vertex of the source face, and we have mapped these vertices into the 2D alignment space. To obtain the motion vectors for vertices of the target face, we again use the previously described triangle mesh interpolation method. The input points are all points of the source face. The input function values are the motion vectors at input points. The triangulation is directly available from the source face model. As this triangulation is defined in 3D, there is a high possibility of overlapping triangles in 2D. To resolve this conflict we choose the

triangle with the smallest depth at the interpolation point. The depth at the interpolation point is obtained by interpolating depth of the three vertices of the triangle.

Once we have the facial motion vectors for each vertex of the target face, we simply add them to the vertex positions and denormalize the target face. This is the animated target face.

### *Antialiasing*

In our interpolation method, the values of the motion vector are known at the points of the source face mesh. This is equivalent to sampling a (presumably smooth) function at a number of discrete, non-evenly distributed points. The density of the source face mesh is the equivalent of the sampling frequency, which usually changes for different regions of the face. From these values, we interpolate – or, in effect, re-sample - the value of the function at a different set of non-evenly distributed points, those of the target face. Again, the density of the target face mesh is equivalent to the frequency after re-sampling. A difference in these two sampling frequencies, i.e. in mesh densities, leads to an aliasing problem manifesting itself in irregularities in the target face motion when mapping from a face with a higher density to a face with a lower density (see Figure 8).

To solve this problem, instead of interpolating the value only at the exact point, we also interpolate at additional points around the original point, then average the result. In effect, we apply a low-pass filter to the interpolation function. The size of the support area of the filter can be varied, but we experimentally found the value of 0.02 to be satisfactory for all cases we tried (the dimensions of the face in the 2D alignment space are limited by the normalization and 2D projection process and therefore similar for all faces, which explains why there is no drastical change in required support area size for the filter).



Figure 8: A: Aliasing artifacts clearly visible on lower lip. B: When filtering is applied, lower lip is smooth.

### *Treating the lip region*

The lip region poses a particular problem for facial motion mapping because the points in the upper and lower lip are very close in the alignment space, but their motion is typically opposite. If we apply the above-described mapping algorithm, the motion of the lower lip would pull the upper lip as well, and vice versa, producing severe artifacts.

We therefore classify all vertices in the face into one of the three categories: upper lip region, lower lip region and out of lip region. When we calculate the motion vector for target face vertices belonging to the lower lip region, we ignore the vertices of the upper lip region of the source face, and vice versa.

To determine the lip regions we use the upper and lower lip feature points, as well as the feature points of the nose and the chin. We define the regions in the projected 2D space (Figure 5C). The upper lip region is determined as the polygon defined by the feature points of the inner upper lip contour and the sides of the nose. The lower lip region is determined as the polygon defined by the feature points of the inner lower lip contour and the sides of the chin, as illustrated in Figure 9.

**Treating eyes, teeth, tongue and global motion**

Eyes, teeth and tongue are typically modeled as separate polygon meshes in a face model, and inserted into the scene hierarchy tree under separate transform nodes. In particular, the MPEG-4 Facial Animation Tables allow for specification of the motion of transform nodes. We therefore attempt to automatically identify the transform nodes responsible for the motion of eyes, teeth, tongue and the global motion, then map these motions onto appropriate transform nodes in the target face.

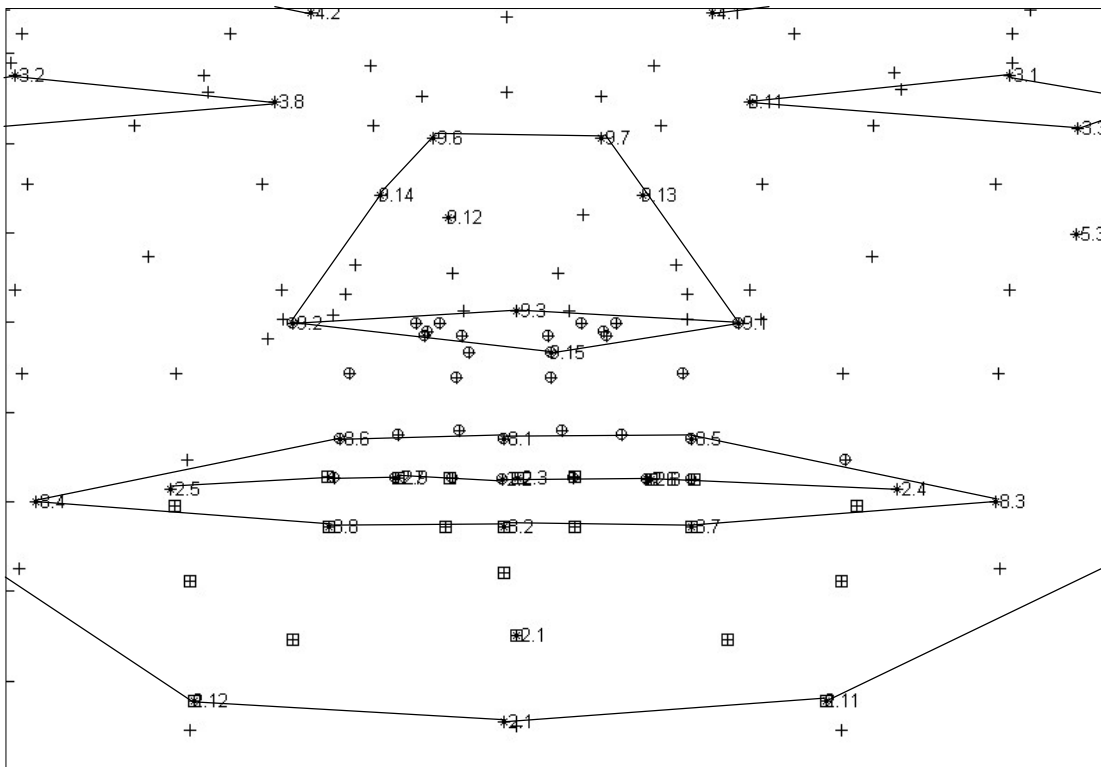


Figure 9: Upper / lower lip region distinction. Upper lip region points marked with circles, lower lip region points marked with squares. Facial features are outlined for better visibility.

The strategy for finding the appropriate transform nodes is to look for the hierarchically lowest transform node that contains some motion, and encompasses the polygon meshes we look for (i.e. an eye, tongue, lower teeth or movable part of the face). The polygon meshes are identified from feature points. For example, to identify the meshes containing lower teeth, we look for a mesh in the model containing one of the lower teeth feature points. Then we look for any connected meshes in case teeth are composed of more than one mesh. While looking for connected meshes, we avoid those that can be identified as other parts of the face, e.g. if the face is modeled with clenched teeth, the upper and lower teeth meshes may appear connected but we resolve this problem by identifying upper teeth as well from their feature points.

In the target face, we find the appropriate transform as the hierarchically lowest transform node that encompasses the polygon meshes we look for. In case of eyes and global transform, we also make sure that their centers of rotation are appropriate (centers of eyes and base of the neck, respectively). If they are not, we install new centers of rotation.

When these correspondences are established, we map rotations of eyes and global motion, and translations of teeth and tongue, from source to target. If any of the transforms in source or target face cannot be identified, these motions are ignored and a warning is issued.

A proper way to treat the tongue would obviously require it to be flexible. The main Facial Motion Cloning algorithm could be applied separately to the tongue. However, at the current stage we had no access to source facial models that actually animate the tongue as a flexible object and we limited the method to rigid object motion for the tongue.

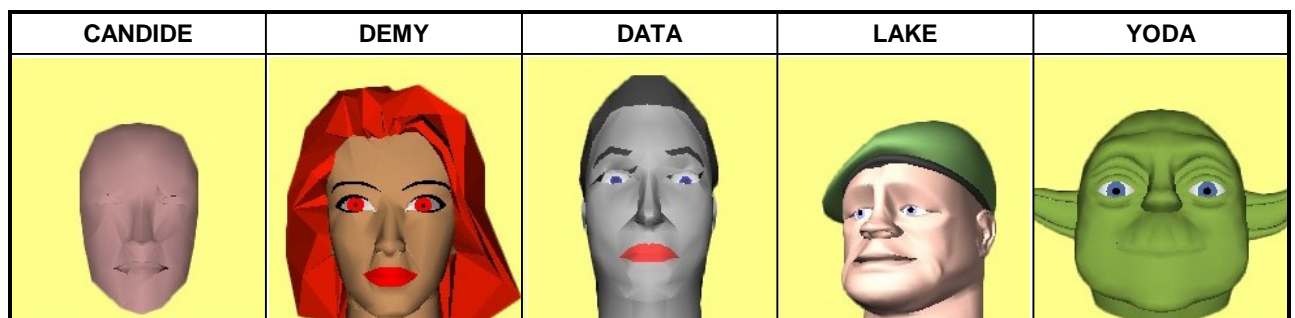


Figure 10: The face models used for testing, shown in neutral position. The sources of the models are: Linköping University for Candide, Sasa Galic for Demy, 3DS Max Tutorial for Lake, Avatara VR ([www.avatara.com](http://www.avatara.com)) for Data and Yoda.

## Results

In the testing of the method we have used five face models shown in Figure 10: Candide, Demy, Data, Lake and Yoda. Data and Yoda were available only in neutral positions. Candide was available with a full range of low-level FAP motions obtained algorithmically [Ahlberg01], but high-level FAPs were not available. The Lake model was available with visemes, but no other motion. Finally, Demy was available with a full set of high- and low-level FAPs. The missing motions for each model were created first by cloning from either Demy or Candide. Thus a full set of high- and low-level FAP motions was available for each model. Then all motions were cloned from each model to each other model, producing a full grid of cloned animated models for each motion. One such grid is shown for the surprise expression in Figure 11, and another for the viseme I in Figure 12. In these grids, the main diagonal contains the source faces, and the rest of each row shows the results of cloning from that source face to all other face models. Therefore, looking at each row shows how an expression is cloned from one face to all other faces; looking at each column shows how the expression is cloned onto the same face from different sources.

It is also interesting to show the expressions produced by combinations of cloned low-level FAPs. One such expression is shown in the cloning grid in Figure 13. These expressions are not cloned directly. Rather, all low-level FAP motions were cloned, and then the face was moved into an expression determined by a particular combination of FAPs. The diagonal contains the source face showing the same combination of low level FAPs.


























|        |         | TARGET  |   |   |   |   |
|--------|---------|---|---|---|---|---|
|        |         | CANDIDE   | DEMY  | DATA  | LAKE  | YODA  |
| SOURCE | CANDIDE |    |    |    |    |    |
|        | DEMY    |    |    |    |    |    |
|        | DATA    |   |   |   |   |   |
|        | LAKE    |  |  |  |  |  |
|        | YODA    |  |  |  |  |  |

Figure 11: Cloning grid for surprise expression.


























|        |         | TARGET  |   |   |   |   |
|--------|---------|---|---|---|---|---|
|        |         | CANDIDE   | DEMY  | DATA  | LAKE  | YODA  |
| SOURCE | CANDIDE |    |    |    |    |    |
|        | DEMY    |    |    |    |    |    |
|        | DATA    |   |   |   |   |   |
|        | LAKE    |  |  |  |  |  |
|        | YODA    |  |  |  |  |  |

Figure 12: Cloning grid for the viseme I.

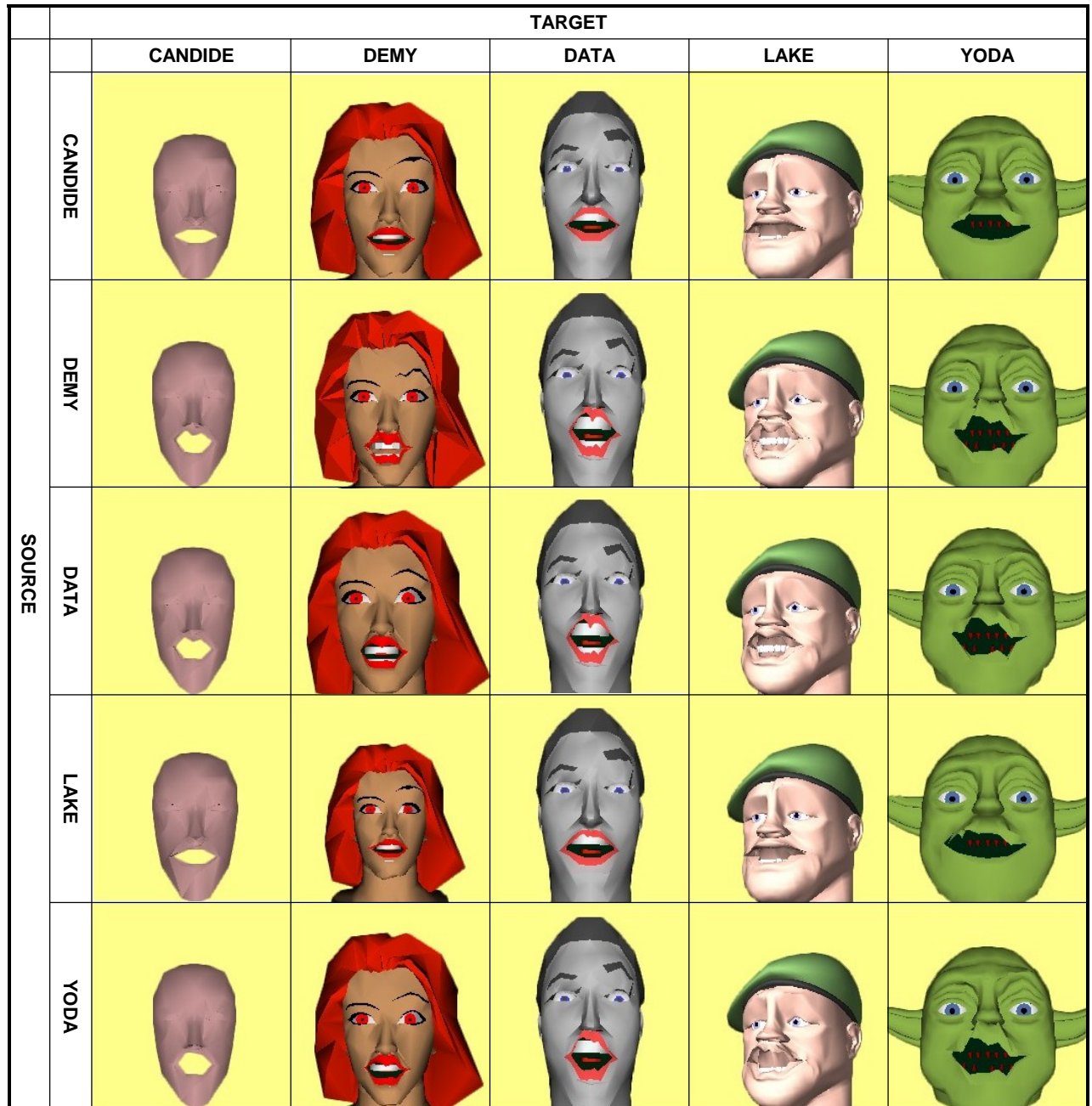


Figure 13: Cloning grid for an expression produced by low-level FAPs. The FAPs were set to the following values: open jaw: 800; stretch left and right lip corners: 150; all inner and outer, upper and lower lip raising/lowering FAPs: -400; raise left eyebrow (middle, inner and outer): 300.

## Discussion

In terms of visual results shown, even though only a small subset of them could be presented here, we believe that most facial movements for expressions and low-level FAPs are copied correctly to the target faces in most cases. Certain problems still appear, in particular when using low level FAPs, as it is visible in Figure 13. This is due to the process of combining low level FAPs to form expressions. If the low level FAPs are not very carefully designed, their combination leads to irregularities in the combined expressions. Such irregularities, if existing in the source face, are

carried over to the cloned face. If the cloned face is then itself used as a source, the irregularities propagate to the next cloned generation and possibly amplify. This is clearly visible in Figure 13, where the original Demy model already shows severe visual artifacts when animated by low level FAPs (see original Demy in the second row). These artifacts are visible in all cloned models in the second row. Furthermore, the Data and Yoda models cloned from Demy were in turn used as sources, and thus the artifacts propagate to all cloned models in rows 3 and 5 as well. By contrast, models cloned from Candide (row 1), and subsequently from Lake cloned from Candide (row 4), show much less artifacts or none at all, because the Candide model originally has much better definitions of low level FAPs than Demy.

We have also noted during the experiments that the method is quite sensitive to correct placement of Feature Points, which was expected.

It is interesting to compare with other similar methods, and the obvious choice in this case is the recent Expression Cloning proposed in [Noh01]. The final results are not easy to compare due to different facial models and expressions used in tests, so we leave the subjective judgement to the readers. We can discuss differences in approach and implementation which are substantial, although the basic idea is very similar. We could summarize the differences of Facial Motion Cloning (FMC) with respect to Expression Cloning (EC) in the following points:

- FMC is specifically aimed at preserving MPEG-4 compatibility of motion.
- FMC treats the motion of the eyes, teeth and tongue.
- FMC treats the aliasing problem due to differing densities of source and target meshes.
- EC uses Radial Basis Functions for alignment of surfaces in 3D; FMC aligns points in 2D and uses a simple interpolation – we suspect that FMC is simpler to implement and lighter in CPU usage; we can currently not conclude if this has a substantial adverse impact on quality.
- EC scales the motion vectors locally and adjusts their direction; FMC scales with respect to MPEG-4 FAPU normalization, i.e. on a larger scale, and it does not adjust motion vector direction as it would destroy the MPEG-4 compatibility – this might have an adverse effect on quality.
- EC proposes heuristic rules to identify the correspondence points between two faces; FMC could possibly use some of these rules, but as it needs exact MPEG-4 Feature Points probably at least some points would still need to be manually defined (in the current implementation the points are identified manually)

In summary, we believe that FMC and EC share the same underlying idea, however with numerous differences in goals and implementations due largely to the MPEG-4 compatibility of FMC.

As a final conclusion, we believe that Facial Motion Cloning offers a dramatic time-saving potential for producing facial animation based on morph targets or MPEG-4 Facial Animation Tables.

## **Acknowledgments**

This research is supported by the VISIT programme of the Swedish Foundation for Strategic Research (SSF). The Demy face model and its animation were created by Sasa Galic. Many thanks to Robert Forchheimer who sparked this idea during a brainstorming.



## References

- [Ahlberg01] "Candide-3 -- an updated parameterized face", J. Ahlberg, Report No. LiTH-ISY-R-2326, Dept. of Electrical Engineering, Linköping University, Sweden, 2001. [www.icg.isy.liu.se/candide](http://www.icg.isy.liu.se/candide)
- [Arai96] "Bilinear interpolation for facial expressions and methamrphosis in real-time animation", Kiyoshi Arai, Tsuneya Kurihara, Ken-ichi Anjyo, *The Visual Computer*, 12:105-116, 1996.
- [Chadvick89] "Layered construction for deformable animated characters", *Computer Graphics*, 23(3):234-243, 1989
- [Escher98] "Facial Deformations for MPEG-4", M. Escher, I.S. Pandzic, N. Magnenat-Thalmann, *Computer Animation 98*, Philadelphia, USA, pp. 138-145, IEEE Computer Society Press, 1998.
- [ISO14496] ISO/IEC 14496 - MPEG-4 International Standard, Moving Picture Experts Group, [www.cseit.it/mpeg](http://www.cseit.it/mpeg)
- [Kalra92] Kalra P., Mangili A., Magnenat-Thalmann N., Thalmann D., "Simulation of Facial Muscle Actions based on Rational Free Form Deformation", *Proceedings Eurographics 92*, pp. 65-69
- [Lavagetto99] "The Facial Animation Engine: Toward a High-Level Interface for the Design of MPEG-4 Compliant Animated Faces", Fabio Lavagetto, Roberto Pockaj, *IEEE Transactions on circuits and systems for video technology*, 9(2):277-289, 1999.
- [Magnenat-Thalmann88] "Abstract muscle actions procedures for human face animation", N. Magnenat-Thalmann, N.E. Primeau, D. Thalmann, *Visual Computer*, 3(5):290-297, 1988.
- [Noh01] "Expression Cloning", Jun-yong Noh, Ulrich Neumann, *Proceedings of SIGGRAPH 2001*, Los Angeles, USA
- [Pandzic01] "A Web-Based MPEG-4 Facial Animation System", I.S. Pandzic, *Proc. ICAV 3D 2001*, demonstration at [www.icg.isy.liu.se/~igor/MpegWeb](http://www.icg.isy.liu.se/~igor/MpegWeb)
- [Parke74] "A Parametric Model for Human Faces", F.I. parke, PhD Thesis, University of Utah, Salt Lake City, USA, 1974. UTEC-CSc-75-047
- [Parke96] "Computer Facial Animation", F.I. Parke, K. Waters, A K Peters Ltd. 1996., ISBN 1-56881-014-8
- [Platt81] "Animating Facial Expressions", S.M. Platt, N.I. Badler *Computer Graphics*, 15(3):245-252, 1981.
- [Tekalp00] "Face and 2-D Mesh Animation in MPEG-4", Tekalp M.A., Ostermann J., *Image Communication Journal*, Tutorial Issue on MPEG-4 Standard, Elsevier, 2000.
- [Terzopoulos90] "physically-based facial modeling, analysis and animation", D. Terzopoulos, K. Waters, *Journal of Visualization and Computer Animation*, 1(4):73-80, 1990.
- [VRML] VRML, ISO/IEC 14772-1:1999, [www.web3d.org/fs\\_specifications.htm](http://www.web3d.org/fs_specifications.htm)
- [Waters87] "A muscle model for animating three-dimensional facial expressions", K. Waters, *Computer Graphics (SIGGRAPH'87)*, 21(4):17-24, 1987.
- [Watson92] Watson D.F., "Contouring – A Guide to the Analysis and Display of Spatial Data", *Computer Methods in Geosciences*, Volume 10, Pergamon