A Feature-Preserved Simplification for Autonomous Facial Animation from 3D Scan Data

Soo-Kyun Kim¹, Sun-Jeong Kim, and Chang-Hun Kim

Dept. of Computer Science and Engineering, Korea University,
1, 5-ka, Anam-dong, Sungbuk-ku, Seoul 136-701, Korea
{nicesk, chkim, mnjeongkim}@korea.ac.kr

Abstract. We propose a new simplification algorithm of facial models for animation. For the facial animation, the models are often simplified from complex scan data based on geometric features, but it leads to decrease the quality and such features are easily noticed by human perception. For example, a lip line and eyebrows easily lose their details by geometry-based simplification. In this paper, facial features are extracted using an image processing of a 2D texture image and the curvature analysis of the 3D geometry, which improves the details around the feature areas of the facial model. Especially if lip contact line is simplified to one or two edges, it may not be proper for lip animation. Finally, we will show that our simplified model can produce as good as a facial animation as the one from the original model.

1 Introduction

An individual head model is usually reconstructed from the scan data acquired from an optical scanning system. Since a scan data is usually large in size, it needs to be simplified, and there are various mesh simplification algorithms to do such a task. However, most of these algorithms are not suitable for facial modeling because they do not take into account the features of a human face. For example, figure 2 shows the facial animation of a model, which is simplified by a geometry-based simplification, that exhibits a poor quality near features like a lip and eyes, although it is simplified to the same number of vertices as that of Figure 1.

In real time or interactive applications, models with millions of polygons are still burdensome even with fast graphics hardware. For this reason, simplification of surfaces has been the subject of a great deal of research. Simplification algorithms [5, 9, 12, 16] based on iterative edge contraction have gained a lot of attention. Since the new vertex position can be controlled to retain the original appearance, retriangulation is not needed.

Ronfard and Rossignac [15] measured the error at a vertex by maximum distance between the vertex and the planes. Instead of the maximum distance, Garland and Heckbert [4, 5] used the sum of the squared distances, along with
Fig. 1. A brief overview of our simplification process: The input model is texture-mapped scan data; at first, facial features are extracted from texture image; then, final features are remained through curvature filtering within bounding area; after our feature-preserved simplification, the animation of the simplified facial model shows higher quality than Figure 2.

a memory-computation efficient algorithm called "Quadric Error Metrics" to accumulate the cost of contraction at an object vertex as the simplification progresses. Garland [5] and Hoppe [9] uses extended error metric that combines geometric and surface attributes (normal, colors, texture coordinate). Cohen [1] did not use a quadric form but proposed a texture error measurement algorithm. Discrete curvature [2, 3] approximated from a geometric reasoning is useful to enhance the shape description of triangular surfaces. Therefore, discrete curvature can be used as a good criterion of simplification that can preserve the shape of an original model well as in [11].

Fig. 2. Low quality of facial animation of a simplified facial model by the previous method [5]. (left) Animation, (right) Close-up of dashed boxes in (left).

Several methods were also proposed to find feature points in meshes by applying image processing techniques to 3D mesh. Guskov [7] applied geometric filter for 3D meshes to enhance the representation of high frequency region. Hubeli [10] proposed a method to find feature edges in a 3D mesh that gives edges weights based on difference value between two normals and then fits polynomials to intersection line between parameter plane and mesh. A drawback of this method is that users must select a threshold value that cannot be found
intuitively. Besides, both of them are not satisfactory in terms of the time spent to extract feature points from 3D mesh.

Facial animation [13, 14, 17, 18] can be embodied by various methods, among which we apply one of performance-driven animations, expression cloning to our simplified results. Expression cloning [14] is a method to retarget facial expression animations to unlike models. So to speak, it adapts approximate animation of a model to another. It enables users to apply the same animation to various models. The anatomical knowledge of faces are not needed, because this technique uses motion vectors.

In this paper, we propose a different approach to model a coarse but feature-preserved head model for facial animation. As we told before, a simplified facial model must be proper to be animated. At first, conspicuous pixels are extracted from a texture image using image processing technique, because those features such as lip line and eyebrows are easy to lose their details by geometry-based simplification. Those pixels find the corresponding vertices by simple parameterization, and then among those vertices the feature points of a facial model are selected through discrete curvature filtering within the bounding area, which is automatically assigned based on pre-knowledge of facial animation. At last, giving weight to one-neighbor vertices of feature points, our algorithm simplifies a facial model preserving features. The proposed method is comparatively simple and easy, and helps to control the quality of a coarse facial mesh generated by feature-preserved simplification.

2 Feature Extraction

The goal of our framework is to extract feature parts in a head scan data, simplify the model with preserving feature points, and to animate the result mesh. Here, we define feature parts and feature points in the face, which include eyes, eyebrows, a nose and a mouth. It is important to simplify them, still keeping their features. Feature points are defined as points that stand out in a facial model such as sharp or ridged points.

2.1 Extraction of Feature Points from a Texture Image

Range scanning system provides a texture image as well as a geometry data. Previous methods have focused on geometric information in order to find feature points and edges, but they failed to obtain superior results and satisfying implementation time. Our method enables us to detect contour edges from texture information and to use them easily.

Edge detection is a commonly used operation which reinforces and extracts edges of images. An edge is a boundary line where images are bordered, or overlapped with the background. We call them edge parts including changes of chroma or gray scale in image. While the Sobel, Prewitt, and Canny algorithms [6] are commonly used for edge detection, but each algorithm has both advantages and disadvantages. Most frequently used the Sobel edge detection is more
sensitive to diagonal lines in edges than to horizontal or vertical lines. The Prewitt appears to have the same result value as the Sobel, but has faster response time. It also exhibits a similar form of a mask to that of the Sobel edge detection but gives a different weight to boundaries of brightness. Given different weight value of mask, the contour edges are embossed. Due to its fast response time and better detection of contours, we employ the Prewitt algorithm in our implementation.

We assign threshold value on R, G, and B to extract prominent pixels by Prewitt edge detection. R, G and B have advantages and disadvantages respectively. One of the most efficient ways is to find all feature points by union of things found from each channel. This approach requires a user-defined value. We assign different threshold values on R, G, and B respectively, and extract the suitable contour edges. Following equation is to assign whether a pixel \((u_j, v_j)\) is a feature or not:

\[
isFeature(u_j, v_j) := |r(u_j, v_j) - \text{threshold}| + |g(u_j, v_j) - \text{threshold}| + |b(u_j, v_j) - \text{threshold}|
\]

In the right figure of Figure 3, white marks indicate that feature edges are detected and black ones the opposite.

![Fig. 3. Simple parameterization of 3D mesh (right) on the texture map (left) using texture coordinates.](image)

After the edge detection process, we must find the 3D coordinates corresponding to the feature edges. An intuitive method is to parameterize 3D geometric coordinates on a texture map with respect to their texture coordinates. In Figure 3, the vertex coordinate \((x_i, y_i, z_i)\) is parameterized on the texture image using the texture coordinate \((u_i, v_i)\). Sometimes pixels of the part marked white are not always on the vertex. The feature pixel \((u_j, v_j)\) is on the edge in Figure 3. In this case, a new vertex has to be inserted precisely to preserve the exact feature line. However, simplification algorithm does not allow the increase of the number of vertices or faces. Therefore, we choose another way. The feature pixels find the closest texture coordinates of a vertex, and then its corresponding vertex coordinates is assigned as a feature vertex. To preserve feature lines, our simplification algorithm puts more weight to both feature vertices and their one-neighbors.
Equation (1) is for a marked pixel \((u_j, v_j)\) to find the closest vertex's texture coordinates \((u_k, v_k)\).

\[
\text{theClosestVertex}(u_j, v_j) = \min_k \sqrt{(u_j - u_k)^2 + (v_j - v_k)^2}
\]

Therefore, because of the marked pixel \((u_j, v_j)\), the vertex \((x_k, y_k, z_k)\) is assigned as a feature point, and then both this vertex and its one-neighbors get more weight for simplification to preserve features in Figure 3.

3 Discrete Curvature Operator

The points extracted from a texture image can be classified as two groups. One contains points located on the important part and the other points that is not important or noise. Here, we use feature points in a texture image and find the best facial feature such as eye, nose, and mouth using discrete curvature operator.

3.1 Bounding Area on a Facial Model

Feature points should consist of only feature parts in the facial model. Hence, we make a bounding area in the facial model, and remove points located on the outer wall of the bounding area.

![Fig. 4. Generation of a bounding area.](image)

Firstly, the local axes of the model are fitted to the world axes to find two points in the facial model. The head top is positive \(y\)-direction, the right ear \(x\)-axis, and the nose tip \(z\)-axis. The head top is the vertex of the highest \(y\)-coordinate. The nose tip is the vertex of the highest \(z\)-coordinate. So we find two points (the head top and the nose tip vertex). Next, we use the two points found above in the facial proportional method (The facial proportional method is acquired by analyzing [8]) and find the necessary points for making a bounding area. Since six points found by the facial proportional method are not the points from real coordinate, we find the points with the shortest distance through equation (1). The following describes the facial proportional method:
Step 1 **Find a point on forehead**: a point on the forehead can be found using a center point of the line segment connecting the points of the head top and of the nose tip. (We find points that become 1.5 times the length when the facial model has hair.) See Figure 4 (left).

Step 2 **Find a point between nose tip and jaw top**: we find the distance from the point on the forehead to the nose tip, and the distance is acquired between the nose tip and the jaw top. The point of the jaw top can be located on 2/3 of the distance. See Figure 4 (left).

Step 3 **Find end points of forehead**: we connect a point on forehead to the nose tip, connect a line segment, and rotate it via the nose tip and find the end points of the forehead. The rotation is based on the following formula (Upper line: about 30 degrees, Lower line: about 35 degrees; these rotation degrees are experimental results). But there is no consideration for z-axis; it can find a point in the back of the head. We find a point of the nose tip and a point of the back of the head, get the average of the two points and find a side for positive direction. See Figure 4 (left) and (center).

Step 4 **Find end points of the jaw**: connect a point on the jaw top to the nose tip, connect a line segment, and rotate it via the nose tip and find the end points of the jaw.

Step 5 **Generate a bounding area**: we generate a bounding area that connects the five points. See Figure 4 (right).

### 3.2 Curvature Filtering

After removing the outer points of a bounding area, remaining points are either feature or noise points. We remove noise points, and thin down the feature points. To do this, we compute the mean curvature of the points extracted from a texture image. To speed up the later computation, we selectively choose some points among the feature points, called curvature filtering. First, we remove the small curvature of points in the mean curvature sense. The equation for computing the mean curvature of a point $v$ extracted from a texture image is as following:

$$\text{curvature}_{\text{sum}} = \sum_{i \in N_v} \frac{1}{s} \sum_{e \in d_i}$$

where $N_v$ is a set of one-neighbor vertices of the feature point $v$, $e_i$ is an edge of a vertex, dihedral angle $d_i$ is an angle between two adjacent surface normals, $A$ is the sum of areas of one-neighbor faces $A = \sum_{i \in N_v} A_{e_i}$.

We compute curvature about extracted points from a texture, divide the number of points extracted from texture and get curvature average after computing, and remove vertices below the curvature average, because they do not form feature marks in the facial model. We define that the feature part with bigger curvature in the facial model. We extract feature points by finding the contour of image and these points are computed by curvature operator. We find
feature points using curvature operator. The following is the pseudo-code of the filtering process described above:

```java
void extract_from_image_points(list)
{
    curvature_average = curvature_sum/#of_extracted_points_from_texture
    for all extracted vertices from a texture image
        if(curvature(v) > curvature_average)
            v1 = vertex.neighbor(v);
            v2 = vertex.neighbor(v1);
            if(v1 == v2)
                vertexlist.insert(v1);
            else
                remove_from_feature_points(v);

} 
```

4 Feature-Preserved Simplification

This method is based on QEM (Quadric Error Metrics) [5], which uses the iterative edge contraction. The QEM defines a quadric $Q_f(v)$ on each face $f$ of the original mesh, which equals to the squared distance of a point $v = (p) \in \mathbb{R}^3$ to the plane containing it. For each vertex $v$ of the original mesh, the sum of quadrics on its adjacent faces weighted by face area is assigned. After each edge contraction $(v_1, v_2) \rightarrow \pi$, the position of new vertex $\pi$ is assigned by minimizing the quadric, and then the next edge contraction is chosen as the one with the lowest such minimum. If a vertex is an auto-detected feature vertex, then we give larger weight onto its squared distance to prevent removing such a feature vertex, which is important for facial animation and may have a small value in quadric error. A feature point $(v_f)$ and its neighbor vertex $(v_j)$. When an edge $(v_1, v_2)$ is contracted with associated quadrics error $Q = w_{v_1}Q_{v_1} + w_{v_2}Q_{v_2}$, the position of $\pi$ is selected to minimize $w_{v}Q_f(v)$.

![Fig. 5. Feature-preserving simplification. (left) Curvature filtering. (center) Geometric model. (right) Textured model.](image-url)
We must consider texture mapping during mesh simplification, because most facial model has a texture image. Our method can achieve good simplified facial mesh considering a texture image and preserving feature points. Before the total error metric $e_i$ is calculated, the weight term of the texture attribute is added to $Q^t(v_j)$. The extended QEM defines a point $v = (p) \in \mathbb{R}^5$.

$$e_i(v \in \mathbb{R}^5) = w_i Q^t(v_i) + \sum_{j \in i} w_j Q^t(v_j)$$ (3)

Figure 5 (left) is the result that uses curvature filtering and removes feature and noise points, curvature filtered feature and its neighbor points. (center) and (right) are the results that use 3 and simplify feature-preserved model.

5 Results and Applications

We tested our algorithm on a PC with a 1.7GHz Pentium 4 processor. The results are shown in Figure 6, which are images from Mr. Shim model. You can see that the feature-preserved simplification can improve the quality of the model better than the QEM-based simplification. Table 1 lists the data size and execution times. In addition, we simplify the model and use Expression Cloning for more effective animation. Expression Cloning [14] is a technique to retarget a model's expression to other models'. In our experiment, opening the mouth was challenging because there is no point on the lips in the scan data due to the occlusion. Therefore, we made additional points around the mouth to generate lip animation. As you can see in the Figure 7, feature-preserved simplification works even more effectively in the facial animation. Figure 7 shows images made by the proposed method and they present a superior animation quality of a simplified model.

6 Conclusion and Future Work

We have presented a new method to create a coarse head model from a dense scan data by an image-based feature detection technique and the feature-preserved simplification. It is a hard and time-consuming process to find feature points and edges automatically from geometric models. But this method is fast, easily extracts a geometric model's feature points from a texture image, and simplifies the facial model while preserving feature points. In addition, this method can be used in mesh editing, facial animation, and so on.

Despite the advantages mentioned, we could not find exact the boundaries of lips when filtering the feature points extracted from an image using discrete curvature, which need to be studied as a future work. Designing a progressive scheme for generating feature points based on a facial mesh is another challenge.

Acknowledgments. This study was supported by a grant of the Korea Heath 21 R & D Project, Ministry of Health & Welfare, Republic of Korea, (02-PJJ-PGI-21400-0001)
Table 1. Data and Running Time (sec.)

<table>
<thead>
<tr>
<th>Number</th>
<th>Mr. Shim.</th>
<th>Mr. Choi.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input # of faces</td>
<td>10,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Extracted points from texture</td>
<td>0.0038s</td>
<td>0.0019s</td>
</tr>
<tr>
<td>Curvature filtering</td>
<td>0.047s</td>
<td>0.053s</td>
</tr>
<tr>
<td>Feature-Preserved Simplification</td>
<td>2.11s/600faces</td>
<td>2.00s/600faces</td>
</tr>
</tbody>
</table>

References

Fig. 6. Comparison between QEM and Feature-preserved simplification (Left to right faces: 10,000, 3,000, 1,500, 600).

Fig. 7. Comparison of facial animation. The face model mesh consists of 600 triangles.