Realistic Real-time Facial Expressions Animation via 3D Morphing Target

Mengzhao Yang
School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China.
School of Computer Information Engineering, Heilongjiang University of Science and Technology, Harbin, China
Email: yangmengzhao@gmail.com

Kuanquan Wang and Lei Zhang
School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China.
Email: wangkq@hit.edu.cn

Abstract— **Realistic facial animation can enhance the immersion characteristic of 3D games. This paper proposes a 3D morphing target method based on GPU for real-time animation of facial expressions. We employ texture mapping to obtain a realistic appearance, and design a fast morphing process to achieve a relatively high FPS by modifying the morph data structure and implementing the morphing algorithm in the shader. The algorithm can be realized and divided into seven steps including creating morph target expressions, evaluating difference vector, initializing morph data structure, loading data into VBO, linking attributes to shader and rendering facial expressions. The rendering system also gives us convenient interaction with the digital character and realizes quick expressions shift for practical application. Experiment results show that the proposed approach yields realistic real-time expressions animation.**

Index Terms— **morphing target, real-time, facial expression, realistic appearance, digital character**

I. Introduction

Realistic expressions animation from digital character can enhance immersion of computer games and give lifelike in film production, computer-aided instruction and other fields [1]. As growing needs for interactivity, the main challenge of computer-based expressions is not only to approximate expressions to give a realistic looking but also develop efficient method to realize it for real-time and implement it easily, so that it can be well integrated with existing GPU pipelines for practical application.

Therefore, we intended to research how to generate realistic facial expression animation using 3D morphing target technology. Furthermore, we hold that the interactive applications should be taken into account, so we decrease complexity in computation and obtain a high frames per second (FPS) to satisfy the needs for 3D game by exploring the parallelism features of the current GPUs.

II. Related Works

Several facial expressions animation algorithms have already been introduced, primarily in psychology and computing research.

Corresponding author: Kuanquan Wang.

Ekman and Friesen [2] firstly introduced psychological studies into facial perception which have helped to identify universal expressions. Then, they proposed a Facial Action Coding System (FACS) for describing facial expressions [3]. The system described the set of all possible basic actions (Action Units) performable on a human face and their effects on facial expressions. FACS has not only informed subsequent studies into observation, but has also informed animation practice and computing projects. Then facial animation can be quickly generated using morphing technology between neutral expression and target expression which is conventional method in both computing [4] [5] and psychology [6] research. They also proposed some effective ways to simplify the transition between neutral expression and target expression, and acquire a fast morphing. Moreover muscle model can generate 3D facial animation by simulating movements of facial muscles [7]. Based on the morphable model, a 3D facial animation model is presented and can effectively generate highly realistic facial animation sequence automatically [8]. Later, Pighin et al. [9] developed a system to reconstruct 3D human face through multiple human images. However, these techniques are not completely investigated into the real-time realization for interactive applications and can't meet the needs of 3D game nowadays.

On the other hand, Yang et al. [10] designed a simple and intuitive facial expression generation system to generate vivid 2D facial expressions. Later, based MPEG-4, Dai et al. [11] proposed a method for human face morphing and expression synthesis by operating a picture of neutral human face. Tang et al. [12] rendered a number of facial images in advance, and continuously displayed the images for facial animations. Recently Shechtman et al. [13] implement a new method for image morphing where the traditional warp and blend approach was replaced with a regenerative approach. The approach does not require manual correspondence, and generates compelling results even when the images are of very different objects. Fu et al. [14] research the human facial features localization with expression and arbitrary face pose in complex background. Wang and Sun [15] analyze

doi:10.4304/jsw.8.2.418-425

the expression features using the optimal kernel marginal fisher. However, these techniques are applicable only for 2D applications and can't be applied to 3D game development.

Therefore, real-time animation for 3D facial expressions is very practical and which is the main interest to our research. In this paper, we focus on realizing and optimizing the process of 3D facial morphing between neutral expression and target expression. Combining the real-time rendering technology based on GPU, we can obtain a high FPS at interactive rates. Also we implement a texture mapping between a texture map containing the eyebrow, teeth, tongue and eyes, and the corresponding position of face, which greatly enhances realistic appearance of face.

The rest of this paper is organized as follows. The proposed method for real-time facial rendering animation is given a detailed description in section III. Experimental results are showed in section IV, and finally conclusions are given in section V.

III. Real-time 3D Morphing Target Method

In order to satisfy the needs for interactivity in 3D game, we implement the real-time morphing target method and acquire a realistic facial animation via programmable vertex and fragment shaders on the GPU. The rendering procedure of our method is designed to realize the facial rendering easily and effectively. Also we modify the morph data structure in the application, so we can easily program the rendering procedure and improve performance for real-time rendering. Using the merits of vertex buffer object (VBO) and per vertex attribute (PVA) provided by the GPU, we can accelerate the rendering and achieve a high FPS when realizing transition between different expressions.

A. Morphing Target Method

Morphing target is a way to modify a mesh with more control than bone-based skeletal animation. A static morphing target is a version of an existing mesh that is slightly different in some way. It can present the process of seamlessly generating a mesh by combining the vertices from a neutral mesh with the equivalent vertices from one or more pose meshes [16]. For example, you can create a smiling version for a character, and import that as a "Smile" morphing target. Then in the game you can apply this morphing target to modify the vertices on the face and make your character smile, but with a great deal of control over how each vertex moves.

The objective of mesh morphing is to perform an interpolation on a per-vertex basis that allows different points to be combined. Therefore, difference vectors delta P are firstly calculated between the neutral head mesh point and the target head mesh point as:

$$\Delta P_i = P_{target} - P_{neutral} \tag{1}$$

where P_{target} is the position of target head mesh, and $P_{neutral}$ is the position of neutral head mesh.

This set of per-vertex difference vectors can be thought of as a mesh of vectors. These difference vectors contained in this mesh not only cover the vertex positions, but also cover other properties such as surface normal, texture coordinates, and tangent vectors. We can adjust these sets of difference vectors by weights and add them to the neutral vector [17]. So creating the final vertex is as simple as computing a weighted sum of all the blend shapes for the present animation as:

$$P_{final} = P_{neutral} + \sum_{i=1}^{k} w_i * \Delta P_i \tag{2}$$

where P_{final} is the final position of morphing mesh, and w_i is weight to adjust the difference vectors.

Finally, we can change the weights of the per-vertex difference between a neutral pose and a target pose to modify the deformation, and acquire the animation of facial expressions.

Traditionally, mesh morphing animation has been prohibitively expensive for complex meshes because it was performed on the CPU, so it is not applicable to real-time 3D game development. In this paper we can now realize interactive rendering for facial animation via graphics hardware GPU, because it has replaced the traditional fixed-function pipeline with programmable vertex and fragment shaders.

B. Real-time Rendering Procedure

All animation features are done totally on the GPU except some data pre-processing. Fig. 1 illustrates the flowchart of the proposed real-time rendering system. From Fig. 1 seven steps are needed to create a real-time animation of facial expression.

- Firstly, we should use the same neutral head mesh to create different facial expressions in Maya or Bend software. Here we create four facial expressions as morphing target.
- Then we can evaluate the difference vectors delta P between the neutral head mesh and the target head mesh as in (1).
- After evaluation of difference vectors we may initialize morph data structure in application through reading into neutral head mesh and all the difference vectors delta P.
- Then we would load all the data into vertex buffer object using massive parallel processing units provided by the GPU.
- Then we link all the attributes from application to the GLSL Shader.
- Later, we can employ weights to adjust the deformation in the vertex shader as in (2) and evaluate each pixel color in the fragment shader.
- At last, we can render the facial expressions in real time, and implement some interactions and shifts between different expressions.

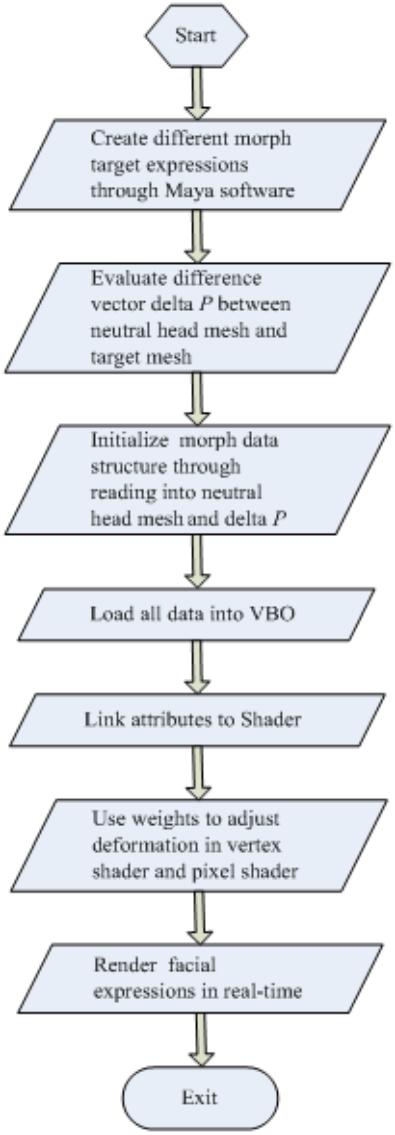

Figure 1. The flowchart of our rendering system

C. Modification to the Morph Data Structure

In order to realize the program easily and improve the real-time performance, we pre-process the difference vectors delta P obtained in the second step of the rendering system. Firstly we compute and store all the delta P data including position and normal of difference vectors into a morph data file. Because we evaluate difference value between the neutral mesh and the target mesh. so this file only includes bunch of vector data which has three float numbers in the direction of three XYZ axis. The file which includes bunch of vector data can be temporarily stored onto the disk and later read into the application.

The data in the morph data file can be partly shown in Table 1. Each row has three float numbers in the direction of three XYZ axis, and vertex value and normal value appear alternately in every row.

Later, we only need to initialize morph data structure in the application by loading neutral obj data and reading all the data in the morph data file using massive parallel GPU processing units. The code for initializing morph data structure can be partly shown in Algorithm 1.

TABLE I.
PART OF DATA IN THE MORPH DATA FILE.

X	Y	Z
.....	.....	.....
0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000
0.03291300	-0.22675014	-0.42169952
-0.00348501	0.09945184	-0.20722115
0.00527000	-1.03810000	-0.06999969
-0.01683601	0.13125449	-0.31993321
0.00726300	-2.56822014	0.11532021
-0.10191403	0.13250518	-0.34031007
-0.02104999	0.03374004	-1.15294981
-0.13700302	-0.48498315	-1.45890927
.....	.....	.....

Algorithm 1 Reading the Morph Data File for Initializing Morph Data Structure

Input: The morph data file which stores all the delta P, fin; The buffer which temporarily stores the morph data, buf; The length of max string for reading morph data, $MAXSTR$; The difference value of each vertex, $vv->mv$; The difference value of each normal , $vv->mn$;

Output: The object of morph data structure containing all the data such as vertex and normal of morphing target and neutral mesh, vv;

1: **repeat**
2: $fgets(buf, MAXSTR, fin)$;
3: $sscanf(buf, v+0, v+1, v+2)$;
4: $vv->mv[0] = v[0]$;
5: $vv->mv[1] = v[1]$;
6: $vv->mv[2] = v[2]$;
7: $fgets(buf, MAXSTR, fin)$;
8: $sscanf(buf, v+0, v+1, v+2)$;
9: $vv->mn[0] = v[0]$;
10: $vv->mn[1] = v[1]$;
11: $vv->mn[2] = v[2]$;
12: **until** $finish\ reading\ all\ the\ morph\ data$;
13: **return** vv;

Finally, the morph data structure has included all the position and normal of neutral mesh and difference vectors delta P, so we can directly implement the interaction between the morph data structure and the shader for real-time rendering.

D. Vertex Buffer Object and Per Vertex Attribute

Vertex buffer object (VBO) is a big chunk of data to hold all necessary vertex and normal vector data for graphics hardware, which can provide method for uploading these data to the GPU for non-immediate-mode rendering. VBO can make data sit in the video device memory rather than the system memory, so it can be rendered directly by the video device and don't worry about bus bottleneck as long as those VBO fits to the

capacity of video memory. We use this merit of VBO very nicely, and stream the vertex and normal vector data to structure $morph\_vert$ as the storage at once. Therefore we can enormously increase GPU performance and acquire real-time morphing facial animation.

There are several ways to manipulate vertex on the GPU using shader. In this paper we use per vertex attribute (PVA) which is one of the most simplest solution. It packs user defined data along with common vertex data set. Improvement of our work here is to pack this lots of numbers in the VBO, so that we can stream it instead that CPU sends it one by one.

E. Shader Implementation

Using weights to adjust the difference vectors, morphing algorithm accumulates difference of vertex position from neutral to target expression as in (2). Both position and normal of each vertex on the mesh are interpolated at the same time. Algorithm 2 shows part code of vertex shader to compute position and normal for morphing as follows.

Algorithm 2 Realization of Morphing Target in the Vertex Shader

Input: The normal of one vertex on the 3D head mesh, gl_Normal; The weight to adjust the morphing between neutral mesh and i^{th} target mesh, $morphWeight(i)$; The difference vector of normal coordinate between neutral mesh and i^{th} target mesh, $normalMorph(i)$; One vertex on the 3D head mesh, gl_Vertex; The difference vector of vertex coordinate between neutral mesh and i^{th} target mesh, $coordMorph(i)$;

Output: Final normal and position value on the head mesh, $Normal$ and $Position$;

1: $n = morphWeight0 * normalMorph0 + morphWeight1 * normalMorph1 + morphWeight2 * normalMorph2 + morphWeight3 * normalMorph3 + morphWeight4 * normalMorph4$;
2: $n = gl\_Normal + n$;
3: $Normal = normalize(gl\_NormalMatrix * n)$;
4: $p.xyz = morphWeight0 * coordMorph0 + morphWeight1 * coordMorph1 + morphWeight2 * coordMorph2 + morphWeight3 * coordMorph3 + morphWeight4 * coordMorph4$;
5: $p.xyz = gl\_Vertex.xyz + p.xyz$;
6: $p.w = 1.0$;
7: $Position = gl\_ModelViewProjectionMatrix * p$;
8: **return** $Normal$ and $Position$;

In the fragment shader, per-pixel lighting and texture mapping is computed and applied. Algorithm 3 gives part code of fragment shader to compute color of per-pixel as follows. Finally, we can acquire each pixel color after morphing and implement an interactive rendering between shader and application.

Algorithm 3 Evaluation of Pixel Color in the Fragment Shader

Input: The texture map containing eyebrow, teeth, tongue and eyes, $faceTex$; The texture coordinate for indexing the texture map, $gl\_TexCoord$; The normal of each vertex on the mesh, $normal$; The light direction from view point, $lightDir$; The color from material of diffuse attribute, $diffuse$; The color from environment, $ambient$;

Output: Each pixel color on the mesh, $gl\_FragColor$;

1: $texel = texture2D(faceTex, gl\_TexCoord[0].st)$;
2: $n = normalize(normal)$;
3: $intensity = max(dot(lightDir, n), 0.0)$;
4: $cf = intensity * diffuse.rgb + ambient.rgb$;
5: $af = diffuse.a$;
6: $ct = texel.rgb$;
7: $at = texel.a$;
8: $gl\_FragColor = vec4(ct * cf, at * af)$;
9: **return** $gl\_FragColor$;

IV. Experimental Results

We have implemented the proposed method to give realistic animation of facial expressions in real time. With AMD Athlon II X4 Four Cores and NVIDIA GeForce GT230, we can realize the algorithm using GLSL Shader and OpenGL programming in VS2008. We can achieve frame rates of approximately 76 frames per second, which is relatively high speed and is valuable practically in the real-time 3D games development.

In order to obtain a realistic appearance of facial expression, we can use a texture map containing the eyebrow, teeth, tongue and eyes to map these skin features onto the corresponding position of face, so we can obtain some realistic features of face. Through the texture mapping and coloring these features we acquire a realistic appearance, which can enhance sense of reality in the 3D game. Texture mapping can be implemented in the fragment shader, and the texture map we used can be seen in Fig. 2.

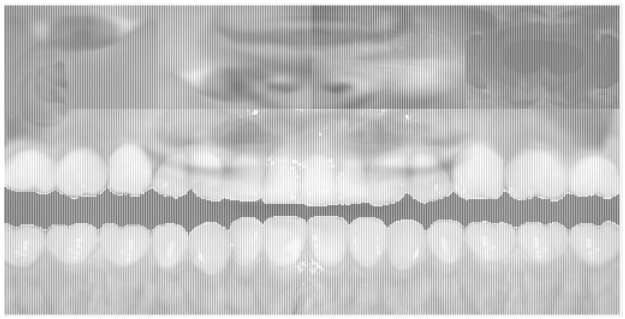

Figure 2. Texture map containing eyebrow, teeth, tongue and eyes.

The weight we use to adjust the deformation can be visualized on the 3D human face before texture mapping. Here we select three basic expressions such as "Cry", "Smile" and "Fear" to see the distribution and changes of weights as shown in Fig. 3, Fig. 4 and Fig. 5. From

three figures we can observe the different colors on the different region of face and these colors reflect big or small weights when implementing the morphing in the shader. Also we can adjust the weight value to control and view different deformation effects on the 3D human face, which greatly improves the interactivity.

Figure 3. The distribution and changes of weights when cry before texture mapping.

Figure 4. The distribution and changes of weights when smile before texture mapping.

Finally, we obtain the realistic animation of facial expressions through texture mapping and 3D morphing target in real time. Here we choose four basic expressions including ”Cry”, ”Smile”, ”Fear” and ”Blink” to render. Fig. 6 shows the dynamic transition process of facial expression on ”Cry”. Fig. 7 illustrates the dynamic transition process of facial expression on ”Smile”. Fig. 8 and Fig. 9 give the dynamic transition process of facial expression on ”Fear” and ”Blink” separately.

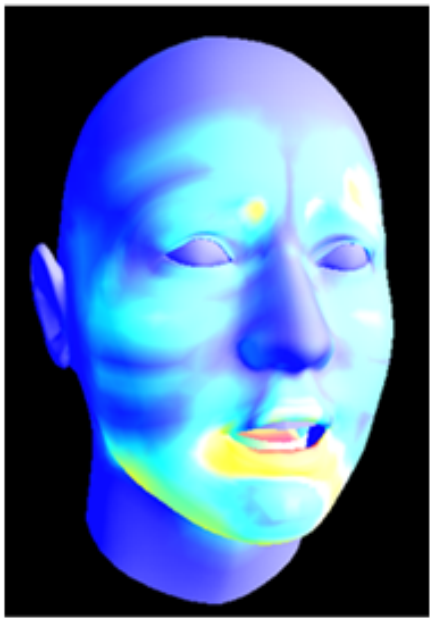

Figure 5. The distribution and changes of weights when fear before texture mapping.

From Fig. 6 to Fig. 9, we can observe the realistic appearance including eyebrow, eyes, teeth and tongue, even double-fold eyelid and eyebrow. Different expressions on the face are natural and realistic to us. We present the transition process of facial expression and give a dynamic vision. Through the quick shift between different expressions we can easily control the digital character, which can improve the interactivity between user and game. Using the modification to the morph data structure and usage of VBO , we also acquire a high FPS which gives us convenient operation with the digital character and realize quick shift from one expression to another. Our rendering system in the experiment clearly shows the effective implementation of the proposed technique. All of merits bring our method into practical value of 3D real-time game development.

V. Conclusion

We have implemented a 3D morphing target algorithm based on the GPU for realistic animation of facial expressions at interactive speed, and combining the texture mapping experiments show that our proposed technique can generate real-time and realistic animation of facial expression. The vision of expression can enhance immersion and realistic appearance of digital character. Moreover, we design a fast morph process and obtain a relatively high FPS, and our method also can be easily integrated into modern GPU pipeline, so it has an important application value for development in 3D game.

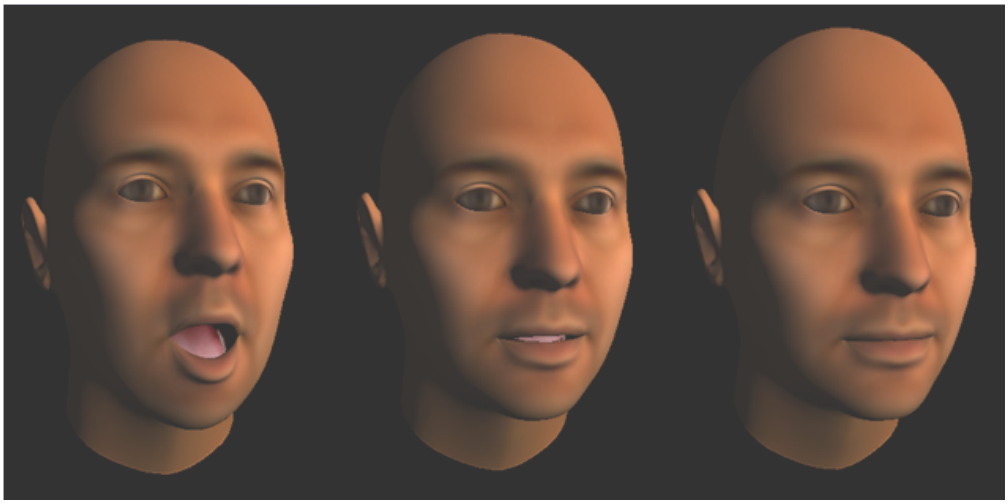

Figure 6. Animation process of expression on cry

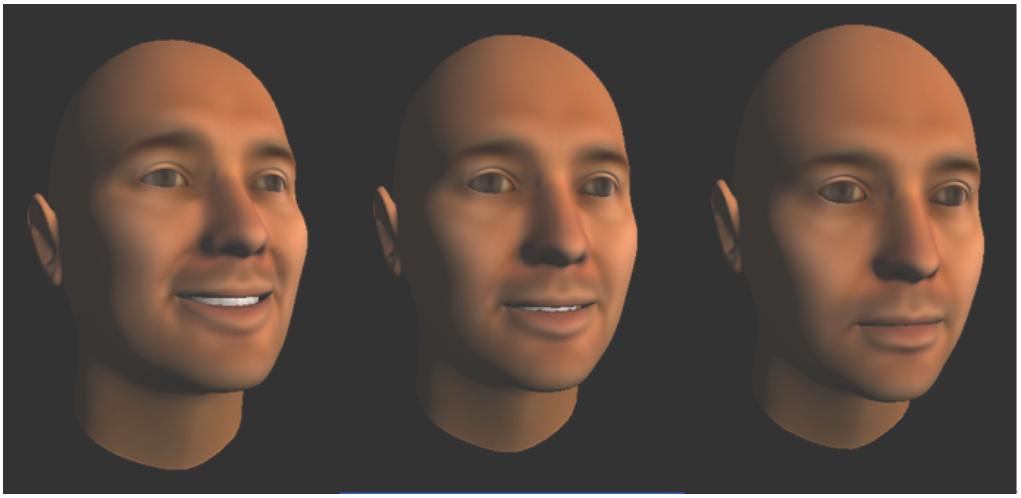

Figure 7. Animation process of expression on smile

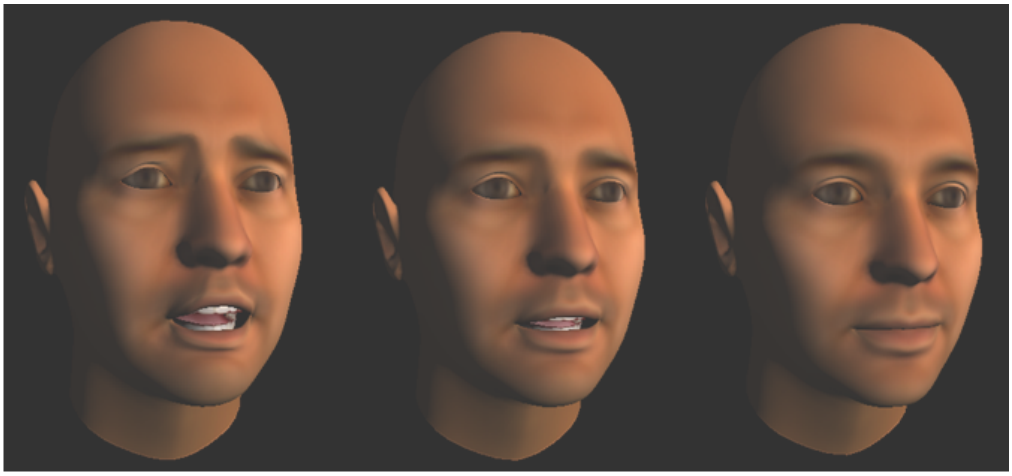

Figure 8. Animation process of expression on fear.

However, there are also disadvantages using morphing target, which is that it have to be created primarily by manual manipulation of the various vertex points in the model, rather than using many of the automatic features now inherent in many 3D modeling and animation programs.

Even though we see our work as a step in allowing designers to quickly render and interact with realistic

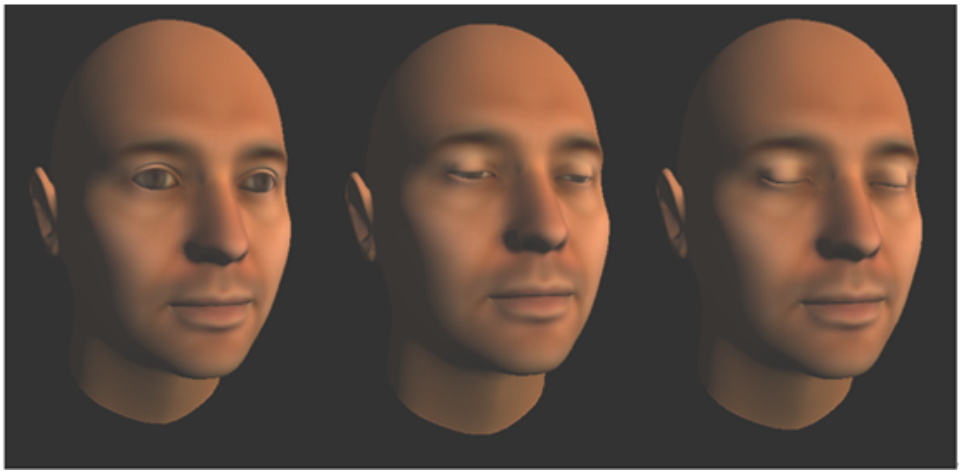

Figure 9. Animation process of facial on blink.

digital character, how the more realistic appearance is approximated is an interesting direction which we are currently working on. In future work, we will try to study the physical properties of the skin and muscles [18], and enhance realistic appearance and real-time performance.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant No.s 61173086 and 61179009, and Science and Technology Foundation of Heilongjiang Province Education Department in China under Grant No. 11551435.

REFERENCES

[1] I. Kerlow, *The art of 3D computer animation and effects*. Wiley, 2004.

[2] P. Eckman and W. Friesen, "Unmasking the face: a guide to recognizing emotions from facial clues," 1975.

[3] P. Ekman and W. Friesen, "Facial action coding system: A technique for the measurement of facial action," *Manual for the Facial Action Coding System*, 1978.

[4] D. Lin and H. Huang, "Facial expression morphing and animation with local warping methods," in *Image Analysis and Processing, 1999. Proceedings. International Conference on*. IEEE, 1999, pp. 594–599.

[5] H. Pyun, Y. Kim, W. Chae, H. Kang, and S. Shin, "An example-based approach for facial expression cloning. acm siggraph," in *Eurographics Symposium on Computer Animation*, 2003, pp. 167–176.

[6] B. Montagne, R. Kessels, E. Frigerio, E. De Haan, and D. Perrett, "Sex differences in the perception of affective facial expressions: Do men really lack emotional sensitivity?" *Cognitive Processing*, vol. 6, no. 2, pp. 136–141, 2005.

[7] K. Zhang, Z. Huang, and T. Chua, "A framework to customize a face model for reusing animation," in *Computer Graphics International, 2003. Proceedings*. IEEE, 2003, pp. 258–261.

[8] B. Yin, C. Wang, Q. Shi, and Y. Sun, "Mpeg-4 compatible 3d facial animation based on morphable model," in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, vol. 8. IEEE, 2005, pp. 4936–4941.

[9] F. Pighin, J. Hecker, D. Lischinski, R. Szeliski, and D. Salesin, "Synthesizing realistic facial expressions from photographs," in *ACM SIGGRAPH 2006 Courses*. ACM, 2006, p. 19.

[10] C. Yang and W. Chiang, "An interactive facial expression generation system," *Multimedia Tools and Applications*, vol. 40, no. 1, pp. 41–60, 2008.

[11] Z. Dai, H. Zhu, S. Zhang, J. Jia, and L. Cai, "Mpeg-4 based facial expression image morphing," *Journal of Image and Graphics*, 2009.

[12] Y. Tang, M. Xu, and Z. Cai, "Research on facial expression animation based on 2d mesh morphing driven by pseudo muscle model," in *Educational and Information Technology (ICEIT), 2010 International Conference on*, vol. 2. IEEE, 2010, pp. V2–403.

[13] E. Shechtman, A. Rav-Acha, M. Irani, and S. Seitz, "Regenerative morphing," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 615–622.

[14] Y. Fu, H. Yan, J. Li, and R. Xiang, "Robust facial features localization on rotation arbitrary multi-view face in complex background," *Journal of Computers*, vol. 6, no. 2, pp. 337–342, 2011.

[15] Z. Wang and X. Sun, "Optimal kernel marginal fisher analysis for face recognition," *Journal of Computers*, vol. 7, no. 9, pp. 2298–2305, 2012.

[16] R. Fernando, "Gpu gemsłprogramming techniques, tips and tricks for real-time graphics," *Recherche*, vol. 67, p. 02, 2004.

[17] T. Lorach, "Directx 10 blend shapes: Breaking the limits," *GPU Gems*, vol. 3, pp. 53–67, 2007.

[18] J. Jimenez, T. Scully, N. Barbosa, C. Donner, X. Alvarez, T. Vieira, P. Matts, V. Orvalho, D. Gutierrez, and T. Weyrich, "A practical appearance model for dynamic facial color," in *ACM Transactions on Graphics (TOG)*, vol. 29, no. 6. ACM, 2010, p. 141.

Mengzhao Yang was born in Lushan, Henan province in 1980. He is currently a PhD student in the Biocomputing Research Centre at the Harbin Institute of Technology, China. He received his MS degree in the School of Information Engineering from the Jiangnan University in Wuxi city in 2006. His research interests include realistic appearance rendering, real-time computing and game development.

Kuanquan Wang is a full professor and PhD supervisor with School of Computer Science and Technology at Harbin Institute of Technology. He is a senior member of IEEE, a senior member of China Computer Federation and a senior member of Chinese Society of Biomedical Engineering. His main research areas include image processing and pattern recognition, biometrics, biocomputing, virtual reality and visualization. So far, he has published over 200 papers and 6 books, got 10 patents, and won 1 second prize of National Teaching Achievement.

Lei Zhang was born Heilongjiang province in 1980. He is currently a PhD student in the School of Computer Science and Technology at the Harbin Institute of Technology, China. He received his MS degree from the Harbin Institute of Technology in 2007. He is aslo a lecture in the Harbin University. His research interests include 3D modeling and rendering, real-time computing and virtual reality.