A Skinning Method in Real-time Skeletal Character Animation

Tianchen Xu [1], Mo Chen [1], Ming Xie [1] and Enhua Wu [1, 2]

[1] Faculty of Science and Technology, University of Macau, Macao, China
[2] State Key Lab of CS, Institute of Software, Chinese Academy of Sciences, Beijing, China

***Abstract*—With regard to skeletal character animation, the question on how to ensure the quality of transition between key frames is of crucial importance. The lack of properly defined motion ranges based on movements would leave the animator with no choice but intervene the result based on camera perspective afterwards, thus creating a lot more work for the animator to modify or clean up the animation curves. Although a number of methods have been raised in these years, such as the Linear Blending Skinning (LBS), they may still have shortcomings in some specific cases, one of which is the obvious unnatural deformation around the joint areas. The primary investigation in this paper is directed to address the problem and improve the framework of rendering in real-time environment with satisfactory skinning effect to the aforementioned scenario, with assistance of GPU computation.**

***Index Terms*—real-time skinning; geometry shading; figure animation; transformed feedback; dual quaternion**

I. INTRODUCTION & RELATED WORK

The field of skeletal animation is gaining momentum as it provides more physical plausibility than key-frame animation. Instead of manipulating every single vertex directly, "bone" as a conceptual structure helps the whole animation model to be controlled in a more predictable manner, while consuming less memory. The characteristic of separating animation data and model mesh brings the animator with more flexibility: the same set of animation data can be applied to many models, while a model is allowed to load different sets of animation data.

In the traditional key frame animation [1] [2], we usually use liner interpolations to handle the parallel movements of vertices. However, when it comes to rotation, this approach will generate weird outcomes. Because in the process of rotation, the movement is not smooth, as it often involves accelerations and decelerations. Moreover, the trace of a vertex is an arc rather than a straight line, which means even the direction, is ever-changing. Under this circumstance, linear interpolations certainly cannot be satisfying.

By far, the best solution to this problem is using the approach of quaternion [3]. The ease of interpolating along a spherical surface will eventually generate a smoother and thus more natural outcome.

Manuscript Received on June 28, 2011
E-mail: x.universe.stars@gmail.com

Most research works mainly focus on two topics: physically based modeling and geometric methods. For physical model simulation, it is to simulate the internal structure and make motion analysis. Not only bones, but also muscles and other physical factors are considered. These methods may be employed to generate relatively realistic results. However the methods are in general restricted to off-line simulations or small domain real-time simulations. For geometry methods, there are five directions:

Fig. 1. Illustration of real-time character animation in Final Fantasy XIII [4].(Color Plate 4)

1) Vertex based blending: The method uses direct interpolation between two sampled meshes in key frame [1] [2].
2) Multi-weight Enveloping [5]*:* Since direct interpolation makes some errors in process of motion. By using multi-weight enveloping, the transformation of vertex is influenced by more than one matrix to make a relatively smooth result.
3) Skeletal Animation: Vertices are bound to bones, and the influencing weight of each bone must be specified.
4) Linear Blending Skinning [6]*:* Blending the matrices of each bone directly with weight bound.
5) Direct Quaternion Blending Skinning [7] [8] & *Dual Quaternion Linear Blending* [9]*:* These methods use quaternion to reduce the error of rotation blending.

In our paper, a set of handling process is presented to the animated character with complex motions, in taking account the consumption for skinning. Besides, it performs high efficiency of rendering by means of dual-quaternion and new characteristics of the Geometry Shading Program (GS), making our algorithm quite suitable for middle scale real-time environment.

The structure of our paper is as follows: after briefly

summarizing the recent works on the topics related to our research interest in section 1, we show an overview and the main contribution of our method, with descriptions of the overall algorithms given in section 2. Some key solutions for implementation and the testing results of our method will be presented in section 3. Finally, we conclude in section 4 with some discussion to the future work.

II. PRINCIPLES & METHODS

Our overall algorithms consist of the following stages as shown in Fig. 2:

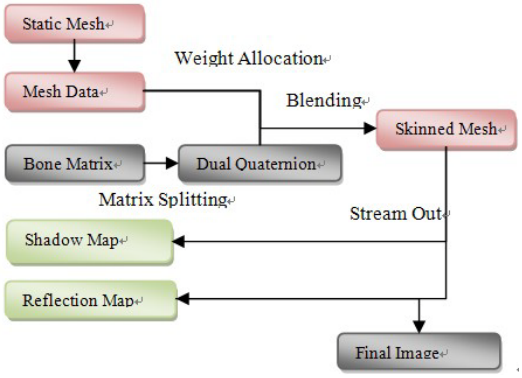

Fig. 2. Overview of our algorithms.

- Data Import (Bones, Hierarchy, and Meshes);
- Weight Allocation;
- Creation of Hierarchy in Memory;
- Updating Frame Transformation;
- Creation of Dual Quaternion from Frame Transformation with scaling factors;
- Skinning with Vertex Shading Program;
- Transformed Feedback (Geometry Stream-Out);
- Multiple Illumination Passes (Basic Illumination, Shadow, Reflection, Gridding, etc.);
- Based on the previous steps, producing a frame buffer for the whole character, via skinning and pixel shading.

2.1 Weight Allocation

In the traditional workflow, after modeling, artist needs to bind the vertices to the bones, and allocate the weight for each vertex manually. For different type of lives, the weight allocation varies, for instance:

- For snakes, fish bodies, and other mollusks, the blending area is relatively large, shown as in Fig. 3. For this case, the weight allocation principle may be expressed by

$$\begin{aligned} B(w) &= w^2\vec{B}_0 + 2(1-w)w\vec{J} + (1-w)^2\vec{B}_1 \\ B'(w) &= 2w\vec{B}_0 + 2(1-2w)\vec{J} + 2(w-1)\vec{B}_1 \\ (B(w) - \vec{p}) &\bullet (B'(w)) = 0. \end{aligned} \quad (1)$$

where B_0, and B_1 are the center position of bone, J is the joint, p is the position of the vertex. We solve w as the unknowns for weight in such case, and the related work will be reported elsewhere.

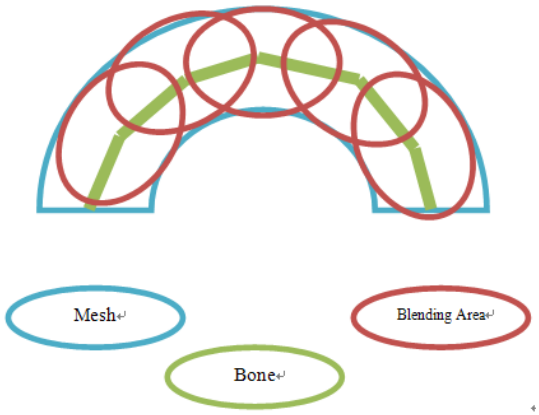

Fig. 3. Weight allocation with large blending influence.

- For limbs of human beings, vertices are almost allocated to one bone, except the joint area. Thus, the blending area is small as shown in Fig. 4. We use (2) to determine the allocation.

$$\begin{aligned} \vec{C}_i &= \frac{r(\vec{B}_i - \vec{J})}{\|\vec{B}_i - \vec{J}\|} \\ B(w) &= w^2\vec{C}_0 + 2(1-w)w\vec{J} + (1-w)^2\vec{C}_1 \\ B'(w) &= 2w\vec{C}_0 + 2(1-2w)\vec{J} + 2(w-1)\vec{C}_1 \end{aligned} \quad (2)$$

where r is the influence area radius. The result of w is clamped to [0, 1]. Our work here mainly focuses on the second type of characters, i.e., human-shape characters.

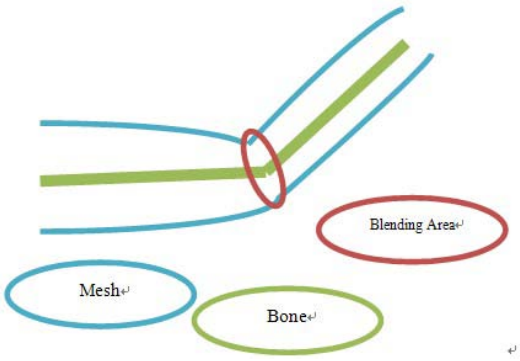

Fig. 4. Weight allocation with small blending influence.

2.2 Hierarchical Frames

Normally, a model is represented in two parts: a collection of interconnected bones, and a surface representation for the whole skeleton.

First, a hierarchy of bones is constructed and developed based on a bone called “the root”, and the rest of bones are

either its siblings or its children. When we do the indexing on the skeleton, normally we would look up for the root first, and then index from it until the specific bone is targeted.

Second, we do the transformation on those bones, so as to generate different movements. The transformations traverse from root. For each frame, we process it in the following steps:

- Transform the vertex from modeling space to the local space of the bone with the inverse of bind-pose transformation.
- Transform the vertex with local transformation of the bone.
- Transform the vertex from bone space back to world space.
- Transform the vertex with parent transformation.

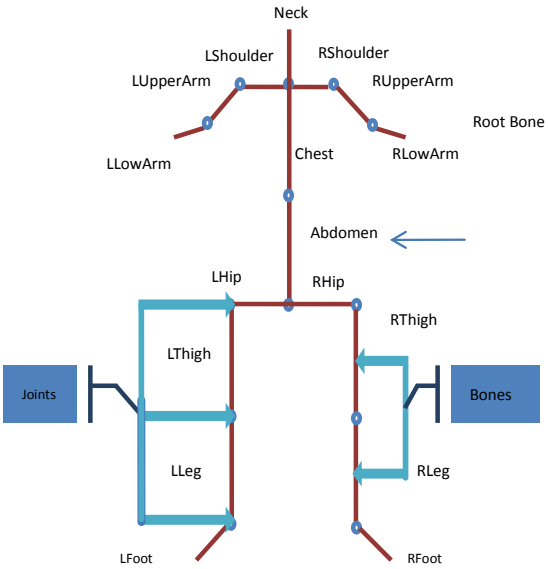

Fig. 5. Hierarchical Frames.

After each transformation, we need to update the whole skeleton so that each new movement is recorded.

2.3 Skinning

After transformations are computed for each bone along the hierarchical frames, we send the matrices to GPU constant buffer as matrix palette. Since the index of bone is bound to each vertex, in vertex shading program, the corresponding matrix can be taken by the indices in the vertex structure from matrix palette. For the vertex with n indices bound, the simplest blending of matrices is Linear Blending Skinning (LBS) [6] (3).

$$p_s = \sum_{i=0}^{n-1} w_i M_i p \tag{3}$$

where w is the weight of influencing bone matrix, M is the transformation of the bone.

However, this blending method may cause wrong twisting when the two blending bones twist by rotating with different directions. Fig. 6 shows an example.

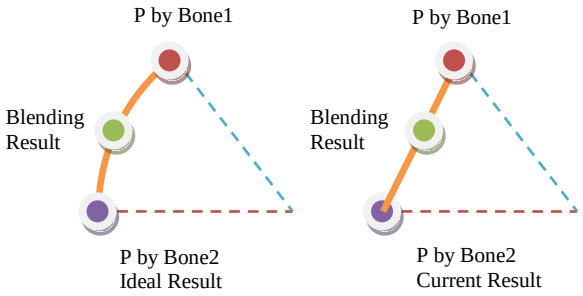

Fig. 6. Ideal Blending Result (Left) & Result by LBS (Right).

To solve this problem, L. Kavan et al. proposed the method of Dual Quaternion Linear Blending (DLB) [9]. Rotation and translation can be denoted as dual quaternion (DQ). The blending is calculated by

$$Q = \sum_{i=0}^{n-1} k_i w_i q_i$$
$$k_i = \begin{cases} -1 & \left(q_{0(r)} \bullet q_{i(r)} < 0\right) \\ 1 & \left(otherwise\right) \end{cases} \tag{4}$$

where w is the weight of influencing bone, q is the dual quaternion, and $q_{(r)}$ is the first row of the dual quaternion.

2.4 Quaternion

1) Unit Quaternion: Rotation can be represented by unit Quaternion:

$$\hat{q} = (\vec{v}, w) = (\vec{u}\sin\phi, \cos\phi) \tag{5}$$

$$R(p, \vec{u}, 2\phi) = \hat{q}p\hat{q}^{-1} \tag{6}$$

In (5) and (6), vector **u** denotes the rotation axis and rotation angle is 2ϕ.

2) Splitting Matrix: Since the modeling software commonly export matrix to represent skeletal animation data, to transform with dual quaternion, we must split the matrix into scaling matrix, rotation matrix and translation matrix.

$$M_T M_R M_S = \begin{bmatrix} S_x M_{R00} & S_y M_{R01} & S_z M_{R02} & T_x \\ S_x M_{R10} & S_y M_{R11} & S_z M_{R12} & T_y \\ S_x M_{R20} & S_y M_{R21} & S_z M_{R22} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$= [C_0 \mid C_1 \mid C_2 \mid C_3]$$
$$S = (S_x, S_y, S_z) = (\|C_0\|, \|C_1\|, \|C_2\|) \tag{7}$$
$$Ortho(M_R) = \left[\frac{C_0}{S_x} \mid \frac{C_1}{S_y} \mid \frac{C_2}{S_z}\right]$$
$$T = C_3$$

3) Matrix to Quaternion: A group of dual quaternion is denoted as (8)

$$dq = q + \varepsilon q_{(\varepsilon)} = \begin{bmatrix} q_x & q_y & q_z & q_w \\ q_{x(\varepsilon)} & q_{y(\varepsilon)} & q_{z(\varepsilon)} & q_{w(\varepsilon)} \end{bmatrix} = \begin{bmatrix} \vec{q}_v & q_w \\ \vec{q}_{v(\varepsilon)} & q_{w(\varepsilon)} \end{bmatrix} \quad (8)$$

When the rotation matrix is split, we generate a quaternion for the first row of DQ.

$$\begin{aligned} q_w &= \frac{1}{2}\sqrt{tr(M_R)} \\ q_x &= \frac{M_{R21} - M_{R12}}{4q_w} \\ q_y &= \frac{M_{R02} - M_{R20}}{4q_w} \\ q_z &= \frac{M_{R10} - M_{R01}}{4q_w} \end{aligned} \quad (9)$$

The translation matrix is transformed into a quaternion form using following calculation:

$$\begin{aligned} q_{x(\varepsilon)} &= \frac{1}{2}\left(T_x q_w + T_y q_z - T_z q_y\right) \\ q_{y(\varepsilon)} &= \frac{1}{2}\left(-T_x q_z + T_y q_w + T_z q_x\right) \\ q_{z(\varepsilon)} &= \frac{1}{2}\left(T_x q_y - T_y q_x + T_z q_w\right) \\ q_{w(\varepsilon)} &= -\frac{1}{2}\left(T_x q_x + T_y q_y + T_z q_z\right) \end{aligned} \quad (10)$$

4) Quaternion to Matrix: After blending the dual quaternion, the transformation quaternion should be transformed back to matrix form.

For unit quaternion, we generate a rotation matrix using (11):

$$M_R = \begin{bmatrix} 1-2\left(q_y^2+q_z^2\right) & 2\left(q_x q_y - q_w q_z\right) & 2\left(q_x q_z + q_w q_y\right) & 0 \\ 2\left(q_x q_y + q_w q_z\right) & 1-2\left(q_x^2+q_z^2\right) & 2\left(q_y q_z - q_w q_x\right) & 0 \\ 2\left(q_x q_z - q_w q_y\right) & 2\left(q_y q_z + q_w q_x\right) & 1-2\left(q_x^2+q_y^2\right) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (11)$$

Translation matrix is generated by

$$M_T = \begin{bmatrix} 1 & 0 & 0 & -2q_{w(\varepsilon)}q_x + 2q_{x(\varepsilon)}q_w - 2q_{y(\varepsilon)}q_z + 2q_{z(\varepsilon)}q_y \\ 0 & 1 & 0 & -2q_{w(\varepsilon)}q_y + 2q_{x(\varepsilon)}q_z + 2q_{y(\varepsilon)}q_w - 2q_{z(\varepsilon)}q_x \\ 0 & 0 & 1 & -2q_{w(\varepsilon)}q_z - 2q_{x(\varepsilon)}q_y + 2q_{y(\varepsilon)}q_x + 2q_{z(\varepsilon)}q_w \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (12)$$

2.5 Illumination

We apply Phong Illumination Model [10] to light up the character.

$$I_p = k_a i_a + \sum_{m \in lights} \left(k_d (\vec{L}_m \cdot \vec{N}) i_d + k_s (\vec{R}_m \cdot \vec{V})^{\alpha} i_s\right) \quad (3)$$

III. IMPLEMENTATION

Our system of skeletal character animation basically has two parts: preprocessing and frame rendering. For the preprocessing part, we use CPU to process the work, and for the frame rendering part, we place the main computation on GPU.

- Preprocessing.

Preprocessing is processed with data initialization, and all the relatively intensive computations should be placed here as much as possible.

1) Data Import: Mesh data is exported from 3D modeling software, which includes vertex data, index data, bone information and hierarchical frames.

2) Hierarchy Construction: We construct hierarchy to hold the index of bones and record the parent-child relation according to the hierarchical frame from the imported data. Matrix container is also created within the hierarchy structure to load the transformation matrices of each frame.

3) *Weight Allocation:* We calculate the center position of each bones and joint position according to the bind-pose matrix which transforms data from bone space into world space. Then, by solving the equations (2), we acquire the weight value to correct the weight binding of joints from modeling software. The result of weight is written into vertex buffer.

- Frame Rendering.

Frame rendering runs through the whole lifetime of the program frame by frame. The efficiency of real-time algorithm is highly demanded.

1) Hierarchy Updating: The coressponding matrices of the current frame is loaded into the matrix container, and the final matrix for each bone is calculated through the hierarchy.

2) Dual Quaternion Construction: Since most of 3D modeling sofwares export animation data in matrix form, while in our implementation, we use dual quaternion.The dual quaternion with scaling data is constructed using (7) (9) and (10). After the dual quaternion palette for current frame is constructed, we send the data in float3x4 to GPU.

3) GPU Vertex Shading for Skinning: This is the most important stage for skeletal animation, where skinning computation runs. First we blend the quaternions, and then transform the quaternion into matrix form or other proper forms of transformation. Finally we output the position in modeling coordinates.

4) GPU Geometry Shading for Feedback: This stage is to accelerate multi-pass rendering workflow for preparation. We output the transformed vertex data back into memory buffer.

5) GPU Vertex Shading: In this stage, we regard the skinned input vertex data as static mesh like other common meshes. The output is transformed into screen coordinates.

6) GPU Pixel (Fragment) Shading: This is the stage to present the illumination effect to final framebuffer or middle data to texture buffer. For illumination, color texturing is sampled combining with the decoding data from normal map, we calculate the final color with Phong Illumination Model.

3.1 Transformations with Scaling

Commonly skeletal animation rarely contains scaling bone transformation. Due to the fact that some exporter does export the animation with scaling factor (e.g. FBX format), we must consider the scaling transformation. For weight blending, we use the dual quaternion, but for final transformation, matrix form is still a direct alternative. However transform quaternion to matrix is still expensive. We use (14) to simplify the calculation in GPU [11]. Since there is no necessity to use rotation matrix and translation matrix finally, we just directly multiply the vertex position by scaling factor instead of matrix calculation, because normal-inverse-matrix generation is still an expensive issue. In details, we handle the transformation and weight blending using following steps in order:

- Rotation quaternion is stored in the first row of DQ.
- Translation quaternion is stored in the second row.
- Scaling factor is stored in the third row.
- Blend DQ using (4).
- Blend scaling factor using k=1 in (4).
- Transform vertex position using scaling factor S.
- Transform normal using 1/S.
- Transform scaled vertex using (14).

$$\begin{aligned} R(\vec{p}) &= \vec{p} + 2(\vec{q}_v \times (\vec{q}_v \times \vec{p} + q_w \vec{p})) \\ T &= 2\left(q_w \vec{q}_{v(\varepsilon)} - q_{w(\varepsilon)} \vec{q}_v + \vec{q}_v \times \vec{q}_{v(\varepsilon)}\right) \\ \vec{p}_s &= R(\vec{p}) + T) \end{aligned} \tag{14}$$

3.2 Multi-pass Geometry Rendering

In order to avoid the overhead of repeat rendering, we enforce multi-pass geometry rendering using the function Transformed Feedback. Transformed Feedback, also called Stream-Out (SO) is enabled since OpenGL 3.0 and DirectX 10 were developed. With geometry shading, a transformed vertex stream can be stored from GPU to a certain buffer for the use of the next passes.

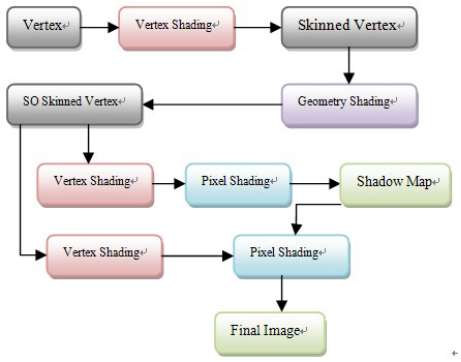

Fig. 7. Multi-pass Work Flow using Multi-pass geometry rendering (Shadow Example).

Due to the expensive consumption of skinning, especially using DLB, we try to make the skinning algorithm run as efficient as possible. Using SO, we can first skin the character only once, and then stream out the skinned vertices to a buffer. For other passes such as shadow map generation, deferred lighting, reflection, we just directly read the skinned vertex buffer as an instance for each rendering pass.

IV. RESULTS

In this section, we show the results acquired by our approach. We show a pair of results to compare the skinning blending methods and an integrated scene with complex models. Our testing machine is equipped with Graphic Card Nvidia® Geforce GT240M and Intel® CoreTM Duo CPU P7450.

4.1 Dual Quaternion Linear Blending with Scaling VS Linear Blending Skinning

The main advantage of Dual Quaternion Linear Blending (DLB) with scaling transformation is to eliminate the error morphing of joints when twisting two bones with opposite directions.

For our DLB (Fig. 8) with scaling transformation, the result is obviously improved from the one using LBS (Fig. 9), especially in the shoulder part of the character, since humans frequently raise and swing their arm, leading the shoulder twist fiercely.

Fig. 8. Skinning Result using DLB with scaling factor (shoulder is normally twisted).

Fig. 9. Skinning Result with LBS (shoulder is abnormally twisted).

Besides, the bone transformations we used in our demonstration concern scaling transformation, thus only using the dual quaternion is not enough, due to the omission of

scaling. In this case, our method also presents the solution with scaling transformation on bones.

4.2 Integrated Example

Here we demonstrate a full system including the figure with complex skeletal animation example (Sword Playing, in Fig. 10, 11 and our attached video) and other 3D models in a complex scaled scene. The example also includes many techniques commonly used in current game.

In our integrated example, the whole scene including trees and bamboos are all modeled in 3D space, with a relatively complex environment. The model statistics is as given in the Table 1. Testing with such environment shows that the algorithm and method proposed is feasible for a general gaming environment in real time operations.

TABLE. 1. ORIGINAL MODEL STATISTICS FOR THE INTEGRATED SCENE.

Name	Vertices	Faces
Character	5138	7355
Scene	13483	10521
Sword	83	161
Total	18704	17876

Fig. 10. Skinning result in an integrated scene.(Color Plate 5)

Fig. 11. Skinning result with effects (Shadow & Fake-HDR).(Color Plate 6)

V. CONCLUSION

In order to provide skinning method for organic models with satisfactory outcomes, we propose a detailed skinning procedure based on dual quaternion while providing strong mathematical validations. Our method can be easily incorporated into many existing real-time animation systems, yielding the benefits of both efficient rendering and highly-realistic results. With this regard, we combine the DLB method with some novel and fast algorithms and models, such as IK solution, motion capture, motion effect with complex character mesh, etc., to enhance the efficiency of computation and improve the interactive effects for real-time character animations.

With the benefits of transformation feedback, skinning computation runs only once, which greatly speeds up the rendering, at the same time avoiding the overhead in the multi-pass rendering workflows. For some pre/post-passes such as shadow mapping, deferred lighting, and so on, the skinned data can be instantiated by those passes.

For future applications, our work has space to improve in the aspects of automatic weight allocation. Since the types of character structures vary, manual weight allocation is still a tough work. To speed up character rigging, we need to enhance our weight allocating algorithm for broad range of applications. To improve the quality of motion, morphing, and emotion of the character, a real-time muscle simulation method with low consumption is also expected.

ACKNOWLEDGEMENT

The authors would like to thank all the reviewers for their constructive comments to improve the manuscript. The work has been supported by NSFC grant (60833007) and the grant of University of Macau.

REFERENCES

[1] J.P. Lewis, M. Cordner and N. Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation, in *Proc. 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, pp.165–172, 2000.

[2] P. P. J. Sloan, C. F. Rose III and M. F. Cohen. Shape by example, in *Proc. 2001 symposium on Interactive 3D graphics*, ACM Press, New York, pp. 135–143, 2001.

[3] W. Clifford. Mathematical Papers, London, Macmillan, 1882.

[4] FINAL FANTASY XIII, Square Enix Co., LTD, 2010.

[5] X. C. Wang and C. Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation, in *Proc. 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, pp. 129–138, 2002.

[6] M. Alexa. Linear combination of transformations, in *Conf. SIGGRAPH '02: Proc. 29th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, pp. 380–387, 2002.

[7] E. Dam, M. Koch and M. Lillholm. Quaternions, interpolation and Animation, in *Technical Report DIKU-TR-98/5*, University of Copenhagen, 1998.

[8] J. Hejl. Hardware skinning with quaternions, in *Game Programming Gems 4*, Charles River Media, pp. 487–495, 2004.

[9] L. Kavan, S. Collins, J. Zara and C. O'Sullivan. Skinning with dual quaternions, in *Proc. 2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, New York, pp. 39–46, 2007.

[10] B. T. Phong. Illumination for computer generated pictures, in *Communications of ACM 18*, vol. 6, ACM Press, New York, pp. 311–317, 1975.

[11] L. Kavan, S. Collins, J. Zara and C. O'Sullivan. Geometric skinning with approximate dual quaternion blending, in *ACM Transactions on Graphics (TOG)*, 27(4), Art. 105, ACM Press, New York, 2008.

Tianchen Xu was born in Ningbo, China, in 1988. He is expected to receive the B.S. degree in Software Engineering from University of Macau in summer of 2011. He was the research assistant in the Software Engineering Lab of University of Macau. Now he is working for the project on figure motion effects in the Motion Capture Lab of University of Macau supervised by Prof. Enhua Wu. His research interests include real-time computer graphics and animation, game engine design, virtual reality and NPR.

Mo Chen was born in Fujian, China, in 1989. He is expected to receive the B.S. degree in Software Engineering from University of Macau in summer of 2011. Now he is working for the project on figure motion effects supervised by Prof. Enhua Wu. His research interests include virtual reality, distributed system and knowledge management.

Ming Xie was born in 1988. She is expected to receive the B.S. degree in Software Engineering in summer of 2011 from University of Macau. Now she is working for the project on figure motion effects supervised by Prof. Enhua Wu. Her research interests focus on computer graphics and animation, web design and development, and marketing.

Enhua Wu undergraduated from Tsinghua University in 1970, Beijing and received his Ph.D degree in 1984 from Department of Computer Science, University of Manchester, UK. He is now a full professor both in the University of Macau, Macao and the State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing. His main interests are Realistic Image Synthesis, Scientific Visualization and Virtual Reality.