Real-Time Animation of Large Crowds

In-Gu Kang and JungHyun Han*

Game Research Center, College of Information and Communications,
Korea University, Seoul, Korea
kangin9@paran.com, jhan@korea.ac.kr

Abstract. This paper proposes a GPU-based approach to real-time skinning animation of large crowds, where each character is animated independently of the others. In the first pass of the proposed approach, skinning is done by a pixel shader and the transformed vertex data are written into the render target texture. With the transformed vertices, the second pass renders the large crowds. The proposed approach is attractive for real-time applications such as video games.

Keywords: character animation, skinning, large crowds rendering, GPU.

1 Introduction

In the real-time application areas such as video games, the most popular technique for character animation is *skinning*[1]. The skinning algorithm works efficiently for a small number of characters. On the other hand, emerging techniques for rendering large crowds[2, 3] show satisfactory performances, but do not handle skinning meshes. The skinning algorithm can be implemented using a vertex shader[4]. Due to the limited number of constant registers, however, the vertex shader-based skinning is not good for rendering large crowds. There has been no good solution to real-time skinning animation of large crowds, where each character is animated independently of the others. This paper proposes a GPU-based approach to independent skinning animation of large crowds.

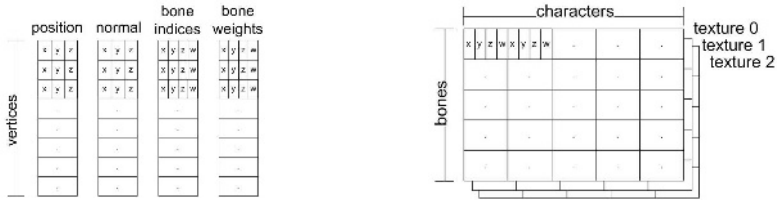
2 Pixel Shader-Based Skinning

This paper proposes a two-pass algorithm for rendering large crowds[5, 6]. In the first pass, skinning is done using a pixel shader and the transformed vertex data are written into the render target texture. With the transformed vertices, the second pass renders the large crowds.

The skinning data for a vertex consist of position, normal, bone indices and weights, and bone matrices. Fig. 1-(a) shows that position, normal, bone indices and weights are recorded in 1D textures. A vertex is influenced by up to 4 bones. The bone matrices are computed every frame, and each row of the 3×4 matrix is recorded in a separate texture, as shown in Fig. 1-(b).

* Corresponding author.

R. Harper, M. Rauterberg, M. Combetto (Eds.): ICEC 2006, LNCS 4161, pp. 382–385, 2006.



(a) vertex textures (b) bone matrix textures

Fig. 1. Texture structures for vertex and matrix data

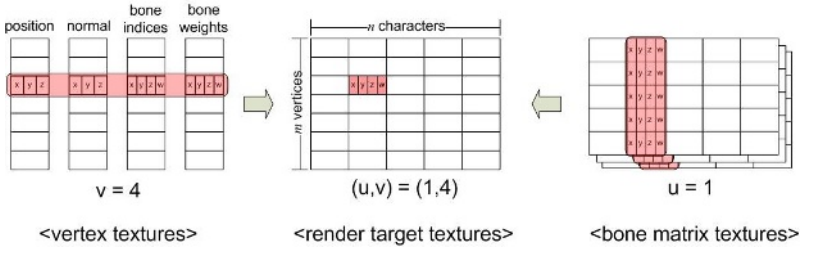


Fig. 2. Skinning and render target texture

Through a single *drawcall*, all vertices of all characters are transformed into the world coordinates, and then written into the *render target texture*. Shown in the middle of Fig. 2 is the render target texture for n characters each with m vertices. For implementing the skinning algorithm in the pixel shader, the vertex shader renders a quad covering the render target. Then, the pixel shader fills each texel of the render target texture, which corresponds to a vertex of a character.

The render target texture in Fig. 2 is filled row by row. All vertices in a row have the identical vertex index. Therefore, the vertex data from the vertex textures are fetched just once, and the cached data are repeatedly hit for processing n-1 characters.

When skinning is done, the render target texture is copied to a vertex buffer object (VBO)[7], and then each character is rendered by the vertex shader using a given index buffer. For all of the render target texture, VBO and pixel buffer object (PBO)[8], 32-bit float format is used for each of RGBA/xyzw for the sake of accuracy.

3 Implementation and Result

The proposed algorithm has been implemented in C++, OpenGL and Cg on a PC with 3.2 GHz Intel Pentium4 CPU, 2GB memory, and NVIDIA Geforce 7800GTX 256MB. Table 1 compares the frame rates of the vertex shader skinning and the proposed 2-pass skinning. For performance evaluation, view frustum culling is disabled and 'all' characters are processed by GPU. Fig. 3 shows snap-

Table 1. FPS comparison of vertex shader (VS) skinning and proposed 2-pass skinning

	soldier		horse	
# characters	VS	2-pass	VS	2-pass
1	2340	1545	2688	1571
16	580	1057	575	1179
64	200	565	163	649
256	56	200	42	219
1024	14	55	10	58
2048	7	27	5	29
4096	3	13	2	14

Fig. 3. Rendering 1,024 soldiers without LOD and frustum culling

Fig. 4. Rendering 10,240 soldiers with LOD and frustum culling

Fig. 5. Rendering 5,120 horses with LOD and frustum culling

shots of rendering 1,024 soldiers. The average FPS is 55, as shown in Table 1. In the current implementation, 3 LOD meshes are used: each with 1,084, 544 and 312 polygons, respectively. Fig. 4 shows snapshots of rendering 10,240 soldiers with LOD applied. The average FPS is 60 with view frustum culling enabled.

Finally, Fig. 5 shows snapshots of rendering 5,120 horses with LOD applied. The average FPS is 62 with view frustum culling enabled.

4 Conclusion

This paper presented a pixel shader-based approach to real-time skinning animation of large crowds. The experiment results show that the proposed approach is attractive for real-time applications such as games, for example, for rendering huge NPCs (non-player characters) such as thousands of soldiers or animals. With appropriate adjustments, the proposed approach can be used for implementing MMOGs (Massively Multi-player Online Games).

Acknowledgements

This research was supported by the Ministry of Information and Communication, Korea under the Information Technology Research Center support program supervised by the Institute of Information Technology Assessment, IITA-2005-(C1090-0501-0019).

References

1. Lewis, J.P., Cordner, M., Fong, N.: Pose Space Deformations: A Unified Approach to Shape Interpoalation and Skeleton-driven Deformation. SIGGRAPH2000 165–172
2. Microsoft: Instancing Sample. DirectX SDK. February 2006
3. Zelsnack, J.: GLSL Pseudo-Instancing. NVIDIA Technical Report. November 2004
4. Gosselin, D. R., Sander, P. V., Mitchell, J. L.: Drawing a Crowd. ShaderX3. CHARLES RIVER MEDIA. (2004) 505–517
5. James, D. L., Twigg, C. D.: Skinning Mesh Animations. SIGGRAPH2005 399–407
6. Dobbyn, S., Hamill, J., O'Conor, K., O'Sullivan, C.: Geopostors : A Real-Time Geometry / Impostor Crowd Rendering System. ACM Transactions on Graphics(2005) 933
7. NVIDIA: Using Vertex Buffer Objects. NVIDIA White Paper. October 2003
8. NVIDIA: Fast Texture Downloads and Readbacks using Pixel Buffer Objects in OpenGL. NVIDIA User Guide. August 2005