Improving the Rendering Speed of 3D Model Animation on Smart Phones

Article *in* Journal of Information and Communication Convergence Engineering · June 2011

DOI: 10.6109/jicce.2011.9.3.266

스마트폰같은게에서 3D model 랜더링

Improving the Rendering Speed of 3D Model Animation on Smart Phones

Cong Jie Ng, Gi-Hyun Hwang, and Dae-Ki Kang, *Member, KIMICS*

Abstract— **The advancement of technology enables smart phones or handheld devices to render complex 3D graphics. However, the processing power and memory of smart phones remain very limited to render high polygon and details 3D models especially on games which requires animation, physic engine, or augmented reality. In this paper, several techniques will be introduced to speed up the computation and reducing the number of vertices of the 3D meshes without losing much detail.**

Index Terms — **3D model animation, Bump Mapping, Level of details (LOD), Graphic Processing Unit (GPU), Keyframe Tweening, Vertex Shader, Vertex Skinning.**

I. INTRODUCTION

WITH the advent of handheld devices such as smart phones and tablet, 3D graphic especially 3D games become more complex which requires high quality 3D graphics and some require physics or augmented reality. The augmented reality and physics engine require intensive computational processing power. It may lead to low 3D animation rendering frame rate. The increasing of complexity in the game requires more processing power and memory.

To render animation of 3D model, the position and normal vector of the vertices of the mesh have to be updated in every frame by using interpolation between keyframes also known as tweening or vertex skinning for skeletal 3D model [1]. It consumes more computational power as the number of vertices in the 3D mesh increases. Not to mention the increasing of number of 3D meshes.

Furthermore, the 3D model which directly ported from PC version usually has very high number of vertices. They require more storage and memory. In addition, they require more processing power to compute the animation. It turns out to be unnecessary, because the screen size is smaller as compared to PC and laptop to notice the fine detail of 3D model.

In this paper, techniques to improve the rendering speed such as utilizing Graphic Processing Unit (GPU), reducing the number of vertices of 3D model without losing too much detail, and level of details of 3D model will be introduced.

II. THE TECHNIQUES

In this section, we will discuss about reducing the memory usage and improving the rendering speed of 3D animation by using several techniques. Firstly, the Graphic Processing Unit (GPU) will be introduced to process the vertices to speed up the rendering. Then, level of details (Lod) of 3D mesh will be discussed to reduce the number of vertices to be processed. And lastly, lighting optimization by using Bump Mapping to maintain the 3D model detail by reducing the number of vertices without much detail lost.

A. Graphic Processing Unit (GPU)

As the 3D model is animating, each of the vertices has to be processed to update its position and normal vector. In character animation, there are two common techniques which are vertex skinning for skeletal animation and keyframe tweening for per-vertex level animation. Skeletal animation relies on joints' position and orientation. Each of the vertices will be influenced by the weighted joints. As for keyframe tweening, it relies on pre-animated keyframe. The keyframe defines the starting point and ending point of the vertices. Interpolations are required to update the intermediate vertices position and normal vector between 2 keyframes [1].

The computation of vertex skinning and interpolation of vertex can be parallelized easily because each of the vertices does not depend on each other. So GPU can be used to parallelize the problem instead of using CPU to compute sequentially. By using this method, not only can speed up the processing by parallelize the problem, it also reduces the memory transfer between main memory and GPU memory as memory transfer causes big overhead to the process.

1) Vertex Skinning on GPU

In vertex skinning, the animation is based on the bone also known as skeletal animation. Each of the vertices position and normal vector will be calculated base on the orientation and position of the influencing bone. To

Manuscript received April 26, 2011; revised May 19, 2011; accepted June 1, 2011.

Cong Jie Ng is a final year undergraduate student in computer science from Multimedia University, Malaysia (Email: ncjlee78@gmail.com)

Gi-Hyun Hwang is with the Division of Computer and Information Engineering, Dongseo University University, Busan, 617-716, Korea (Email: hwanggh@gdsu.dongseo.ac.kr)

Dae-Ki kang (corresponding author) is with the Division of Computer and Information Engineering, Dongseo University University, Busan, 617-716, Korea (Email: dkkang@dongseo.ac.kr)

perform the vertex skinning on GPU, the bone matrices can be calculated by using CPU and it will be sent to GPU in every frame to process the vertices [2]. The bone matrices contains transformation matrix of every bone of the skeletal model. While processing the vertices, the GPU will refer to the corresponding bone matrix that influence the vertex and compute the position and normal vector of each of the vertices.

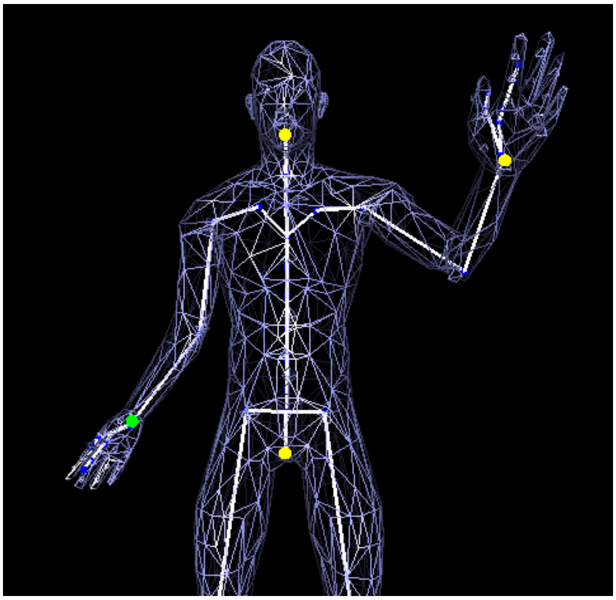

Fig. 1. Vertex skinning.

뼈대가 어떤 마디로 연결되어 있는지
joint . 어떤 영향

Vertices will be updated base on the bones (white color). Each of the vertices will be influenced by one or more bones base on the influencing weight. The model is created by using MakeHuman, the bone specification is based on H-Anim and it is loaded by using Cald3D library [3-5].

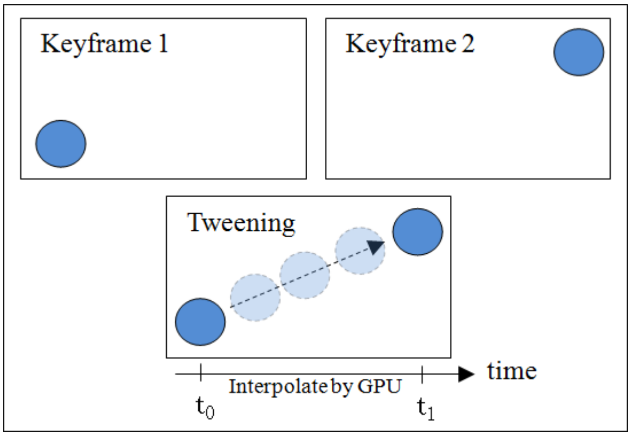

Fig. 2. Illustration of the keyframe tweening on GPU. Two keyframes are sent to GPU to be interpolated in between t0 and t1.

2) Keyframe Tweening on GPU

As the Keyframe defines the starting point and ending point of each of the vertex, it is reasonable to let the GPU to do the work to interpolate the position of the vertex until the next keyframe. It can be processed on GPU by sending 2 consecutive keyframes into the GPU in every transition of keyframes. During the interpolation time, it can reduce the CPU workload until the next keyframe. This illustrated in figure 2.

3) Application on device in practice

The latest version of OpenGL ES which is version 2.x supports programmable graphic pipeline which runs on GPU and can be programmed by using GLSL language [6]. We can utilize vertex shader to process the vertices. The GLSL language supports a number of fast built-in functions for mathematic and geometry computation. One of the built-in function can be used to do linear interpolation for keyframe tweening is “genType mix(genType x, genType y, genType a)” [7]. The equation 1 denotes equation of linear interpolation. It can be computed by using the GLSL code 1 where p, p0, p1 and t denote interpolated position, keyframe 1, keyframe 2 and time factor respectively. This approach is recommended by Khronos Group as it may be optimized for certain hardware [8].

$$p = p_0.(1-t) + p_1.t \qquad \text{Equation (1)}$$

$$p = mix(p0, p1, t); \qquad \text{GLSL code (1)}$$

Unfortunately, not all devices support OpenGL ES 2.0. An alternative to this is OpenGL ES 1.1 GL_MATRIX_PALETTE extension which uses GPU [8]. Again it has a lot of limitations. It can be used in vertex skinning, but it cannot be used on keyframe tweening as the interpolation does not require transformation matrix. The number of supported joints per vertex skinning and number of supported palette matrices are limited to a certain number base on hardware specification. The number of palette matrices can be regarded as number of bones supported. For Samsung Galaxy S it only supports up to 4 joints per vertex and 32 palette matrices and for iPhone 3Gs it supports 4 vertex unit and 11 palette matrices [2]. It is useless if the skeletal 3D model is complex which the vertex requires more than the maximum number of supported influencing joint.

The number of supported Vertex Unit can be checked by querying GL_MAX_VERTEX_UNITS_OES and the palette matrices can be checked by querying GL_MAX_PALETTE_MATRICES_OES. If the number of bones is more than the maximum supported palette matrices, it can be solved by performing the same process for more than 1 passes [2]. However, if the required vertex unit is more than the maximum supported value, we have to use CPU instead with several optimizations such as using fixed point operation and memory copy operation on native code for Android device [9].

B. Level of Details (LOD)

Level of details is very useful in speeding up the 3D rendering when the scene contains a lot of 3D meshes. It is used to manage the level of details of 3D meshes on different distance. As the distance between the object and viewport is getting further, it's reasonable to render the object with less detail because the details are too small for a pixel to render, and it is not noticeable [10]. By using this method, we can reduce the work of the rendering pipeline and reduce the number of vertices to be processed for animation purposes as discussed in previous technique.

There are mainly two types of managing level of details, discrete level of details and continuous level of details [10]. The discrete level of details is a traditional yet simple approach. It creates multiple version of the 3D model with different level of details from coarse to more complex. These 3D models are generated during offline process. It can be either created by using 3D modeling software or algorithm to simplify the 3D model into multiple version of LOD. The 3D model will be chosen based on the distance from viewport [11]. Apart from its simplicity, this approach requires more memory to store the 3D models of different LOD and the granularity of LOD is low.

As for continuous LOD, instead of generating several versions of LOD of the 3D mesh, the mesh is simplified during runtime. This approach can save memories by freeing up the unneeded vertices based on the level of details to allow more memory to load more meshes [10]. It also saves the space of storing the preprocessed meshes, which can further reduce the size of the installation file of the program or game. The number of polygon can be reduced linearly proportion to the distance between viewport. The model simplification is based on data structure of storing the 3D meshes. There are many techniques to simplify the 3D meshes based on local simplification operators. One of the operator calls Edge collapse, which proposed by Hoppe. This operators collapse an edge to a single vertex [12].

Figure 3 illustrates the Continuous LOD. On the left the LOD is 100% which is the original model, on the right is the simplified model with LOD 20% and the vertices is further reduced on runtime. This picture is taken from the screen shoot of Cal3D Library Sample Program [5].

1) Application of LOD on device in practice

One of the Open source libraries that support continuous LOD is Cal3D library. This library is widely used in games, as well as in researches. It is an open source 3D character animation library for skeletal model animation [5]. It is developed on C++, so it can be easily ported into smart phone platform like Android and Apple. In Android, it can be compiled by using Android NDK revision r5 with the variable APP_STL in Application.mk file set to stlport_static to support C++ STL because the library uses C++ STL library [13]. As for Apple it can be ported by using Objective C++ it supports C++ STL too [14].

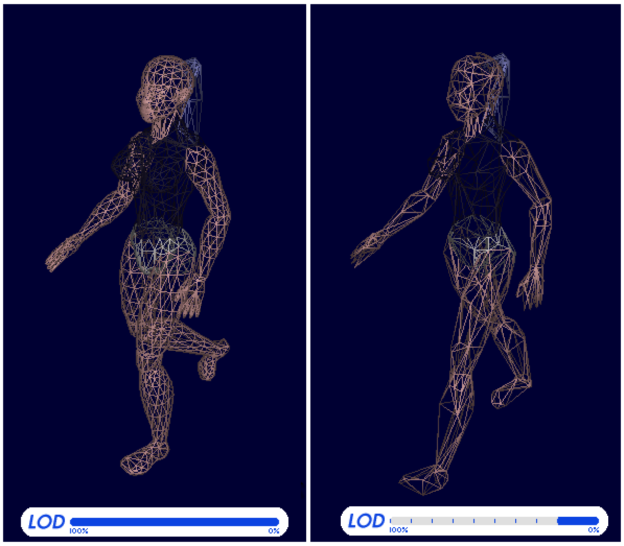

Fig. 3. Illustration of Continuous LOD.

C. Bump Mapping

It would be nice if we are able to render high resolution 3D meshes without system slowing down. But it is not practical especially on smart mobile devices with slower processing power and limited memory. By using bump mapping, we are able to render low polygon with visually high resolution 3D model as illustrated in figure 4. A 3D model with low polygon looks as if made of very high number of polygon. By using this technique, we can reduce the number of vertices without losing much detail and it in turn speed up the rendering and save the memory. It is especially useful if the surface is bumpy, such as a brick wall or a wooden wall. This technique is widely use in PC and console games. It was popularized by Doom 3, which uses low polygon mesh to achieve high resolution appearance [15].

Figure 4 illustrates bump mapping on a low polygon 3D mesh and a production a high resolution appearance 3D mesh. The 3D model is taken from Doom3, ID software [15] without texturing, and it is rendered on Samsung Galaxy S.

The bump mapping does not change the shape or the structure of the 3D model, the silhouette of the 3D model remains the same. The only thing has been changed is the per-pixel lighting. It simulates the lighting by irregulating the normal vector. It was first introduced by Blinn in year 1978 [16]. Rendering the bumpy surface geometrically will be impractical as it requires a very high resolution of 3D mesh. A bumpy brick wall can be achieved with a flat plane and a bump map. The bump map stores the normal value of the mesh surface as an image representing by

Red Green and Blue, axis x, y, and z respectively. The normal value can be either in tangent space or object space. Tangent space is the local space of the particular point on a surface. Tangent space bump map requires more computational steps to calculate the tangent and binormal. These vectors are used to transform the light direction and view direction into tangent space in every frame. It is required for the shading function to calculate the final color [2]. Although object space bump mapping is faster since no extra computation is needed, it is not suitable for deformable 3D model. It works well for static model. Hence tangent space bump map will be the choice of deformable 3D mesh.

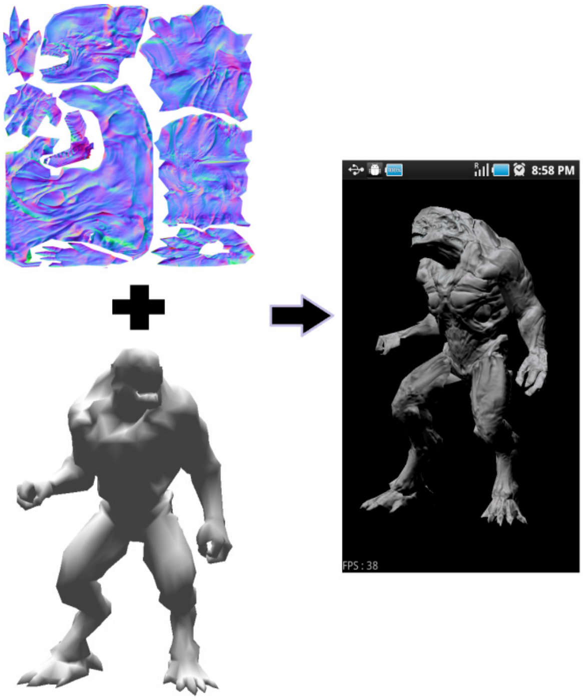

Fig. 4. Bump mapping on a low polygon 3D mesh and production of a high resolution appearance 3D mesh.

1) Application of Bump Mapping in Practice

To use bump mapping, we need a normal map of the 3D model as shown in figure 4, the bluish color image. There are many tools available that simplify the work to generate the bump map, such as NVIDIA Melody and XNormal. To generate the bump map, two 3D models with one high resolution version and another low resolution model are needed [17, 18].

As for the implementation of loading bump map into a 3D model, it can be done by using OpenGL ES 2.0 and OpenGL ES 1.1. OpenGL ES 2.0 supports programmable shader by using GLSL language. There are 2 types of shader in GLSL, which are vertex shader and fragment shader [7]. Bump mapping can be done on fragment shader to compute the pixel color based on Phong Shading model or Blinn-Phong Shading model by applying the obtained normal vector from bump map on the calculation of Lambertian reflectance of the diffuse parameter [16].

As for OpenGL ES 1.1, it does not support programmable shader instead it used fixed function pipeline [6]. So another approach has to be used. It's the same as explained in previous paragraph just that it is less control over other lighting parameter. We have to calculate the Lambertian reflectance to irregulate the diffuse lighting. The calculation of lambertian reflectance requires dot product of the surface normal vector and the light direction. This operation can be done by using GL_DOT3_RGB operation on glTexEnvf parameter with the bump map as first argument, and light direction as second argument [2]. The dot product operation on GLSL can be done in one line of code. On the other hand, OpenGL ES 1.1 needs 6 lines of codes to accomplish the same operation.

III. CONCLUSIONS

Although, the smart phones today is getting more powerful which supports duo core processors and high end graphic chipset, optimizations are needed as the demand of graphical effects and complexity of the 3D game increase. This paper presents several techniques to reduce the memory usage and speed up the rendering process with practical approaches which support most today smart phones such as Android OS and Apple IOS platform. This is required to gain better performance and higher complexity of the 3D graphic animation. As for the implementation, OpenGL ES 2.x supports programmable graphic pipeline. It is advisable to develop on OpenGL ES 2.x instead of older version of OpenGL ES if the game or 3D graphics require complex and high resolution 3D model and effects. Since more hardware today supports OpenGL ES 2.x. It is a good start for future graphic programming on smart phone devices and the programmable graphic pipeline provides more flexibility over the fix function pipeline.

ACKNOWLEDGMENT

This research was conducted in TPR3311 industrial training program of Multimedia University, Malaysia and was supported by Business for Academic-industrial Cooperative establishments funded Korea Small and Medium Business Administration in 2010. (Grants No. 00040456)

REFERENCES

[1] D. Gosselin (2002), *Character Animation with Direct3D Vertex Shaders.* Shader, Wordware Inc.

[2] P. Rideout, *iPhone 3D Programming - Developing Graphical Applications with OpenGL ES.* O'Reilly, 2009.

[3] MakeHuman. (n.d.). Open Source tool for making 3D Characters [Online]. Available: http://www.makehuman.org/

[4] Humanoid Animation Working Group. (2011, April 22). H-Anim [Online]. Available: http://www.h-anim.org/ .

[5] Cal3D. (2006, Jun). 3D Character Animation Library[Online]. Available: http://home.gna.org/cal3d/ .

[6] M. Segal and K. Akeley, *The OpenGL Graphic System A Specification.* p. 354, October 22, 2004.

[7] R. J. Simpson, and J. Kessenich, *The OpenGL ES Shading Language*. p. 72, May 12, 2009.

[8] Khronos Group (2011, March 19), GLSL: common mistakes [Online]. Available:: http://www.opengl.org/wiki/GLSL_:_common_mistakes

[9] C. J., Ng., "Speeding up the 3D model renderin on Android Device," *Proc. 35th Korea Information Processing Society Conference,* May 2011

[10] D. Luebke, et al., *Level of Detail for 3D Graphics.* Elsevier Science, 2002.

[11] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Communications of the ACM,* vol. 19, no. 10, pp. 547-554, 1976.

[12] H. Hoppe, "Progressive meshes," *Proc. ACM SIGGRAPH 1996,* pp. 99-108, 1996.

[13] Google Inc. (2011, January). Android NDK [Online]. Available: http://developer.android.com/sdk/ndk/index.html .

[14] Apple Inc. (n.d.). Options Controlling Objective-C and Objective-C++ Dialects [Online]. Available:: http://developer.apple.com/library/mac/#documentation/DeveloperTools/gcc-4.0.1/gcc/.

[15] ZeniMax Media Inc. (n.d.). *Doom 3* [Online]. Available: http://idsoftware.com/games/doom/doom3/.

[16] J. F. Blinn, "Simulation of Wrinkled Surfaces," *Computer Graphics,* vol. 12, no. 3, pp. 286-292, August 1978.

[17] NVIDIA Corporation. (n.d.). NVIDIA Melody [Online]. Available: http://www.nvidia.com/object/melody_home.html.

[18] Santiago Orgaz & co. (2011, February 9). xN bakes your maps! [Online]. Available: http://www.xnormal.net.

Cong Jie Ng is a final year undergraduate student in computer science from Multimedia University, Malaysia in 2011. He has been actively participated various programming contests in Asia regional level and national level, and earned good ranking in national level. His interests include parallel processing, 3D computer graphics, computer vision and machine learning.

Gi-Hyun Hwang received his M.S. and Ph.D. in Electrical Engineering, Pusan National University. Recently, he is a Ph.D. He is currently a professor at Dongseo University. His research interested lies in applications of intelligent control to power system, RFID, and Embedded System. E-mail: hwanggh@gdsu.dongseo.ac.kr

Dae-Ki Kang is an assistant professor at Dongseo University in South Korea. He was a senior member of engineering staff at the attached Institute of Electronics & Telecommunications Research Institute in South Korea. He earned a PhD in computer science from Iowa State University in 2006. His research interests include social network services, machine learning, relational learning, statistical graphical models, metaheuristics, ontology learning, Tower of Hanoi, multimedia systems, intrusion detection, Web firewall, and computer vision. Prior to joining Iowa State, he worked at a Bay-area startup company and at Electronics and Telecommunication Research Institute in South Korea. He received a science master degree in computer science at Sogang University in 1994 and a bachelor of engineering (BE) degree in computer science and engineering at Hanyang University in 1992.