A Methodology for Virtual Human Animation

Francisco A. Madera[1], Carlos G. Herrera[2] and Francisco Moo-Mena[3]

[1,2,3] Facultad de Matemáticas, Universidad Autónoma de Yucatán
Mérida, Yucatán 97110, México

Abstract

We propose a method to animate a 3D virtual human generating keyframes of the model and interpolating to achieve a realistic animation. The first two steps refer to the human modeling, the rigging and the skinning. The third step deals with the human in a programming environment to obtain the animation by quadratic interpolation and consists of the parallel version to display the human movements. The method has shown to produce real animations of an articulated object in an easy and fast manner, optimizing the memory usage of the GPU parallel implementation.

Keywords: *Computer Graphics, Object Modeling, Virtual Human.*

1. Introduction

There exist graphics applications such as videogames, security programs, educational systems, that include virtual humans which we call avatars. This requires modeling, animation and scene adjustments to interact with other objects. There are several tools to help: modeling and animation packages, programming languages, libraries, game engines, physics engines.

Our motivation lies on the creation of a method to facilitate the avatar modeling and animation in order to manipulate it on a programming environment to have all the geometry features available.

Three methods are commonly utilized. The first method employs 3D modeling packages, but it is limited to the environment package rules and the access to the avatar's geometric features results complicated. The second method consists on the use of a programming language where different algorithms of modeling, rigging, skinning should be implemented, so that it requires a hard work besides the expensive computation with the matrix operations required by the articulated motion. The third method suggests the use of a game engine to import the animations obtained by a modeling package, but it requires adjustments with the affine transformations to have the avatar placed in the scene in the correct manner.

An avatar is basically a geometric object formed by polygons. The avatar can be modeled by a 3D modeling package (3D package) and it can be imported by a program of a specified Integrated Development Environment (IDE). However, the animation of the avatar generated by a 3D package causes problems when it is imported by a programming language, in particular with the translation and the rotation transformations of the articulated motion.

To animate an avatar, we require a virtual skeleton to guide the vertex movements. The mesh M of an avatar is defined by a set of n polygons and m vertices $M=(\Delta,V)$, where the set of polygons (triangles) and the set of vertices are $\Delta=\{\Delta_0, \Delta_1, \ldots, \Delta_{n-1}\}$ and $V=\{v_0, v_1, \ldots, v_{m-1}\}$ respectively. A triangle is formed by three different vertices labeled in anti- clock wise order $\Delta_i=\{v_j, v_k, v_h\}$.

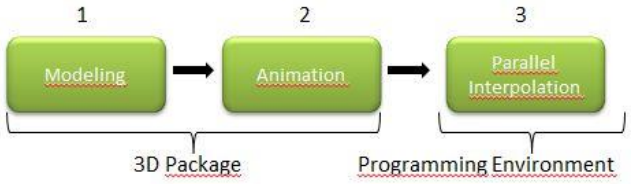

Fig. 1 Stages of the method proposed to animate an avatar in a 3D virtual environment.

The contribution of this work is the methodology to obtain an avatar animation; it starts from the modeling and finishes with the animation in a programming language environment (Figure *1*). In Section *2* previous work is presented, Section *3* details the avatar modeling process, in Section *4* the animation step is described. Interpolating mathematical foundation is shown in Section *5* and the conclusion is presented in Section *6*.

2. Previous Work

There exists a variety of object modeling techniques that can be classified in one of the following approaches: creative or re-constructive. The former refers to the use of a 3D modeling package such as Maya™, 3DMax™, Blender™, ZBrush™, GoogleSketchUp™. The model can have several features, as accurate as required; for instance, in medical applications bones and muscles are modeled, while in fashion applications [4] cloths are regarded. In the

second approach, the model is captured by using tridimensional scanner devices; even the mesh generated is a well approximated shape of the real human, there are some adjustments to be done.

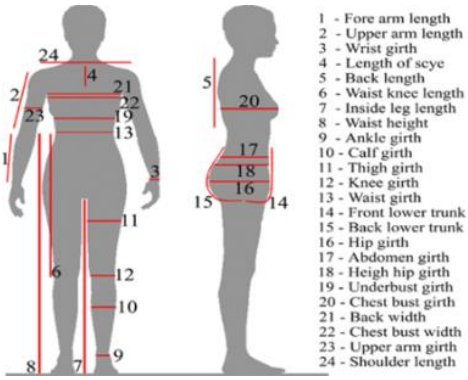

Fig. 2. Basic human body measurements for technological design [27].

In [3], the human is scanned in specific points, defined by the human anthropometric measures. Figure *2* shows the *24* parameters of the anthropometric measures specified by the ISO-7250:2008 [27] norm which refers to anthropometry and biomechanics. The skeleton creation (rigging) implies the construction of a hierarchy of the bones, where the root node is the parent of the skeleton hierarchy and mostly is placed on the center of the avatar. Thus, the vertices of the mesh can be grouped to belong to one or more bones. Similar to the work of James and Twigg [6], we want to build a geometric transformation which approximates a sequence of the avatar's meshes with the vertices displacements. The Skeletal Blend Skinning (SBS) method is widely utilized in videogames due to its ease of use and its efficiency in SIMD (Single Instruction Multiple Data) parallel implementation.

The bone's motion can be achieved by applying transformations with Quaternions, Matrices, spherical coordinates among others. In the case the skeleton is unavailable, the motion can be done with the position of the vertices along the time. The motion Capture technique (MoCap) [8] consists on capturing the information from some sensors placed on a real human, who moves freely. Some research labs upload their MoCap files in Internet, such files contain the set of vertices in time *0*, time *1*, and so on: V^0, V^1, … Interpolation strategies should be employed to make the transitions from V^i to V^{i+1}.

In this work, the skeleton and the animation are created with two 3D packages. After that, the file meshes are exported: mesh in time *0* (keyframe *0*), mesh in time *1* (keyframe *1*), etc. The files obtained are utilized in a programming environment, so that the skeleton is not required anymore. Lu et. al [10] introduce a deformation that makes the transitions among meshes along the time. Other motion strategies can be achieved using procedural deformation [9] or interpolation [10].

In this work we propose a methodology to animate an avatar in an easy and rapid way in order to be used in a programming environment and then, have the geometric features completely available. First of all, the avatar is constructed; secondly, the skeleton is built (rigging), vertices are assigned to the bones (skinning). Poses are created, walk, run, swim, etc. Thirdly, in the environment program, we employ the mesh files generated to implement a quadratic interpolation to move the avatar. 3D packages used are Blender™ and MakeHuman™; programming tools are C++, OpenGL, GLSL, CUDA.

3. Modeling (Stage 1)

The 3D model is constructed by using photos or by using a 3D scanner. The Anthropometry, the biologic science to measure the human body [4], is widely used to define the parameters of the avatar. As the first option we could create a 3D avatar using a method of digital sculpture such as edge-loop, polygon handling, curves definition, Boolean operators approach [12].

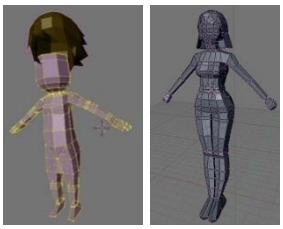

Fig. 3. Two avatars utilized in the MATYA videogame [1].

Definitely, these techniques require time and skills to have an avatar model ready. We have worked in the construction of two avatars, the mayan warrior and the mayan princess (Fig. *3*), which were used in the MATYA videogame [1]. We employed the polygonal handling method for modeling, the animation was constructed with Blender™ and it was imported with XNA™ game engine [21] for their adjustments in a programming environment using C#.

The second option would be to download an avatar from internet. For instance, the avatar illustrated in Figure *4* was modified to be adjusted to the application.

Fig. 4. The original model [19] and several modifications: traditional dress, water effect, geometric analysis.

We use the third option: an avatar modeling package where human model templates are ready. We employ MakeHuman™ [16] which allows to create virtual humans specifying some parameters: size, sex, weight, and some of the anthropometric measures specified in Fig. *2*. This model is exported in OBJ format in order to be worked with Blender™. This file contains the vertices and polygons specifications. The avatar obtained is depicted in Figure *5*.

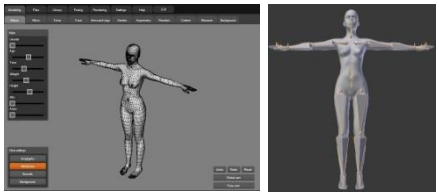

Fig. 5. Left: the avatar created with MakeHuman™ [16], right: the avatar skeleton is created with Blender™.

4. Animation (Stage 2)

The avatar skeleton is a hierarchical structure formed by connected bones. Each union or joint specifies a tridimensional transformation that is inherited by the children bones in the hierarchy. A bone has a position and an orientation (rotation). The skeleton is a hierarchy of bones, where two bones are connected by a joint. A bone is a matrix transformation due to it describes the transformation of a specified point.

The avatar imported by Blender™ is displayed and the skeleton is added automatically. Many people consider both, the avatar modeling and the skeleton construction as a time consuming process so that we have shown an easy way to create it. The skeleton contains *31* bones grouped in a hierarchy whose central bone is located in the torse of the avatar and it is labeled as *Root*. *Root*'s children are *Hips* and *Spine1* located in the center of the avatar. The *Hips* children bones are the lower extremities : *UpLeg_R* and *UpLeg_L*.

It is important to know the avatar motion to create its poses. Bones are described with matrices as shown in Figure *6*. Rotations are specified in the following expression:

$$x' = x \cos\theta + y(-\sin\theta),\ y' = x \sin\theta + y \cos\theta \qquad (1)$$

4.1 Rigging

The rigging consists on the construction and motion of the skeleton. The bone includes two articulations. The *i* bone's *4x4* matrix h_i is placed in local coordinates, and when it is modified to translate or rotate, the matrix transformation (or reference matrix) B_i^{-1} is obtained in skeleton coordinates.

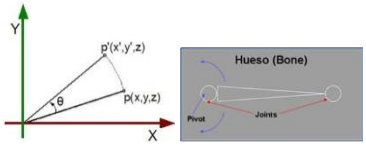

Fig. 6. Left: Bone's rotation matrix, right: bone's articulation.

Each joint in the reference posture is associated with a local coordinate system. In the animated posture, the joints are transformed by rotation. The transformation from the reference position and orientation of joint *j* to its position and orientation in the animated posture can be described by a homogeneous matrix. This matrix can be expressed as a multiplication of successive joint transformations $M_i = h_i B_i^{-1}$ [23].

4.2 Skinning

The skinning is the process to add the vertices to the skeleton. A vertex can be assigned to one or more bones, depending on the position it occupies. There is some work about skinning such as [24], where the method proposed used the mesh deformation to be displayed with the ambient occlusion technique. Also, with graphics hardware we can find the work of Kavan [25] who introduced the skinning with dual quaternions.

The simplest deformation assign to each vertex to a joint. That is, in vertex $v'=M_i v$, the vertex v is transformed to its new position v'. v is the reference vertex associated to the joint i and v' is the position of the deformed mesh. A vertex which belongs to two bones will have *50%* of weight on each of them: $v'=w_0 M_i v + w_1 M_i v$, where $w_0 = w_1 = 0.5$. Fortunately, Blender™ makes the skinning automatically so that we could create poses moving the skeleton and therefore the mesh.

4.3 Poses

In Figure *7*, three poses were created. The original mesh (left) is defined by the vertices $V^0 = \{v_0^0, v_1^0, \ldots, v_{m-1}^0\}$, pose *1* (central) is defined by the vertices $V^1 = \{v_0^1, v_1^1, \ldots, v_{m-1}^1\}$, pose *2* (right) is defined by the vertices $V^2 = \{v_0^2, v_1^2, \ldots, v_{m-1}^2\}$.

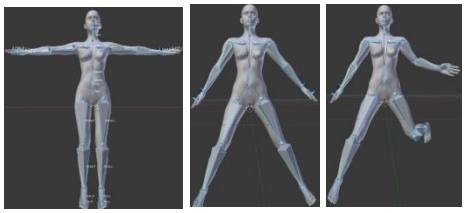

Fig. 7. Three poses: (lef) original position, (central) the 4th keyframe, (right) the 8th keyframe.

The poses $V^0,\ldots,V^p$ are exported to PLY or OFF files, which serve in the next stage. The type of polygon (quads or triangles) mesh is irrelevant since we are working with the vertices. This files define the vertices by its position *X,Y,Z* with float numbers, and the polygons by a set of integer numbers indicating the vertices of such polygon.

5. Parallel Interpolation (Stage 3)

The last stage of the methodology involves the Bézier interpolation basic concepts, the parallel implementation and the program execution.

5.1 Bézier Interpolation

The morphing method [26] consists on converting an object from an initial position to a target position, taking control points to make the transition. These control points are utilized to re-calculate the position of the vertices in time *i* with a transformation.

Given a set of points, we can obtain a polynomial that passes by the control points. Given a function *f* with known values $v_0, v_1, \ldots, v_{m-1}$, we call the polinomial interpolation to find $p_r(x)$ of lower degree of *r* that satisfies $p_r(v_k) = f(v_k)$ for $k=0,1,\ldots, m-1$.

To deform a vertex from mesh M^i to M^{i+2} we apply the Bézier Interpolation, getting intermediate values to achieve a smooth transition. This is a quadratic interpolation which requires three values of the same vertex to be applied: M^i, M^{i+1}, M^{i+2}.

Bézier curves are a type of splines. A spline function is formed by several polinomials, each one defined by an interval, joined by a continuity constraint. Spline curves are used to produce similar results to the polinomials, requiring a low polynomial degree and avoiding oscilations.

Given the points $v_0, \ldots, v_{m-1}$, the *m* degree Bézier curve is defined as follows: a *3*-degree Bézier curve is parametrically defined by a function *q(n);* when *u* varies from *0* to *1*, the values of *q(u)* are displaced along the curve. The formula is the following:

$$q(u) = B_0(u)p_0 + B_1(u)p_1 + B_2(u)p_2 + B_3(u)p_3 \quad (2)$$

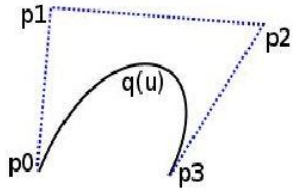

Fig. 8. Bézier curve with *4* control points.

The *n*-degree Bézier curve can be generalized as follows:

$$q(u) = \sum_{k=0}^{m-1} \binom{m-1}{k} u^k (1-u)^{m-1-k} \quad (3)$$

5.2 Parallel Implementation

According to the interpolation algorithm analysis, we decided to implement the parallel version without passing by the sequential implementation version. The avatar animation was implemented with quadratic interpolation for Bézier curves, using OpenGL and CUDA. OpenGL buffers called as VBO (Vertex Buffer Objects) can be mapped as much as required in CUDA memory spaces. The CPU-GPU data interchange is minimized by having the data in the GPU for processing (CUDA) and rendering (OpenGL).

Three keyframes are required to apply equation (*2*) and obtain a movement as illustrated in Figure *10*. VBOs are employed to store the vertex and color of the object mesh. A VBO is required for each keyframe, therefore *3* VBOs are required for the Bézier Interpolation.

The actual position is calculated and updated in the rendering loop, calling a Bézier interpolation kernel. A kernel is a procedure executed in GPU. At the start, the three control points are initialized to indicate the vertex

position in time *i, i+1, i+2*. Let ψ be the number of blocks, ϑ be the number of threads, and *m* be the number of vertices, then we have

$$\psi = \frac{m}{\vartheta} + \{0,1\} \quad (4)$$

Where the value of the second {0,1} depends on the rest of the first factor. The vertices are well distributed among the threads to have a load balancing work (Figure *9*). In this sense, the complexity is $O(\frac{m}{\vartheta\psi})$.

If the time exceeds the upper bound defined to make the animation, then only two of the three keyframes will be updated, and this third keyframe will be the first keyframe of the next movement.

Fig. 9. The number of threads per block is designed to contain all the vertices of the mesh.

The algorithm of the Bézier Interpolation is explained in Algorithm *1*. The actual vertex position is computed during the animation in its correspondent VBO.

Algorithm 1 Bézier Interpolation

```
1: procedure InterpolaBezier(VBO1, VBO2, VBO3, M, u, v).
2:     int i = blockIdx.x * blockDim.x + threadIdx.x;        ▷ in parallel
3:     if (i < v) then
4:         M_i = (1-u)^2 VBO1_i + 2u(1-u) VBO2_i + u^2 VBO3_i;
5:     end if
6: end procedure
```

Each thread updates a vertex; if ϑ is the number of threads in each block factor of *32*, then the ψ blocks can be obtained as follows:

$$\psi = \left\lceil \frac{m}{\vartheta} \right\rceil \quad (5)$$

5.3 Program Execution

The algorithm was run in a PC desktop AMD FX 6100 X6 with 2.0 GB RAM DDR, operating system GNU/linux X86-64 Ubuntu 12.04, kernel 3.2.029, GeForce GT 520 1 GB DDR Graphics Card.

At the start three frames are loaded: ξ_0, ξ_1, ξ_2. The animation speed can vary depending on the application. We choose *2* seconds to animate an avatar with 11 keyframes. The kernel execution time is *130* μs with *68* fps (frames per second).

We proceed to execute the program with another Graphics Card GeForce GTX 650 with 1 GB DDR5. The kernel execution time was 30 μs with *68* fps. The time was reduced in 100 μs, a relevant enhancement. Four keyframes are illustrated in Figure *10*.

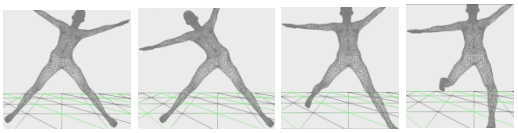

Fig. 10. Avatar Animation.

6. Conclusion

A methodology to develop an avatar animation was proposed, including three stages: modeling, animation, and parallel implementation. The focus was to fill the gap of the animation methods used in 3D modeling packages and in the programming languages implementation. In the 3D modeling packages, the object's geometry can not be easily accessed and the process becomes slow, depending on the features of the same packages. The use of programming languages implies a complicated task with the matrices and quaternions operations, and it requires a high control among the bones hierarchy.

We take the advantages of the 3D packages and the programming languages. 3D packages facilitate the modeling, rigging and skinning processing. Programming languages allows an accurate operation, immediately access to the object's geometry, control of the animation speed, and independency of the environment (platform and libraries).

Interpolation algorithm was implemented in parallel using GPUs, obtaining $O(\frac{m}{\vartheta\varphi})$ time complexity. We could vary some parameters to control the animation speed. We run the algorithm in one avatar, but other avatars could be employed.

The work can be extended using other parallel programming techniques such as the use of the texture memory to reduce the memory access. Also, other interpolation methods can be utilized, with more keyframes required to have smoothy movements. Also, other 3D packages could be used, and other programming tools can be proved: OpenCL, and graphics APIs such as DirectX.

Acknowledgments

We would like to thank to the Universidad Autónoma de Yucatán and the CONACYT México for their financial support.

References

[1] F. Madera and L. Basto and R. Uicab, "Virtual Chichen Itzá" X Congreso Internacional de Informática y Computación. 2011, Colima, México.
[2] M. Kasap, N. Magnenat-Thalmann. "Parameterize Human Body Model for Real-Time Applications", Proceedings of the 2007 International Conference on Cybreworlds, pp. 160 - 167, 2007.
[3] H. Seo, N. Magnenat-Thalmann. "An example-based approach to human body manipulation", Graphical Models, vol. 66, No. 1, 2003.
[4] H. Seo, N. Magnenat-Thalmann "An automatic modeling human bodies from sizing parameters", Proceedings of the 2003 Symposium on Interactive Gaphics I3D'03. Monterey, California.
[5] T. Pejsa, I.S. Pandzic. "State of the art in example-based motion synthesis for virtual characters in interactive applications". Computer Graphics Forum, vol. 29, No. 1, pp. 202-226, 2010.
[6] Doug L. James and Christoper D. Twigg. "Skinning mesh animations". ACM Transactions on Graphics, vol. 24, N. 3, 2005.
[7] L. Kavan, J. Zára. "Spherical blend skinning: a real-time deformation of articulated models". Proceedings of the 2005 symposium on Interactive 3D graphics and games, Washington, District of Columbia, 2005.
[8] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, J. Davis. "SCAPE: shape completion and animation of people". ACM Trans. Graph. vol. 24, No. 3, pp.408-416, 2005.
[9] H. Mitake, K. Asano, T. Aoki, S. Marc, M. Sato, S. Hasegawa. "Physics-driven Multi Dimensional Keyframe Animation for Artist-directable Interactive Character". Computer Graphics Forum, vol. 28, No. 2, 2009.
[10] D. Lu, X. Ye, G. Zhou. "Animating by example". Computer Animation and Virtual Worlds, vol. 18, No. 4-5, pp. 247-257, 2007.
[11] Revista Dibujarte, No. 23, 1998. México. https://es-es.facebook.com/RevistaDibujarte
[12] Kelly L. Murdock and Eric M. Allen. "Edgeloop Character Modeling". Wiley, 2006.
[13] P. Schneider and David Eberly. "Geometric Tools for Computer Graphics", Morgan Kaufmann, 2003.
[14] Edward Angel. "Interactive Computer Graphics, a top down approach using openGL". Addison Wesley, 2008.
[15] G. Maestri, "Digital Character Animation 3", First Edition, New Riders Publishers (2006).
[16] Make Human: An Open Source tool for making 3D characters. http://www.makehuman.org/, 2013.
[17] Blender: An Open Source tool for Modeling and Animation. http://www.blender.org/, 2013.
[18] A Manga Girl 3D Model. http://blenderartists.org/forum/showthread.php?95491-Maid San-(Manga-style-girl-in-a-maid-costume), 2013.
[19] A Blender Girl 3D Model. http://www.blender-models.com/model-downloads/humans/id/low-polyfemale-model/, 2013.
[20] COLLADA: Digital Asset and FX Exchange Schema. http://collada.org, 2013.
[21] XNA Tutorial. http://www.xna-tutorial.com/, 2013
[22] Bones in Blender. http://wiki.blender.org/index.php/Doc:2.6/Manual/Rigging/Armatures/Bones, 2013.
[23] K. Erleben, J. Sporring, K. Henrikisen, H. Dohlmann. "Physics-Based Animation". Charles River Media Publisher, 2005.
[24] Adam G. Kirk, Okan Airkan. "Real-Time Ambient Occlusion for Dynamic Character Skins". Proceedings of the 2007 symposium on Interactive 3D Graphics and games. Seattle, Washington, pp 47 -52, 2007.
[25] Ladislav Kavan, Steven Collins, Jiri Zara, Carol O'Sullivan. "Geometric Skinning with Approximate Dual Quaternion Blending". ACM Transactions on Graphics, vol. 27. No. 4, pp. 1-23, 2008.
[26] Won-sook Lee, Nadia Magnenat-Thalmann. "Virtual Body Morphing". Proceedings of the Fourteenth Conference on Computer Animation. pp 158 - 166, Seoul, 2001.
[27] ISO 7250-1:2008. "Basic human body measurements for technological design, part I: Body measurement definitions and landmarks". http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=44152

Francisco A. Madera received his B. Sc. Degree from the Universidad Autónoma de Yucatán, México; his PhD from the University of East Anglia, UK. He is the Computer Science postgraduate Chairman at the University of Yucatán. Dr. Madera teaches subjects related to computer graphics and videogames development; and his research is focused on collision detection and GPU programming.

Francisco Moo-Mena is a Professor in Computer Sciences at Universidad Autónoma de Yucatán, in Mérida, Mexico. From the Institute National Polytéchnique de Toulouse, in France, he received a Master Degree in Computer Science and a PhD, in 2003 and 2007, respectively. He also received another Master Degree in Distributed Systems from the Instituto Tecnológico y de Estudios Superiores de Monterrey, Mexico, in 1997. He received a BS in Computer Systems Engineering from the Instituto Tecnológico de Mérida, Mexico, in 1995. His research interests include Parallel and Distributed Computing, CUDA, Self-healing systems, and Web services Architectures.

Carlos G. Herrera is studying a master degree in Computer Sciences at the Universidad Autónoma de Yucatán. His research interest focuses on Graphics and GPU Programming.