

# Progressive Skinning for Character Animation

Simon Pilgrim  
University College London  
Electronic Arts UK Studio  
S.Pilgrim@cs.ucl.ac.uk

Anthony Steed  
University College London  
A.Steed@cs.ucl.ac.uk

Alberto Aguado  
Electronic Arts UK Studio  
AAguado@europe.ea.com

## ABSTRACT

Previous works have shown that for characters with obvious articulation, animations comprising the blending of skeletal bone transformations, significantly reduce the computation and data requirements, permitting much greater visual detail. However increasingly, animators and engine designers must produce animations that can run on platforms with quite different run-time capabilities. They may simply author multiple skeletons and sets of animations, or reduce the temporal or spatial detail to achieve acceptable frame rates.

In this paper we show how these skeletal skinning methods can be extended to support a continuous level of animation detail. We show that by manipulating the skeletal hierarchy and skinning parameters we can throttle the computational load of a character model in real-time according to its position in a scene and any hardware constraints. Our method is compatible with additional geometric level of detail methods.

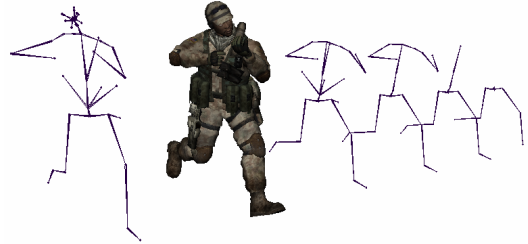
## Keywords

animation, skinning, compression, level of detail

## 1. INTRODUCTION

Accurate, efficient character animation is vital for modern real-time computer graphics applications. Domains such as video games and film production rely on methods such as skeletal blend skinning and vertex blending to permit the efficient authoring and playback of a wide variety of animations. Blend skinning is very commonly used due to the ease with which it reproduces rigid motion; it can easily be re-targeted to a variety of character models; is mathematically simple; and is ideal for the SIMD transformation of vertex geometries on CPU or GPU based hardware.

However, even with recent graphics hardware, the number of vertices that can be processed in a complex scene is severely limited. Thus, important visual details are often



**Figure 1: A picture showing the concept of progressive skinning. On the left is the original skeleton rig and the character skin. To the right are a sequence of reduced skeletons, which can be used to animate the same skin on less powerful hardware, or within an LOD mechanism. Progressive skinning allows smooth blends between these reduced skeletons.**

lost. Generally, the simplification necessary on each character is achieved either by reducing the number of vertices and polygons in the model or by creating a simpler skeleton 'rig' with fewer bones and the relevant animations re-authored until a balance of quality and performance is found. When a character model appears in a scene at different positions, each instance requires a higher or lower level of detail (LoD) to be created. Furthermore, simplification of a character mesh is not related to motion structure and when several instances are created it is difficult to define congruent animation data. Whatever the method, every additional LoD represents additional costs to production and requires the storage of extra data.

This paper introduces an evolution of blend skinning, that we call *progressive skinning*. In an offline, automatic conversion process, we replace the authored, (anatomic) skeleton hierarchy with a progressive hierarchy that re-orders and re-parents the bones to minimize the total vertex error. At runtime this allows each character skeleton to be progressively animated, with the most important bones being computed first. Based upon the resources available to interpolate and prepare the bone transformation data, we ensure that for a given resource allocation, each character may be drawn with minimal error. Furthermore, by gradually blending

the lower priority bones in or out of the active hierarchy, we introduce a continuous LoD mechanism to the skeletal animation system. As well as prioritizing the skeletal data, we also demonstrate that the blend skinning weight data can also be prioritized to increase fidelity or reduce computation time. This technique not only permits animation data to be used as an effective LoD runtime component but can reduce the costs of production by allowing better reuse of existing high detail skeleton data. Additionally, by being orthogonal to any geometric LoD mechanism, both methods can be used in parallel based upon the LoD requirements of each scene. Our work primarily concerns character animation for video games systems, typically with CPU and GPU hardware; however there is nothing in this method that prevents it being used for other character animation applications.

The rest of this paper is organized as follows. In Section 2 we briefly review related work. In Section 3 we describe a modern playback system for video game character animations. We identify the potential bottlenecks for computation and data flow, the reasons for the bottlenecks and their relative effect on the overall system performance. Section 4 then describes the features of progressive skinning, and how they may be used to alleviate each of the bottlenecks. We show how this method may be used with existing blend skinning systems with little or no alteration. Section 5 presents a typical example of a character animation and the effect of manipulating the skeletal and skinning parameters. We conclude with some directions for future work.

## 2. PREVIOUS WORK

Working particularly well for rigid, articulated character animations, skeletal blend skinning (notably Linear Blend Skinning[15]) is widely used in video games systems due to its ease of use and high performance on modern SIMD hardware. A complex vertex geometry may be animated with relatively few skeletal bones. Matrix Palette Skinning [3] is a simpler version of blend skinning where the vertices are each transformed by a single pre-blended matrix. This decreases the per-vertex computation cost in exchange for a smaller increase in data storage requirements for the matrices. Alternative blend skinning methods such as Quaternion [5], Spherical [9] and Dual Quaternion Skinning [8] work in a similar fashion, but replace the linear blend with various methods to improve rigid rotation blending in exchange for a small performance penalty. Methods to convert vertex animations to skinning representations [14, 10, 7] have demonstrated notable improvements in performance, but can not easily optimize existing skinning data.

In more complex scenes it is necessary to utilize a LoD method to maintain performance in frame rate and hardware resource allocation. [2] introduced LoD as part of a visibility, culling and clipping scheme based upon maintaining a hierarchy of the geometric data in a scene and also demonstrated the drawing of each model at a specific LoD depending upon its projected size. [4] developed the concept of maintaining a consistent rendering frame rate through the manipulation of each objects' geometric complexity based upon a cost-benefit scheme. [1] demonstrated the idea of clustering bone animation data to determine common postures. Only the clusters that have significantly altered since the previous frame are reanimated, the remainder keep their

previous values. Such an approach requires careful scrutiny of the bone rotation clusters to prevent notable rotation interpolation errors and does not allow for limitations in the skinning methods resulting in non-minimal errors in the vertices. Our method minimizes the squared vertex errors, thus avoids introducing bone and skinning errors.

To reduce the computational load for precomputed animations, most methods only consider simplifying the underlying mesh geometry. They include techniques that have extended progressive meshes [6, 12] and techniques that have obtained simplified representations based on vertex transformations [13] and principal component analysis [11]. Unfortunately, the high data and computation costs of progressive meshes [6] hinder their use in current gaming systems. Typically a game is limited to choosing from a number of pre-computed character meshes LoDs. Thus, performance is highly dependent on the animation and skinning with the bone matrices. By manipulating the high computational cost of the animation data (instead of the high data cost of the geometry) our method is more suited to modern hardware which are dominated by CPU (not GPU) bottlenecks.

## 3. CHARACTER ANIMATION PIPELINE

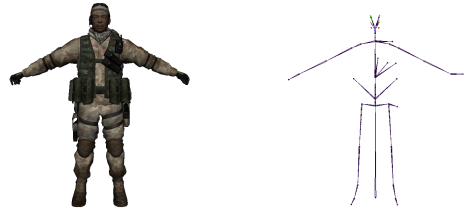


Figure 2: A picture showing a textured video game character model and its skeletal rig in the bind pose.

Linear blend skinning derives the position of vertex  $v_{i,f}$  in animation frame  $f$  from the normalized weighted sum  $w_{i,b}$  of each bone  $b$  inverse bind pose  $M_{b,bind}^{-1}$  and frame  $M_{b,f}$  matrices and the vertex's bind pose  $v_i$  (Equation (1)). Typically, each vertex in a video games character will have no more than four non-zero weight values, permitting the blend matrix weights and IDs data to each fit into one SIMD vector4 type. The vertex normals are similarly transformed using the blended inverse-transpose of the bones' upper 3x3 sub matrices.

$$v_{i,f} = v_i \sum_{b=1}^B w_{i,b} M_{b,bind}^{-1} M_{b,f} \quad (1)$$

$$\sum_{b=1}^B w_{i,b} = 1 \quad 0 \leq w_{i,b} \leq 1 \quad (2)$$

Each bone matrix is derived from the forward kinematic calculation of the bone's local transformation (typically restricted to Scale-Rotation-Translation (SRT) components) with its parent's bone matrix  $M_{p,f}$  (Equation (3)), recursively calculated from the root bone transformation. A

bone’s transformation for a given frame is determined from an interpolation of keyframes.

$$M_{b,f} = (S_{b,f}R_{b,f}T_{b,f}) M_{p,f} \quad (3)$$

In most video game systems, the bone preparation is performed on the CPU before being uploaded to the GPU vertex shader where the vertex blend skinning then occurs. The main performance bottlenecks during skinning playback are therefore: the interpolation and preparation of the bone matrices, the transfer of the bone matrices to the GPU, the blending of the bone matrices and the complexity of the mesh geometry. Additional bottlenecks such as texture fetch, pixel fill rate and rasterization are almost independent of the skinning method and are not investigated in this paper.

### 3.1 CPU Bottlenecks

Processing on the CPU represents the most common bottleneck in the character animation pipeline. Although much processing can be transferred to the SIMD based GPU, the CPU is still necessary for the (non-parallel) bone calculations.

*Keyframe Interpolation.* represents the biggest potential bottleneck in the calculation of the bone matrix data (Table 1). The linear interpolations of the scale and translation vector3 data are relatively low cost SIMD operations. However the rotation data is typically linearly or spherically interpolated using quaternions. Both methods involve complex (costly) operations such as trig or sqrt functions, which do not apply well to SIMD data and require many cycles to complete.

Typical methods of reducing the cost of rotation interpolations include storing the rotation values for every frame (which increases the data access and storage costs), ignoring quaternion normalization errors (which reduces quality) and trigonometric approximations (which reduces quality).

*Matrix Preparation.* concerns the conversion of the SRT components into the bone’s 4x3 local transformation matrix (Equation (4)). Again the major limitation is the conversion of the quaternion rotation data into a 3x3 rotation matrix and then its multiplication with the scale data; a typical implementation requires 21 ADD + 9 MUL scalar operations per bone. The translation data merely requires a direct copy

Transform	Operations
Scale	1 SUB, 1 MADD
Translate	1 SUB, 1 MADD
Rotation-Slerp	4 TRIG, 1 DOT, 2 MADD
Rotation-Lerp	1 SUB, 2 MADD, 1 RSQRT

**Table 1: A table of the typical (scalar or vector) operations necessary for the interpolation of the SRT transformation components of each bone.**

into a matrix row or column.

$$S_{b,f}R_{b,f}T_{b,f} = \begin{pmatrix} S_x R_{11} & S_x R_{12} & S_x R_{13} \\ S_y R_{21} & S_y R_{22} & S_y R_{23} \\ S_z R_{31} & S_z R_{32} & S_z R_{33} \\ T_x & T_y & T_z \end{pmatrix} \quad (4)$$

*Forward Kinematics.* calculation of the bone world matrix requires a minimum of  $(N_{Bones} - 1)$  matrix multiplies (12 MADD vector operations each) through traversing the hierarchy and storing all bone matrices upon calculation, either in an allocated matrix cache or on the program stack using a recursive calculation method. The resulting bone matrices must then be multiplied with their bind pose values (these are static and maybe pre-computed), which requires an additional matrix multiply per bone (12 MADD vector operations each).

*Total Computation Cost.* of calculating each bone is therefore approximately 60 ‘simple’ operations (MADD, DOT, etc.) and up to 4 ‘complex’ operations (RSQRT, TRIG).

*Matrix Transfer.* performance from the CPU to the GPU vertex shader constant table is difficult to predict as memory block transfers can only take so many matrices at a time, and the underlying hardware drivers will often multiplex various graphics data to/from the GPU in irregular combinations. It is necessary to accept that this is a known performance cost, as all the matrix data must be transferred.

*Matrix Palette Creation.* is an alternative step that must be performed on the CPU when Matrix Palette skinning is being used, typically for older GPU systems with limited vertex shader programmability. Each palette entry is the result of the matrix blend calculation in Equation (1) (without the vertex transformation). It is these entries that are then transferred to the GPU vertex shader constant table, with each vertex transformation only needing to reference one entry. CPU performance is therefore directly related to the number of entries and the number of blend weights per entry, with each entry requiring approximately 4 MADD vector operations per blend weight. It is these blended matrices that are then transferred to the GPU. As the vertex shader does not perform any blending, its performance should then be considered a fixed cost per vertex.

### 3.2 GPU Vertex Shader Bottlenecks

The performance of the GPU vertex shader will (obviously) have a direct correlation with the number of vertices in the character model. As most SIMD hardware performs best without any branching, we should focus all optimizations on ensuring that each vertex takes a similar, minimal number of cycles to complete its transformation.

*Matrix Blending.* requires approximately 4 MUL + 4 ADD vector operations per blend weight (Table 2). Most GPU

implementations permit no more than four blends and depending upon the implementation it may be more efficient to blend the bones matrices or blend the transformed vertices.

*Number of Polygons.* and their layout has a notable effect on skinning performance, not only due to the number of vertices referenced but poor use of the GPU vertex cache may require many recalculations of the vertices. This is a hardware specific limitation; tools exist to optimize polygonal meshes for target hardware and greatly assist with general vertex shader performance.

## 4. PROGRESSIVE SKINNING

We present an extension to the blend skinning method that provides a continuous LoD playback for bone-based animations. The aim is to maintain the computational load of a given character in proportion to its visual priority in the scene. The method can be used with existing (highly optimized) skinning algorithms for either CPU or GPU computation. An existing LoD control system should be used to determine the priority of each character in each frame and specify their LoD according to available resources.

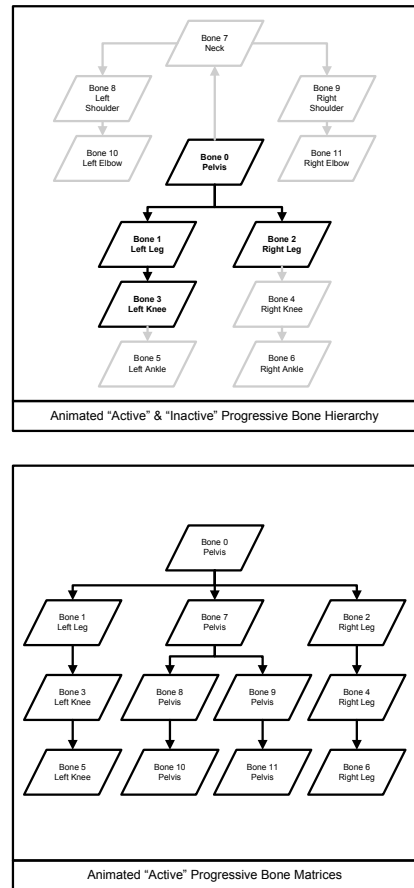
As explained in Section 3, the bottlenecks due to character animation fall into two categories: the cost of calculating the bone matrices in each frame; and the cost of blending the bone matrices and transforming each vertex from its bind to frame position. Progressive Skinning provides the ability to alter the computational load of both. A character’s progressive bone hierarchy is pre-computed so that each additional bone calculation in the hierarchy minimizes the outstanding vertex error in the animation. The character need only animate the bones the LoD system has allocated it, the remaining bones in the hierarchy copy their parents’ bone matrices. Similarly on the GPU, each vertex’s bone blending is reordered so that when fewer blends are used, the weight data is correctly redistributed to minimize error. The LoD control system sends a value representing the % of maximum computation that the character is permitted, which progressive skinning uses to determine the % of bones and blend weights to calculate.

### 4.1 CPU Progressive Bone Hierarchy

The principal aim of progressive skinning is to *only calculate the bones there is time for*. On a per-animation basis, the bones are ordered in terms of contribution to the animation error, and the parentage set so that bones may inherit optimal transformation data from their new parent bone. There is a minute increase in data storage from storing the extra parentage data of one integer value per bone per animation. During the playback of each frame, the LoD control system will indicate the number of *active bone* matrices that should

<b>Blends</b>	1	2	3	4
<b>GPU Operations</b>	20	35	43	50

**Table 2: A table of the typical GPU operations required for linear blend skinning with one to four active blend weights (n.b. with one weight no blending is necessary).**



**Figure 3: Diagrams showing a simple progressive bone hierarchy for a walking animation, with only resources to animate four key leg bones. The upper diagram shows the anatomical hierarchy. The active bones are shown in bold. The lower diagram shows the actual matrices used by each bone, showing inheritance. In this example the bones have not been re-parented.**

be calculated, which will be at the top of the progressive hierarchy. The remaining *inactive bones* in the hierarchy can copy their matrix values from their active parent bone (Figure 3) or use a default value for the bone depending upon the situation. As our method does not rely on any specific feature of the linear blend skinning method it will work with more recent blend skinning methods [5, 9, 8].

This method is simple to implement in a runtime system and works to overcome all of the CPU bone preparation bottlenecks identified in Section 3:

*Keyframe Interpolation.* calculations are only performed on the active bones in the frame. As many animations only actively use a fraction of their bones (e.g. a walking animation mainly concerns the leg bones), a large proportion of the bones can be deactivated and only introduce marginal

errors.

**Matrix Preparation.** must be performed on all the active bones. The inactive bones however may either be treated as a direct copy of their parent (equivalent to setting their local transformation to the Identity) or may use a per-animation default, pre-computed transformation (bind pose or other) and reduce the computation to a direct matrix copy.

**Forward Kinematics.** requires the active bones to be correctly computed by traversing the bone hierarchy. However, the progressive hierarchy ensures that no inactive bones will be included, minimizing the number of necessary matrix multiplies. Building on the matrix preparation stage, the inactive bones may directly copy their matrix from their parent or will need to be included at the end of the forward kinematic multiplication if they use a default transformation matrix value. Whether inactive bones use their default transformations or copy their parent's matrix should be determined based upon the cost of matrix multiplication, and could possibly be an extra runtime decision based upon the character's current LoD status.

**Matrix Transfer.** is a fixed cost however we derive the value for each bone matrix - it must still be transferred to the GPU for vertex skinning.

#### 4.1.1 Bind Pose Optimization

The bones' default transformations can be viewed as a fixed alteration to the character's mesh bind pose for that animation. It follows therefore that should a majority of the animations share similar values for a bone's default transformation, the mesh bind pose should be altered directly in the offline conversion process. This avoids having to include the default value for that bone in many of the animations.

#### 4.1.2 Continuous LoD

The activation and deactivation of bones in the skinning skeleton may cause notable LoD 'popping' effects. However, as the deactivation of a bone is equivalent to setting its local transformation to the Identity, it follows therefore that we can continuously blend on or off by interpolating with the Identity (or default value). This does require an additional (simplified) interpolation for the duration of the LoD transition, and means that the rate of change of each character's LoD is limited but avoids any notable visual errors.

### 4.2 GPU Progressive Bone Blending

In a similar fashion to the bone hierarchy, progressive skinning can vary the number of blend weights being used for vertex skinning. It is assumed that each model has a maximum of four blend weights and that the values sum to one (Equation (2)). The number of blends can be reduced by transferring the later blend weights to earlier blends. Figure 4 defines a fixed blend hierarchy used by progressive skinning for the redistribution of blend weights as the number of blend bones varies between one and four. A continuous LoD of matrix blending is possible by linearly interpolating between the four blend levels, requiring one additional MADD

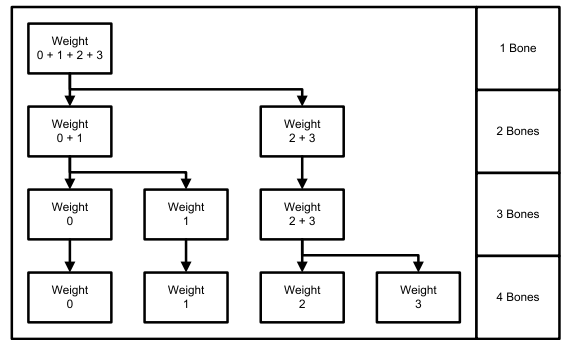


Figure 4: A diagram showing the hierarchy of how the blend weights are redistributed as the number of blend bones varies between one and four.

operation in the vertex shader.

**Matrix Palette Creation.** on the CPU can operate in an identical fashion to the GPU implementation, but with the extra ability to directly copy those palette entries that reduce to the same values with fewer active blend weights, avoiding repeated computations.

### 4.3 Data Preparation

After each skeleton animation has been authored, an offline conversion is used to convert the bone hierarchy to the progressive skinning format for playback. A complete search is performed to determine the progressive hierarchy that results in the minimal squared error in the vertex positions of the skinned character mesh with each additional active bone. As video games skeletons are typically quite small, containing less than 64 bones, we found that a more efficient search method was not necessary for this one off calculation. Any bones not being referenced by the character are removed. We found that a bone's default transformation calculated from its average value across all frames of the animation was sufficient. The mesh bind pose was updated if enough animations used a similar default bone value and the hierarchy reordered as necessary. Finally, the blend weights and IDs are sorted to minimize vertex error across all the animations, either for a specific number of progressive bones or for all progressive bone counts.

A basic version of the converter merely reorders the bones, without altering their parentage. This is necessary if the animation is non-rigid - as complex use of the scale transformation requires the original parentage.

An advanced version of the converter transforms all the bone transformations to character world space, determines a minimal error root bone and then reorders and re-parents the bones to minimize the error with each additional bone. The bones are then transformed back to local space based upon their new parentage.

### 4.4 Additional Factors



Figure 5: A picture showing the crowd animation playback demo of sixteen separate instances of the soldier model.

*Geometric LoD.* methods can generally be used in parallel to progressive skinning's animation LoD method. The geometry LoD determines which polygons and vertices are drawn and possibly manipulates the vertex bind positions [12] in the vertex shader. Progressive skinning then animates each vertex as described above. Continuous geometric LoD methods sometimes increase the number of vertices or GPU operations performed during transition between LoDs and it would be necessary for the LoD control system to be aware of this to avoid it trying to over compensate.

*Animation Blending.* is a costly procedure, requiring the interpolation of two separate animations and then interpolating the results to get the actual bone matrices. Progressive Skinning will reduce the costs, by only interpolating a proportion of the bones for each animation. However it will still often be necessary to interpolate a large number of bones. In most games only a few, principal characters ever use animation blending, with the remainder merely concatenating separate animations together.

## 5. RESULTS & DISCUSSION

Figures 1 and 2 showed an example of a video game 'soldier' character model. The model contains 5703 vertices, 11000 triangles and a 44 bone skeleton; although highly detailed for a current generation video game character, this is expected to be a typical level of quality for next generation titles. All calculations were performed on a Xeon 3.0 GHz with 1Gb RAM and a 6800 Ultra AGP graphics card at XGA resolution.

The soldier character includes a variety of animations, including running, kneeling down, crawling and firing a gun. For each animation the skeleton rig was converted to a progressive skin hierarchy. Sixteen separate instances of the soldier were placed in a basic DirectX crowd scene (Figure 5). For each test the frame rate and total squared vertex error was determined. A basic LoD system was used to maintain frame rate based upon each character's distance from the camera and the average drawing time of the last five frames.

Figure 6 shows the effect on the frame rate of varying the number of progressive bones and blend weights. For high bone counts, the performance is totally CPU bound due,

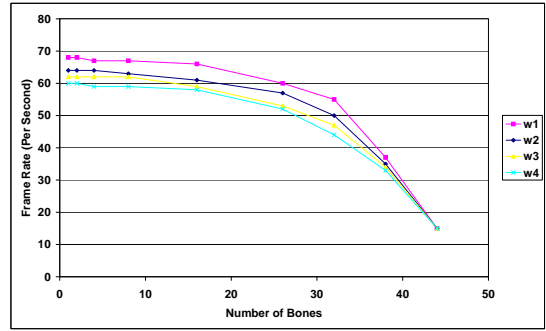


Figure 6: A graph showing the frame rate (fps) of the crowd demo for various numbers of active bones and blend weights

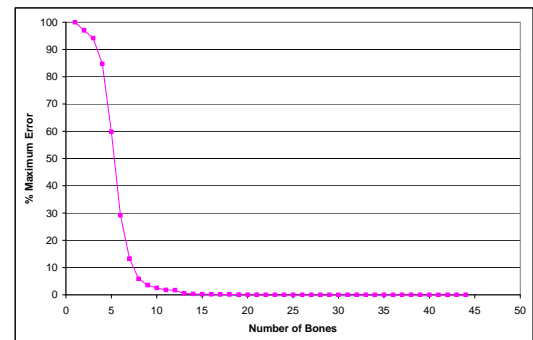


Figure 7: A graph showing the % of maximum squared vertex error of the soldier animation crowd demo for various numbers of active bones.

with bone blending showing more of an effect as more bones are deactivated.

Figure 7 shows the error as a % of the maximum across all the animations and blend weight LoDs. It is very clear that each animation animates only a small portion of the skeleton bones and the remainder are well suited to being set to a default value (or a mesh bind pose adjustment).

## 6. CONCLUSIONS

This paper has demonstrated a method of extending skeletal skinning character animation to incorporate a continuous animation LoD with a progressive bone hierarchy. This enables the CPU and GPU computational load of a given animation to be altered at runtime based upon the character's priority in the scene. We have described how to pre-compute a progressive skinning hierarchy as an offline operation and shown there is a discernible improvement in runtime frame rate performance for an insignificant increase in vertex error. Finally, we have explained how this method is simple to implement and compatible with existing skinning and geometric LoD methods that may already be in use.

Progressive skinning is well suited to video games systems, and we would like to investigate other character animation applications. Potential future work includes using the method with more advanced LoD control systems for complex crowd systems; demonstrating it in use with recent skinning methods and investigating better integration between geometric and animation based LoD representations.

## Acknowledgments

We would like to thank Electronic Arts UK Studio for sponsoring this project, which was part of UCL's Engineering Doctorate in Virtual Environments, Imaging and Visualisation (EngD VEIV). Particular gratitude goes to Ian Shaw, Martin Usoh and Kenny Mitchell for their ongoing support and advice.

## 7. REFERENCES

- [1] J. Ahn and K. Wohn. Motion level-of-detail: A simplification method on crowd scene. *Proceedings Computer Animation and Social Agents (CASA)*, pages 129–137, 2004.
- [2] J. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.
- [3] B. Freidlin. DirectX 8.0: Enhancing real-time character animation with matrix palette skinning and vertex shaders. *MSDN Magazine*, June 2001.
- [4] T. Funkhouser and C. Sequin. Adaptive display algorithms for interactive frame rates during visualization of complex virtual environments. In *ACM Transactions on Graphics (SIGGRAPH 2005)*, pages 247–254, 1993.
- [5] J. Hejl. Hardware skinning with quaternions. *Game Programming Gems*, 4, 2004.
- [6] H. Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108, 1996.
- [7] D. James and C. Twigg. Skinning mesh animations. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3):399–407, 2005.
- [8] L. Kavan, S. Collins, J. Zara, and C. O'Sullivan. Skinning with dual quaternions. In *2007 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM Press, April/May 2007.
- [9] L. Kavan and J. Zara. Spherical blend skinning: a real-time deformation of articulated models. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 9–16, New York, NY, USA, 2005. ACM Press.
- [10] A. Mohr and M. Gleicher. Building efficient, accurate character skins from examples. *ACM Transactions on Graphics* 22, 3 (July), pages 562–568, 2003.
- [11] W. Müller and M. Alexa. Representing animations by principal components. *Computer Graphics Forum (EuroGraphics 2000)*, 19(3):411–418, 2000.
- [12] P. Sander and J. Mitchell. Out-of-core rendering of large meshes with progressive buffers. In *ACM SIGGRAPH 2006: Proceedings of the conference on SIGGRAPH 2006 course notes*, pages 1–18, New York, NY, USA, 2006. ACM Press.
- [13] A. Shamir and V. Pascucci. Temporal and spatial level of details for dynamic meshes. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 77–84, 2001.
- [14] X. Wang and C. Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 129–138, 2002.
- [15] A. Watt and F. Policarpo. *3D Games: Animation and Advanced Real-time Rendering*. Addison Wesley, 2003.