Progressive Skinning for Video Game Character Animations

Simon James Pilgrim* VECG Lab, CS Dept., UCL & Electronic Arts UK

Alberto Aguado† Electronic Arts UK

Kenny Mitchell‡ Electronic Arts UK

Anthony Steed§ VECG Lab, CS Dept., UCL

Introduction We present a linear blend skinning method that provides a continuous level of detail (LOD) playback for bone-based animations. The purpose is to reduce the load of streaming animation data to a rendering card. The proposed animation data can still use existing (highly optimized) skinning algorithms for either CPU or GPU computation. Thus, animations can be used on current generation gaming systems with little to no alteration.

Previous works [Mohr and Gleicher 2003; James and Twigg 2005] generate a specific bone hierarchy and skin animation for a given number of bones. In our approach, we produce a progressive character skin, that is one skin with a skeleton where the number of bones can be increased or decreased at run-time. Thus, we can balance computational load against vertex errors without the extra data requirements of storing multiple sets of skinning skeletons and weights. Additionally, we smoothly blend a bone on & off through a small number of frames. This avoids popping artifacts that could otherwise occur when switching between static skeleton LODs. We have found this mechanism very useful for crowd animations, permitting many characters to use the same skinning data but consume computational time in proportion to their visual importance.

The main performance bottlenecks during skinning playback are: the animation of the bone matrices (CPU bottleneck), the number of non-zero bone weights per skin vertex (CPU or GPU) and the number of mesh polygons (CPU or GPU). Unfortunately, the high data and computation costs of progressive meshes [Hoppe 1996] 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. Our method is aimed at handling these bottlenecks by manipulating the LOD of the bone animation. LOD selection can be based on CPU/GPU budgets and scene state.

Playback 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 transformation $M_{b,f}$ with the vertex's bind pose v_i (eq. (1)). Typically, a vertex in a games character will have no more than four non-zero weight values.

$$v_{i,f} = \sum_{b=1}^{B} w_{i,b} v_i M_{b,f} \qquad \sum_{b=1}^{B} w_{i,b} = 1 \qquad 0 \leq w_{i,b} \leq 1 \qquad (1)$$

As with standard skinning, we use a hierarchical data structure so the bones inherit transformation data. But progressive skinning also uses the hierarchy to propagate the weighting data, so a selected LOD can be played. Bones in the hierarchy are ordered by decreasing complexity in the animation. The LOD playback animates the number of bones that the system has time for and copies these 'active' bone matrix values down the hierarchy to their 'inactive' children. Thus, in avoiding costly alterations to the vertex stream data, the method is suitable for current GPU skinning implementations. Additionally, the non-zero weight values are ordered in the vertex stream based upon minimal error. Thus, a progressive skinning system may transfer later weight values to earlier weights to reduce the number of non-zero weight values and overcome the remaining performance bottleneck.

Figure 1: One frame from an animation compressed using progressive skinning. From the left, using 1, 2 & 4 bones. On the right is the original 42 bone rig.

Generation Given the operation of the playback system, the generation of skinning data is comparatively open ended. We have implemented two initial methods, which are appropriate for different situations.

Collapse Existing Skin In this approach, we restructure the artist's rigid skeleton keeping the original weights to permit perfect reproduction of the original animation. A new bone structure is obtained by reordering bones to minimize accumulated 'collapse' error (similar to polygonal edge collapse). Alternatively, the bone matrices can be flattened to world space and we then build them into a new minimal-error hierarchy. For both methods we adjust the mesh bind pose and bone matrices to minimize bone collapse errors.

Build From Scratch For meshes with no existing skeleton, we implemented a top-down strategy for adding bones one at a time. This method is often capable of finding visually acceptable 'approximate' solutions using only a few bones. The bind pose is set as the average of the vertex positions and the first 'root' bone (weight $w_{i,0} = 1$) found by a least squares analysis of each animation frame. Additional bones use least squares on the outstanding errors to determine what proportion of the weight to draw off from its parent and then derive the transformation for each animation frame.

References

HOPPE, H. 1996. Progressive meshes. In *Proceedings of ACM SIGGRAPH 1996*, ACM Press / ACM SIGGRAPH, New York, Computer Graphics Proceedings, Annual Conference Series, ACM, 99–108.

JAMES, D., AND TWIGG, C. 2005. Skinning mesh animations. *ACM Transactions on Graphics 24*, 3, 399–407.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Transactions on Graphics 22*, 3, 562–568.

*e-mail:S.Pilgrim@cs.ucl.ac.uk

†e-mail:AAguado@europe.ea.com

‡e-mail:KMitchell@europe.ea.com

§e-mail:A.Steed@cs.ucl.ac.uk