A Dynamic Caching System for Rendering an Animated Crowd in Real-Time

W. Lister, R. G. Laycock and A. M. Day

School of Computing Sciences, University of East Anglia, UK

Abstract

We present a method to accelerate the rendering of large crowds of animated characters. Recent trends have seen matrix-palette skinning become the prevalent approach due to its low memory overhead and fully dynamic geometry. However, the performance of skeletal animation remains modest in comparison to static rendering since neither temporal nor intra-frame coherency can be exploited. We cast crowd rendering as a memory-management problem and allocate a small geometry cache on the GPU within which animated characters can be stored. This serves to augment matrix-palette skinning with baked geometry and allows animation frames to be re-used by multi-pass rendering, between multiple agents and across multiple frames. Our method builds its cache dynamically and adapts to the current simulation state through use of the page-replacement algorithms traditionally employed by virtual-memory systems. In many cases this negates the need for skinning altogether and enables thousands of characters to be rendered in real-time, each independently animated and without loss of fidelity.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism — Animation.

1. Introduction

As real-time virtual environments continue to strive towards photorealism, their enrichment with crowds of high quality and diverse characters becomes essential for the provision of an immersive experience. Commonly applied to urban simulations [CLM05, DHOO05] and cultural heritage visualizations [dHCSMT05, RFD05] to populate otherwise sterile worlds, the gaming industry has demonstrated that crowds need not be confined to passive roles but can instead become fundamental to the success of a game [Ubi07].

This paper considers the problem of how best to render large crowds of animated characters. Linear-blend skinning is a commonly applied technique and is favoured due to its low memory requirements and fully dynamic geometry. In this approach, each vertex of a mesh is bound to an animated skeleton using a set of bone assignments. Vertices are transformed independently by a weighted summation of the corresponding bone keyframe matrices and this lends itself well to parallel implementation. In [Lee07], seven variants of the basic GPU algorithm are identified. Of these, matrix-palette skinning stores an animation library within a vertex texture and when combined with instancing, is a powerful method by which to render many thousands of characters.

Although matrix-palette skinning can be implemented efficiently on both current-generation graphics hardware [Dud07] and future architectures supporting a tessellation pipeline [SBOT08], its performance remains modest when compared to that of rendering static meshes. Our work accelerates the current state-of-the-art by introducing a baked geometry caching system.

Baked geometry has previously been used as a standalone rendering technique when hardware limitations saw it impractical to skin an entire crowd in real-time. Therefore the accepted approach was to avoid having to perform dynamic animation at all and Ulicny et al. [UdHCT04] describe how an animated character can be cached on the GPU as a set of pre-transformed meshes which are played-back during simulation. This method requires all keyframes to be pre-computed and so memory constraints naturally serve to limit locomotive diversity. However, for the small number

of animations that can be baked, rendering is extremely efficient since the system has only to interpolate vertices. By way of example, we implemented both baked geometry and four-bone matrix-palette skinning as preliminary work and found the former to run approximately twice as fast. Whilst it is easy to envisage a hybrid system that incorporates both methods, treating them as distinct techniques still constrains characters rendered using the baked approach to a limited selection of animations created offline.

2. Skinning a Crowd

Matrix-palette skinning is illustrated in Figure 1. The bone transformations for each keyframe of a given animation are stored as **3x4** matrices within a single slice of a texture array. A vertex constructs its final transformation matrix by blending those of its assigned bones; keyframe interpolation is achieved using hardware texture filtering.

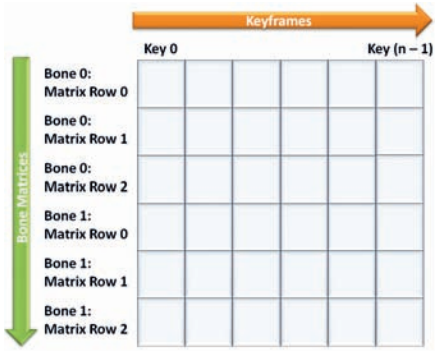

Figure 1: *Matrix-palette skinning animation texture layout.*

For a given animation, the horizontal texture coordinate at normalised time **t** can be calculated using Equation 1 where **n** is the number of keyframes in the target animation and **w** is the width in texels of the texture array. All animations are assumed to be regularly sampled but are not required to use the same sampling frequency.

$$\text{tex.u} = \frac{(\mathbf{n}-1)\mathbf{t}+0.5}{\mathbf{w}} \qquad (1)$$

Skinning an entire crowd of characters every frame is not only expensive but also unnecessary. Given a target framerate of 60 fps and a simple walk-cycle animation running at 7.5 keyframes per second, successive keyframes are interpolated for eight rendered frames. Instead of skinning the interpolated keyframe, the same transformation can be achieved by skinning those keyframes immediately preceding and following the current animation time and blending between them. For the example given, this reduces eight expensive skinning operations to two skinning and eight low-cost interpolations, without loss of fidelity. A net performance gain can be expected provided that baked keyframes remain valid for long enough so as to offset the cost of their generation. A similar philosophy is applied by a dynamic-impostor crowd rendering system in [ABT00] and is the premise that underpins our technique.

3. Geometry Caching

We use matrix-palette skinning to populate and maintain a cache of baked meshes from which characters are rendered by interpolating pre-transformed keyframes. The vertices baked by our system are determined solely by the current state of the crowd simulation; only those actively required by crowd members need exist within the cache. Furthermore, cache size can be explicitly controlled and thereby provides a mechanism by which to balance memory consumption and computational performance.

A single cache is shared by all crowd members and this enables locomotive coherencies within the crowd to be exploited. These can be categorised into two distinct types; temporal coherency, whereby a single character interpolates the same animation keyframes for several rendered frames and intra-frame coherency, whereby multiple characters interpolate the same keyframes within a single rendered frame. There also exists another type of coherency, a hybrid of those described, whereby the keyframes currently required by a character have been recently used by others within the crowd. Our system can exploit all three scenarios without distinction and this allows baked animation frames to be reused by multi-pass rendering, between multiple agents and across multiple frames.

We describe our method by way of an example which uses a crowd of 14 characters, a single animation of 7 keyframes and an available cache size of 4 meshes. The simulation begins at time **t** and its current state is illustrated in Figure 2. The white numbers denote how many characters are currently interpolating the enclosing keyframes; we denote the regions between successive keyframes as temporal bins.

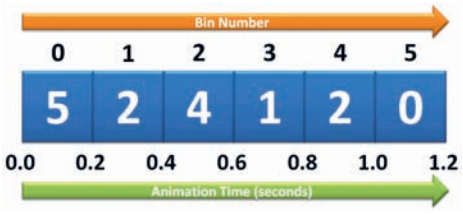

Figure 2: *Crowd simulation state at time **t**.*

To ensure maximum utility of baked keyframes, the cache is populated by first sorting temporal bins by size and then adding their enclosing keyframes in descending order until the cache is full. Thus, the set of keyframes used by bins 0 and 2 { 0.0, 0.2, 0.4, 0.6 } are rendered to the cache.

Crowd members are subsequently filtered dependent upon whether they can be rendered from the cache. Clearly, those characters from bins 0 and 2 qualify as it was their keyframes which were used in its construction. The characters in bin 1 also note the presence of their keyframes. However, the character in bin 3 has only one baked keyframe available and those in bin 4 have neither. Thus, for this frame, a total of 11 characters can be rendered by interpolating cached keyframes and the remaining 3 use skinning. The frame has a total cost of 7 skinnning operations (4 cache updates and 3 skinned characters) and 11 blending operations.

At time **t + dt**, the simulation state is as shown in Figure 3 and at this timestep, the set of keyframes belonging to bins 1 and 3 { 0.2, 0.4, 0.6, 0.8 } are required to exist within the cache. Of these, { 0.2, 0.4, 0.6 } are already present; only keyframe { 0.8 } needs to be baked and replaces keyframe { 0.0 }. As before, 11 characters can be rendered from the cache and 3 use skinning. However, due to the exploitation of temporal coherency, the total cost of this frame is reduced to 4 skinning operations (1 cache update and 3 skinned characters) and 11 blending operations.

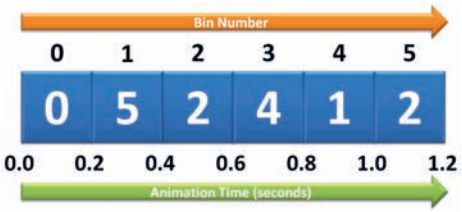

Figure 3: *Crowd simulation state at time **t + dt**.*

3.1. Page-Replacement Algorithms

Given this mechanism for accessing and updating the cache, the question remains of how to manage it effectively so as to extract maximum performance. This is a well studied problem within the field of virtual-memory systems and for which there are many existing algorithms. Within the context of this work, a baked mesh is analogous to a page of memory and there are two factors which pertain to caching efficiency; that of when a baked mesh was last used and that of how frequently it is accessed.

The least-recent used (LRU) page-replacement algorithm assigns a timestamp to each element within the cache and is updated each time the element is accessed. Whenever a new item is added to the cache, that with the oldest timestamp is replaced. By contrast, the not frequently used (NFU) algorithm assigns a counter to each element and is incremented each time the element is accessed. When replacing a cached item, that with the lowest frequency is overwritten. The incorporation of page-replacement algorithms enables our system to perform a minimal number of cache updates and adapt to the current simulation state transparently. The performance of both is evaluated in the following section.

4. Performance Analysis

We implement the geometry cache as a vertex texture. Each baked mesh is stored by a single row of texels and an index table is maintained by the CPU. Cache updates are performed by binding the texture to a framebuffer object and rendering animated vertices to the required row. Multiple keyframes can be baked simultaneously using instancing.

The characters within our crowd share a consistent texture parameterization and this enables rendering from the cache to be initiated by instancing a set of indexed texture coordinates. Each instance loads its source and target vertex attributes and blends between them using linear interpolation. The unified shader architecture of current-generation GPUs ensures that vertex-texture fetch (VTF) is sufficiently fast so as not to become a bottleneck when processing vertex data. Assuming that meshes are optimized for post-transform efficiency and buffer locality [NVI04], vertices are spatially close and benefit from texture caching. We incur minimal overhead when rendering using this approach in comparison to streaming static attributes directly from VBOs.

Figure 4 shows the effect of a variable cache size upon our system and compares its rendering performance to that of static meshes, baked geometry and matrix-palette skinning. The presented results relate to a crowd size of 1,024 characters using five animations; we observe the same trendline across various crowd sizes.

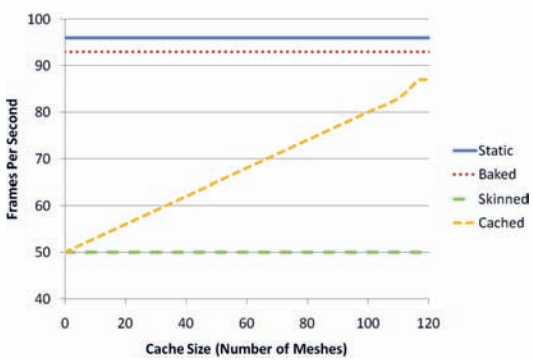

Figure 4: *A comparison of four rendering techniques as applied to a crowd of 1,024 characters.*

The framerates attained by our technique increase almost linearly as the cache size is increased. When the cache size becomes sufficient to store all keyframes of the animations in use (117 keyframes) our system emulates a baked renderer since cache updates and skinning become unnecessary. We never quite achieve the performance of baked geometry due to the overhead of temporal binning.

We found negligible difference between the memory-management algorithms tested. This is because we simulate a worst-case scenario and initialise each character with a random position in a different animation. This leads to the fre-

quency and temporal keyframe distributions being broadly uniform and imposes a lower-bound upon the coherencies that can be exploited.

Once the geometry cache has been populated, relatively few updates are required per frame in order to maintain it. Given a generous cache size, the majority of a crowd can be rendered directly from the cache at minimal cost. This saves precious GPU cycles for those characters requiring specific locomotive individualisation, such as motion blending, or non-linear animation techniques.

To test the scalability of our system we render a textured crowd of 10,000 characters and light the scene with per-pixel lighting. Using a 2.4 Ghz Intel Core 2 Duo, 2 GB of memory and a Nvidia GeForce 8800 GTX we attain 5 fps using matrix-palette skinning, 8 fps using dynamic caching with a cache size of 64 meshes and 10 fps with a cache size of 128 meshes. The latter cache size is sufficient to store all of the baked meshes required by the crowd. Each mesh is approximately 3,300 triangles and 2,100 vertices. This yields a throughput of 330M textured and lit triangles per second and the latter cache size requires 6.15 MB if vertices are assumed to have position and normal attributes. Using the vertex compression scheme described in [SBOT08], this can be reduced to 4.10 MB whilst also including a tangent attribute.

5. Conclusions and Future Work

We have described a caching system which accelerates the rendering of real-time crowds by exploiting both temporal and intra-frame coherency. Our technique retains the memory efficiency, animation fidelity and versatility of skinning whilst approaching the performance of rendering static geometry. In contrast to pure baked geometry systems, we animate characters dynamically and operate a geometry cache which is governed by a page-replacement policy. The system adapts to the crowd simulation state and can be configured to balance memory utilization and computational cost.

Rendering performance is tempered primarily by cache size but we have yet to incorporate additional attributes such as animation diversity and crowd size. Instead of defining the memory available to our system heuristically, we would prefer a formal relationship between these parameters. It is likely that probabilistic analysis could then be applied and used to determine an appropriate cache size, predict run-time performance and author cache-aware crowds.

The technique can be further improved by incorporating cache compression and the use of keyframe reduction techniques to extend the longevity of baked meshes. Additionally, our system assumes that all crowd members are geometrically identical, future work will address this constraint.

6. Acknowledgments

This programme of research is supported by EPSRC grant EP/E035639/1.

References

[ABT00] AUBEL A., BOULIC R., THALMANN D.: Real-time display of virtual humans: Levels of detail and impostors. *IEEE Transactions on Circuits and Systems for Video Technology 10*, 2 (2000), 207–217.

[CLM05] COIC J.-M., LOSCOS C., MEYER A.: Three lod for the realistic and real-time rendering of crowds with dynamic lighting. *Research Report, LIRIS, Lyon University, France* (2005).

[dHCSMT05] DE HERAS CIECHOMSKI P., SCHERTENLEIB S., MAIM J., THALMANN D.: Reviving the roman odeon of aphrodisias: Dynamic animation and variety control of crowds in virtual heritage. In *Proceedings of the 11th International Conference on Virtual Systems and Multimedia* (2005).

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: a real-time geometry / impostor crowd rendering system. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM, pp. 95–102.

[Dud07] DUDASH B.: Animated crowd rendering. In *GPU Gems 3* (2007), Addison-Wesley, pp. 39–52.

[Lee07] LEE M.: Seven ways to skin a mesh: character skinning revisited for modern gpus. *Gamefest Unplugged (Europe)* (2007).

[NVI04] NVIDIA: http://www.developer.nvidia.com/. *NvTriStrip Library* (2004).

[RFD05] RYDER G., FLACK P., DAY A. M.: A framework for real-time virtual crowds in cultural heritage environments. In *VAST '05: Short Papers Proceedings* (2005), pp. 108–113.

[SBOT08] SHOPF J., BARCZAK J., OAT C., TATARCHUK N.: March of the froblins: simulation and rendering massive crowds of intelligent and detailed creatures on gpu. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes* (2008), pp. 52–101.

[Ubi07] UBISOFT: Assassin's creed (2007).

[UdHCT04] ULICNY B., DE HERAS CIECHOMSKI P., THALMANN D.: Crowdbrush: Interactive authoring of real-time crowd scenes. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 243–252.