# Geometric Diversity for Crowds on the GPU

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

School of Computing Sciences
University of East Anglia
Norwich, NR4 7TJ, UK

{ w.lister, Robert.Laycock, Andy.Day } @uea.ac.uk

## ABSTRACT

Pure geometric techniques have emerged as viable real-time alternatives to those traditionally used for rendering crowds. However, although capable of drawing many thousands of individually animated characters, the potential for injecting intra-crowd diversity within this framework remains to be fully explored. For urban crowds, a prominent source of diversity is that of clothing and this work presents a technique to render a crowd of clothed, virtual humans whilst minimising redundant vertex processing, overdraw and memory consumption. By adopting a piecewise representation, given an assigned outfit and pre-computed visibility metadata, characters can be constructed dynamically from a set of sub-meshes and rendered using skinned instancing. Using this technique, many thousands of independently clothed, animated and textured characters can be rendered at 40 fps.

## Keywords

Crowd rendering, crowd diversity, GPU techniques.

## 1. INTRODUCTION

As real-time virtual environments continue to strive towards photorealism, their enrichment with high-quality and diverse crowds becomes essential for the provision of a truly immersive experience. Applied by urban simulations [CLM05, DHOO05, TLC02a] and cultural heritage visualizations [dHCSMT05b, RFD05] to populate otherwise sterile worlds, recent work within the gaming industry has shown that crowds need not be confined to passive roles but can instead become fundamental to the success of a game [Ubi07].

This work focuses purely upon the rendering aspect of crowd simulation which typically requires three criteria to be addressed; namely the quantity, quality and diversity of characters. With current techniques capable of rendering many thousands of agents, it can be contended that the former objective has largely been achieved and the principal problem is now that of how to increase the individuality and fidelity of each. However many previous works are constrained by their dependency upon pre-computation; memory limitations naturally restrict the number of impostors and baked-meshes that can be stored.



To this end, skinned instancing (discussed in Section 3) has recently emerged as a viable alternative to the hybrid techniques traditionally employed by real-time simulations. The capability of skinned instancing to render large numbers of animated characters has been demonstrated on both current-generation graphics hardware [Dud07] and future architectures supporting a tessellation pipeline [SBOT08]. However, besides colour modulation and multi-texturing, the potential for intra-crowd diversity is yet to be explored.

Within an urban setting, a prominent source of crowd diversity is that of clothing and ideally each agent should be individually dressed. However, for crowds comprised of impostors and baked-geometry, this is inhibited by pre-computation which typically serves to consolidate characters and clothing into a single representation. Thus, the opportunities for variety are limited to image-space techniques since if a different outfit is required then an alternative set of impostors and baked-meshes must be provided. In contrast, our approach decouples characters from clothing and defers outfit assignment until rendering.

The contributions of this work can be summarised as follows.

- A method is presented to add geometric diversity to urban crowds through the addition of clothing.

Given a skinned template mesh and a selection of pre-fitted outfits, characters can be independently clothed by assigning a different outfit to each at run-time. The technique is memory efficient and fully compatible with skinned instancing.

- Rendering clothing directly over a character leads to redundant vertex processing and overdraw. To address this, a piecewise representation is adopted whereby a character is divided into a set of sub-meshes and the visibility of each is tagged for all outfits as a pre-process. This metadata can then be used to construct characters dynamically during rendering whilst ensuring that only visible regions are drawn, irrespective of the clothing assigned.

## 2. RELATED WORK

Recent years have witnessed the problem of real-time crowd generation receiving significant research interest. In the following, prominent rendering and intra-crowd diversity techniques are discussed.

With the geometric complexity of a typical crowd scene being prohibitively high for graphics hardware of the time, many early works explored image-space techniques as a means to accelerated rendering. In [ABT00], the application of dynamic impostors was proposed. In this approach, a character is animated, rendered into a texture and then mapped onto a camera-oriented quad to be displayed in place of the original model. By exploiting temporal coherency, the cached image (impostor) may be reused in subsequent frames until sufficient variation in posture or viewing angle evokes an update of the texture.

Subsequent work by Tecchia et al. [TLC02a] describes a system within which impostor generation is performed as a pre-process by sampling from the surface of a hemisphere at 32 positions and at 8 elevations. For each frame of animation, samples are stored within a compressed texture and the most appropriate is selected during rendering, dependent upon the current viewport and animation frame. Variety is achieved using colour modulation together with multi-pass rendering whereby the alpha channel of an impostor texture is used to mask the regions of a character shaded by each pass. Later work described in [TLC02b] considers both normal mapping and shadow generation.

The primary advantage of impostors over the alternative techniques discussed by the remainder of this section is their efficiency of rendering. An implementation by Millan et al. [MR07] using pseudo-instancing [NVI04a] achieves interactive framerates for crowds in excess of one million individuals, suggesting impostors to be ideal for large-scale applications such as within stadia.

However, this performance comes at the expense of memory consumption, image-quality and diversity. For example, despite using texture compression, the system of Tecchia et al. [TLC02a] requires 256 KB of memory per-impostor, per-frame. Consequently, they limit the resolution to a maximum of $256^2$ texels and permit only ten frames of animation.

For applications requiring higher fidelity rendering, Ulicny et al. [UdHCT04] introduces the concept of baked-geometry and describes how an animation can be cached on the GPU as a set of pre-computed meshes. By providing a separate mesh for each frame and sorting for temporal locality, efficient rendering is achieved using display lists. In a novel method, crowd variety can be introduced through interactive authorship of a scene.

Dobbyn et al. [DHOO05] builds upon the work described in [UdHCT04] to construct a hybrid system within which crowds are comprised predominantly of impostors and enriched by baked-geometry. Through the introduction of LOD, this enables the efficient rendering of a crowd whilst affording increased quality for agents local to the viewer. A ‘pixel-to-texel’ ratio is used to switch between representations when an impostor becomes sufficiently close such that a single texel maps to more than one pixel on the screen. In the spirit of Tecchia et al. [TLC02a], crowd variety is introduced by encoding the customisable regions of an impostor within the alpha channel of its texture. However, afforded by the luxury of shaders, colour modulation can be applied in a single pass by using region identifiers to address a one-dimensional palette texture. Additional textures allow impostors to be clothed in different outfits and normal mapping is used to increase shading accuracy. The later work of Coic et al. [CLM05] notes that the visual disparities between impostor and geometry representations are often sufficient as to induce artifacts during LOD transitions. Consequently, Coic et al. introduced an intermediate LOD through the adaptation of layered impostors [Sch98].

Motivated by the rapid evolution of graphics hardware, recent research has increasingly sought to remove the limitations imposed by pre-computation and focused upon the development of dynamic, geometry-based crowd rendering systems capable of running primarily on the GPU. As the first work within this area, Gosselin et al. [GSM04] developed several techniques, many of which were to later become fundamental to the current state-of-the-art [Dud07, SBOT08]. Of particular interest, Gosselin et al. implemented a limited form of instancing to reduce the API overhead associated with rendering thousands of meshes, performed skeletal-subspace deformation (SSD) within a vertex shader (hardware skinning) and promoted the use of 3x4 matrices for

storing bone transformations. Applied to a crowd of soldiers, variety is achieved through the standard approach of colour modulation and masking although, in contrast to the work described in [DHOO05], a two-dimensional palette texture is used whereby texels on adjacent rows denote the range of tints that may be applied to a given character. Further diversity is added through the application of decals and image-quality addressed by the use of pre-computed ambient occlusion, normal mapping and pre-generated shadow maps. In [dHCSMT05a], together with geometric LOD, hardware skinning is later used to animate a crowd of Romans within a cultural heritage simulation.

At the forefront of current research, work described in [Dud07, SBOT08] demonstrates how geometry instancing [Car05] can be used to apply hardware skinning across a crowd of characters with far greater efficiency than was possible at the time of [GSM04].

## 3. INSTANCED CROWD RENDERING

Previous work has demonstrated the emergence of skinned geometric crowds as a viable alternative to those traditionally realised by impostor and baked-geometry systems. Efficient rendering can be achieved by instancing a crowd from a limited set of template meshes and using shaders to perform skinning, world-space transformations and variety introduction on a per-agent basis. However, within this framework, the incorporation of visual diversity remains a challenging problem and current methods operate predominantly at the fragment level through the adoption of image-space techniques such as masked colour modulation and multi-texturing.

Whilst this is sufficient for the work described in [Dud07, SBOT08], which both consider crowds of animated fantasy characters, within an urban setting the incorporation of geometric diversity is necessary to enable characters to be individually clothed. Such is the focus of this section which, following a review of skinned instancing, presents a compatible method for the rendering of clothed, real-time crowds.

### Skinned Instancing

Skinned instancing is a hybrid rendering technique that uses geometry instancing to apply skeletal-subspace deformation (SSD) [MTLT88] across a large number of characters. The former refers to a mechanism through which an application can render multiple instances of a template mesh using a single draw call. When applied to the problem of crowd simulation, this high-level functionality can relieve a CPU from the burden of issuing expensive rendering instructions for every agent. Instead, a buffer object containing per-instance parameterizations is bound and used by the GPU to customise the template mesh for each instance.

$$x = \left( \sum_{i=0}^{k-1} w_i \, T_i \, T_{o,i}^{-1} \right) u \qquad (1)$$

Presented formally by (1), SSD transforms each vertex, $u$, of an animated mesh by a weighted blend, $w$, of the bone transformations, $T_i T_{o,i}^{-1}$, from which it receives influence. As shown in [Dud07, SBOT08], this can be mapped to current-generation GPUs by multiplying the inverse pose-space transformation, $T_{o,i}^{-1}$, of each bone by the corresponding bone transformation, $T_i$, for each keyframe and storing the results within a floating-point texture. This is illustrated in Figure 1. If all transformations are assumed to be affine then it is only necessary to store the upper three rows of each matrix.

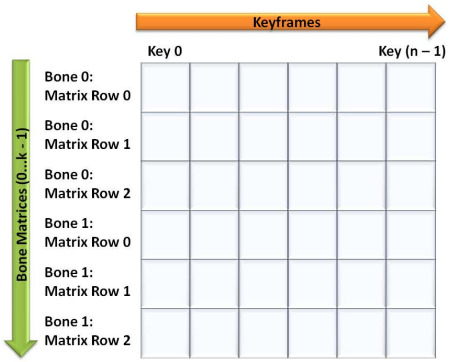


**Figure 1.** Skinned instancing animation texture.

### Adding Diversity through Clothing

Given a skinned template mesh and a set of pre-fitted clothing, this section seeks to use the template as the basis for a crowd and inject diversity by assigning a different outfit to each character. Conceivably, this can be achieved in three ways.

#### *3.1.1 Naïve*

An initial approach, conceptually similar to that of [Dud07], is to create a list of the characters which use a given outfit, instance the template mesh for each character and then for each outfit, draw the required number of instances. Although simple to implement, this method is inefficient since large regions of the template are occluded by clothing. Clearly, outfits should be drawn first to minimise overdraw but a more significant problem is that of having to process the occluded faces at all. This implies the redundant transformation of skinned vertices which is especially undesirable given their high cost of evaluation.

### 3.1.2 *Pre-fitted*

An alternative technique could instead embed a copy of the template mesh within each outfit and remove hidden faces as a pre-process. However, this requires additional memory since a near-complete version of the mesh is stored for each outfit and much of the resulting geometry arises from duplication.

### 3.1.3 *Tagged*

The method proposed by this work is to segment the template mesh into sub-sections and as a pre-process, determine the visibility of each for all of the outfits provided. This metadata can be used during rendering to ensure that only the visible regions of a character are drawn, irrespective of the clothing assigned.

Naturally, there are many ways in which the template mesh can be segmented and most of the impostor works discussed previously allude to the use of anatomical divisions when defining the image-space masks used by colour modulation techniques. By contrast, for geometric rendering, meshes are often divided into a minimum set of sub-regions as a result of material assignments. This is illustrated by the leftmost image in Figure 2 and provisionally these regions are used to avoid further mesh fragmentation.

As is illustrated by the center and rightmost images in Figure 2, sub-region boundaries are typically not in direct correspondence with those of the clothing and as a result, some segments are partially occluded. In this case, the entire region could be drawn but this is undesirable for those where only a small proportion is visible. Instead, a copy of the sub-region is provided, specific to the outfit, with the occluded faces removed. This allows redundant vertex processing to be minimised whilst having only a modest impact on memory usage since, in contrast to the previous method, only visible regions which differ from those of the template are stored. The approach naturally emits a piecewise representation whereby the majority of a character is rendered using sub-meshes from the template mesh and for partially visible segments, outfit specific sub-meshes are used.

## Content Preparation

The character and clothing models used throughout this work are intended primarily for offline use and as a consequence, are too highly tessellated for real-time rendering. However, for the character, multiple LODs are available and that used by our implementation contains 3.3K faces. All clothing is fitted to this mesh using software provided by [DAZ08]. After rigging and skeletal animation, a copy of the character mesh is embedded within each outfit and occluded faces are removed.

Our tagging algorithm processes this data to generate a set of unique sub-meshes and the corresponding region visibility metadata for each outfit. For all of the clothed characters, each sub-region is compared to the corresponding region on the generic template and if the same numbers of faces are present then it is known to be fully visible. Conversely, if a sub-region of the template is absent in the outfit specific mesh then it is known to be fully occluded. For the case where a sub-region is found but the number of faces differs, the sub-region is only partially occluded and the specialised copy is stored. After tagging, for each outfit there is a list of the fully visible template mesh regions and a set of dedicated sub-meshes for those which are partially occluded.

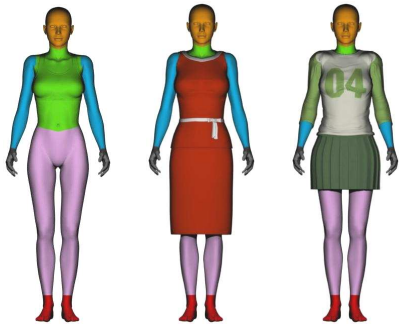

**Figure 2.** Left: Character template mesh with color-coded sub-regions and fitted clothing.

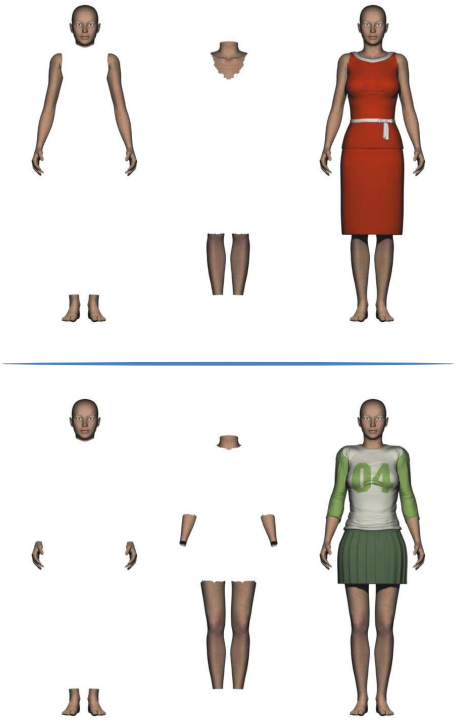

**Figure 3.** Left: Fully visible sub-meshes for the corresponding outfit. Center: Outfit specific sub-meshes determined by visibility. Right: The final composite character.

## 4. RENDERING

On the CPU, this work represents a crowd as a list of characters and stores a 3x4 world matrix, an outfit identifier and an animation time offset for each.

Every frame, the world matrices and animation time for a maximum of *n* characters are consolidated and stored within a constant buffer on the GPU. As described in [Dud07], *n* is bound by the GPU constant buffer size of 4,096 **float4** vectors. We use a total of four **float4** vectors per character and this permits agents to be rendered in batches of up to 1,024 at a time.

From the preceding section, the renderer receives a set of sub-meshes and metadata specifying those that should be rendered for characters wearing a given outfit. To each sub-mesh, an empty list of character identifiers is assigned. Render queues can then be generated by iterating through the CPU character list and for each, appending its position within the list to that of each sub-mesh used by the character. After all characters have been processed, for each sub-mesh it is known how many instances should be drawn and for each instance, the position of its per-character data within the constant buffer on the GPU.

| Sub-mesh | Character IDs |
|---|---|
| Head (generic) | { 0, 1 } |
| Hands (generic) | { 0, 1 } |
| Feet (generic) | { 0, 1 } |
| Arms (generic) | { 0 } |
| Arms (jumper) | { 1 } |
| Torso (generic) | { } |
| Torso (dress) | { 0 } |
| Torso (skirt) | { 1 } |
| Legs (generic) | { } |
| Legs (dress) | { 0 } |
| Legs (skirt) | { 1 } |
| Dress | { 0 } |
| Jumper | { 1 } |
| Skirt | { 1 } |

**Table 1.** Render queue generation.

As an example, in reference to Table 1, consider the two characters shown previously in Figure 3. The first, is wearing a dress and the second, a jumper and skirt. The head and hands of both characters are fully visible and so their character IDs are appended to the corresponding generic sub-mesh lists. For the first character, the arms are also fully visible whereas the legs are partially occluded. Thus, the character ID is appended to the generic arms and dress-specific sub-meshes. For the second character, the arms and legs are both partially occluded and use the jumper and skirt specific sub-meshes respectively. The former case is noteworthy since if a dedicated mesh was not provided, the entire generic arms mesh would need to be drawn even though only a small segment is visible.

The crowd is rendered by iterating through each sub-mesh in turn, uploading character IDs to a constant buffer on the GPU and then drawing the appropriate number of instances; this is given by the length of the character IDs list. Within a shader, each instance use the character ID buffer to map the *instanceID* system variable generated by the GPU to the corresponding per-character data in the buffer shown in Figure. 1. The world transformation matrix and attributes can then be retrieved and skinned instancing implemented as described in Section 3.

The additional layer of indirection is necessary since, due to the piecewise construction, instance *x* of each sub-mesh does not necessarily belong to the same character. For example, in the case described above, instance 0 of the skirt sub-mesh belongs to character 1, not character 0. As a consequence, addressing the per-character buffer using *instanceID* directly would access the attributes for an unintended character.

## 5. RESULTS

To appraise our technique, we compare it to those described in Section 3 with respect to both rendering performance and memory consumption. As the models used by this work are relatively complex, only crowds up to a maximum of 5,000 characters are evaluated. The test system is a 2.4 GHz Intel Core 2 Duo with 2 GB of memory and a Nvidia GeForce 8800 GTX graphics card.

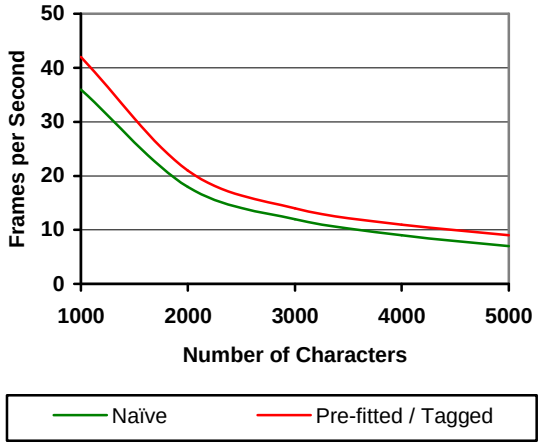


**Figure 4.** Performance tests for the three techniques.

 ISBN 978-80-86943-93-0

Figure 4 compares the three rendering techniques as applied to crowds of up to 5,000 characters. Our test scene is illustrated in Figure 5. As anticipated, performance is increased by removing occluded faces although there is negligible difference as to whether this is accomplished as a pre-process or using the tagged approach of this work.

| Occlusion Technique | Total Number of Faces Stored | | | | |
|---|---|---|---|---|---|
| | Template | C1 | C2 | C3 | Total |
| **Naïve** | 3266 | 2486 | 3568 | 2550 | **11,870** |
| **Pre-fitted** | n/a | 5024 | 5739 | 5001 | **15,764** |
| **Tagged** | 3266 | 2745 (259) | 3788 (220) | 2785 (234) | **12,583** |

**Table 2.** The number of faces stored by each technique for three arbitrary outfits: C1, C2 and C3. Those quoted in brackets denote how many of each total belong to dedicated sub-meshes. C1 and C2 correspond to those outfits illustrated in Figure 3 (top and bottom of figure respectively).

However, the principal advantage of our method is the reduced amount of memory required to attain the same performance. As shown in Table 2, for the pre-computed approach a generic template is unnecessary and a specialised copy for each outfit with the occluded faces removed is stored instead. For the examples presented, this raises memory consumption by 33% over the naïve approach. In contrast, the tagged method increases memory requirements by just 10% since the only additional mesh data stored is that of the specialised sub-meshes for partially occluded regions.

**Figure 5.** A screenshot showing 1,000 characters rendered with geometric diversity in real-time.

## 6. CONCLUSIONS

This work has presented a technique to render a crowd of clothed, virtual humans whilst minimizing redundant vertex processing, overdraw and memory consumption. By adopting a piecewise representation, given an assigned outfit and pre-computed visibility metadata, characters can be constructed dynamically from a set of sub-meshes and rendered using skinned instancing. Using this technique, a geometric crowd of 1,000 individually clothed, animated and textured characters can be rendered in excess of 40 fps.

For the examples shown, culling the regions of a character that are known to be occluded by clothing increased rendering performance by approximately 20%. The advantage of the presented method is that this can be achieved without the provision of a dedicated character mesh for each outfit. In addition to reducing memory requirements, this approach also assists in the provision of supplementary content. For example, if a new outfit was provided for an in-game character, only clothing, replacement sub-meshes and metadata would need to be downloaded. In contrast, the pre-fitted method would use additional bandwidth since a near-complete copy of the template mesh would also be required.

Although the results from Section 5 demonstrate the effectiveness of the technique, diversity is limited since only three clothing sets are used. If diversity was to be increased and ten outfits provided, using the average sizes from Table 2 the total numbers of faces can be extrapolated for both the pre-fitted and tagged approaches. This is shown in Figure 6 and it can be seen that the relative memory consumption of the techniques is inversely proportional to diversity. Our method becomes increasingly memory efficient as additional diversity is introduced and at the limit, requires just 59% of the memory used by pre-fitting.

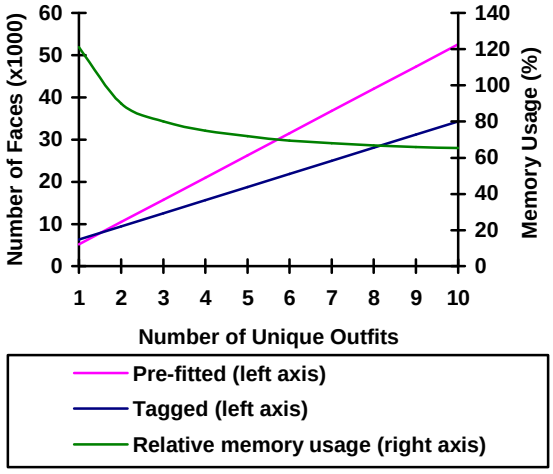


**Figure 6.** Comparing the relative memory usage of the tagged technique against the pre-fitted method.

As enforced by this paper, geometric methods are a viable way by which to render an urban crowd, even for those comprised of relatively complex character meshes. Although only a small crowd of 5,000 agents was demonstrated, we anticipate that much larger crowd will be possible since LOD techniques are yet to be incorporated. Thus, future work will continue to address both the quality and diversity of characters. Morph targets coupled with hardware tessellation may offer potential within this context.

## 7. ACKNOWLEDGMENTS

The models and textures illustrated by this work were purchased from DAZ Productions [DAZ08]. This programme of research is supported by EPSRC grant EP/E035639/1.

## 8. 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.

[Car05] Carucci F.: Inside geometry instancing. In *GPU Gems 2* (2005), Addison-Wesley, pp. 47-67.

[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).

[DAZ08] DAZ Productions: http://www.daz3d.com. *DAZ 3D* (2008).

[dHCSMT05a] de Heras Ciechomski P., Schertenleib S., Maim J., Thalmann D.: Real-time shader rendering for crowds in virtual heritage. In *VAST '05: Proceedings of the 6th International Symposium on Virtual Reality, Archaeology and Cultural Heritage* (2005).

[dHCSMT05b] de Heras Ciechomski P., Schertenleib S., Maim J., Thalmann D.: Reviving the roman Odeon of aphrodisiac: 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'Conner K., O'Sullivan C.: Geopostors: a real-time geometry/ impostor crowd rendering system. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (2005), pp. 95-102.

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

[GSM04] Gosselin D., Sander P., Mitchell J.: Drawing a crowd. In *ShaderX 3: Advanced Rendering with DirextX and OpenGL* (2004), Charles River Media, pp. 505-517.

[MR07] Millan E., Rudomin I.: Impostors, pseudo-instancing and image maps for gpu crowd rendering. *The International Journal of Virtual Reality 6,* 1 (2007), 35-44.

[MTLT88] Magnenat-Thalmann N., Laperrière R., Thalmann D.: Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88* (1988), pp. 26-33.

[NVI04a] NVIDIA: Technical report: Glsl pseudo-instancing. In *NVIDIA SDK 9.52* (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 20008 Classes* (2008), pp. 52-101.

[Sch98] Schaufler G.: Per-object image warping with layered impostors. In *Proceedings of the 9th Eurographics Workshop on Rendering* (1998), pp. 145-156.

[TLC02a] Tecchia F., Loscos C., Chrysanthou Y.: Image-based crowd rendering. *IEEE Computer Graphics and Applications 22,* 2 (2002), 36-43.

[TLC02b] Tecchia F., Loscos C., Chrysanthou Y.: Visualizing crowds in real-time. *Computer Graphics Forum 21,* 4 (2002), 753-765.

[Ubi07] Ubisoft: http://assassinscreed.uk.ubi.com. 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.