Volume xx (200y), Number z, pp. 1–19

A Survey of Real-Time Crowd Rendering

A. Beacco[1,2], N. Pelechano[1] and C. Andújar[1]
[1]Universitat Politècnica de Catalunya
[2]EventLab, Universitat de Barcelona

Abstract

In this survey we review, classify and compare existing approaches for real-time crowd rendering. We first overview character animation techniques, as they are highly tied to crowd rendering performance, and then we analyze the state of the art in crowd rendering. We discuss different representations for level-of-detail (LoD) rendering of animated characters, including polygon-based, point-based, and image-based techniques, and review different criteria for runtime LoD selection. Besides LoD approaches, we review classic acceleration schemes, such as frustum culling and occlusion culling, and describe how they can be adapted to handle crowds of animated characters. We also discuss specific acceleration techniques for crowd rendering, such as primitive pseudo-instancing, palette skinning, and dynamic key-pose caching, which benefit from current graphics hardware. We also address other factors affecting performance and realism of crowds such as lighting, shadowing, clothing and variability. Finally we provide an exhaustive comparison of the most relevant approaches in the field.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

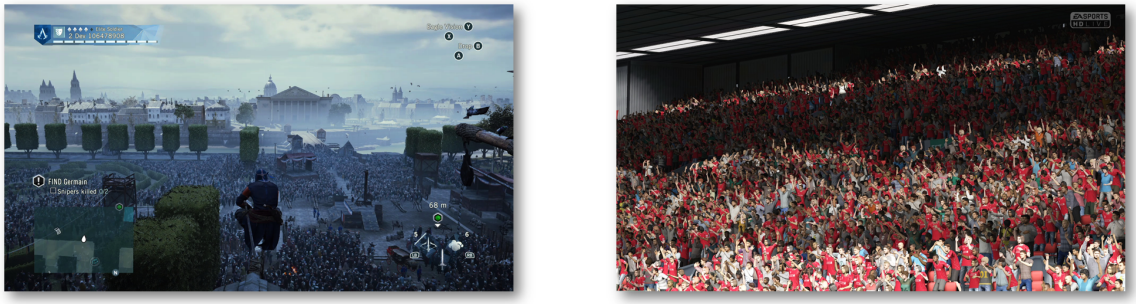
1. Introduction

Crowd simulations [PAB08, TM13] are becoming increasingly important in many computer graphics applications. Although the most prominent use of crowd simulations is found in video games (particularly in the sandbox genre, but also in sport and strategy games), crowd rendering is also crucial in a variety of applications including evacuation planning (like Thunderhead's *Pathfinder* software), crowd management training, phobia treatments, and psychological studies. These applications often need to render in real-time hundreds or thousands of moving agents with a certain level of visual quality and plausibility.

A close look at the video game industry reveals that each new console generation gives rise to a new predominant genre that fully exploits the hardware improvements. While in the past generation (PS3, Xbox 360) first person shooters were predominant, the current console generation (PS4, Xbox one) has lead to a substantial increase of game titles in the sandbox genre. One example is the LEGO's *Traveler's Tales* series, which has evolved from simple action-platformed games, to evolving sandbox environments e.g. in *LEGO Batman 2 DC Super Heroes* and *LEGO Marvel Super Heroes*. This is due to the trend of providing players with more alive and interactive environments with each new game. In order to allow the player to feel immersed inside game environments such as virtual stadiums, villages and cities, the scenes need to be populated with crowds of people that make the environment both alive and believable. In sandbox games this is almost a requirement, and recent games from bestselling series such as Ubisoft's *Assassin's Creed* or RockStar's *Grand Theft Auto* have pushed the limits on the amount of agents shown on screen in real-time. The recent *Assassin's Creed Unity* (Figure 1-a) claims to show up to 12,000 agents in real-time, although just 120 of them are rendered using high resolution models.

Moving to strategy games, we can find massive armies of up to 100,000 soldiers animated in real-time, as in Creative Assembly's *Total War: Rome 2* . Distant soldiers are not required to have individual appearance, animation and behavior, which makes it easier to reach such a high performance. In sport games such as Electronic Arts' *FIFA 15* (Figure 1-b), stadiums can also be filled with up to 120,000 animated virtual spectators, although they remain seated in the same place with no navigation or collision avoidance.

Real-time realistic crowds have thus a massive impact in the current game industry. Despite the substantial advances



(a) (b)

Figure 1: Crowds are an important element in videogames of different genres. (a) *Assassin's Creed Unity*. ©2014 Ubisoft Entertainment. All Rights Reserved. Assassin's Creed, Ubisoft and the Ubisoft loco are trademarks of Ubisoft Entertainment in the U.S. and/or other countries. All other rights are reserved by Ubisoft Entertainment. (b) *FIFA 15*. ©2014 Electronic Arts.

in the field, virtual crowds still suffer from a number of easy-to-spot artifacts, including popping effects when switching between level-of-detail (LoD) representations, limb self-intersections, prominent clones (sharing visual appearance and/or animation), and poorly simulated behaviors. It is therefore necessary to devise new techniques to simulate, animate and render large numbers of autonomous agents in real time, minimizing visual artifacts and optimizing the use of CPU, GPU and bandwidth resources, which have an obvious impact in performance, energy consumption and, in mobile devices, battery life.

Regarding crowd rendering, there has been a large amount of work in this field since the last survey [RD05]. There are also other surveys and books covering partly this topic [ASDB08, PAB08, TM13]. Efficient rendering of crowds is an area with a significant amount of techniques introduced in the last years, and continues to be in need of further innovation as the complexity of virtual environments increases along with the capabilities of the hardware. This survey aims at providing a complete, up-to-date overview of the state of the art in crowd rendering, with an in-depth comparison of techniques in terms of quality, resources and performance.

For a given graphics hardware, there is an obvious trade-off between the number of animated characters that can be simulated in real-time, and the different factors affecting their visual quality, including geometric detail, materials, motion (path within the scenario) and animation (e.g. walking or running cycles). Achieving high-quality real-time crowd rendering is therefore a major challenge, but it is important to note that the rendering process is strongly tied to the simulation process and also to animation computations. Although each of these fields deserves its own survey, we will provide a short overview of character animation techniques (Section 2), keeping the focus on crowd rendering.

Although there is a large variety of crowd rendering acceleration techniques, LoD rendering is the most effective and frequently-used approach. LoD rendering refers to the possibility of drawing objects using different representations with varying complexity and accuracy [Cla76, LWC*02]. In the context of character animation, each character type can be represented with different LoDs; at runtime, the most suitable representation for each character instance is chosen according to an estimation of its contribution to the image, which depends, among other factors, on the distance to the camera.

Current LoD representations for characters can be roughly classified into polygon-based techniques, point-based techniques, and image-based techniques. *Polygon-based techniques* provide various LoD representations of the character geometry in the form of a polygonal mesh (faces are often limited to triangles and quads). These techniques are the de-facto standard in character rendering, despite the difficulties in generating high-quality simplified meshes for animated characters automatically.

Point-based techniques represent characters through point clouds with no explicit connectivity [LW85, Bær05] which can be rendered using a surface splat primitive [ZPvBG01]. *Image-based techniques* replace 3D characters by simple textured-mapped primitives called impostors [ABT98, TC00, DHOO05, KDC*08, BAPS12]. LoD techniques are generally lossy and thus the primary goal is to maximize performance while minimizing visible artifacts.

Besides LoD techniques, a variety of (mostly) lossless techniques can be applied to accelerate crowd rendering with no impact on image quality. *Culling techniques* aim to avoid processing invisible geometry [COCSD03], either because it falls outside the viewing frustum (frustum culling [Cla76, BEW*98, AM00]) or because it is hidden by occluding geometry (occlusion or visibility culling [KS01, WB05]).

Although individual characters have little chances to occlude relevant geometry, the aggregation of many characters, as well as the environment they inhabit, offer great opportunities to optimize rendering performance in a conservative way. Current hardware features offer new possibilities for optimizing the rendering of large crowds. These techniques include primitive instancing, palette skinning, and key-pose caching [MR06b, Dud07b, LLD10].

The remainder of this survey is organized as follows. Section 2 provides an overview of character animation, covering both skeletal and non-skeletal (cages, blend shapes) techniques. Section 3 discusses LoD representations for animated characters, including polygon-based, point-based and image-based techniques. Lossless acceleration techniques, including culling and instancing are described in Section 4. Section 5 reviews lighting characters [JVES*12] and casting shadows onto the agents [TLC02] and Section 6 discusses cloth simulation [MDCO06] and variation of the avatars' appearance [MYT09]. Section 7 provides a comprehensive comparison of all major approaches. Conclusions and a discussion of open problems are provided in Section 8.

2. Character Animation

In this section we give an overview of general aspects of character animation that influence rendering performance. These aspects range from how the character is represented to how this representation is modified to handle animations.

Some character animation methods focus on achieving highly-realistic, physically-accurate mesh deformations for applications without real time requirements. Physically-based methods simulate the internal structures of the body (bones, tendons, muscles and fat tissues [JEOG11, SKP08, MCC11, AT00]), achieve a high level of realism, and might even support dynamic effects such as muscle bulges, but at a high computational cost. Despite the high visual quality achieved by these methods, they are too expensive for rendering a large number of characters under real-time constraints. Since we focus on crowd rendering, we will limit our review of character animation methods to those that can be applied in real time to large groups of people.

2.1. Skeletal animation

The most extended approach for animating 3D characters is *skeletal animation* [MTLT88]. In this technique, the character is represented by a mesh or 'skin', and an underlying skeleton. The skin is an arbitrary polygonal mesh and the skeleton is a hierarchy of bones carefully placed so that they fit inside the skin. Initially both of them are designed in a reference pose, and during run time the mesh representing the skin will be deformed following the bones' movement (see Figure 2).

To determine how the vertices of the mesh will be deformed, each bone is associated with a portion of the mesh.

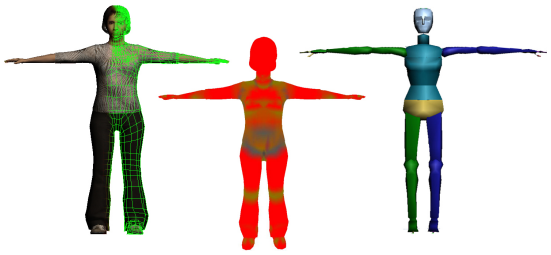


Figure 2: Skeletal Animation: a character is composed by a 3D mesh representing the skin (left) and a skeleton (right). The skinning process assigns each vertex of the mesh to one or more bones of the skeleton with a weight (center). Vertices influenced exclusively by one bone are represented in red, and other colors indicate vertices that are influenced by several bones.

For example, all the vertices forming the left hand will be linked to the left hand bone. In some places, a portion of the mesh is associated to more than one bone, by defining a weight (influence) associated to each one. These vertices will then be deformed based on the weights of all its linked bones that are being moved. Therefore, an animation can be defined by the movement of the bones, and the associated vertices will move along with the skeleton. The process of defining these weights, as well as the skeleton fitting, is called *rigging* and it is typically done manually, although some recent procedures [BP07, RLS08, PS12] simplify the rigging process considerably. Recently developed online tools allow users to rig any biped character within minutes [Mix14]. There is also recent work to transfer animations between characters with different skeletons [BTST12], a process known as retargeting.

An animation can be defined by a series of *keyframes*, each one defining a different *pose* for an instant of time t. Poses consist of a geometric transformation for each *bone* of the *skeleton*. These geometric transformations are usually rotations encoded as matrices, resulting in one matrix per bone and per keyframe. During the animation, new poses can be computed at arbitrary times by spherically interpolating the rotations of the bones between the two closest keyframes.

The actual deformation of the skin geometry for a given skeleton pose is known as geometric skinning, or GPU skinning when its computation is performed in the GPU. *Linear blend skinning* is the de-facto standard for low-cost skinning. Transformations are represented by the skeleton matrices, which are blended linearly according to the applied rigging. Besides skin deformations, linear blend skinning can be used to animate other deformable elements such as cloth, since it is typically faster than physically based cloth simulation [CMT05].

The direct linear combination of matrices is known to suffer from blending artifacts in the deformed skin. A typical

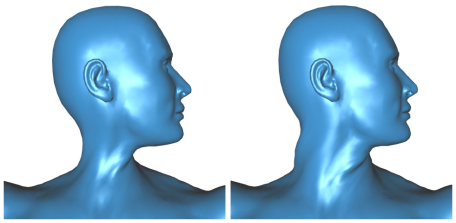


Figure 3: Linear blend skinning (left) produces several artifacts such as the "candy-wrapper" effect, because the resulting matrix no longer represents a rigid transformation. Dual quaternions methods (right) can reduce these artifacts [KS12] (image courtesy of Ladislav Kavan).

artifact is the “candy wrapper” effect, where the skin collapses into itself (see left part of Figure 3). This occurs because the weighted sum of matrices representing rigid transformations (with neither scale nor shear) is not necessarily another rigid transformation, but a general affine transformation. Rigid transformation matrices can be decomposed into quaternion plus translation pairs, which can be independently blended linearly to always get rigid transformations [Hej04, Kv05]. The problem of doing it separately is that transformations become dependent with the body-space coordinate system, meaning that vertices are going to rotate around the origin of the body-space instead of the actual pivot point of the closer joint. *Dual quaternions* approaches [McC90, KCvO07, KCvO08] reduce these artifacts in an elegant way, by blending quaternions whose elements are dual numbers (see right part of Figure 3). Another advantage of quaternions is that they have a more compact representation compared to matrices, thus saving bandwidth for GPU skinning.

Implicit skinning [VBG*13] is a post process applied over a geometric skinning (such as linear blending or dual-quaternions) to handle skin contact effects and muscular bulges in real-time. The typical artifacts of geometric skinning techniques are avoided through an implicit surface representation. Every frame they approximate the mesh by a set of implicit surfaces, which are combined in real-time and used to adjust the position of mesh vertices starting from their smooth skinning position. This process is performed without any loss of detail and can seamlessly handle contacts between skin parts. Since it is a post-process, the method can be added to any standard animation pipeline. The method requires no intensive computation step such as collision detection and can achieve real-time performance for simple animations, although it is not fast enough for crowd rendering.

2.2. Cage-based animation

In recent years cage-based deformations have been applied to character animation. The main advantages of cage-based deformation techniques are their simplicity, relative flexibility and speed. The idea is to use one or more cages enclosing the model to facilitate the animation while preserving the smoothness of the deformed meshes. Chen et al. [CLTL11] presented an efficient approach that can generate both low and high-frequency surface motions such as muscle deformation and vibrations with little user intervention. Given a surface mesh, they construct a lattice of cubic cells embracing the mesh and apply lattice-based smooth skinning to drive the surface primary deformation with volume preservation. Secondary deformations are handled through lattice shape matching with dynamic particles. Gonzalez et al. [GGPCP13] recently proposed a versatile deformation scheme, allowing the usage of heterogeneous sets of coordinates and different levels of deformation, ranging from a whole-model deformation to a very localized one. This locality allows for faster evaluation and a reduced memory footprint, and thus outperforms single-cage approaches in flexibility, speed, and memory requirements for complex editing operations.

2.3. Per-vertex animation

An alternative approach is *per-vertex animation*, also referred to as *shape interpolation*, *blend shapes* or *morph targets*, where vertex positions are stored not only for the reference pose, but also for each animation keyframe [Lor07, WDAH10]. Vertex positions are then interpolated within keyframes to obtain new frame deformations. An advantage of morph target animation over skeletal animation is that it provides artists with more control over the movement because they can define arbitrarily the individual positions of the vertices within a keyframe, without being constrained by skeleton joints. This can be useful for animating cloth, skin, and facial expressions because it can be difficult to conform those shapes to the bones that are required for skeletal animation. In the field of crowd rendering, Ulicny et al. [UCT04] avoid computing the deformation of a character mesh by storing pre-computed deformed meshes for each keyframe of the animation, and then carefully sorting these *static meshes* to take cache coherency into account. Switching consecutive meshes creates the illusion of an animation. Unfortunately, these techniques require a large amount of memory to store the animations and they are seldom used for crowd rendering.

2.4. Animation Individuality

Individuality in crowd animation refers to the possibility of having as many different animations as possible so that individuals within the crowd can be animated with multiple speeds, styles and gaits. In many situations crowds are just animated with a handful of animation clips that are run with a certain time offset to avoid synchronized animations. Although animations have a small memory footprint, the computation of all the blended poses can become a major performance bottleneck. Moreover, the final pose is represented by a set of matrices that must be used to transform all the avatar

vertices. This transformation is usually performed in the vertex shader. Matrices can be computed in the CPU and then sent to the GPU, but this approach consumes a significant amount of CPU-GPU bandwidth. Alternatively, keyframe matrices can be preloaded onto the GPU, at the expense of GPU memory space [SBOT08]. Both approaches thus benefit from matrix compression techniques. It is also possible to reduce the frequency at which animations are updated with no visible artifacts [PO11, MNO07].

3. Level-of-detail representations for characters

A well-known crowd acceleration technique is level-of-detail (LoD) rendering [Cla76, LWC*02], where the appropriate representation of each character is chosen according to its image contribution [PPB*97, HMDO05]. The basic idea is that, as characters are placed farther away from the camera, less details can be perceived on their screen projection and thus simpler, cheaper representations can be used (see Figure 4). Ulicny et al. [UCT04] replaced full geometrical models with lower resolution ones, and were able to create complex scenes with thousands of characters. Pettre et al. [PDHCM*06] significantly improved performance by using four discrete LoD meshes for the humans in their crowd.

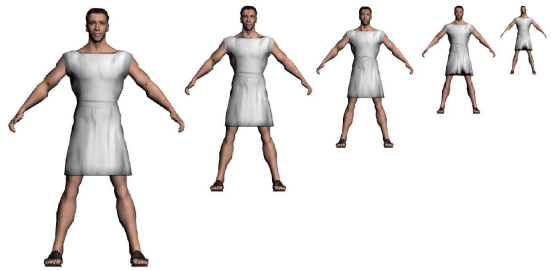


Figure 4: Five models of the same avatar with decreasing number of polygons as they are placed further away.

3.1. Polygon-based techniques

The automatic generation of simplified models is a long-standing problem, and many algorithms have been proposed in the literature for the general case of static meshes. Vertex clustering methods [RB93] group vertices of the input mesh according to a surrounding grid, and then discard the resulting degenerate triangles. Progressive meshes [Hop96] use edge collapse operations to dynamically and progressively reduce the geometric complexity of the mesh. The method stores a sequence of the edge collapses (indeed, their inverses, i.e., pair contractions) and allows a smooth choice of the desired LoD. Unfortunately, conventional mesh simplification techniques typically perform poorly on animated meshes, which require careful preservation of per-vertex attributes such as texture coordinates (specially when these refer to a texture atlas) and, most importantly, blending weights. Failing to preserve vertices around joints leads to severe animation artifacts as even sophisticated skinning techniques produce inconsistent deformations. Larkin et al. [LO11] explored the perception of texture, silhouette and lighting artifacts of different character LoDs. Mesh simplification can be performed in the GPU [DT07] which allow for real time simplification of the geometry. Even recent progressive encoding schemes [LJBA13] are not suitable for mesh animations.

Only a few works address the problem of simplifying animated characters. Schmalstieg and Furhmann [SF99] break the mesh surface into single bone regions (for vertices associated only to one bone) and additional regions for multiple-weighted bones. These surfaces were simplified separately and then stitched together. Some methods try to minimize simplification errors from not just the resting pose, but a set of example poses [MG03, DR05]. Other approaches work with a model and a set of frames with the deformed vertex positions, to build a multiresolution hierarchy, letting the surface change topology for each frame, and therefore showing a simplified surface for each frame [KG05, HCC06, PHB07, ZW07]. The main problem of all these methods is that they are limited to a predefined animation set. Landreneau and Schaefer [LS09] proposed an edge collapse method guided by an error metric that measures deviation from the original deformed shape considering positions and weights. They produce not only new vertices, but also skin weights, and therefore can create keyframes for new animations with new poses from the original ones. Willmott [Wil11] extends classic vertex clustering to handle attribute discontinuities and preserve animation features.

Applications using LoD techniques also need to decide the number of representations to encode (continuous LoD approaches such as progressive meshes are a notable exception), with clear implications in both memory footprint and potential popping artifacts when switching representations. The encoding of the different LoDs of an avatar is also critical to minimize the number of state changes. For example, if the different LoD levels are just reduced versions of the same geometry, and share attribute names, shaders and textures, it makes sense to store the different meshes in the same data structure or to have access to all of them from the same class instance in order to just change the draw call and share the same state.

Tessellation shaders, available in consumer graphics cards since 2011, are able to subdivide faces and thus add detail to existing models without increasing the memory footprint or requiring additional bandwidth. Tessellation shaders can be applied to crowd rendering to generate high-quality representations on-the-fly from a coarse mesh, in a view-dependent manner. Multiple dynamic levels of detail are possible by adjusting the tessellation levels used by the tessellation primitive generator. An interesting fact about tessellation is that when animating a tessellated character, only the base original vertices are transformed, so the new ones

are generated inside the transformed ones. This provides higher resolution for animated models with the same vertex shader performance. Tatarchuk et al. [TBP08] applied these concepts to obtain highly detailed characters for close-up views (see Figure 5). Tessellation can be used in combination with displacement maps to add geometry detail rather than to smooth surfaces. In this case, texture seams can lead to noticeable artifacts unless conveniently handled [TBP08].

3.2. Point-based Techniques

Levoy and Witted's report [LW85] early suggested the use of points as a new primitive to render geometry. The idea is to render a surface using a vast amount of points. A Gaussian filter or surface splatting [ZPvBG01] can be performed to fill in the possible gaps. Kobelt and Botsch presented a survey on point-based techniques [KB04]. But it was Bærentzen [Bær05] who proposed to use point-based models to replace objects that are far away from the camera. Point-based rendering is more useful and faster when the triangles of a model cover a pixel or less (as there is neither triangle setup nor interpolation).

Point-sampled objects do not need to store and maintain globally-consistent topological information. Therefore they are more flexible when compared to triangle meshes. Nevertheless this technique has some limitations. For instance, if the point samples are the result of decimating the mesh for LoD, they become independent from the original mesh and loading animations becomes difficult.

An alternative multi-resolution representation for animated geometry is proposed by Wand and Strasser [WS02] who combine pre-filtered point samples and triangles arranged into an octree. Their randomized sampling scheme guarantees that sample points are distributed sufficiently uniformly on the animated geometry at any time during the animation (see Figure 6), at the expense of requiring a separate multi-resolution hierarchy for each pair of consecutive keyframes. Larkin et al. designed a time-critical system where point samples are distributed for every agent depending on its selected LoD [LO11].

Toledo et al. presented a system where an additional skeleton contains an octree per limb [TDGR14]. Each level of the octree represents a different LoD, and animations can be automatically transferred to each node, thus reducing the memory consumption (see Figure 7).

Figure 5: Tessellation allows rendering large crowds of characters with extreme details in close-up (left). The same character without using tessellation looks significantly less detailed (right) [TBP08] (images courtesy of AMD, Budirijanto Purnomo and Natalya Tatarchuk).

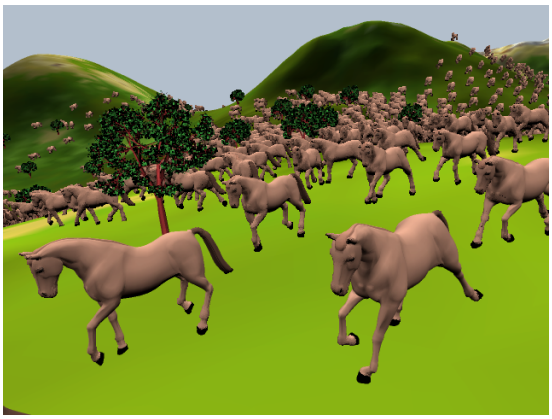


Figure 6: While close-up characters use triangle meshes, background characters can be rendered with point splats [WS02] (image courtesy of Michael Wand).

Figure 7: Some approaches allow animations to be applied to both point-based and polygon-based representations [TDGR14] (images courtesy of Leonel Toledo).

3.3. Image-based Techniques

An impostor is in essence a simple primitive that has the capacity to fool the viewer. As opposed to polygon-based representations, impostors are not just a simplified version of the original geometry, but a different primitive conceived to replace it under appropriate viewing conditions. Impostor representations range from simple billboards (3D sprites) textured with an image of the rendered object, to a small set

of textured polygons allowing the recovery of surface details and parallax effects. Although early impostors were designed for static objects, they can be also used to render animated objects and crowds of agents.

Millan and Rudomin performed a strict comparison between point-based techniques and image-based techniques [MR06a]. Their point-based characters required a variable number of points between 3 and 280, thus resulting in a more inefficient render than an impostor image-based approach using only a quad or two triangles per agent.

Since impostors are essentially images, there are two main approaches: to generate dynamically these images at runtime, or to pre-compute and store them into a texture atlas and access them when necessary.

3.3.1. Dynamic Impostors

The virtual human impostor used by Aubel et al [ABT98] is a simple textured quad which rotates to continuously face the viewer. A snapshot of the virtual human is mapped onto it and re-used over several frames (see figure 8).

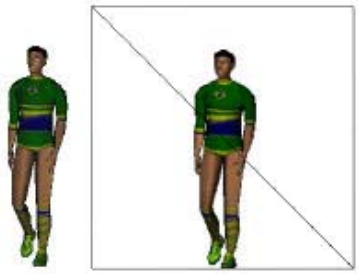
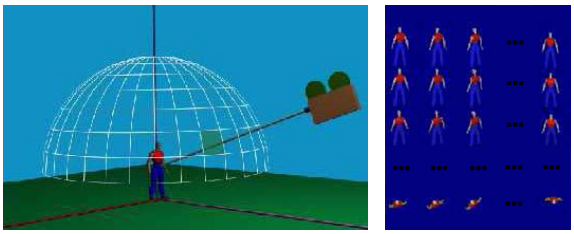


Figure 8: A football player and its (somewhat oversized) impostor [ABT98] (image courtesy of Daniel Thalmann).

As the humanoid moves or the camera moves, the mapped texture might need to be refreshed. To take updated snapshots an off-screen buffer is set up and a multiresolution virtual human is placed in front of the camera in the right pose. The virtual human is then rendered and copied into texture memory and so ready to be mapped onto the billboard.

To decide whether or not to refresh the texture they proposed two fast algorithms. The first one tests distance variations between some pre-selected points in the skeleton, so they can decide if the pose has changed significantly. This obviously sub-samples the animation.

The second algorithm does not test independently the camera motion and the character's orientation because it is not important to know what factor caused the visual variation. Instead, they test the variations of the modelview matrix corresponding to the transformation under which the viewer sees the virtual human. These impostors are dynamic in the sense that they are not pre-computed, but they change dynamically depending on the results of the two algorithms above at every frame.



(a) (b)

Figure 9: Discretising the view direction between the object and the viewpoint (a) allows to generate a texture with all the captured directions for one frame of one animation (b). The process must be repeated for every animation frame [TC00] (images courtesy of Franco Tecchia).

The off-screen buffer can be set up in a pre-process, adjusting the frustum to the character. The stored impostor can also be re-used for other human meshes, and since posing up the character would have been done with the whole geometry, the approach is not slower than rendering the 3D geometry. But even if the impostor is re-used, after a few frames it will finally be discarded.

The main limitation of these kind of approaches is that replacing the whole geometry by a textured plane might introduce occlusion problems. For example, imagine a character sitting on a chair as two independent meshes. If one, or both are replaced by a textured quad, it is not clear how they can be arranged spatially to avoid visibility problems. This is due to the depth values of the impostor fragments which are unlikely to be the same as those of the actual geometry.

3.3.2. Pre-generated impostors

Pre-generated impostors were first used by Tecchia et al. [TC00] by rendering each character from several viewpoints and for every animation frame of a simple animation cycle (see Figure 9). The images were stored in a single texture atlas, and each crowd agent was rendered as a single polygon with suitable texture coordinates according to the view angle and frame.

Pre-generated impostors with improved shading have also been used by Tecchia et al. in [TLC02] by adding shadows to each agent. Since the shadow is just the projection onto the ground of the character's silhouette, they can project the polygon of the impostor onto the ground, using the shadow coverage to darken the ground (see Figure 10). This fake shadow is valid only on planar geometry with a parallel light source, but gives plausible results with small overhead.

Pre-generated impostors can achieve rendering of crowds consisting of tens of thousands of agents. Unfortunately, although image and texture compression techniques can be applied to the resulting texture atlases, they still require large

Figure 10: A scene with shadowing pre-generated impostors [TLC02] (image courtesy of Yiorgos Chrysanthou).

amounts of memory due to the per-view, per-frame replication. Some memory savings can be achieved by removing intermediate frame textures, and generating them online using morphing techniques [YYBE13]. An additional limitation of pre-generated impostors is that, depending on the texture resolution, close-up characters appear clearly pixelated. These impostors do not allow interpolation or blending between two or more different animations (e.g. combining a walk clip with a hand waving clip).

3.4. Hybrid techniques

3.4.1. Geopostors

Dobbyn et al. [DHOO05] introduced the Geopostors, a hybrid system combining pre-generated impostors with a polygon-based representation. Figure 11 shows how impostors are used for far agents while the ones close to the camera are rendered with full geometry.

Figure 11: Geopostors. Far agents are rendered with impostors while closer ones are rendered with geometry [DHOO05] (image courtesy of Carol O'Sullivan).

The switching between the mesh and the impostor is based on the impostor image pixel size to impostor texel size ratio. Ideally this ratio should be 1:1, because aliasing starts when a texel is bigger than a pixel.

An extension of this approach was made by Pettre et al. [PDHCM*06], combining the animation quality of dynamic meshes with impostors and adding a third LoD using the high performance offered by static meshes, i.e. meshes where animated poses were already computed.

3.4.2. Layered impostors

In geopostors, the visual gap between flat impostor and geometry might, for some view directions, be too large to completely avoid popping artifacts. Coic et al., [CLM07] described a similar hybrid system but with three LoDs, adding an intermediate layered impostors LoD between flat impostor and geometry to help achieving continuity during transitions. Instead of a single textured polygon, an adaptive number of layers of the color texture are drawn, depending on the texel's depth (see Figure 12). These layers fill a volume in the 3D scene and can be shaded dynamically using color textures enriched with depth and normal channels.

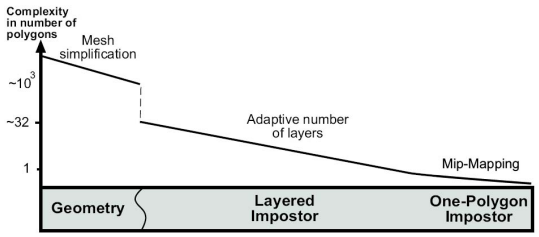


Figure 12: Volumetric layered based impostors rendering scheme: between geometry and the one-polygon impostor, an adaptive number of layers is used for a layered impostor [CLM07] (image courtesy of Celine Loscos).

Layered impostors are rendered in layers parallel to the image plane. For each of these layers, the pixels that correspond to a certain depth are selected. After selecting the required number of layers, they divide the volume captured during preprocessing into as many intervals as the number of layers, defining the intervals of depth for selecting pixels in the color texture. The selection of the right pixels for each depth interval is done in a fragment shader, where lighting is also computed.

To extend the validity of the layered impostors, overlapping depth intervals can be used. Without overlapping, cracks appear on the layered impostor as soon as the viewpoint slightly differs from the pre-computed one. By drawing a small part of the previous and next layers, these gaps are avoided, extending the lifetime of the layered impostor and decreasing the density of precomputed views (see Figure 13).

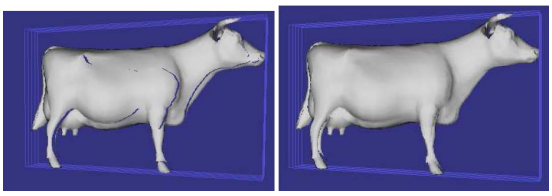


Figure 13: A cow rendered with 5 layers and dynamic lighting, without (left) and with overlapping (right) [CLM07] (image courtesy of Celine Loscos).

Although this approach improves visual quality and fills the gap between the polygonal representation and the flat impostors, it adds additional channels (normal, depth and several layers), which worsens the memory problem. Layered impostors are slower to render than one-polygon impostors.

3.4.3. Polypostors

Polypostors were introduced by Kavan et al. [KDC*08] to reduce the memory requirements of pre-generated impostors whilst mantaining rendering performance. Each polypostor consists of a collection of 2D (planar) primitives, each one representing an individual body part for a given view direction (thus avoiding a per-frame memory consumption).

The original 3D character is cut into several body parts in order to minimize occlusion issues. The original skeletal animation is applied to the body parts. The composition of these body parts gives the same animation as the one provided originally. For the first frame of the animation, each body part is rendered and enclosed within textured 2D polygons, using a standard contour tracing algorithm. For all subsequent frames, an algorithm based on dynamic programming shifts the vertices of the 2D polygons so that they approximate the actual rendered image as closely as possible (see Figure 14). This algorithm matches two textured polygons in an optimal way with respect to a chosen error metric.

At run-time, the deformed polygons are composited in depth order, creating the illusion of an animated 3D character. Since polypostors approximate the animation by deforming their texture, they are not as accurate as other impostors. They can be applied only to animations that can be described as deformations of the initial key-frame. They can produce artifacts with views where there is a lack of texture information in the first key-frame, due e.g. to disocclusion effects.

3.4.4. Per-joint impostors

Two recent approaches by Beacco et al. adopt the polypostor idea of having impostors per body parts instead of per-character. They maximize performance by using a collection of pre-computed impostors sampled from a discrete set of view directions. The first method is based on relief impostors [BSAP11] and the second one on flat impostors [BAPS12]. Characters are animated by applying the joint rotations directly to the impostors, instead of choosing a single impostor for the whole character from a set of predefined poses. This representation supports any arbitrary pose and thus the agent behavior is not constrained to a small collection of predefined clips.

In [BSAP11] each character is encoded through a small collection of textured boxes storing color and depth values (Figure 15). At runtime, each box is animated according to the rigid transformation of its associated bone and a fragment shader is used to recover the original geometry using a dual-depth version of relief mapping [OBM00], recovering surface details and reproducing view-motion parallax effectively, at the expense of some per-fragment overhead. Beyond a certain distance threshold, this compact representation is much faster to render than traditional level-of-detail triangle meshes. Their user study demonstrated that replacing polygonal geometry by per-joint relief impostors produces negligible visual artifacts [BSAP11]. Although this method provides a high-quality representation for distant meshes, its applicability is limited to relatively far-away characters due to the per-fragment cost of the fragment shader. So for close-up agents it is usually faster to render fully animated 3D skinned characters.

A more efficient approach is presented in [BAPS12]. Instead of using six orthogonal relief maps for each joint, which requires multiple dependent texture accesses per fragment, they use flat impostors created by sampling each joint from multiple view directions. These view directions correspond to the faces of a subdivided icosahedron. A spherical Voronoi map is computed from it, and a cube map is built by projecting the Voronoi cells onto the cube faces, thus encoding for each texel the ID of its nearest discrete sample. At runtime, for each fragment, a single cube map texture lookup is enough to retrieve this sample ID, and another one to retrieve the color of the fragment from that sample. Since per-joint impostors are intended to be valid for any pose, a key issue is to properly define which part of the geometry influenced by each joint must be represented as opaque pixels in the corresponding impostor. These parts are encoded

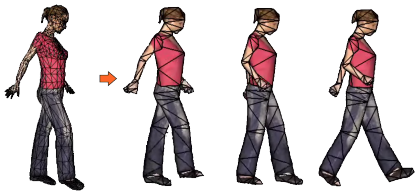


Figure 14: An example of a polypostor animation, overlaid with wireframe. Note that the character animation is created simply by displacing polygon vertices (stretching the texture accordingly) [KDC*08] (image courtesy of Ladislav Kavan).

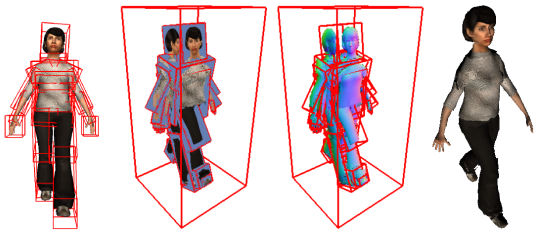


Figure 15: Relief impostors: During pre-process, color, normal and depth information are projected onto the 6 box faces. During real-time, each character is rendered through relief mapping [BSAP11].

through opacity masks (see Figure 16), which are computed by considering how the geometry of each bone is affected by the transformation of neighboring joints.

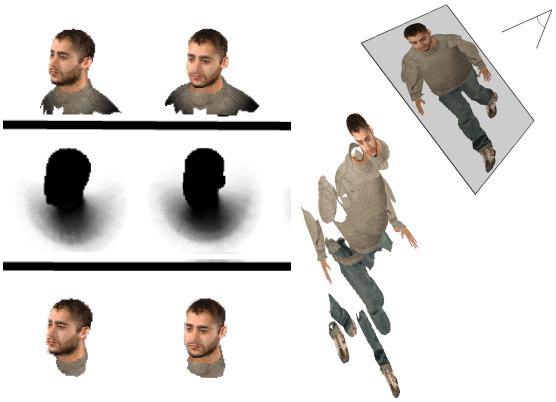


Figure 16: On the left, pre-process to obtain each per-joint and per-view impostor for the head of the character after applying the corresponding mask. On the right, real time rendering compositing the per-joint textures for a given view point [BAPS12].

Previously, in the same spirit, Aubel et al. [ABT00] divided each character into coherent parts by using the natural segmentation of joints. However, their subdivision was used exclusively for handling visibility issues rather than for animating each part separately as in [BAPS12]. Maim et al. [MYT09] sampled individual parts from multiple view directions too, but their animation-independent impostors are limited to rigid accessories such as hats, wigs or backpacks.

3.5. LoD selection

Assuming multiple LoDs are available, the application has to decide the most suitable LoD for each agent, when to switch from one LoD to another, and how to do this switch as seamlessly as possible to avoid popping artifacts. Early LoD selection techniques relied on object-space distance thresholds defining ranges for each LoD. A better approach is to select the LoD according to (a conservative estimation of) the area of the screen projection of the character. Hardware occlusion queries report pixel counts and thus can be used also to select a proper LoD considering not only screen-projected area, but also partial visibility (if an object is hardly visible, a lower resolution model can be used). However, naive occlusion queries often have a detrimental effect on performance unless more sophisticated techniques exploiting temporal and spatial coherence of visibility are used [MBW08].

Zach and Karner presented an automatic and dynamic selection of LoD by computing an estimation of the rendering cost and the perceptual benefits [ZMK02]. Hernández et al. implemented a dynamic LoD selection and view frustum culling on the GPU, by using the new transform feedback mechanism [HR11]. More recently Toledo et. al. implemented a tiling of the environment, tagging each agent with the code of the tile they are occupying, and making it easy and fast to dynamically update it [TDGR14]. The tagged code allows for a fast distance computation to the camera and LoD selection.

A related problem is how to switch from one LoD to another, as this has a significant impact on visual quality [SG03]. If LoDs are selected based on a range of distances, all agents crossing the boundary immediately suffer a LoD switch and thus pop-up effects will become more localized and noticeable. On the other hand, an agent walking over such boundary lines will be constantly switching between two LoDs and might easily catch the eye. A solution is to add a minimum validation time, biased with a random offset, that the agent needs to be in the new zone before further switches are allowed. The random offset allows more unpredictable and asynchronous changes of LoDs. Another possibility to reduce pop-up artifacts is to alpha-blend the renders of two LoDs in the vicinity of switch threshold values. This approach has the potential to mitigate popping artifacts, at the expense of rendering two representations instead of one (at least for a subset of the characters), and thus reducing performance.

4. Lossless acceleration techniques

In contrast to the LoD techniques reviewed above, the acceleration techniques described in this section introduce no further visual artifacts.

4.1. Culling Techniques

Culling techniques aim at discarding objects or parts of objects which are not visible [COCSD03]. In this section we show how these algorithms can be applied to crowd rendering and the problems that can be encountered.

Frustum culling is a very common technique for discarding all of the objects that are not inside the camera frustum,

thus avoiding sending them to the graphics card and speeding up the rendering [Cla76, BEW*98, AM00]. The test to determine if a point or a bounding volume (box, spheres and cylinders are typical shapes) is partially or totally inside the frustum can be performed before the render instruction. In the case of crowd rendering we can simply use the bounding sphere of the agents to approximate their shape. Although this test is very fast, it has to be performed for every agent in the crowd, which can be very large (tens or hundreds of thousands). In the case of having most of the agents within the frustum, then this test will consume a large amount of processing time that could have been used to render more agents.

With the goal of applying *frustum culling* to crowd rendering there are some strategies that can be applied. If the simulation and the rendering run in parallel, the frustum test can be performed in the simulation thread which sets a visibility flag. Another possibility consists of keeping some data structure that can speed up the process of determining which agents need to be rendered, such as spatial hashing [Rey06] or hierarchical representations [TDGR14].

Visibility or *occlusion culling* algorithms determine if an object is occluded by some other geometry (mainly static geometry) before rendering it [KS01]. The visibility test can be performed using preprocessed data (requiring an organization of the whole geometry), using structures such as *kd-trees* that encapsulate all the scene visibility information. In the case of crowds, since agents are moving around a virtual environment, the visibility test needs to be performed against the static geometry of the scene. However, due to the dynamic nature of the agents, it is not enough to use specific methods that often apply exclusively to static geometry. One efficient way to perform occlusion culling is to first render the scene and then render the agents discarding the occluded fragments with a simple depth test. Notice that this still requires sending them to the graphics card to be rendered since culling is performed in the GPU.

Another possibility to significantly increase performance is to use hardware-based occlusion queries [WB05] by sending simpler geometry to the GPU such as bounding volumes. This allows for conservative culling, since discarding objects whose bounding volume is completely occluded is safe, but we might fail to discard some invisible agents.

When having a very large crowd, and depending on the kind of camera motion we have (for example a first person camera), it is very common to have agents occluding other agents. The early z-culling [MS04] feature, implemented in most of the recent GPUs, allows for a fragment to be discarded before it is processed by the fragment shader (although some practices, like modifying the fragment's depth programmatically, disable this feature). One could think that depth test and early z-culling should be enough to handle these situations and avoid rendering thousands of agents, but depth algorithms depend highly on the order in which primitives are rendered. Clustering techniques or spatial data structures allowing an efficient front-to-back traversal of the crowd agents, such as KD-trees or BSP-trees, can benefit from early z-culling. We must remember though, that crowd agents will be moving around the environment, and therefore these structures should be dynamically updated every frame according to the new positions of the agents. Hernández et al. use the new transform feedback mechanism of modern GPUs to perform view frustum culling efficiently [HR11].

4.2. Instancing and pseudo instancing

Primitive instancing [Car05] optimizes rendering by drawing multiple copies of an object using a single call. Through instancing, the graphics processor deals with per-instance geometry transformations and appearance modifications, releasing the main processor from this task. A GPU acceleration crowd rendering is presented by Millan and Rudomin in [MR06b], alternating the use of impostors with polygonal meshes drawn through pseudo-instancing (see Figure 17).

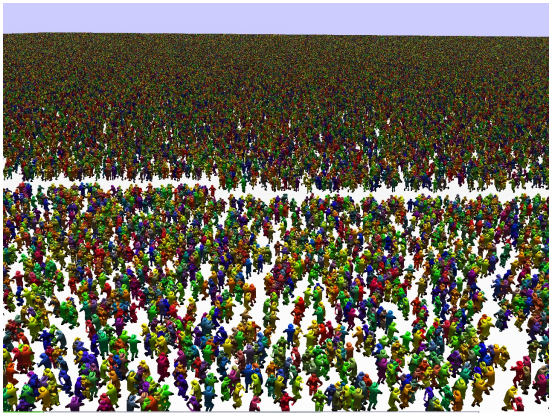


Figure 17: One million characters rendered with pseudo-instancing [MR06b] (image courtesy of Isaac Rudomin).

Even when instancing is originally designed for static objects, a similar technique may be used to render large crowds of animated characters. To achieve this goal, a pseudo-instancing technique is used, where geometry is updated on every animation frame and sent to the graphics memory to be used later for rendering nearby characters. Pseudo-instancing takes advantage of the efficiency of using persistent vertex attributes, such as color or transformations, to provide information for an entire instance.

However, this model update implies copying information into graphics memory. Therefore, to maximize the outcome of this technique, several copies of the same object must be rendered in every frame. This is a problem when using animated models, since every different animation pose needs to be sent to the graphics memory. As a workaround, a few poses can be selected, and nearby characters are rendered using the closest pose to the ones selected.

The main difference is that, in instancing, only one call is used to render all primitives, while pseudo-instancing requires one call to a display list to render each instance. However, these calls are very efficient in OpenGL, so similar performance levels are achieved by both techniques. When having crowds with hundreds or thousands of agents, it is often desired to have each agent to play different animations with different poses. Although animations have a small memory footprint, the computation of all the blending poses can become a major bottleneck and ruin performance. With the introduction of programmable pipelines, a number of costly CPU computations were moved to the GPUs. Beeson and Bjorke highly accelerated the skinning process by computing it directly in the GPU [BB04]. Skinning in the GPU requires transfer of both the original vertices of the avatar and the set of matrices of each animation frame. The final pose is a set of matrices that must be used to transform all the avatar vertices in the vertex shader, so the GPU bandwidth is critical.

Matrix palette skinning, introduced by Dudash [Dud07b], avoids sending to the GPU the transformation matrices for each bone and character instance. In matrix palette skinning, bone matrices for each frame and for each animation are stored in graphics memory. This allows each agent to have its own distinct pose and animation [Dud07a]. Note however that palette skinning only saves memory bandwidth; it does not affect the number of matrix operations in the vertex shader.

4.3. Dynamic caching

Recently, Lister et al. [LLD10] improved the efficiency of linear-blend skinning by using the temporal and intra-crowd coherencies that are inherent within populated scenes. They achieved it through the allocation of a small geometry cache within which transformed key-poses can be stored. These key-poses are then re-used by multi-pass rendering, between multiple agents and across multiple frames.

The cache of skinned key-poses is a maintained fixed-sized cache, from which crowd members can be reconstructed by interpolation. Generic poses may be shared amongst crowd members to significantly reduce the number that must be stored.

This cache size becomes also a trade off between the rendering performance and the memory usage, because it is the number of characters that have the key-poses stored in the cache that will have the greatest effect on the rendering performance. Clearly, the choice of which key-poses to store is critical to maximize the potential of the approach. Since this is a NP-hard problem, they present a greedy algorithm suitable for real-time applications.

5. Lighting, shading and shadowing

Lighting and shading improve the way we perceive the characters. Jarabo et al. recently made a perception study on how important lighting is for the overall perceived realism of dynamic scenes [JVES*12]. Self-shadowing and self-interreflection can help the human eye to interpret the animation and expression of the avatars. Two main techniques are well-known for casting shadows: shadow maps and shadow volumes. Williams introduced shadow maps [Wil78], using a depth map build from each light source to determine whether a fragment is illuminated or not. Two main problems arise from this. First, since textures are used, aliasing problems appear. Second, an additional render of the scene is required for each light source, since animated characters invalidate precomputed shadow maps. Shadow volumes, introduced by Crow [Cro77], are more expensive to calculate, but resolve the aliasing problems by using a semi-infinite frustum. This frustum is extended back from the silhouette of the object away from the light. As mentioned in 3.3.2, shadows for animated crowds can be rendered using impostors [TLC02] , but this technique does not support characters casting shadows onto each other. Self-shadowing for texture-based impostors can be computed using parallax occlusion mapping and its self-shadowing contribution [Tat06]. For more information about real-time shadow techniques see [EAS*13].

6. Clothing and Crowd Variability

Traditionally 3D characters are modeled clothed, with their clothes being part of the human mesh since having them modeled as separate elements and adding clothing simulation is very expensive. Detecting collisions with the human body [CFW13] is prohibitively expensive for real-time crowds. McDonnell et al. [MDCO06] perceptually evaluated different LoD representations of humans wearing physically simulated clothing. They show that impostors can depict the deformation properties of clothing. Some recent games in-

Figure 18: Dynamic caching accelerates the rendering of an animated crowd [LLD10] (image courtesy of Andy Day).

Figure 19: Crowd using two template models with color variation.

clude cloth simulation for the main character or a small amount of them, but real-time cloth simulation for crowds is beyond the state-of-the-art.

A related problem is to give each agent of the crowd an individual aspect. McDonnell et al. [MLD*08] performed perceptual studies to determine which aspects are more critical to identify clones. The ideal would be to have unique instances of each avatar, but due to obvious memory and modelling budgets, repetitions are inevitable. Some approaches attempt to add variability and create new differentiated instances of the same base character. Maim et al. [MYT09] proposed a method for attaching accessories to individual agents, and a generic technique for adding detailed color variety and patterns, by using segmentation maps over the human and accessory meshes. Their approach is scalable for all LoDs, including impostors. McDonnell et al. [MLH*09] use selective color variation to generate the illusion of variety as full color variation (see Figure 19). Lister et al. also [LLD10] add geometric diversity using tangent space morph targets.

Another important aspect to consider when increasing individualism, is the relevance of motion variety. In [PO11] perception studies were carried out to determine how people detect motion clones.

7. Comparison of crowd rendering techniques

7.1. General Comparison

Table 1 summarizes the crowd rendering approaches discussed in previous sections, considering the underlying representation and animation technique. We have also summarized the limitations of each approach, highlighting the parameter or element that contributes most to the performance-quality trade-off. In the latter columns we evaluate (as high, medium or low) the limitations of each method in terms of memory space, visual artifacts and time efficiency.

The explanation of each column follows:

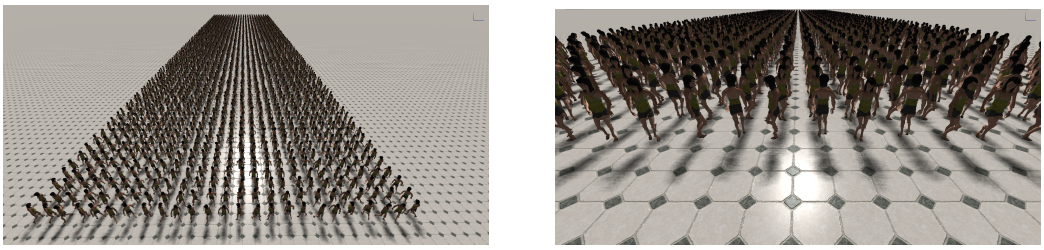
- **Type:** Polygon-based, Point-based, Image-based or Hybrid.
- **Representation:** The geometric representation(s) used to render each agent.
- **Animation:** The animation technique used to animate each agent.
- **Tradeoff:** The main parameter of the approach that implies a tradeoff between visual quality and performance.
- **Limitations:** The main limitations of the approach in terms of memory, visual quality and time efficiency, and other aspects with a high impact on the final results or the implementation.
  - **Memory:** We give the required memory space of each approach by pointing which parameters it depends upon:
    - Number of agent types (A)
    - Geometric complexity (G)
    - Number of frames (F)
    - Number of views, if it has a discrete number of views, like planar impostors (V)
    - Number of joints (J)
    - Texture resolution (R)
  - **Artifacts:** We classify visual and animation artifacts in three categories:
    - Image Quality: blocky aspect (Bl), pixelization (Pix), cracks or gaps (Crk), and animation artifacts due to inconsistent geometry deformation ($Anim_D$).
    - Temporal Discontinuity: popping when changing of LoD (Pop_L), when changing the view point (Pop_V), when changing the frame (Pop_F).
    - Spacial Consistency: visibility or occlusion problems with the scene ($Occl_S$), and with the agent itself ($Occl_A$).
  - **Cost:** The time efficiency is affected by some of the following elements:
    - Number of agent types (A)
    - Total number of vertex operations (V_x)
    - Total number of fragments ($Frag$)

7.2. Performance Comparison

We measured the performance of the most representative LoD techniques for crowd rendering in different scenarios. We used two test views, an aerial view with little overdraw and a street view with a significant amount of overdraw (Figure 20). In both views the crowd was placed at a fixed distance range from the 60-degree camera, with close-up characters at approximately 10 meters and the farthest ones at 100 meters. When testing different crowd sizes, we kept the area covered by the characters fixed, and only increased the density. The test hardware was a desktop PC equipped with an Intel Core i7-2600K @3.40GHz with 16 GB of RAM and one NVidia GeForce GTX 560 Ti. All renders were performed at a rather high-quality profile: 1920×1080 resolution, with a single static light casting soft shadows on the

Approach	Year	Reference	Type	Representation	Animation	Tradeoff	Limitations			
							Memory	Artifacts	Cost	Other
LoD	1997	[PPB*97]	Polygon	Mesh	Skeletal	Distance or pixel size	$A \times G$	Bl, $Anim_D$, Pop_L	V_x	Simplification problem
Dynamic Impostors	1998	[ABT98]	Image	Oriented billboard	Skeletal	Refresh criterium		Pix, $Occl_S$, Pop_V, Pop_F	A	Limited reusability
Pre-generated Impostors	2000	[TC00]	Image	Oriented billboard	Texture cyclical	# of views	$A \times F \times V \times R$	Pix, Pop_V, Pop_F, $Occl_S$	A	
Point-based Impostors	2002	[WS02]	Point	Triangle and Point Mesh	Mesh cyclical	Distance or pixel size	$A \times G$	Bl, Pix, Pop_L	V_x	Aliasing
Static Geometry	2004	[UCT04]	Polygon	Oriented billboard and mesh	Mesh cyclical	Distance or pixel size	$A \times G \times F$	Pop_F	V_x	
Geopostors	2005	[DHOO05]	Hybrid	Oriented billboard and mesh	Texture cyclical and skeletal	Distance or pixel size	$A \times F \times V \times R$	Pop_L, Pop_V, Pop_F, $Occl_S$	A	
Pseudo-Instancing	2006	[MR06b]	Polygon	Mesh	Skeletal	#Meshes in the GPU	$A \times F \times V \times R$	Pix, Pop_V, Pop_F	A	Too much data in the GPU
Volumetric Layered Impostors	2007	[CLM07]	Hybrid	Oriented layered billboards and Mesh	Texture cyclical and skeletal	Distance or pixel size	$A \times F \times V \times R$	Pop_V, Pop_F, $Occl_S$	A	
Polypostors	2008	[KDC*08]	Hybrid	1 Oriented billboard per body part, and mesh	Texture deformation and skeletal	Distance or pixel size	$A \times F \times V \times R$	Crk, Pop_V, $Occl_A$	A	
Dynamic Caching	2010	[LLD10]	Polygon	Mesh	Closest pose (skeletal)	Caché size	$A \times G$	Pop_L, Pop_F	A	
Relief Per-Joint Impostors	2011	[BSAP11]	Hybrid	6 relief impostors per body part, and mesh	Skeletal	Distance or number of fragments	$A \times J \times R$	Crk, Pop_L	$Frag$	High per fragment cost
Flat Per-Joint Impostors	2012	[BAPS12]	Hybrid	One oriented billboard per body part, and mesh	Skeletal	Distance and # of view angles	$A \times V \times J \times R$	Pop_L, Pop_V	A	
Hierarchical Point-Based	2014	[TDGR14]	Hybrid	Octree per limb, and mesh	Skeletal	Distance and hierarchy level	$A \times J$	Crk	A	

Table 1: Comparison of the limitations of the main approaches on crowd rendering in the literature. *Quick reference: MEMORY:* **A**: agent type; **G**: geometry complexity; **F**: number of frames; **V**: number of views; **J**: number of joints; **R**: texture resolution. *IMAGE QUALITY:* **Bl**: blocky aspect; **Pix**: pixelization; **Crk**: cracks; $\mathbf{Anim_D}$: animation deformation; $\mathbf{Pop_L}$: popping changing of LoD; $\mathbf{Pop_V}$: popping changing of view; $\mathbf{Pop_F}$: popping changing of frame; $\mathbf{Occl_S}$: occlusion problems with the scene; $\mathbf{Occl_A}$: occlusion problems with the agent itself. *COST:* **A**: agent type; $\mathbf{V_x}$: vertex operations; **Frag**: number of fragments



(a) (b)

Figure 20: Aerial (a) and street (b) view used in our performance tests. Further characters are at 100 meters from the camera.

	Number of Agents	2500		5000		10000	
	View	Aerial	Street	Aerial	Street	Aerial	Street
Approach	Original mesh (25 K triangles)	1.4	1.4	0.7	0.7	0.3	0.3
	Simplified mesh (5 K triangles)	2.9	2.9	1.5	1.5	0.8	0.7
	Simplified mesh (2.5 K triangles)	6.0	6.0	3.1	3.1	1.6	1.6
	Relief Per-Joint Impostors [BSAP11]	20	17	11	9	5	4
	Polypostors [KDC*08]	43	38	25	21	13	11
	Flat Per-Joint Impostors [BAPS12]	45	41	26	22	15	12
	Pre-generated Impostors [TC00]	90	81	51	48	26	25

Table 2: Performance comparison of different crowd rendering techniques in frames per second (fps).

ground, and each character being animated by blending two different animation clips. The pose of each character was updated at every application frame. Agents were animated in place to evaluate exclusively rendering and animation performance, without considering collision detection and simulation, which would affect all techniques. Several hardware optimizations were implemented for all techniques, including deferred rendering (soft shadows), instancing and palette skinning. Occlusion and frustum culling though were disabled since for the two test views they offered no performance gain.

For the sake of fairness, we used identical or similar parameters whenever possible across impostor representations. The original mesh had 25 K polygons, 2048×2048 textures, and a 67-bone skeleton. All impostor textures were 128×128 texels. Polypostors, Relief and Flat Per-Joint Impostors used a 21-bone skeleton. For view-dependent impostors 72 view directions were used for sampling. Relief mapping was configured to execute up to 8 linear search steps, and up to 2 bilinear search steps. Animation clips of pre-generated Impostors [TC00] were sampled at 30 fps.

Table 2 reports the results of our benchmark with different crowd sizes. As expected, image-based techniques clearly outperformed mesh-based approaches, even after significant simplification of the mesh. Pre-generated Impostors offered the highest performance in our setup, but due to their visual-quality and animation limitations, this representation should be reserved to very distant characters. From the point of view of performance, both Polypostors and Flat Per-Joint Impostors are good for not-so-distant characters.Relief per-joint impostors provided the highest-quality among image-based representations, but their output-sensitive nature and their high per-fragment cost makes them suitable for characters with small to medium-size screen projection.

8. Conclusions

We have reviewed and compared a large number of crowd rendering approaches. Overall, each technique falls somewhere within the triangle representing the quality/memory/performance trade-offs.Polygon-based approaches offer the best visual quality, but their performance depends strongly on the number of polygons per agent. Current GPU techniques such as instancing, palette skinning and dynamic caching can help to speed up the rendering, but as the size of the crowd increases, performance will be severely affected.

There is currently no standard solution for representing far away characters, though most crowd rendering systems use at least a different representation for these. LoD techniques are commonplace to improve performance as the number of

agents increases. Some issues specific to character rendering are how to generate simplified models of avatars that need to be deformed by animations, and how to switch between LoDs without popping artifacts.

Using only points as primitives reduces the rendering cost, but the animation can suffer from visual artifacts when drastically reducing the number of primitives. This problem can be alleviated by proper point sampling, at the expense of having a different sampling for each keyframe.

Image-based representations offer the highest performance. We can trade off memory for quality to provide a better sampling of view directions and animation frames to minimize popping artifacts, but in practice image-based representations are suitable only for distant characters.

Hybrid approaches combine mesh-based and image-based or point-based representations to display characters at different viewing distances. Special care must be taken to prevent visual artifacts when switching from one representation to another. Hybrid approaches have now reached a finer granularity, from using a single texture for the whole character, to using per-joint impostors at different skeleton levels, but it is unclear how much further these techniques can go in adding details while still being more efficient than the original geometry.

From an implementation point of view, all the new hardware improvements such as instancing need to take important considerations into account. Appropriate grouping data structures for avatars, textures and animations, and minimizing 3D API state changes are critical to get all the benefits of these improvements. The general trend is to move as many computations as possible to the GPU, achieving higher parallelization of per-agent computations and releasing the CPU for other tasks. Skinning and animation blending can be performed efficiently in the GPU. As we free the CPU from rendering and animation tasks, the CPU can spend more resources to the crowd simulation, although lately there has been a tendency towards moving also some simulation tasks to the GPU. We will probably reach a point where all crowd simulation, animation and rendering will be performed in the GPU at different shader stages. In this scenario, GPU memory and CPU-GPU bandwidth could become major limiting factors.

Adding shadows to the crowd scene increases realism, but at the high cost of having to render the scene for each light. Impostors can be used to do so and have approximate shadows, although they do not support self shadows. Clothing the characters also adds realism but there is still no physical clothing simulation fast enough for crowd rendering. Repeating instances of the same avatars is inevitable, specially to benefit from instancing and similar techniques, but some variety can be added to the crowd by attaching different accessories or by adding color variety and editable patterns to some base meshes.

There are still many open problems in this field, such as having individuality at all levels (appearance, animation and even behavior). For the individuality problem, tessellation shaders could become a powerful tool in order to add geometric variation to close-up characters, using stochastic techniques. Hardware tessellation might also accelerate the skinning process by rigging and animating a mid-resolution mesh that can be refined dynamically into a high-resolution mesh.

Despite hardware improvements, we believe that optimized crowd rendering would continue to be a challenging problem in the years to come. Even with hardware able to handle large groups of agents in real-time, optimized primitives will free CPU and GPU, leaving more resources for other tasks, minimizing energy consumption and increasing battery life.

Beyond some maximum viewing distance, a 3D character projects into a few pixels. At such distances, we may not be visualizing microscopic simulations anymore, but crowds may be moving at a macroscopic level. In such a case, crowd rendering should be approached in a completely different way, representing massive groups of people flowing, and not individual agents anymore. There are other questions about what humans are able to perceive within their field of view, such as what is the real maximum amount of agents that can be perceived simultaneously by a single viewer, what is the minimum visual quality they need to have, or what are the visual queues users are able to distinguish. As we have seen, some perceptual studies already attack the perception problem but focusing on agent variability. Additional perception studies and psychophysical experiments are definitely required to answer the questions above and to discover the boundaries of perception in the context of real-time crowd rendering.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness and FEDER under grant TIN2014-52211-C2-1-R. A. Beacco was also supported by the grant FPU AP2009-2195 (Spanish Ministry of Education).

References

[ABT98] AUBEL A., BOULIC R., THALMANN D.: Animated impostors for real-time display of numerous virtual humans. In *VW '98: Proc. of the First International Conference on Virtual Worlds* (London, UK, 1998), Springer-Verlag, pp. 14–28. 2, 7, 14

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

[AM00] ASSARSSON U., MÖLLER T.: Optimized view frustum culling algorithms for bounding boxes. *J. Graph. Tools 5*, 1 (Jan. 2000), 9–22. 2, 11

[ASDB08] AZAHAR M., SUNAR M., DAMAN D., BADE A.: Survey on real-time crowds simulation. In *Technologies for E-Learning and Digital Entertainment*, Pan Z., Zhang X., El Rhalibi A., Woo W., Li Y., (Eds.), vol. 5093 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 573–580. 2

[AT00] AUBEL A., THALMANN D.: Realistic deformation of human body shapes. In *Proc. Computer Animation and Simulation 2000* (2000), pp. 125–135. 3

[Bær05] BÆRENTZEN J.: Hardware-accelerated point generation and rendering of point-based impostors. *J. Graphics Tools 10*, 2 (2005), 1–12. 2, 6

[BAPS12] BEACCO A., ANDÚJAR C., PELECHANO N., SPANLANG B.: Efficient rendering of animated characters through optimized per-joint impostors. *Journal of Computer Animation and Virtual Worlds 23*, 2 (2012), 33–47. 2, 9, 10, 14, 15

[BB04] BEESON C., BJORKE K.: Skin in the "dawn" demo. In *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics* (2004), Pearson Higher Education, pp. 45–62. 12

[BEW*98] BISHOP L., EBERLY D., WHITTED T., FINCH M., SHANTZ M.: Designing a pc game engine. *IEEE Comput. Graph. Appl. 18*, 1 (Jan. 1998), 46–53. 2, 11

[BP07] BARAN I., POPOVIĆ J.: Automatic rigging and animation of 3d characters. *ACM Trans. Graph. 26*, 3 (2007), 72. 3

[BSAP11] BEACCO A., SPANLANG B., ANDUJAR C., PELECHANO N.: A flexible approach for output-sensitive rendering of animated characters. *Computer Graphics Forum 30* (2011). 9, 10, 14, 15

[BTST12] BHARAJ G., THORMÄHLEN T., SEIDEL H.-P., THEOBALT C.: Automatically rigging multi-component characters. *Computer Graphics Forum 31*, 2pt4 (2012), 755–764. 3

[Car05] CARUCCI F.: Inside geometry instancing. In *GPU Gems 2* (2005), Pharr M., (Ed.), Addison-Wesley, pp. 47–67. 11

[CFW13] CHEN Z., FENG R., WANG H.: Modeling friction and air effects between cloth and deformable bodies. *ACM Trans. Graph. 32*, 4 (July 2013), 88:1–88:8. 12

[Cla76] CLARK J. H.: Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10 (Oct. 1976), 547–554. 2, 5, 11

[CLM07] COIC J., LOSCOS C., MEYER A.: *Three LOD for the Realistic and Real-Time Rendering of Crowds with Dynamic Lighting*. Research Report RN/06/20, Université Claude Bernard, LIRIS, France, April 2007. 8, 9, 14

[CLTL11] CHEN C., LIN I., TSAI M., LU P.: Lattice-based skinning and deformation for real-time skeleton-driven animation. In *Proc. of the 2011 12th International Conference on Computer-Aided Design and Computer Graphics* (Washington, DC, USA, 2011), CADGRAPHICS '11, IEEE Computer Society, pp. 306–312. 4

[CMT05] CORDIER F., MAGNENAT-THALMANN N.: A data-driven approach for real-time clothes simulation. *Computer Graphics Forum 24* (2005), 173–183. 3

[COCSD03] COHEN-OR D., CHRYSANTHOU Y. L., SILVA C. T., DURAND F.: A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics 9*, 3 (July 2003), 412–431. 2, 10

[Cro77] CROW F.: Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph. 11*, 2 (July 1977), 242–248. 12

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: a real-time geometry / impostor crowd rendering system. In *I3D '05: Proc. of the 2005 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2005), ACM, pp. 95–102. 2, 8, 14

[DR05] DECORO C., RUSINKIEWICZ S.: Pose-independent simplification of articulated meshes. In *Proc. of the 2005 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2005), I3D '05, ACM, pp. 17–24. 5

[DT07] DECORO C., TATARCHUK N.: Real-time mesh simplification using the gpu. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 161–166. 5

[Dud07a] DUDASH B.: Animated crowd rendering. In *GPU Gems 3* (2007), pp. 39–52. 12

[Dud07b] DUDASH B.: Skinned instancing. In *NVIDIA SDK 10* (2007). 3, 12

[EAS*13] EISEMANN E., ASSARSSON U., SCHWARZ M., VALIENT M., WIMMER M.: Efficient real-time shadows. In *ACM SIGGRAPH 2013 Courses* (New York, NY, USA, 2013), SIGGRAPH '13, ACM, pp. 18:1–18:54. 12

[GGPCP13] GONZÁLEZ GARCÍA F., PARADINAS T., COLL N., PATOW G.: *cages:: A multilevel, multi-cage-based system for mesh deformation. *ACM Trans. Graph. 32*, 3 (July 2013), 24:1–24:13. 4

[HCC06] HUANG F.-C., CHEN B.-Y., CHUANG Y.-Y.: Progressive deforming meshes based on deformation oriented decimation and dynamic connectivity updating. In *Proc. of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), SCA '06, pp. 53–62. 5

[Hej04] HEJL J.: Hardware skinning with quaternions. In *Game Programming Gems 4*, Kirmse A., (Ed.). Charles River Media, 2004, pp. 487–495. 4

[HMDO05] HAMILL J., MCDONNELL R., DOBBYN S., O'SULLIVAN C.: Perceptual evaluation of impostor representations for virtual humans and buildings. *Computer Graphics Forum 24*, 3 (2005), 623–633. 5

[Hop96] HOPPE H.: Progressive meshes. In *Proc. of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 99–108. 5

[HR11] HERNÁNDEZ B., RUDOMIN I.: A rendering pipeline for real-time crowds. In *GPU Pro 2*, Engel W., (Ed.). A K Peters, 2011, pp. 369–383. 10, 11

[JEOG11] JIMENEZ J., ECHEVARRIA J. I., OAT C., GUTIERREZ D.: *GPU Pro 2*. AK Peters Ltd., 2011, ch. Practical and Realistic Facial Wrinkles Animation, pp. 15–27. 3

[JVES*12] JARABO A., VAN EYCK T., SUNDSTEDT V., BALA K., GUTIERREZ D., O'SULLIVAN C.: Crowd light: Evaluating the perceived fidelity of illuminated dynamic scenes. *Computer Graphics Forum (Proc. EUROGRAPHICS 2012) 31*, 2 (2012). 3, 12

[KB04] KOBBELT L., BOTSCH M.: A survey of point-based techniques in computer graphics. *Comput. Graph. 28*, 6 (Dec. 2004), 801–814. 6

[KCvO07] KAVAN L., COLLINS S., ŽÁRA J., O'SULLIVAN C.: Skinning with dual quaternions. In *Proc. of the 2007 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 39–46. 4

[KCvO08] KAVAN L., COLLINS S., ŽÁRA J., O'SULLIVAN C.: Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph. 27*, 4 (Nov. 2008), 105:1–105:23. 4

[KDC*08] KAVAN L., DOBBYN S., COLLINS S., ŽÁRA J., O'SULLIVAN C.: Polypostors: 2d polygonal impostors for 3d crowds. In *I3D '08: Proc. of the 2008 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2008), ACM, pp. 149–155. 2, 9, 14, 15

[KG05] KIRCHER S., GARLAND M.: Progressive multiresolution meshes for deforming surfaces. In *Proc. of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2005), SCA '05, ACM, pp. 191–200. 5

[KS01] KLOSOWSKI J., SILVA C.: Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Trans. Vis. Comput. Graph. 7*, 4 (2001), 365–379. 2, 11

[KS12] KAVAN L., SORKINE O.: Elasticity-inspired deformers for character articulation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA) 31*, 6 (2012), 196:1–196:8. 4

[Kv05] KAVAN L., ŽÁRA J.: Spherical blend skinning: A real-time deformation of articulated models. In *Proc. of the 2005 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2005), I3D '05, ACM, pp. 9–16. 4

[LJBA13] LIMPER M., JUNG Y., BEHR J., ALEXA M.: The pop buffer: Rapid progressive clustering by geometry quantization. *Comput. Graph. Forum 32*, 7 (2013), 197–206. 5

[LLD10] LISTER W., LAYCOCK R., DAY A.: A key-pose caching system for rendering an animated crowd in real-time. *Computer Graphics Forum 29*, 8 (2010), 2304–2312. 3, 12, 13, 14

[LO11] LARKIN M., O'SULLIVAN C.: Perception of simplification artifacts for animated characters. In *Proc. of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization* (New York, NY, USA, 2011), APGV '11, ACM, pp. 93–100. 5, 6

[Lor07] LORACH T.: Gpu blend shapes. *NVidia Whitepaper* (2007). 4

[LS09] LANDRENEAU E., SCHAEFER S.: Simplification of articulated meshes. *Computer Graphics Forum 28*, 2 (2009), 347–353. 5

[LW85] LEVOY M., WHITTED T.: *The Use of Points as a Display Primitive*. UNC report. University of North Carolina, Department of Computer Science, 1985. 2, 6

[LWC*02] LUEBKE D., WATSON B., COHEN J., REDDY M., VARSHNEY A.: *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002. 2, 5

[MBW08] MATTAUSCH O., BITTNER J., WIMMER M.: Chc++: Coherent hierarchical culling revisited. *Comput. Graph. Forum 27*, 2 (2008), 221–230. 10

[McC90] MCCARTHY J. M.: *Introduction to Theoretical Kinematics*. MIT Press, Cambridge, MA, USA, 1990. 4

[MCC11] MCLAUGHLIN T., CUTLER L., COLEMAN D.: Character rigging, deformations, and simulations in film and game production. In *ACM SIGGRAPH 2011 Courses* (New York, NY, USA, 2011), SIGGRAPH '11, ACM, pp. 5:1–5:18. 3

[MDCO06] MCDONNELL R., DOBBYN S., COLLINS S., O'SULLIVAN C.: Perceptual evaluation of lod clothing for virtual humans. In *Proc. of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), SCA '06, Eurographics Association, pp. 117–126. 3, 12

[MG03] MOHR A., GLEICHER M.: *Deformation Sensitive Decimation*. Tech. rep., University of Wisconsin Graphics Group, 2003. 5

[Mix14] MIXAMO: Mixamo auto-rigger. https://www.mixamo.com/auto-rigger, 2014. 3

[MLD*08] MCDONNELL R., LARKIN M., DOBBYN S., COLLINS S., O'SULLIVAN C.: Clone attack! perception of crowd variety. *ACM Trans. Graph. 27*, 3 (2008), 1–8. 13

[MLH*09] MCDONNELL R., LARKIN M., HERNÁNDEZ B., RUDOMIN I., O'SULLIVAN C.: Eye-catching crowds: saliency based selective variation. *ACM Trans. Graph. 28*, 3 (July 2009), 55:1–55:10. 13

[MNO07] MCDONNELL R., NEWELL F. N., O'SULLIVAN C.: Smooth movers: perceptually guided human motion simulation. In *Symposium on Computer Animation* (2007), Gleicher M., Thalmann D., (Eds.), Eurographics Association, pp. 259–269. 5

[MR06a] MILLAN E., RUDOMIN I.: A comparison between impostors and point-based models for interactive rendering of animated models. In *Proc. of the International Conference on Computer Animation and Social Agents (CASA)* (2006), University Press. 7

[MR06b] MILLAN E., RUDOMIN I.: Impostors and pseudo-instancing for gpu crowd rendering. In *GRAPHITE '06: Proc. of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (New York, NY, USA, 2006), ACM, pp. 49–55. 3, 11, 14

[MS04] MITCHELL J., SANDER P.: Siggraph 2004 - real-time shading course applications of explicit early-z culling, 2004. 11

[MTLT88] MAGNENAT-THALMANN N., LAPERRIRE R., THALMANN D.: Joint-dependent local deformations for hand animation and object grasping. In *In Proc. on Graphics interface'88* (1988), pp. 26–33. 3

[MYT09] MAIM J., YERSIN B., THALMANN D.: Unique character instances for crowds. *Computer Graphics and Applications, IEEE 29*, 6 (nov.-dec. 2009), 82 –90. 3, 10, 13

[OBM00] OLIVEIRA M., BISHOP G., MCALLISTER D.: Relief texture mapping. In *SIGGRAPH '00: Proc. of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 359–368. 9

[PAB08] PELECHANO N., ALLBECK J., BADLER N.: *Virtual Crowds: Methods, Simulation, and Control*. Morgan & Claypool, 2008. 1, 2

[PDHCM*06] PETTRÉ J., DE HERAS CIECHOMSKI P., MAÏM J., YERSIN B., LAUMOND J., THALMANN D.: Real-time navigating crowds: scalable simulation and rendering: Research articles. *Comput. Animat. Virtual Worlds 17*, 3-4 (2006), 445–455. 5, 8

[PHB07] PAYAN F., HAHMANN S., BONNEAU G.-P.: Deforming surface simplification based on dynamic geometry sampling. In *Shape Modeling and Applications, 2007. SMI '07. IEEE International Conference on* (June 2007), pp. 71–80. 5

[PO11] PRAŽÁK M., O'SULLIVAN C.: Perceiving human motion variety. In *Proceedings of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization* (New York, NY, USA, 2011), APGV '11, ACM, pp. 87–92. 5, 13

[PPB*97] PRATT D., PRATT S., BARHAM P., BARKER R., WALDROP M., EHLERT J., CHRISLIP C.: Humans in large-scale, networked virtual environments. *Presence 6*, 5 (1997), 547–564. 5, 14

[PS12] PANTUWONG N., SUGIMOTO M.: A novel template-based automatic rigging algorithm for articulated-character animation. *Computer Animation and Virtual Worlds 23*, 2 (2012), 125–141. 3

[RB93] ROSSIGNAC J., BORREL P.: Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in Computer Graphics* (1993), Falcidieno B., Kunii T. L., (Eds.), IFIP Series on Computer Graphics, Springer, pp. 455–465. 5

[RD05] RYDER G., DAY A. M.: Survey of real-time rendering techniques for crowds. *Computer Graphics Forum 24*, 2 (2005), 203–215. 2

[Rey06] REYNOLDS C.: Big fast crowds on ps3. In *Proc. of the 2006 ACM SIGGRAPH Symposium on Videogames* (New York, NY, USA, 2006), Sandbox '06, ACM, pp. 113–121. 11

[RLS08] RAMIREZ J., LLIGADAS X., SUSIN A.: Automatic adjustment of rigs to extracted skeletons. In *Articulated Motion and Deformable Objects*, Perales F., Fisher R., (Eds.), vol. 5098 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 409–418. 3

[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 *ACM SIGGRAPH 2008 Games* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 52–101. 5

[SF99] SCHMALSTIEG D., FUHRMANN A.: *Coarse View-Dependent Levels of Detail for Hierarchical and Deformable Models*. Tech. rep., Vienna University of Technology, 1999. 5

[SG03] SOUTHERN R., GAIN J.: Creation and control of real-time continuous level of detail on programmable graphics hardware. *Computer Graphics Forum 22* (2003), 35–48. 10

[SKP08] SUEDA S., KAUFMAN A., PAI D. K.: Musculotendon simulation for hand animation. *ACM Trans. Graph. (Proc. SIGGRAPH) 27*, 3 (2008). 3

[Tat06] TATARCHUK N.: Dynamic parallax occlusion mapping with approximate soft shadows. In *I3D '06: Proc. of the 2006 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2006), ACM, pp. 63–69. 12

[TBP08] TATARCHUK N., BARCZAK J., PURNOMO B.: Gpu tessellation for detailed, animated crowds. SIGGRAPH Asia 2008 Sketch. 6

[TC00] TECCHIA F., CHRYSANTHOU Y.: Real-time rendering of densely populated urban environments. In *Proc. of the Eurographics Workshop on Rendering Techniques 2000* (London, UK, 2000), Springer-Verlag, pp. 83–88. 2, 7, 14, 15

[TDGR14] TOLEDO L., DE GYVES O., RUDOMÍN I.: Hierarchical level of detail for varied animated crowds. *The Visual Computer 30*, 6-8 (2014), 949–961. 6, 10, 11, 14

[TLC02] TECCHIA F., LOSCOS C., CHRYSANTHOU Y.: Image-based crowd rendering. *IEEE Comput. Graph. Appl. 22*, 2 (2002), 36–43. 3, 7, 8, 12

[TM13] THALMANN D., MUSSE S.: *Crowd Simulation, Second Edition*. Springer, 2013. 1, 2

[UCT04] ULICNY B., CIECHOMSKI P. D. H., THALMANN D.: Crowdbrush: interactive authoring of real-time crowd scenes. In *Proc. of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), SCA '04, Eurographics Association, pp. 243–252. 4, 5, 14

[VBG*13] VAILLANT R., BARTHE L., GUENNEBAUD G., CANI M., ROHMER D., WYVILL B., GOURMEL O., PAULIN M.: Implicit skinning: real-time skin deformation with contact modeling. *ACM Trans. Graph. 32*, 4 (July 2013), 125:1–125:12. 4

[WB05] WIMMER M., BITTNER J.: Hardware occlusion queries made useful. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Pharr M., Fernando R., (Eds.). Addison-Wesley, Mar. 2005. 2, 11

[WDAH10] WINKLER T., DRIESEBERG J., ALEXA M., HORMANN K.: Multi-scale geometry interpolation. *Computer Graphics Forum 29*, 2 (May 2010), 309–318. Proc. of Eurographics. 4

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph. 12*, 3 (Aug. 1978), 270–274. 12

[Wil11] WILLMOTT A.: Rapid simplification of multi-attribute meshes. In *Proc. of the ACM SIGGRAPH Symposium on High Performance Graphics* (New York, NY, USA, 2011), HPG '11, ACM, pp. 151–158. 5

[WS02] WAND M., STRASSER W.: Multi-resolution rendering of complex animated scenes. *Computer Graphics Forum 21*, 3 (2002), 483–491. 6, 14

[YYBE13] YUKSEL K., YUCEBILGIN A., BALCISOY S., ERCIL A.: Real-time feature-based image morphing for memory-efficient impostor rendering and animation on gpu. *The Visual Computer 29*, 2 (2013), 131–140. 8

[ZMK02] ZACH C., MANTLER S., KARNER K.: Time-critical rendering of discrete and continuous levels of detail. In *The ACM symposium on Virtual reality software and technology (VRST)* (2002), Shi J., Hodges L. F., Sun H., Peng Q., (Eds.), ACM, pp. 1–8. 10

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proc. of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 371–378. 2, 6

[ZW07] ZHANG S., WU E.: Deforming surface simplification based on feature preservation. In *Entertainment Computing - ICEC 2007*, Ma L., Rauterberg M., Nakatsu R., (Eds.), vol. 4740 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, pp. 139–149. 5