VDR-AM: View-Dependent Representation of Articulated Models

Pio Claudio
Dept. of CS, KAIST
pioclaudio@gmail.com

Dohyeong Kim
Purdue University
karkayan@gmail.com

Tae-Joon Kim
Dept. of CS, KAIST
tjkim.kaist@gmail.com

Sung-eui Yoon
Dept. of CS, KAIST
sungeui@gmail.com

ABSTRACT

We present a novel, View-Dependent Representation of Articulated Models (VDR-AM), and show its main benefits in the context of view-dependent rendering integrated with occlusion culling for large-scale crowd scenes. In order to provide varying resolutions on each animated, articulated model, we propose to use a cluster hierarchy in the VDR-AM for an articulated model. The cluster hierarchy serves as a dual representation for both view-dependent rendering and occlusion culling. For a high-performance view-dependent rendering and occlusion culling, we construct each cluster of the cluster hierarchy to contain a spatially coherent portion of the mesh that also has similar simplification errors. To achieve our goal, we present an error-aware clustering method for articulated models. We also identify a subset of animation poses that well represents the original pose data and perform the well-known quadrics-based simplification to efficiently compute our representation, while achieving a high quality simplification. At runtime, we choose a LOD cut from the cluster hierarchy given a user specified error bound in the screen space and render all the visible clusters in the LOD cut. We implement our method in GPU and achieve interactive performance (e.g., 40 frames per second) for large-scale crowd scenes that consist up to thousands of articulated models and 242 M triangles, without noticeable visual artifacts.

Keywords
View Dependent Rendering, Character Animation, Level of Detail

1 INTRODUCTION

Owing to advances of data capture and modeling technologies, detailed articulated models are easily generated and widely used in many different applications. Moreover, various crowd simulation techniques have been designed and a high number of articulated models are frequently used in large-scale crowd scenes [TOY$^+$07, NGCL09]. In typical crowd scenes with lots of articulated characters, it can require high computation costs of rendering and performing other operations (e.g., collision detection) for handling those articulated characters.

A significant amount of research has been put in order to improve the performance of rendering and conducting various operations for polygonal models. Some of them include designing multi-resolution representations of polygonal models [LRC$^+$02, YGKM08], performing visibility culling [COCSD03], collision detection [TCYM09], etc. Unfortunately, most of these prior techniques assume polygonal models and do not easily apply to articulated models. This is mainly because

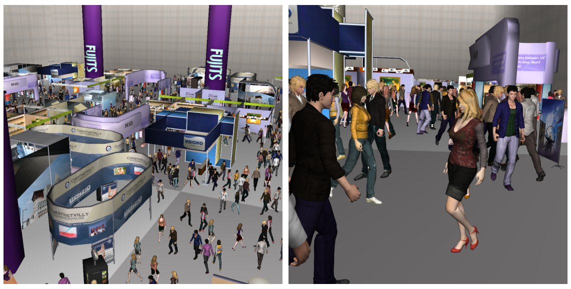
Figure 1: This figure shows two images of an exhibition crowd scene that has 1 K articulated characters. All the characters and the scene have 83 M triangles. Our method achieves 46 frames per second (fps) on average for this model with 0.5 pixel-of-error (PoE) in a 1280 by 720 HD screen resolution.

articulated characters are dynamically animated with an underlying deformation model (e.g., skeleton) that changes the shapes of characters.

In terms of rendering articulated models, many techniques have been proposed to improve the rendering performance. At a high level, these techniques can be classified as image-based [DHOO05, KDC$^+$08] and mesh simplification approaches [MG03, DR05, LS09] for articulated models. Image-based approaches have been reported to achieve a high rendering performance by rendering textures instead of triangles. However, these techniques may provide low quality rendering results for certain views (e.g., overhead or near views [DHOO05, KDC$^+$08]) or may not be used for other geometric applications such as collision detection.

Geometric simplification methods for articulated models, on the other hand, can provide high-quality polygonal meshes that can be used for various views. However, these simplification methods have focused on computing different versions of level-of-detail (LOD) meshes from an input articulated mesh, and have not been applied widely to view-dependent rendering that uses different LODs for portions of the mesh depending on viewing information. To the best of our knowledge, there have been no prior methods that adopt view-dependent rendering integrated with culling for large-scale scenes consisting of hundreds or thousands of articulated models [RD05].

Main contributions. In this paper we propose a novel, View-Dependent Representation of Articulated Models, VDR-AM. We show its benefits in the context of rendering, especially view-dependent rendering integrated with occlusion culling. VDR-AM consists of a cluster hierarchy that serves as both a multi-resolution representation and a bounding volume hierarchy. We use its multi-resolution representation for view-dependent rendering, and its bounding volume hierarchy for occlusion culling. We construct each cluster of the cluster hierarchy to contain a spatially-coherent small portion of the mesh that also has similar simplification errors. To construct such clusters, we propose an error-aware clustering method. Given the cluster hierarchy of VDR-AM, we can compute a LOD cut that satisfies a user-specified screen space error in terms of pixels-of-errors (PoE) at runtime. We also perform occlusion culling for clusters in the LOD cut in an efficient manner that utilizes the temporal coherence between consecutive frames.

We have implemented our method and applied it to three large-scale crowd scenes that consist of up to five thousand articulated models that have 242 M triangles in total. Even though our VDR-AM representation is not mainly designed for static polygonal models, it can be naturally applied to handling those models without major modifications. Therefore, we have also applied our representation even for static models that are parts of tested crowed scenes. Our method shows 16 to 50 frames per second (fps) without visible artifacts in a 1280 by 720 HD screen resolution. Compared with a base rendering method that uses the original articulated models combined with view-frustum culling, our method achieves four to eight times improvement.

2 RELATED WORK

In this section we give a background on animating articulated models based on skinning, and briefly review previous work related to our method. For detailed information about crowd rendering in general, refer to an excellent survey by Ryder et al. [RD05].

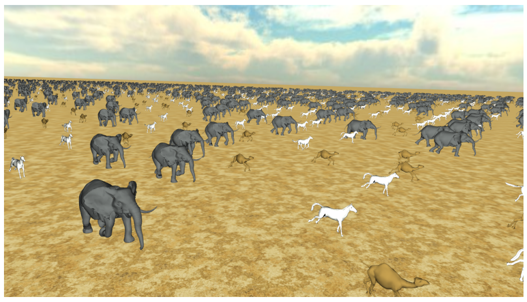

Figure 2: This figure shows a stampede crowd scene that has 5 K articulated characters and 242 M triangles. Our method can achieve 16 frames per second (fps) on average for this large-scale crowd scene.

2.1 Background of Articulated Models

Articulated models are typically designed with skinning and skeleton animations. In this case an articulated model consists of a base mesh, a skeleton, and vertex weights. The base mesh, also known as skin, is a 3D polygonal mesh, and the skeleton is a hierarchical representation of bones. The vertex weights associated with vertices of the base mesh represent the skin-to-skeleton binding. One of the most popular techniques used to achieve interactive animation is linear blend skinning.

The animation of the base mesh of an articulated model is defined by a series of poses. Each pose is defined by a 4 by 4 transformation matrix representing positions and orientations of bones of the skeleton. A vertex position, v, in the base mesh is then moved to a new position, $\hat{v}$, by linear blend skinning with a pose, based on the following equation:

$$\hat{v} = \sum_i w_i M_i v, \tag{1}$$

where M_i is the transformation of the ith bone associated with the vertex with the weight w_i given the input pose. There are more advanced skin blending methods such as skinning using the dual quaternions, and they can be also used with our method.

2.2 Mesh Simplification

Mesh simplification has been extensively studied for polygonal meshes, and numerous techniques have been presented. Among them the quadric error metric (QEM) and edge collapse operations [GH97] have been most widely used for high-quality mesh simplifications.

Many simplification techniques have been also proposed for dynamic models, but some of them [KG05, HCC06] are not directly applicable to articulated models that are animated with an underlying deformation model. In this section we focus on simplification methods that can handle articulated models with skinning.

Simplification for articulated models. Mohr and Gleicher [MG03] applied the QEM method for simplifying articulated meshes. Their method takes a skinned mesh and a series of example poses. It computes the quadric error for each vertex by considering all the example poses. DeCoro and Rusinkiewicz [DR05] improved this method by considering the bone transformations and computing the quadric error in a base reference pose. Recently, Landreneau and Schaefer [LS09] perform the simplification by considering weights associated with vertices and thus achieve higher simplification quality over other techniques. Our simplification method is based on the work of DeCoro and Rusinkiewicz [DR05], but can be improved by adopting techniques proposed by Landreneau and Schaefer [LS09]. In addition, Pilgrim et al. [PSA07] proposed a progressive skinning technique that progressively uses a subset of original bones of articulated models. This technique can be combined with our method for higher rendering performance.

Image-based simplification methods. As an alternative representation for polygonal representations, image-based representations [DHOO05, KDC+08] like impostors have been proposed for articulated models and reported to achieve a high rendering performance, while maintaining a reasonable memory requirement [YYBE13]. As downsides, these techniques may provide low quality rendering results for certain views (e.g., overhead views) or require a high storage requirement. In addition, image-based representations may provide low-quality results for various geometric operations such as collision detection. Nonetheless, these image-based representations can be used together with polygonal representations including ours, to achieve a higher rendering performance and quality. For example, one can use polygonal representations in a near field and use imposters in a far field, as suggested by Kavan et al. [KDC+08]. As a result, our method is orthogonal to image-based techniques in the context of rendering.

2.3 View-Dependent Rendering and Culling

View-dependent rendering (VDR) uses lower resolutions for portions of the mesh that are located farther away from the viewer. This technique aims to reduce the number of rendered triangles of complex models depending on viewing configurations and has been extensively studied for polygonal meshes [YGKM08]. VDR originated as an extension of the progressive mesh (PM) that is a linear sequence of encoding finer meshes [Hop97]. Hoppe [Hop97] improved this method by organizing the PM as a vertex hierarchy instead of a linear sequence. The vertex hierarchy is known to provide very smooth LOD transitions, while requiring high runtime computations. This technique has been extended to large-scale polygonal models that consist of hundreds of millions of triangles [YSGM04]. However, VDR has not been applied widely to articulated models for large-scale crowd rendering.

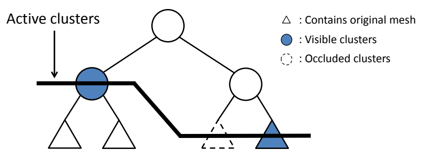

Figure 3: This figure shows our cluster hierarchy of our VDR-AM representation with active clusters (i.e. LOD cut) including visible and occluded clusters. Note that leaf clusters contain the original mesh.

Visibility culling also has been well studied to reduce the number of rendered triangles in scenes that have a high depth complexity [COCSD03]. For general environments, image-based occlusion representations are widely used, and high-performance culling algorithms use GPUs to perform occlusion culling [YSM03].

Hybrid algorithms for rendering acceleration. Many hybrid algorithms [RL00, YSGM04, CBWR07] that combine simplification with visibility culling for static polygonal models have been proposed. These techniques have integrated various visibility techniques into VDR frameworks of static polygonal meshes. Our VDR-AM representation can enable hybrid rendering methods for articulated models.

3 OVERVIEW OF OUR METHOD

In this section we explain our representation, followed by an overview of its construction and our runtime rendering algorithm.

3.1 Dual Representation

We use a *cluster hierarchy* (Fig. 3) for an articulated model, as a dual representation. The cluster hierarchy serves both as a multi-resolution hierarchy for View-Dependent Rendering (VDR), and as a bounding volume hierarchy for occlusion culling, view-frustum culling, and other geometric operations.

We call each node of the hierarchy a *cluster*. Each cluster serves as a main processing unit for VDR and occlusion culling. Each leaf cluster of the hierarchy contains a portion of the base mesh of the articulated model. For an intermediate cluster of the hierarchy, we merge two sub-meshes contained in its two child clusters, simplify them, and store them in the intermediate cluster. Therefore, each leaf cluster can provide the original resolution of a portion of the mesh, while an intermediate cluster can provide a low resolution for a portion of the mesh. Also, each cluster is associated with a bounding volume that contains all the geometry throughout the animation.

Each cluster records its maximum geometric simplification error that is caused by simplifying the sub-mesh of the cluster, while considering poses of an animation for the articulated model. In order to allow drastic simplifications on the articulated model given an error bound, it is critical to cluster vertices that have similar simplification errors, thus leading to smaller geometric simplification errors for each cluster. In order to provide a high culling ratio, each cluster should be constructed such that it contains a spatially coherent portion of the base mesh of the articulated model.

3.2 Construction

In order to construct the cluster hierarchy of an articulated model, we first decompose the base mesh of the model into clusters, which become the leaf clusters of the hierarchy. We perform our error-aware clustering method to construct each cluster to have a spatially-coherent portion of the mesh whose vertices have similar simplification errors (Sec. 4.2). For the simplification of sub-meshes contained in clusters, we use the well-known edge-collapse and quadric-based simplification methods, which also consider poses of the animation of the model. Since the simplification process can take a large amount of time for the animation that consists of many poses, we propose a pose selection method that chooses representative poses for the animation and simplify the mesh by considering only those representative poses (Sec. 4.1).

3.3 Rendering Algorithm

To show benefits of our VDR-AM representation, we apply it to view-dependent rendering integrated with occlusion culling for articulated models. At runtime, we compute a new position and orientation (e.g., animation pose) of each articulated model in the scene. To perform VDR, we maintain *active clusters* (i.e. a LOD cut) that represent the articulated model with the lowest number of triangles, given a user-specified error bound (Sec. 5.1). To determine active clusters, we compute a screen-space projected simplification error for each cluster and compare it with the user-specified error bound. To perform occlusion culling we compute a conservative set of visible clusters based on the bounding volume information encoded in the cluster hierarchy and render them for the final images (Sec. 5.2).

4 CLUSTER HIERARCHY CONSTRUCTION

In this section we explain our cluster hierarchy construction method. We also compute clusters from the static polygonal models of scenes, as we compute clusters from the animated, articulated models.

Pose space reduction. Since an articulated model can have many bones (e.g., 30 to 60 bones for human-like characters), each pose can be considered as a point in a high dimensional pose space. Unfortunately, any operations on these high dimensional points can be very difficult and expensive because of the well-known curse of dimensionality. To ameliorate this issue, we reduce the dimensionality of the pose space. In particular, we use the multi-dimensional scaling, a statistical tool for reducing dimensions of data [McG68]. This method maps data into a reduced dimensional space, while preserving the distance between data in the original dimensional space. For all the tested models in the paper, we reduce the pose space of such models to 9 dimensional space, as suggested by Assa et al. [ACCO05]. Though Assa et al. suggested mainly for human-like models, we found that 9 dimensional space works well to other tested animal models.

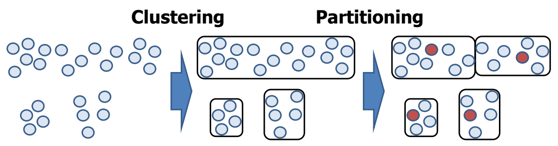

Figure 4: This figure shows the process of our pose selection method that performs the clustering and partitioning. The red points are the selected final poses that represent the distribution of the original poses.

4.1 Representative Pose Selection

In order to reduce the time taken on simplifying models, we propose to consider only *representative poses* instead of considering all the original poses during the simplification process. As the number of representation poses decreases, we can improve the performance of the simplification, but can underestimate the simplification error more, leading to a low-quality simplification.

To minimize this negative effect, we define *pose groups*, each of which contains a coherent set of poses. We then choose a representative pose from each pose group. After reducing the dimension of the pose space, we partition poses into pose groups based on our *clustering and partitioning* framework (Fig. 4).

We start with creating a pose group for each pose point. Our pose selection method consists of two stages: clustering and partitioning stages. In the clustering stage, we recursively merge two groups into a new pose group, if any two points chosen from those two pose groups are within the Euclidean distance of d. In order to efficiently perform this operation, we can construct a kd-tree from poses in the reduced pose space and find nearest neighboring pose points [Ben75].

The pose groups computed from the clustering stage can be quite big (e.g., the top group computed from the clustering step shown in Fig. 4), since we compute pose groups based only on the local distance information between poses. We adopt a second, partitioning stage that splits big groups into smaller groups. More specifically, we check whether the diagonal size of the bounding box computed from a pose group, c, is bigger than a threshold (e.g., $1.5 \times d$). If so, we recursively split the group c into two pose groups by using a median spatial partitioning, which divides the longest width of the bounding box of the group c into half. For each final pose

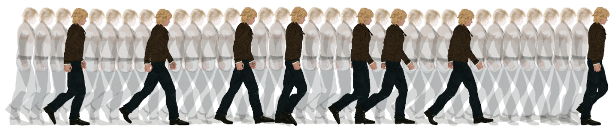
Figure 5: This figure shows eight poses chosen out of 35 poses for the walking animation based on our pose selection method.

group, we choose a representative pose (e.g., red circles shown in Fig. 4) that is closest to the center of the group.

We tried a well-known clustering method, K-means, for computing representative poses. We chose our clustering and partitioning framework instead of K-means, mainly because it is hard to set the target number of pose groups that well represents the distribution of the original poses for different animation patterns (e.g., walking, running, etc.).

4.2 Error-Aware Clustering Method

To construct the cluster hierarchy of an articulated model, we first decompose the base mesh of the model into a set of clusters. These computed clusters become leaf clusters of the cluster hierarchy.

We identify two different criteria and consider them for the cluster construction. First, each cluster should be a spatially coherent portion of the mesh, in order to keep the bounding box of the cluster small and thus achieve a high culling ratio. Second, each cluster should contain vertices that have similar simplification errors. For example, if a cluster contains two different mesh regions such as a joint and upper arm, highly deforming and less deforming regions respectively in the animation, then the cluster can get a high simplification error caused by the joint region, even though some portions (e.g., the upper arm) contained in the cluster may have a much lower simplification error. In this case, we may have to render a higher number of triangles even though some of those triangles can be simplified further given the user-specified error bound.

We, therefore, propose an error-aware clustering method that considers the simplification error as well as the spatial coherence for the sub-mesh contained in each cluster. To consider the simplification error for the geometry contained in each cluster during clustering, we have to simplify the mesh and compute simplification errors. However, this causes a chicken-and-egg problem, since we have to construct clusters first before simplifying clusters.

In order to avoid this problem, we measure how much a vertex deforms during the animation as a *deformation level* for the vertex, and use it as a rough approximation of the simplification error for the vertex. This is because as a vertex deforms more, it tends to generate a higher simplification error computed by considering different poses of animations.

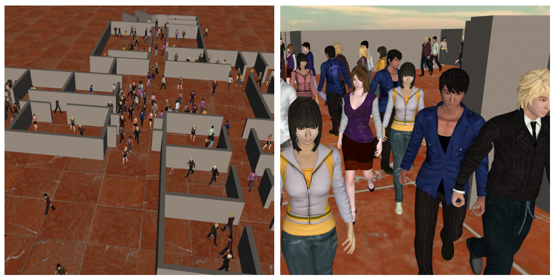
Figure 6: This figure shows two images of an office evacuation crowd scene with two hundred articulated characters. This scene consists of 16.4 M triangles. Our method can achieve 49 frames per second (fps) on average for this scene.

To measure the deformation level of a vertex, we identify a bone, b_a, that is a least common ancestor for bones that affect the vertex and then compute a tightest bounding box that contains the trajectory of the vertex during the animation in the reference frame of the bone b_a. The deformation level of the vertex, then, is computed as the diagonal length of the bounding box. The computed deformation level does not fully capture the simplification error, which is also affected by the neighboring triangles of the vertex. Nonetheless, we have found that it works well for our tested benchmarks and is very easy to compute the deformation level during the clustering stage.

For computing clusters of the base mesh, we adopt a clustering and partitioning framework that is similar to the one we used for computing pose groups. In the first clustering stage, we group adjacent vertices (i.e. spatially coherent vertices) with similar deformation levels into a cluster. We define that two vertices have the similar deformation levels, when their deformation levels differ within the range of 10%. We continue this process until we cannot merge vertices any more.

Clusters computed from the clustering stage can have a very high number of triangles. Therefore, we apply the partitioning stage, which recursively splits clusters that have more than s (e.g., 100) triangles based on the median-based spatial partitioning. Fig. 7-(b) shows computed leaf clusters based on our method for an input model.

4.3 Hierarchy Construction

We construct the cluster hierarchy in a bottom-up manner, starting from leaf clusters computed in Sec. 4.2. We construct the hierarchy by merging two adjacent clusters that have similar deformation levels. We define two clusters to be adjacent, if they have adjacent triangles that share the same vertices. We also define the deformation level of a cluster to be the maximum of the deformation levels of vertices contained in the cluster.

Particularly, among adjacent clusters for a cluster, c, we identify a cluster, c_m, that has the minimum difference

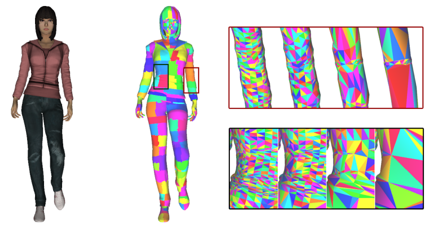

Figure 7: This figure shows an original model (a) and its leaf clusters generated by our error-aware clustering method (b). (c) show zoomed regions of boxed parts in (b). The top of (c) shows a deforming elbow region with increasing simplification levels. The bottom shows a non-deforming torso region, resulting in more aggressive simplification.

of the deformation level to that of the cluster c, and merge c_m and c into their parent cluster. We repeat this procedure until we construct the root node of the cluster hierarchy.

Simplification. Once we construct the hierarchy, we then perform the simplification for each cluster by traversing clusters in a bottom-up manner. For each leaf cluster, we simplify the sub-mesh contained in the cluster such that the number of the triangles in the cluster is reduced into half. For the simplification, we apply the pose-independent simplification method [DR05] that uses the well-known quadric simplification error metric. During the simplification, we consider only representative poses computed by our pose selection method (Sec. 4.1).

5 GPU-BASED RENDERING

In this section we explain how we can design an interactive GPU-based VDR method integrated with occlusion culling based on our VDR-AM representations for articulated models.

5.1 LOD Selection

In order to perform VDR and culling, we first compute a LOD cut in the cluster hierarchy that represents the articulated model given the error bound. To compute the LOD cut, we measure the geometric simplification error of each cluster in the screen space. To do that, we pre-compute a sphere whose diameter corresponds to the maximum Hausdorff distance between the original mesh and its corresponding simplified mesh contained in each cluster. For efficient computation, the maximum Hausdorff distance is approximated as the square root of the maximum quadric error associated with each cluster. We project the sphere associated with each cluster to the screen space. If the diameter of the projected sphere is equal to or smaller than the user specified pixels-of-error (PoE) value, we treat the cluster to have an enough resolution that can represent the articulated model given the PoE value. This LOD cut selection method takes only a minor CPU time (e.g., less than 1 ms) even for the exhibition crowd scene (Fig. 1) that consists of 1 K articulated models and 83 M triangles.

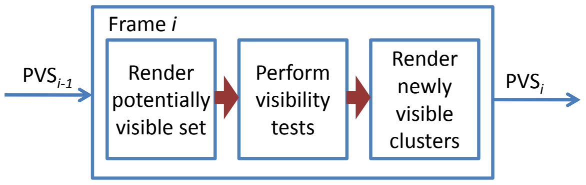

Figure 8: This figure shows our runtime rendering architecture.

5.2 Rendering Algorithm

We use an image-based conservative culling algorithm [YSM03] that is based on the frame-to-frame coherence and hardware-accelerated occlusion queries, in order to achieve an efficient performance of culling. We briefly explain how we enable such image-based culling based on our VDR-AM representation for articulated models.

Before we perform the culling, we decompose active clusters of the hierarchy into two disjoint sets: *potentially visible set* (PVS) and *potential occludee set*. Clusters in the PVS are treated to be visible and are used to create an occlusion map for visibility tests of clusters. We use PVS_i to denote the PVS at frame i. The overall architecture of our rendering algorithm is shown in Fig. 8.

Occlusion map generation. At frame i, we start with PVS_{i-1}, the PVS at the previous frame $i-1$. As Step 1 of our algorithm, we first update, i.e. simplify or refine, clusters of PVS_{i-1} to meet the error bound with our LOD selection method (Sec. 5.1) that considers the current view information. We set those clusters as PVS_i. We then render clusters of PVS_i into color and depth buffers. The depth buffer information computed with PVS_i serves as an occlusion map for visibility tests in the next rendering step.

Visibility tests. All the clusters of the LOD cut that are not in the PVS_i are used for the potential occludee set. We also update all the clusters in the occludee set. In Step 2 of our rendering algorithm, we check the visibility of clusters of the occludee set by using hardware accelerated occlusion queries. For clusters that passed the standard view-frustum culling, we use the bounding volumes of those clusters with the occlusion queries against the occlusion map. These bounding volumes serve as a conservative proxy to the geometry contained in corresponding clusters. We call those clusters of the occludee set that are identified as visible clusters with occlusion queries *newly visible* clusters. Fig. 9 shows an example of culling results in one of our benchmark scenes.

Rendering newly visible clusters. As the final step of our algorithm, we render newly visible clusters, to

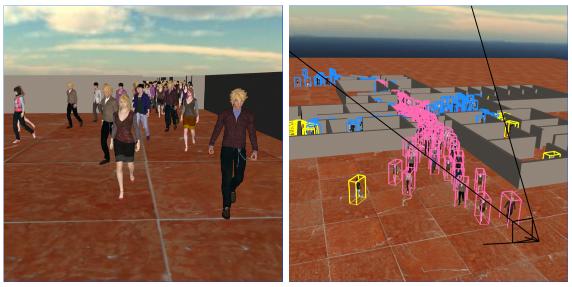
Figure 9: The left figure shows the image created at a first person view. The right image shows a third person view with occlusion results. The black lines show the view frustum of the first person view. The pink, blue, and yellow boxes are visible, occlusion culled, and view-frustum culled clusters.

create the final image. Also, these newly visible clusters are added to the PVS_i. We use the PVS_i for the next frame. Note that by taking advantage of temporal coherence, we only update and reset the PVS every n frames.

Instancing and static models. It is common to use instancing to create large-scale crowd scenes. We also store various data of cluster hierarchies such that we can efficiently utilize the GPU-based instancing. More specifically, we adopt the pseudo-instancing method [Zel04] for our VDR representation. In addition, our VDR-AM representation can be easily applied to handling static models. In this case our method considers only base meshes of static models. As a result, our rendering method can be applied to handling both static and articulated models.

6 RESULTS

To show benefits of our representation, we have implemented our construction and runtime rendering algorithms on a 3.00 GHz Intel quad-core PC with a GeForce 8800 GTX GPU that has 768 MB. We store all the transformation matrices of bones of various articulated models with all the poses in a 1 D texture buffer, which is easily accessible in the GLSL vertex shader that implements our runtime skinning method. For all the performance tests, we use the HD image resolution of 1280 by 720. In this image resolution, we use the PoE value of 0.5, in order to avoid visual artifacts to viewers. Since the used PoE causes only sub-pixel errors in the screen space, we do not perform any expensive geomorphing [Hop97].

Benchmark scenes. We have tested our method with three different crowd scenes that consist of human and animal articulated characters. Our first benchmark represents an office evacuation scenario (Fig. 6). This office scene consists of 200 instanced virtual human characters and 16.4 M triangles. Our second benchmark represents an exhibition scenario (Fig. 1). This exhibition scene consists of 1 K instanced characters and 83 M triangles. The third scene is a stampede scenario (Fig. 2) consisting of 5 K instanced animal characters and 242 M triangles. In the first and second crowd scenes, we use 20 different virtual human characters that have 35 bones and 73 K to 110 K triangles for their base meshes. Also, they are animated by using a walking animation pattern that consists of 35 different poses. In the third crowd scene, we use horse, elephant and camel models that consist of 17 K, 85 K, and 44 K triangles respectively. They have 30 to 40 bones and are animated in a running pattern with 15 to 80 poses.

Pose selection parameter setting. In our pose selection method, the parameter d trades-off between the simplification quality and performance of our overall construction method. If we set d to be too high, we would choose too small number of representative poses, causing faster construction, but leading to a worse simplification quality. On the other hand, if we set d to be too low, we would get the reverse effects: slower construction, but higher simplification quality. Given this trade-off space, we found that the range of 10% to 30% of the diagonal size of the bounding box of the model for d strikes a good balance in our tested benchmarks.

In this setting, for a walking animation that consists of 35 poses, our method selects 8 different poses. For a snake crawling animation with 81 poses, our method selects 13 poses. As a result, our method achieves 3 to 5 times performance improvement for our construction method. Also, in terms of the simplification quality, we found that the RMS distance between the original mesh and the simplified mesh computed only from considering computed representative poses is within 1% to 2% difference to the RMS distance between the original mesh and the simplified mesh computed from considering all the poses; we use the metro tool [CRS98] to measure the RMS distance. Fig. 5 shows eight chosen poses out of 35 poses of the walking animation. Note that more poses are chosen during a period that the human character switches his pivoting leg.

Hierarchy construction comparisons. In order to show the benefits of our error-aware clustering method, we additionally implemented a naive clustering method that uses an octree. More specifically, we construct clusters by recursively partitioning the base mesh until the sub-mesh contained in each node of the octree has s triangles. This naive method considers only the base mesh computed from a pose and does not consider any simplification errors. Once we compute clusters from the octree, the hierarchy construction and simplification that we have applied to to our error-aware clustering method are performed in the same manner to the naive, octree-based clustering method.

Compared to this naive method, our error-aware clustering method shows only 4% slower clustering performance at preprocessing, but shows 50% higher runtime rendering performances, by rendering 50% fewer triangles given the same error bound in our tested benchmarks. Since we cluster vertices that have similar sim-

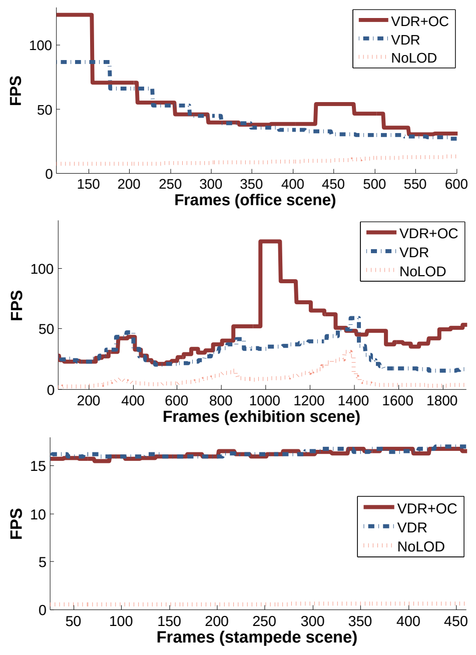

Figure 10: These figures show fps graphs of different methods: our VDR method, **VDR**, our VDR method integrated with occlusion culling, **VDR+OC**, and **NoLOD** that uses the original resolutions and view-frustum culling.

plification errors, we can allow more drastic simplification if possible, over the naive octree-based clustering method. Fig. 7-(c) shows our simplification results on different portions of a walking character.

Construction time and memory requirement. We set each cluster to have less than 100 triangles. In this setting, our method creates 1.8 K leaf clusters and takes 19 min in CPU to compute our representation for the biggest virtual character that has 110 K triangles. Also, our representation requires 72 bytes per each triangle for an articulated model. Since the original mesh requires 30 bytes per each triangle, our representation requires 140% more space over the original mesh. Since our representation stores simplified triangles, whose number is similar to the number of original unsimplified triangles, our representation requires at least 100% more space over the original mesh. Therefore, the memory overhead 140% of our representation is not significantly high.

Comparison configurations. We measure the performance of three different methods: 1) a base rendering system, **NoLOD**, that uses the original resolutions and performs only view-frustum culling, 2) our VDR rendering system, **VDR**, that performs the VDR on the base system, and 3) our VDR system integrated with occlusion culling, **VDR+OC**. We measure the frames per second (fps) of these methods with pre-defined paths, which are shown in the accompanying video. The fps graphs of these methods with our benchmark scenes are shown in Fig. 10.

Benchmark	FPS			#. Rendered Tri. (M)		
	NoLOD	VDR	VDR+OC	NoLOD	VDR	VDR+OC
Office Fig. 6	10.27	43.29	49.21	12.46	2.29	2.06
Exhibition Fig. 1	7.73	29.15	46.08	24.66	3.14	1.94
Stampede Fig. 2	0.55	16.38	16.23	148.09	4.77	4.70

Table 1: This table shows the frames per second (fps) and the number of rendered triangles on average while rendering scenes. **NoLOD**, **VDR**, and **OC** represent rendering with the original resolutions, our view-dependent rendering method, and our occlusion culling technique respectively.

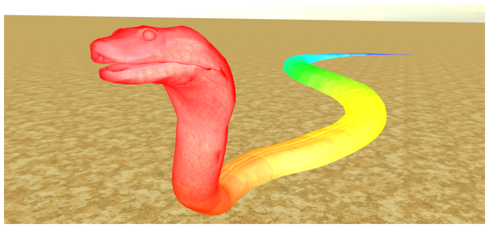

Figure 11: This figure shows adaptively varying resolutions computed from our VDR-AM representation. We render each vertex of the model with colors computed from the well-known heat color map; the red color indicates the highest resolution, while the blue color refers to the lowest resolution.

6.1 Rendering Performance

On average, **VDR+OC** achieves interactive performances, 49 fps and 46 fps in the office and exhibition crowd scenes respectively (Table 1). In the exhibition scene, compared to the base rendering system, we achieve 4 times performance improvement by enabling the VDR, and achieve 6 times performance improvement by enabling VDR and occlusion culling together. We also observe a similar performance gain with the office scene. These performance improvements are mainly caused by reducing the number of triangles that we have to process for the skinning and rendering operations. More specifically, the base rendering system renders 458 K clusters that contain 24.66 M triangles for the exhibition scene. On the other hand, **VDR** renders 12 K clusters with 3.14 M triangles and **VDR+OC** renders 8 K clusters with 1.94 M triangles.

In the stampede scene our method achieves interactive performance, 16 fps, even though the original model consists of more than 200 M triangles. Also, it demonstrates high performance improvement (up to 30 times) by enabling VDR over the base rendering method. Nonetheless we show a minor, but lower performance by enabling occlusion culling over VDR. This is mainly because we do not have much depth complexity in the tested view point.

6.2 Discussions

To highlight the benefit of our method, we show varying resolutions of our view-dependent representation for a snake articulated model that consists of 28 K triangles and 81 poses for its crawling animation (Fig. 11). We use the heat color map to show how the resolution of the model varies given the first person view. As a portion of the model is farther away from the viewer, it gets lower resolution as indicated by showing colors close to the blue one. At the given view our method requires only 18 K triangles for rendering the snake model.

Relationships with LODs. Many view-dependent representations have been proposed, as discussed in Sec. 2. These techniques have not been widely applied to articulated models. This may be mainly because that many prior view-dependent techniques have high computational overheads. Nonetheless, our technique reduces the computational overhead by providing view-dependent resolution at a granularity of clusters consisting of around 100 triangles, not each triangle of the mesh. Note that this kind of approach is inspired by efficient LOD rendering techniques such as HLODs (Hierarchical levels of detail) [EMB01] designed for large-scale static models.

Breakdown of each rendering component. We also measure how much percentage each component of our method takes over the total rendering time. The CPU-based selection of the LOD cut given a viewing configuration takes less than 1.5 ms. The GPU-based skinning, rendering, and occlusion culling components take 14%, 82%, and 4% over the total rendering time on average across all the tested benchmarks.

Limitations. Even though our method shows performance improvements over the base rendering method, there is no guarantee that our VDR integrated with occlusion culling always improves the performance of various crowd scenes. This is mainly because performing VDR and occlusion culling has overheads. Also, our method may show visual artifacts with PoE values bigger than 1, while we were able to achieve interactive performance without noticeable artifacts by using 0.5 PoE for our tested models. Geomorphing can ameliorate *popping* artifacts by providing smooth transitions between different LODs.

7 CONCLUSION

We have proposed view-dependent representation for articulated models, VDR-AM, and presented how we can use it for an interactive view-dependent rendering method integrated with occlusion culling in large-scale crowd scenes. VDR-AM consists of a cluster hierarchy that serves both as a multi-resolution representation for VDR and a bounding volume hierarchy for occlusion culling. We also presented an error-aware cluster construction method to allow drastic simplifications on portions of meshes of articulated models. We were able to achieve 16 to 49 fps on average for large-scale crowd scenes that consist of thousands of articulated models and hundreds of millions of triangles without noticeable visual artifacts.

There are many avenues for future work. In addition to addressing current limitations of our method, we would like to first handle larger crowd scenes by designing a more drastic simplification method (e.g., volumetric simplification methods) as well as to simplify skeletal and behavioral models of characters [RCBS10]. One can combine our method with image-based representations [DHOO05, KDC$^+$08]; use our method in a near field and use impostors allowing more drastic simplifications in a far field. Also, we used deformation levels to estimate simplification errors. We would like to extend this concept to identify vertices that have similar rotational sequences [JT05], to more accurately cluster vertices that have similar simplification errors. In addition, it would be interesting to conduct a user study measuring perceptual errors of our method. Finally, we would like to apply our VDR-AM to other geometric applications such as collision detection. Since our representation is based on a polygonal representation, we believe that it can be easily applied to collision detection in a similar spirit to the collision detection method designed for dynamic simplification [YSLM04].

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their constructive feedbacks. This work was supported in part by MCST/KOCCA/CT/R&D 2011, MKE/KEIT [KI001810035261], MKE/MCST/IITA [2008-F-033-02], BK, DAPA/ADD (UD110006MD), MEST/NRF/WCU (R31-2010-000-30007-0), KMCC, MSRA, and MEST/NRF (2012-0009228).

8 REFERENCES

[ACCO05] Assa, J., Caspi, Y., and Cohen-Or, D., Action synopsis: pose selection and illustration. ACM Trans. Graph., 24, no. 3, pp. 667–676, 2005.

[Ben75] Bentley, J.L., Multidimensional binary search trees used for associative searching. Communications of the ACM, 19, pp. 509–517, 1975.

[CBWR07] Charalambos, J.P., Bittner, J., Wimmer, M., and Romero, E., Optimized hlod refinement driven by hardware occlusion queries. In Advances in Visual Computing, Springer, 2007, pp. 106–117.

[COCSD03] Cohen-Or, D., Chrysanthou, Y., Silva, C., and Durand, F., A survey of visibility for walkthrough applications. IEEE Tran. on Visualization and Computer Graphics, 9, pp. 412–431, 2003.

[CRS98] Cignoni, P., Rocchini, C., and Scopigno, R., Metro: Measuring error on simplified surface. Computer Graphics Forum, pp. 167–174, 1998.

[DHOO05] Dobbyn, S., Hamill, J., O'Conor, K., and O'Sullivan, C., Geopostors: a real-time geometry/impostor crowd rendering system". ACM Transactions on Graphics, 24, no. 3, pp. 933–933, 2005.

[DR05] DeCoro, C., and Rusinkiewicz, S., Pose-independent simplification of articulated meshes. In Symp. on Interactive 3D Graphics, 2005, pp. 17 – 24.

[EMB01] Erikson, C., Manocha, D., and Baxter, B., Hlods for fast display of large static and dynamic environments. Proc. of ACM Symposium on Interactive 3D Graphics, 2001.

[GH97] Garland, M., and Heckbert, P., Surface simplification using quadric error bounds. ACM SIGGRAPH, pp. 209–216, 1997.

[HCC06] Huang, F.C., Chen, B.Y., and Chuang, Y.Y., Progressive deforming meshes based on deformation oriented decimation and dynamic connectivity updating. In ACM Symp. on Computer Animation, 2006, pp. 53–62.

[Hop97] Hoppe, H., View dependent refinement of progressive meshes. In ACM SIGGRAPH, 1997, pp. 189–198.

[JT05] James, D.L., and Twigg, C.D., Skinning mesh animations. ACM Trans. on Graphics (SIGGRAPH), 24, no. 3, 2005.

[KDC$^+$08] Kavan, L., Dobbyn, S., Collins, S., Zara, J., and O'Sullivan, C., Polyposters: 2d polygonal impostors for 3d crowds. In ACM Symp. on Interactive 3D Graphics and Games, 2008, pp. 149–155.

[KG05] Kircher, S., and Garland, M., Progressive multiresolution meshes for deforming surfaces. In Symp. on Computer animation, 2005, pp. 191–200.

[LRC$^+$02] Luebke, D., Reddy, M., Cohen, J., Varshney, A., Watson, B., and Huebner, R., Level of Detail for 3D Graphics. Morgan-Kaufmann, 2002.

[LS09] Landreneau, E., and Schaefer, S., Simplification of articulated meshes. Computer Graphics Forum, pp. 347–353, 2009.

[McG68] McGee, V.E., Multidimensionnal scaling of n sets of similarity measures : A nonmetric individual differences approach. Multivariate Behavioral Research, 3, pp. 233–248, 1968.

[MG03] Mohr, A., and Gleicher, M., Deformation sensitive decimation. University of Wisconsin Graphics Group. Technical Report, 2003.

[NGCL09] Narain, R., Golas, A., Curtis, S., and Lin, M.C., Aggregate dynamics for dense crowd simulation. In SIGGRAPH Asia, 2009, pp. 1–8.

[PSA07] Pilgrim, S., Steed, A., and Aguado, A., Progressive skinning for character animation. Computer Animation and Virtual Worlds, 18, no. 4-5, pp. 473–481, 2007.

[RCBS10] Rodriguez, R., Cerezo, E., Baldassarri, S., and Seron, F.J., New approaches to culling and lod methods for scenes with multiple virtual actors. Computers and Graphics, 34, no. 6, pp. 729 – 741, 2010.

[RD05] Ryder, G., and Day, A.M., Survey of real-time rendering techniques for crowds. Computer Graphics Forum, 24, no. 2, pp. 203–215, 2005.

[RL00] Rusinkiewicz, S., and Levoy, M., Qsplat: A multiresolution point rendering system for large meshes. SIGGRAPH, pp. 343–352, 2000.

[TCYM09] Tang, M., Curtis, S., Yoon, S.E., and Manocha, D., Iccd: Interactive continuous collision detection between deformable models using connectivity-based culling. Visualization and Computer Graphics, IEEE Transactions on, 15, no. 4, pp. 544 –557, 2009.

[TOY$^+$07] Thalmann, D., O'Sullivan, C., Yersin, B., Maim, J., and McDonnell, R., Populating virtual environments with crowds. In Eurographics Tutorial, 2007.

[YGKM08] Yoon, S.E., Gobbetti, E., Kasik, D., and Manocha, D., Real-Time Massive Model Rendering. Morgan & Claypool Publisher, 2008.

[YSGM04] Yoon, S.E., Salomon, B., Gayle, R., and Manocha, D., Quick-VDR: Interactive View-dependent Rendering of Massive Models. In IEEE Visualization, 2004, pp. 131–138.

[YSLM04] Yoon, S., Salomon, B., Lin, M.C., and Manocha, D., Fast collision detection between massive models using dynamic simplification. In Eurographics Symposium on Geometry Processing, 2004, pp. 136–146.

[YSM03] Yoon, S., Salomon, B., and Manocha, D., Interactive view-dependent rendering with conservative occlusion culling in complex environments. In Proc. of IEEE Visualization, 2003.

[YYBE13] Yuksel, K., Yucebilgin, A., Balcisoy, S., and Ercil, A., Real-time feature-based image morphing for memory-efficient impostor rendering and animation on gpu. The Visual Computer, 29, pp. 131–140, 2013.

[Zel04] Zelsnack, J., Glsl pseudo-instancing. Tech. rep., NVIDIA Corporation, 2004.