Computers & Graphics 34 (2010) 729–741

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

New approaches to culling and LOD methods for scenes with multiple virtual actors ☆

Rafael Rodriguez [a], Eva Cerezo [b], Sandra Baldassarri [b,*], Francisco J. Seron [b]

[a] Brainstorm Multimedia, Valencia, Spain
[b] Advanced Computer Graphics Group (GIGA), Computer Science Department and Engineering Research Institute of Aragon (I3A), University of Zaragoza, Spain

ARTICLE INFO

Article history:
Received 3 December 2009
Received in revised form
15 July 2010
Accepted 15 July 2010

Keywords:
Computer animation
Synthetic actors
Real-time
Culling
LOD

ABSTRACT

Despite the ever-increasing power of graphics workstations, rendering and animating virtual actors remains a very expensive task. Current scenegraphs are clearly oriented toward the management of static scenarios instead of toward scenes with multiple dynamic elements guided by complex behaviors, as in the case of virtual actors, human or not. Applications that deal with the management of multiple actors require the development of specific methods that reduce not only the number of polygons sent to graphics hardware but also the calculations involved in the management of multiple reference systems and the behavior of actors. In this paper we propose a culling method based on the use of different types of bounding spheres, which minimizes the number of calculations related to the behavior of actors and allows better use of the capacities of present-day graphics hardware. With the same aim in mind, we also propose a method for managing levels of detail which acts not only on a geometric level but also on skeletal and behavioral levels. These ideas can be implemented on top of traditional scenegraphs, resulting in significant improvements in computational costs. This improvement is analyzed, as well as the results obtained when managing scenes with thousands of virtual actors.

1. Introduction: virtual actors and scenegraphs

The use of a scenegraph-based library is the most standard method of implementing a simulation application. However, the way in which present day scenegraphs are defined and processed clearly reveals that they are geared toward static scenarios or scenarios with a very small number of reference systems. A typical example of an application of this kind would be a flight simulator, in which the majority of objects are static (terrain, buildings, trees, etc.), and there is a very small number of mobile elements (airplanes and other types of vehicles), which are not articulated, and which, moreover, have simple behaviors. Traditional scenegraphs are geared toward the management of this type of application. They focus on controlling the number of polygons sent to the graphics hardware. However, they pay little attention to the costs derived from the existence of multiple reference systems, and do not perform behavioral management in any way whatsoever. Fortunately, as will be shown, the traditional scenegraph can be seen as a low-level tool on which new methods can be built.

The characteristics of a scene involving virtual actors are very different: such a scene involves many reference systems (a single actor may have 40 different reference systems, more than are usually required for a complete traditional simulation), and the computational cost of the behavioral management of a virtual actor tends to be very high. Just like a traditional scenegraph, a scenegraph which manages multiple virtual actors has to control the number of polygons sent to the graphics hardware, but it must pay equal or greater attention to minimizing the calculations derived from the existence of multiple reference systems (management of transformation matrixes, processing of bounding volumes, etc.), and performing behavioral management. In fact, at the present time, reducing the number of calculations and the number of polygons sent to the graphics hardware to facilitate interactive work in scenes with multiple actors is still a challenge. In Section 2 of this article, we review the efforts undertaken in this direction.

The paper presented here is the continuation of a previous paper [1] which proposed a data structure geared toward the management of virtual actors in real time applications. This paper is based on those structures (recalled in Section 3 of the present article), but focuses on the adaptation of several aspects of scenegraph processing to real-time applications. In particular, two new methods for minimizing the number of operations required and making better use of the graphics hardware's capacities are proposed: a culling management method (presented in Section 4)

☆ This article was recommended for publication by Gladimir Baranaski.
* Corresponding author.
E-mail addresses: rafael@brainstorm.es (R. Rodriguez), ecerezo@unizar.es (E. Cerezo), sandra@unizar.es (S. Baldassarri), seron@unizar.es (F.J. Seron).

0097-8493/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2010.07.006

and a level of detail management method (Section 5). The article follows with a theoretical estimate of the computational improvement yielded by these methods in relation to traditional scenegraph processing, which is presented in Section 6, along with an example of application and of the results presented in Section 7. Our conclusions and possible lines of future research are put forward in Section 8, and finally an Appendix with extra data and notation is included.

2. Related work

Despite the ever-increasing power of graphic workstations, rendering and animating virtual actors remains a very expensive task. Even very high-end graphics systems can have trouble sustaining a sufficient frame rate when they have to render numerous moving figures commonly made up of thousands of polygons.

In Computer Graphics, reducing the number of polygons sent to graphics hardware is a problem that has received much attention, and several techniques have been developed in relation to it. Generally speaking, acceleration techniques for rendering large environments can be subdivided into three main categories: visibility culling methods, level of detail (LOD) methods and image-based rendering (IBR) techniques. What they all have in common is the aim of reducing the complexity of the scene while preserving its visual characteristics. They can often be combined to produce better results as well. Nonetheless, most of these techniques were developed for the efficient management of large static polygonal models, so their application to the management of thousands of complex dynamic entities such as virtual actors is not a trivial matter.

Culling algorithms basically discard objects or parts of objects that are not visible in the final rendered image: if an object lies outside the view frustum it is not sent to the graphics pipeline (visibility culling) [2]. The most common culling techniques are: backface and clustered backface culling, hierarchical view frustum culling and portal culling [3]. Nevertheless, these traditional ways of culling render their application to scenes with multiple actors rather ineffective, since they are implemented after performing a high number of calculations on all the actors. The method proposed in Section 4 tries to avoid this problem. Another possible culling method is based on eliminating those elements occluded by other parts of the scene (occlusion culling) [3–5]. A major drawback of occlusion culling algorithms is that they usually require a specific organization of the entire geometry database: the scene is typically divided into smaller units or cells in order to accelerate the culling process, so in its native form it is not well suited to the treatment of individual, deformable and mobile objects. Yoon et al. [6] combine conservative occlusion culling with view-dependent rendering for interactive display of complex environment, but in their case the occlusion culling algorithm does not work well with high movement either.

The basic principle of Image-Based Rendering is to replace certain parts of the polygonal content of the scene with images. There have been several attempts to apply IBR to scenes with virtual actors, such as the work of Tecchia et al. [7], which uses pre-generated impostors, and that of Aubel et al. [8], which proposes dynamically generated impostors to diminish the required memory. Ulicny et al. [9] succeed in rendering complex scenes involving thousands of animated individuals at interactive frame rates by storing pre-computed deformed meshes in OpenGL display lists and then carefully sorting them using the OpenGLScene-Graph 3D toolkit to take cache coherency into account. In [10] they improved on their performance by using four level-of-detail meshes for their model, thereby reaching a frame rate which is several times higher.

Geometric level of detail (LOD) attempts to reduce the number of polygons rendered by using several decreasingly complex representations of an object. The appropriate model or resolution is selected for each frame. The typical selection criterion is the distance from the viewer, although most games use a combination of geometric LOD approaches [11]; object motion is also taken into account (motion LOD) in some cases [12]. Pratt et al. [13] have applied geometric LODs to virtual humans in large-scale networked virtual environments. They used a heuristic to determine the viewing distances that should trigger a LOD switch. Each simulated human has four different resolutions, ranging from approximately 500 polygons down to only three. However, their lowest resolutions are probably too coarse to be actually used unless the virtual human is only a few pixels high, in which case a technique other than polygon rendering may be considered. Dobbyn et al. [14] have presented a hybrid system for real-time rendering of large-scale animated crowds. This system implements a Level Of Detail approach by using image-based representation for virtual humans at a distance, and switching to geometric representation once the human is within a certain distance threshold based on a texel to pixel ratio. The shading of impostors and the introduction of variety into the virtual human's geometric and impostor model takes place at run-time through programmable graphics hardware, and the system allows the animation of large crowds at interactive frame rates.

Even though the research on levels of detail for simulations of actors has taken place mainly on a geometric level, approaches that reduce the complexity of the motions generated have also been set forth. Cozot et al. [15] suggested a level of detail architecture that comprised a pipeline of sub-models, in which each sub-model performs a given task during the animation process. The architecture is applied to a model capable of walking on complex terrain. Level of detail is selected according to two criteria: complexity of the environment and distance from the camera. Carlson and Hodgins [16] introduced the idea of simulation levels of detail, and applied it to the animation of one-legged creatures at multiple levels of detail, according to the importance of a creature and its visibility. Simulation methods include rigid-body dynamics and hybrid kinematics/dynamics, depending on the accuracy required. Developing this idea further, Brogan and Hodgins [17] use simulation LOD to control the movements and actions of large groups. By providing a simplified version (reducing the number of degrees of freedom) of a physically simulated character, they were able to simulate large groups by dynamic switching between these LODs. O'Sullivan et al. [18] describe a framework called ALOHA (Adaptive Level Of detail for Human Animation), which incorporates levels of detail for both geometry and motion. Random key-frame animations are chosen for those characters which are not highly rated, while more sophisticated motions synchronized with speech are applied to more salient characters. The switch between animation levels is performed according to a set of predefined rules, in an attempt to minimize degradation to the simulation viewer's perception. However, they do not specify the number of actors they are able to work with, nor do they estimate the computational reduction of their proposal. They also propose the addition of a LOD-AI, Artificial Intelligence LOD, based on the use of role passing techniques [19], but only preliminary results have been reported. As commented before, more recent work has focused on the geometric LOD management [14]. Their work is directly related to ours, which proposes integrated management of a geometric, skeletal and behavioral LOD. In our case, the ideas are implemented within a standard framework like Performer, computational reduction is assessed and we provide an example

application which manages 6000 actors at interactive frame rates. The skeleton pruning presented in [20] is a similar idea to our skeleton LOD. Because of the way the authors store and manage joint transforms, they limit the simplification to cut-off or prune bones or entire sub-trees of bones at the end of the skeleton when the details are no longer visible.

3. Data structure for actor management

This paper is based on a previous paper [1] in which we proposed two new types of specially designed nodes for the integration of synthetic actors in real-time simulation applications. The main advantage of using these new structures instead of standard ones such as H-Anim [21] or MPEG4 [22] lies in the circumstance that our proposal allows for the development of any generic articulated structure, whether the latter is an object, an animal or a person. Our intention is to provide a basic specification that can help to create a new standard, more generic and real-time centered, based on the implication of how some de-facto standard work (OpenGL Performer [23], OpenSceneGraph [24]). H-Anim, on the contrary, was specifically designed for the creation and exchange of generic human beings. The H-Anim structure is very well defined, but it is too complex to be used in real-time simulations as well as with hundreds of individuals. MPEG4 also focuses its attention on the encoding of 3D humanoid characters' facial and bodily animation [25]; however, it does not standardize the geometric models representing the face or the body, but the parameters on the basis of which a neutral face or body can be personalized and animated in real time.

We do not intend to delve deeply into these structures once again here, although we will briefly summarize them to facilitate the understanding of the following sections.

There are two types of nodes, called ***Actor*** and ***Skeleton*** that provide the user with high-level control over virtual actors and implement low-level management, thus minimizing their computational cost.

The *Actor* node acts as a control center of all the actions of virtual actors and as a basic reference system for all the nodes that make up each actor. For the latter purpose, we considered it convenient that the actor employ two groups of transformations, the first of which defined the position of a point called *Reference Point*, normally associated to the actor's center of mass, and a second one, which defines the position of the *Skeleton Root,* a fixed point located somewhere on the actor's bone structure, which operates as a basic reference system for the rest of its articulated structure (see Fig. 1). The critical point for the integration of virtual actors in a simulation application is no doubt the drawing of the articulated structure. The Actor node functions as the root node of a hierarchy of *Skeleton* nodes. Each Skeleton node defines an articulation point on the actor, such as the elbow, the knee or the neck. As we shall see in the following sections, the application of culling and traditional LOD methods to virtual actors is inadequate; therefore, we will propose specific methods. The Actor node will be responsible for performing part of this management, as well as for storing several fields destined to this purpose.

4. Culling method for actors

Traditional processing of a 3D scene for real-time involves, first of all, applying all the transformations of the objects (commonly known as APP), and then selecting the objects within the view frustum (commonly known as CULL) and sending said objects to the graphics hardware to be drawn (commonly known as DRAW). This is, for example, the traditional APP/ CULL/DRAW Performer

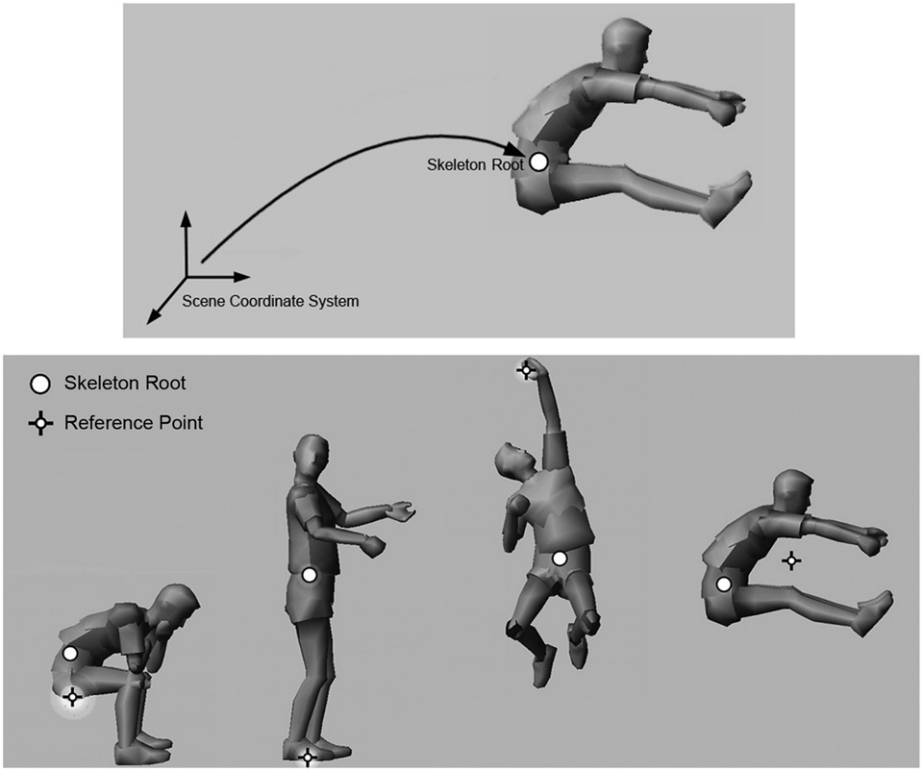

Fig. 1. Skeleton Root storing information on the location of the actor within the scene (above). Examples of the location of the Reference Point in different positions in relation to the actor (below).

pipeline [26,27]. This work outline, geared towards reducing the number of polygons, is adequate for a traditional simulation application, but is a waste of many resources when dealing with a simulation involving elements with complex behaviors or many transformation matrices. In the case of a scene with virtual actors, both these circumstances concur.

Let us picture a scenegraph with Actor and Skeleton nodes processed according to the traditional procedure. Under these circumstances, the drawing phase of a scene involving multiple virtual actors would be preceded by the following phases:

Phase 1: Execution of the code required for calculating the values of the Actor and Skeleton fields of all the actors present in the scene (behavioral management). These calculations take up very large amounts of resources (dynamic and reverse kinematic calculations, facial expression, artificial vision, etc.)
Phase 2: Generation of the Actor and Skeleton transformation matrices on the basis of the values calculated in the previous phase. In the modus operandi of the traditional culling process these multiplications have to be performed in the CPU.
Phase 3: The transformation matrices generated by each Actor and Skeleton node must be adequately multiplied during the traversal of the scenegraph.
Phase 4: Hierarchical recalculation of the bounding spheres (bspheres) of the scene nodes is required: the bsphere of each node is calculated on the basis of the bspheres of its descendant nodes.
Phase 5: Verification of intersection between different bspheres has to be carried out with the vision frustum to determine which parts of the actors are inside the vision frustum, which are then stored in the list of objects to be sent to the graphics hardware.

As can be seen, in a traditional scenegraph it is impossible to know if an actor is inside the frustum or not until its behavior has been calculated, the matrices of its nodes have been generated and applied, and hierarchical recalculation of its bspheres has taken place. In order to avoid this, the culling method has been modified in order to determine if an actor is outside of the vision frustum without resorting to these costly calculations. To achieve this, we propose the use of static bspheres (Section 4.1) and modification of culling management (Section 4.2). The connection between the culling of actors proposed and the traditional procedure is discussed in Section 4.3.

4.1. Types of bounding spheres used in actor culling

In order to establish whether actors are outside the vision frustum without managing their behavior or undertaking any operation whatsoever with their Skeleton nodes, we propose that the Actor node use static bounding spheres fixed during a phase prior to the simulation. Bounding spheres have been selected instead of bounding boxes or line swept spheres [18] since they behave very well when the orientation of the geometries is changing (as in the pieces of a virtual actor) and also because they are used for the default culling in the most standard scenegraphs like Performer or OpenSceneGraph.

Therefore, in this work, three types of bounding spheres will be used for the management of a virtual actor, two of which are static, while the third one is a dynamic bounding sphere similar to those used in traditional scenegraphs. Let us now look at the features of the three types of bspheres in more detail (from now, Bspheres with a capital letter indicates the proper name of a bsphere):

The ***Refpoint Bsphere*** is a sphere which is capable of encompassing a virtual actor and its movements. The size and position of the *Refpoint Bsphere* depend on the type of movement performed by the actor. This type of bounding sphere makes sense in the context of actors in which the movement of the Skeleton *root* in relation to the *Refpoint* takes place within a restricted area. This would be the case of an actor executing a certain key-framing table, or some kind of movement taking place within a limited space (such as, for example, the movement of a child on a swing).

The ***Sklsroot Bsphere*** is a sphere located within the actor's *Skeleton Root*, with a fixed radius large enough to completely encompass the entire virtual actor regardless of the position he or she adopts. For the moment, we assigned the radius of this sphere manually, however this process could be done automatically with little effort.

Internal Bspheres are the ensemble of bspheres that encompass the different geometries that represent a virtual actor. The position and size of these bspheres are dynamically modified according to changes in the degrees of freedom. These bspheres are organized hierarchically, so in the case of a human actor the first bsphere to be calculated would be the one corresponding to the foot; the next bsphere to be calculated would be the one resulting from the addition of the foot bsphere and the calf bsphere, following with the addition of the latter to the bsphere of the thigh would ensue, and so on until the result is a bsphere which fits the global geometry of the actor, and which would adapt to it in a similar way to which the *Sklsroot Bsphere* does so. However, as we have seen, in order to calculate this bsphere, prior calculation of the values of the actor's degrees of freedom, as well as the transformation and composition operations of the bounding spheres have to be completed. In Fig. 2 an example showing the different types of bspheres used in the processing of a virtual actor can be seen.

Culling verification in actors is carried out in cascade fashion: the intersection of the *Refpoint Bsphere* with the vision frustum is verified first, and the next verification—the calculation of the Skeleton root and the analysis of the *Sklsroot Bsphere* in relation to

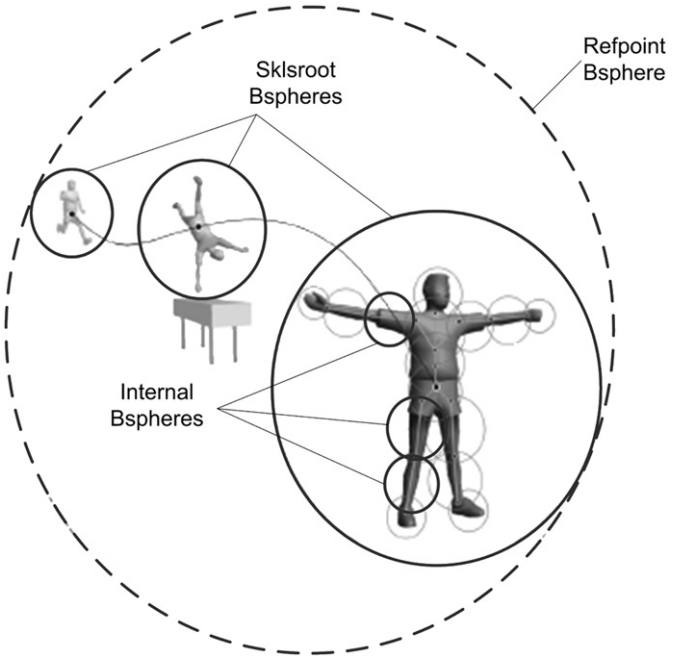

Fig. 2. Representation of the different types of bounding spheres used in the culling management of a virtual actor which is performing a pirouette on the basis of the values stored in a *key-framing* table.

the vision frustum—is performed only in the event that the latter is partially inside it. Only in the event that the *Sklsroot Bsphere* were partially inside would it be necessary to perform verifications of the *Internal Bspheres*, which surround the different geometries that constitute the actor internally and are calculated according to the traditional procedure.

4.2. Organization of actor culling in phases

In traditional processing there is a clear difference between the process that manages the behavior of the objects in the scene (APP) and the process that determines which polygons are sent to the graphics hardware (CULL). The use of static *bounding volumes*, and the fact that actors' behavior is managed after verifying if they are really inside the vision frustum, requires modification of traditional task organization in the form of a APP/CULL/DRAW pipeline so that culling and behavioral management of actors can be jointly performed, as opposed to the two independent phases resorted to by traditional scenegraphs.

Behavioral management of a virtual actor can be divided into three types of operations: calculation of Reference Point transformations, calculation of Skeleton root transformations and calculation of the degrees of freedom that determine the actor's posture. When a traditional scenegraph is used, these operations are performed on all the actors in the scene prior to the culling phase (see Fig. 3, top). The culling method proposed here, however, is divided into three phases, as shown in the bottom part of Fig. 3.

According to the diagram, these are the operations performed in each phase:

1. Calculation of the values of the transformation matrixes of the *Refpoints* of all the *Actor* nodes in the scene.
2. ***First CULL phase***, in which the transformations are applied to the *Refpoint Bspheres*, and the relation of these transformed bspheres to the frustum is verified, which results in the rejection of all the actors whose *Refpoint Bsphere* is outside of the vision frustum.
3. Calculation of the values of the transformation matrices of the *Sklsroot* of the remaining actors.
4. ***Second CULL phase***, in which the absolute positions of the *Sklsroot Bspheres* of the Actor nodes are calculated and the relation of said bspheres to the frustum is verified, which results in the rejection of all actors whose *Sklsroot Bspheres* are outside the vision frustum.
5. Behavioral management is performed on the actors that pass the second CULL phase, which generates specific values for the degrees of freedom of the Skeleton nodes. On the basis of these values, the resulting transformation matrices are then generated and applied.
6. A ***third CULL phase*** that determines which specific parts of actors have to be drawn is completed. This entails hierarchical recalculation of *Internal Bspheres* and intersection verifications with the frustum.
7. Only those parts of the actors which are inside the vision frustum are sent to the graphics hardware.

In both the traditional and the proposed methods, the number of polygons sent to the graphics hardware is the same. However, with the new method, behavioral management calculations are drastically reduced. In a typical scene, 70% of actors may be blocked during the first culling phase, along with an additional 10% during the second phase, i.e., it is a common occurrence that only 20% of the original actors ever require the calculation of their degrees of freedom and management of their Skeleton nodes.

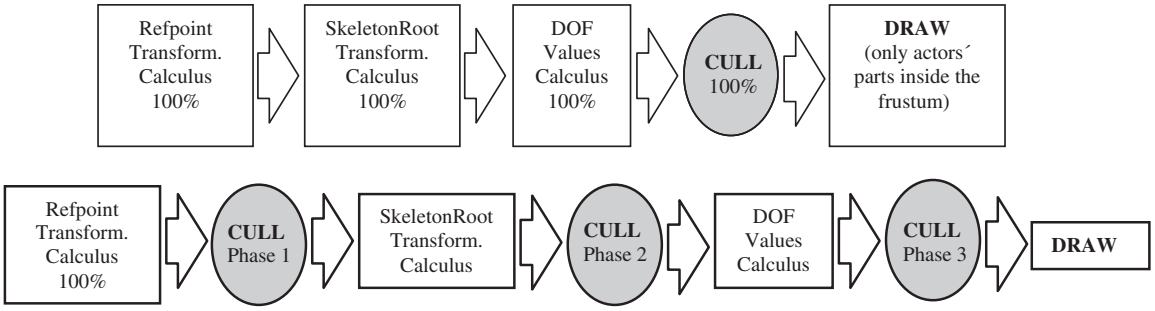

Fig. 3. Processing of actors with a traditional scenegraph (top). Processing of actors with the new culling method proposed (bottom).

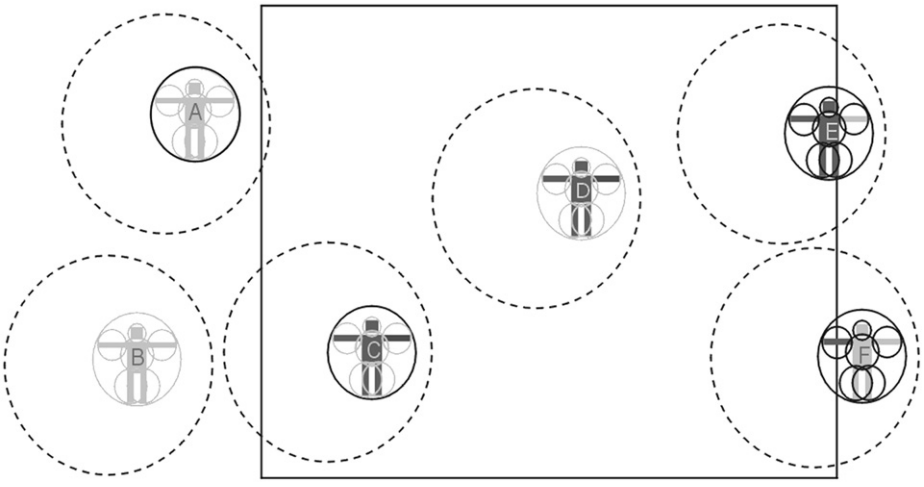

Fig. 4. Culling process on actors with different positions in relation to the frustum.

Fig. 4 offers an example that clarifies how the culling method interacts with the processing of virtual actors. In the figure, the frustum is represented by a rectangle, and the different bounding spheres are represented by dark circles to indicate that intersection with the frustum has been verified, and by clear circles to indicate that this is not the case. In similar fashion, those parts of the actors which are going to be drawn are represented in dark colors while those which are not are represented in clear colors. Clearly, a great deal of work can be avoided.

4.3. Interaction with generic scene culling

In the previous sections, we analyzed the most adequate form of performing operations related to the culling of virtual actors, given that the remaining elements of the scene which are not virtual actors are to be processed using traditional culling methods. In practice, both culling methods may prove interdependent: culling of virtual actors may affect elements in the scene and culling of elements in the scene may affect virtual actors.

The interaction between the culling of virtual actors and traditional culling may be classified as belonging to one of two types:

- A piece of scenegraph depends on a virtual actor. This would be the case of an actor that decides to hold an object which belongs to the scene (a tool, for instance).
- An actor depends directly on a generic scenegraph node. This may happen in case there is a superior structure grouping a set of actors together (a flock of birds, for example), or a group of actors and a set of traditional scene elements (for example, a building with all its structure and furniture, with several people inside it, or a vehicle with several occupants).

In the first case, when a culling verification with the *Refpoint Bsphere* or the *Sklsroot Bsphere* of the actor is performed, one must keep in mind that this operation will necessarily affect the tool being held by the actor. This is done by increasing the *Sklsroot Bspheres* radius so that it includes the actor and the object dependent on him or her, and also increasing, in a coherent way, the radius of the *Refpoint Bspheres*.

In the second case, traditional culling affects the processing of actors. Thus, in the example of the vehicle, the culling of the interior determines if the actors within it have to be processed or not.

5. Specific LOD management methods for virtual actors

The level of detail within a scenegraph is usually managed by special nodes called LOD (level of detail). This kind of node has several descendant branches, each of which represents the same object with a different level of complexity. When one of these nodes is reached during the traversal of the scenegraph, the distance between the viewpoint and the reference point of the LOD (the central point on the LOD) is calculated, and the descendant branch to be drawn according to that distance (and other criteria) is selected. In a traditional scenegraph all level of detail management is performed by this type of nodes.

Databases used in traditional simulations usually involve few transformations and very elementary behavioral management. This confines the management of the level of detail to verifying the number of polygons sent to the graphics hardware. In the case of virtual actors, the performance of this type of control, which we shall call ***geometric level of detail***, is necessary, but so is carrying out the management of the ***skeletal level of detail*** which determines the number of transformations applied by an actor, as well as management of the ***behavioral level of management*** that allows distance to reduce the computational cost of behavioral management. In the following sections we specify the effect that these three types of level of detail management have on virtual actors.

To facilitate all three of these types of level of detail management, the Actor node will store the ***eyeDistance*** and ***priority*** fields. The *eyeDistance* field stores the distance of the actor's *Skeleton Root* from the viewpoint and is used to select the appropriate levels of detail. The *priority* field is a normalized value which determines the level of detail selection criteria, allowing for an increase of the quality with which those actors, that are performing a particularly important activity for the user, are represented.

5.1. Geometric level of detail management

When managing the geometric level of detail of a virtual actor, it might be thought that it would suffice to have three or four different models of the same actor, and for each change from one level of detail to the next to take place simultaneously in all parts of the actor's body. However, this approach does not take into account that a virtual actor is made up of different geometries, not all of which are equally important in visual terms: in order to represent a human actor's head, it may be appropriate to use five different levels of detail, whereas the forearm could be correctly represented by using only three. In keeping with this, the distances at which transition between levels takes place should also be different.

Besides, the use of traditional LOD nodes may entail excessive computational cost. This is because each LOD node requires the calculation of the distance between its central point and the viewpoint in each frame. There are two drawbacks to this: on the one hand, it requires knowledge of the absolute position of the LOD's central point, which forces operations with transformation matrices to be performed in the CPU, and on the other, it entails a high computational cost of the distance calculus operation itself.

Both these drawbacks can be avoided by observing that the central points of all the LOD nodes that depend on the same virtual actor always remain close to each other. Therefore, if the distance between the viewpoint and the central point of the LOD node corresponding to the head of a human actor is 50 m, the distance between the central point of the LOD node which corresponds to the forearm would be within a range of 50 ± 1 m. The solution to the problem lies in performing a single distance calculus for each virtual actor and having all the LOD nodes that depend hierarchically on him or her employ it. In order to apply this mechanism, a slight modification of the modus operandi of the LOD nodes is required, for which purpose we have defined an auxiliary data structure called ***vaLod***, which replaces traditional LOD nodes. The *vaLod* structure has a pointer to the actor node on which it depends, which allows it to use the value of the distance from the camera stored in the *eyeDistance* field of the *vaActor* node. This distance will be used by all the LOD nodes that depend hierarchically on this virtual actor, thus finishing with the need to perform multiple distance calculations for each actor and, above all, with the need to perform operations with matrices in the CPU.

5.2. Skeletal level of detail management

A virtual actor is usually constituted by a considerable number of Skeleton nodes. However, the effect of some of these Skeleton nodes on actors that are far away from the camera is almost negligible and consequently, managing them is a waste of calculation resources. Therefore, a method which allows the reduction of the topological complexity of an actor as the latter's

distance from the camera increases is required. The solution proposed in this paper is to provide each Skeleton node with a distance beyond which it ceases to be processed, and therefore its degrees of freedom will not be evaluated, its matrix will not be generated, and the subgraph depending will not be traversed.

The need to perform this type of management is obvious if we consider the example of the hands of a humanoid actor: the definition of the articulated structure of a hand involves the use of 16 Skeleton nodes (three for each finger and an additional one for the wrist joint). However, it makes no sense whatsoever to perform any type of management with the fingers of an actor which is 500 m away from the camera. Beyond a certain distance, the different distances that define the hand and fingers have to be automatically replaced by a single geometry which represents the entire hand, and the different Skeleton nodes which represent the articulation points of the phalanxes of the fingers have to be deactivated.

Of course, there must be coordination between the deactivation distances of an actor's Skeleton nodes and the transition distances of the *vaLod* nodes which manage the level of detail of their geometry. This is shown in Fig. 5, where a scenegraph containing a very simple humanoid actor is represented.

Three skeletal levels of details are allowed: the first one, formed by nine Skeleton nodes, the second, formed by four and the third one, formed by none. The nodes containing geometry are represented by human figures with the enclosed geometry colored. The different types of lines show what parts of the scenegraph are processed in every level of detail. For example, for LOD level 1, there is a separate representation of head and body, but in LOD level 2, they share the same geometry. Therefore, it would be necessary that the deactivation distance of the head Skeleton node be the same as the distance used to change from geometric LOD level 1 to 2. Beyond that distance the Skeleton node will not be processed. This will reduce the number of Skeleton nodes to be processed, but also it will affect the number of vertices to be rendered as well as the behavioral management, as there is no need to calculate anything related to a deactivated articulation point.

5.3. Behavioral level of detail management

In the case of a simulation application involving virtual actors, it is common to dedicate more than half of the CPU's resources to the behavioral management of the actors (it must be kept in mind that complex behaviors that incorporate reverse kinematic calculations, artificial vision mechanisms, voice synthesis, etc., may be being managed). When a virtual actor is far away, the ability to appreciate the details of his or her geometry is lost, and likewise the ability to perceive the details of his or her behavior. The level of behavioral detail affects the precision of the mechanisms which manage the behavior of actors, reducing the computational cost as their distance from the viewpoint increases.

Let us delve deeper into this type of level of behavioral detail management by considering the example of the control mechanism of a human actor's visual activity. The mechanism responsible for an actor's visual activity has to control not only the movement of the eyes, but also that of the neck and even the spinal column. Let us suppose that the deactivation distance for the Skeleton nodes of the eyes is 50 m, and that the deactivation distance for the neck is 500 m. As seen in the previous section, this would require that the *vaLod* node that controls the geometry of the head to have a model that becomes activated at a range of 50 m, and the eyes to be incorporated in the geometry of the head itself.

Likewise, the LOD which controls the geometry of the thorax would have to have a model that becomes activated at a range of 500 m, in which the geometry of the thorax would incorporate a representation of the head. Such topological changes produced, on a virtual actor due to the deactivation distances of the Skeleton nodes, directly affect behavioral management mechanisms. Thus, the visual activity management model can be deactivated at distances superior to 500 m (the deactivation distance for the neck's Skeleton node), and it would make no sense for it to perform calculations regarding the orientation of eyesight for distances superior to 50 m. Changes in the topology of the actor define the minimum number of behavioral detail levels; however, the number of these may be superior to those imposed by the disappearance distances of the Skeleton nodes. In the case of visual activity mechanisms, the deactivation distances of the neck and eyes requires the existence of LOD change points at 50 and 500 m, but there may be additional ones, the aim of which is to ensure better control over the precision and speed of calculations. These additional behavioral levels could be based, in our vision management example, on different inverse kinematics or approximation methods.

As mentioned before, the distance used to select the proper level of detail is stored in the Actor node. The management of behavioral

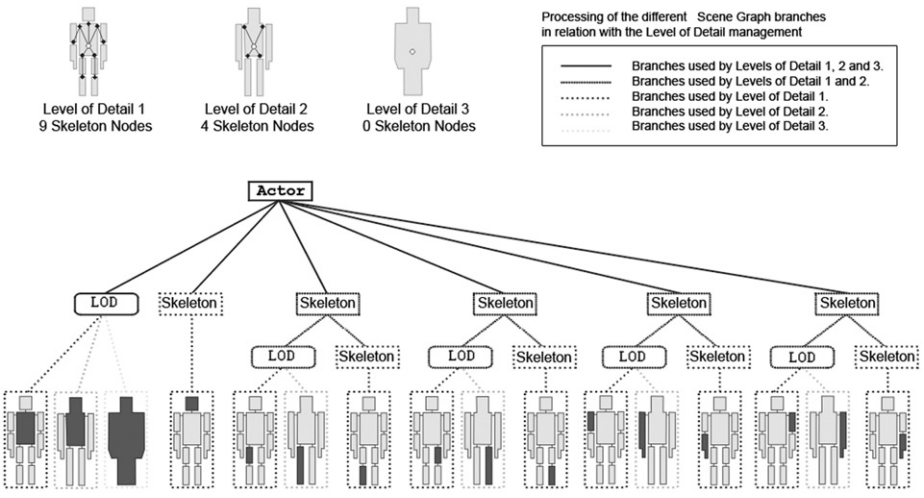

Fig. 5. Combining geometric and skeletal LODs.

levels of detail is essential for developing an application in which there are several virtual actors with average behavioral complexity.

6. Estimate of computational improvement

In this section we will analyze the computational improvement produced by the structures and methods proposed in the preceding sections.

Many matrix multiplications have to be implemented in simulation applications containing multiple virtual actors; these affect the speed of the simulation considerably and sometimes cause bottlenecks. This limitation can be considerably curtailed if efficient use is made of the capacity of graphic cards to perform 4×4 matrix multiplications. The main drawback of these capacities is that the result of the multiplications remains within the graphics hardware, and present day scenegraph management methods require access to these matrices: culling calculations (knowledge of the absolute position of the nodes' bounding spheres in regard to the view frustum), LOD calculations (knowledge of the reference point of the LOD is required to calculate its distance in relation to the viewpoint) and specific needs of the application, such as detection of collisions between objects, inverse kinematic calculations, etc.

The methods presented in the previous sections not only decrease the number of calculations but also allow better use of the capacities of present-day graphics hardware. This is due to the fact that they allow a great deal of the processing of virtual actors to be implemented without the CPU having to know the result of the multiplication of matrices. This makes the use of graphics hardware for this purpose possible, avoiding the characteristic bottlenecks associated with this type of operations.

In order to accurately estimate the cost of the operations, the unit of cost measurement, FMULT, is defined as the time invested to multiply two floating point numbers with the following equivalences:

1 mult = 1 FMULT,
1 add = 1 FMULT,
1 div = 5 FMULT,
1 sqrt = 7 FMULT,
1 multiplication of two 4×4 matrices in the graphic hardware = 10 FMULT.

Characterization of the different operations involved in the processing of virtual actors that take part in the scene is a necessary part of the estimation. The number of these operations and the cost of each one are described in Table 1. Obviously, in a real system they only represent one aspect of the performance but they allow us to quantify the improvement achieved by using the methods proposed.

Quantitative estimates of the computational improvement will be highlighted by using the "Global Improvement Factor", which is defined as the ratio between the Global Cost of Traditional Processing (**Trad_Proc**) of a scenegraph and the Global Cost of Actors' Processing (**Act_Proc**) following the methods proposed in this paper.

$$\textbf{GlobalImprovementFactor} = \frac{\textbf{GlobalCostTrad\_Proc}}{\textbf{GlobalCostAct\_Proc}} \quad (1)$$

In both terms of the ratio of expression **(1)**, the Global Cost is computed considering the Behavioral Management Cost and the Scene Graph Management Cost, as follows:

$$\textbf{GlobalCost} = \textbf{BehavioralManagementCost} + \textbf{SceneGraphManagementCost} \quad (2)$$

To obtain a quantitative estimation of these costs, a complex scene with 100 actors is considered. A complete description of the selected scene is given in Section 6.1, while Section 6.2 presents a comparison of the results of processing the selected scene with traditional methods and with the approach proposed in this work.

Table 1
Types of operations (number and cost) to be performed during virtual actor processing.

Notation		Description
NO_O1		Number of operations required to obtain the matrix of the articulation of a Skeleton Node
	CO_O1a	Cost of calculation of the values of the degrees of freedom (FMULT)
	CO_O1b	Cost of generation of the transformation matrix (FMULT)
NO_O2		Number of operations required to obtain the position/orientation matrices of the Reference Point
	CO_O2a	Cost of calculation of the values of the degrees of freedom (FMULT)
	CO_O2b	Cost of generation of the transformation matrix (FMULT)
NO_O3		Number of operations required to obtain the position/orientation matrices of the Skeleton Root
	CO_O3a	Cost of calculation of the values of the degrees of freedom (FMULT)
	CO_O3b	Cost of generation of the transformation matrix (FMULT)
NO_O4a		Number of operations required for the recalculation of the bsphere on the basis of the bspheres of its descendant nodes for culling processes in one node
NO_O4b		Number of operations required for the verification of the relation between the bsphere and the frustum for culling processes in one node
	CO_O4a	Cost of recalculation of the bsphere of the node on the basis of the bspheres of its descendant nodes (FMULT)
	CO_O4b	Cost of verification of the relation between the bsphere and the frustum (FMULT)
NO_O5		Number of operations required to obtain the distances for level of detail management
	CO_O5	Cost of calculation to obtain the distances for level of detail management (FMULT)

6.1. Description of the example scene

In order to estimate the computational improvement produced by our approach, we propose the use of a specific complex scene made up of 100 actors with different characteristics. The variables and values for the example scene are shown in Tables 2 and 3. For more details see the Appendix.

Each of the actors is comprised of 40 Skeleton nodes, with an average of 2 DOFs per Skeleton node, and 41 LOD nodes, one for each Skeleton node and an additional one for the Actor node. We have realized that an average reduction of 50% in the number of skeleton nodes per actor is suitable for the example scene after applying our skeletal LOD management method (described in Section 5.2).

6.2. Computational improvement

The costs of the behavioral and the scenegraph management, necessary to solve the **Global Cost** of Eq. (2), must be calculated considering the number and the cost of the operations involved in each case, either for the **Trad_Proc** and for the **Act_Proc**.

The values for the standard selected scene have been studied separately, as follows.

Table 2
Description of the characteristics of the 100 actors that make up the example scene.

Description	Number
Total number of actors **(Total Actors=A+B+C)**	100
A Number of actors with *Refpoint Bspheres* totally outside the frustum	20
B Number of actors with *Refpoint Bspheres* within frustum	30
C Number of actors with *Refpoint Bspheres* intersecting with frustum edge **(C=C1+C2+C3)**	50
C1 Number of actors with *Sklsroot Bspheres* totally outside the frustum	20
C2 Number of actors with *Sklsroot Bspheres* within frustum	20
C3 Number of actors with *Sklsroot Bspheres* intersecting with frustum edge **(C3=C3.1+C3.2)**	10
C3.1 Number of actors with *Internal Bspheres* totally outside or totally within the frustum	8
C3.2 Number of actors which need culling with *Internal Bspheres*	2

Table 3
Description of the characteristics of each actor in the example scene.

Description	Number
Number of Skeleton nodes per actor	40
Number of *Internal Bspheres* per actor	80

6.2.1. Behavioral management cost (BMC)

In order to obtain the Behavioral Management Cost only the computation of the values related with the degrees of freedom must be taken into account (see Table 1), so that BMC=NO_O1 × CO_O1a+NO_O2 × CO_O2a+NO_O3 × CO_O3a

The values of the number of operations and the cost of each operation can be found in the Appendix.

If we also suppose that the cost average (called **AverageCost**) of the calculation of the values of the degree of freedom is the same for [CO_O1a, CO_O2a, CO_O3a] in the case of traditional processing, and half (**AverageCost/2**) for CO_O1a and CO_O3a due to the average reduction of the number of Skeleton nodes of an actor by LOD in the case of actor processing (behavioral level of detail management), then the resulting expressions for the cost of behavioral management in the standard selected scene are:

$$\textbf{Trad\_ProcBMC} = 4.200 \times \text{AverageCost} \quad \text{(expressed in FMULT)}$$

$$\textbf{Act\_ProcBMC} = 315 \times \text{AverageCost} \quad \text{(expressed in FMULT)}$$

6.2.2. Scene graph management cost (SGMC)

The Scene Graph Management Cost must take into account: the generation of the transformation matrix operations, the culling operations and the calculation for obtaining the distances for the level of detail management. Because the configuration of the bodies changes very little from one time step to the next in the scenegraph management, the temporal Coherence factor and the Tracking factor have also been taken into account:

- The temporal Coherence factor (**fCoh**) is the value resulting from substracting from 1 the ratio between the time that one of the matrices remains unaltered and the total simulation time expressed as a fraction of 1. In other words, the coherence factor is 1 if the matrix of each frame is updated and 0 if it is never modified.
- The Tracking factor (**fTrack**) is the ratio between the number of matrix multiplications not performed through hardware in relation to the total of matrix multiplications required. In other words, the tracking factor is 0 if all matrix multiplications are performed through hardware and 1 if they are performed outside the graphic hardware.

Therefore, considering the values of the number of operations (see Appendix) and the cost of each operation (see Appendix), the final cost of scenegraph management will be

$$\textbf{Trad\_ProcSGMC} = 4.206.240 \text{ (expressed in FMULT)}$$

$$\textbf{Act\_ProcSGMC} = 400 \times \text{CO\_O1b} + 100 \times \text{CO\_O2b} + 30 \times \text{CO\_O3b} + 23.100$$

which, if Coherence and Tracking factors are taken into account, can be expressed as

$$\textbf{Act\_ProcSGMC} = 11.600 \times \textbf{fTrack} \times \textbf{fCoh} + 21.200 \times \textbf{fTrack} + 12.350 \times \textbf{fCoh} + 35290 \text{ (expressed in FMULT)}$$

6.2.3. Global Cost

In order to obtain the result of the **GlobalCost** from expression (2), the previously calculated **Trad_Proc** and **Act_Proc** of both **BehavioralManagementCost** and **SceneGraphManagementCost** must be taken into account.

Hence, the global cost of managing the actors of the standard scene according to the traditional management method can be expressed as

$$\textbf{GlobalCostTrad\_Proc} = \textbf{Trad\_ProcBMC} + \textbf{Trad\_ProcSGMC}$$

$$\textbf{GlobalCostTrad\_Proc} = 4200 \times \textbf{AverageCost} + 4.206.240 \text{ (expressed in FMULT)}$$

And the global cost of actor management according to the methods proposed in this paper can be expressed as

$$\textbf{GlobalCostAct\_Proc} = \textbf{Act\_ProcBMC} + \textbf{Act\_ProcSGMC}$$

$$\textbf{GlobalCostAct\_Proc} = \textbf{315} \times \textbf{AverageCost} + 11.600 \times \textbf{fTrack} \times \textbf{fCoh} + 21.200 \times \textbf{fTrack} + 12.350 \times \textbf{fCoh} + 35290 \text{ (expressed in FMULT)}$$

Therefore, the global improvement factor of expression (1) can be expressed as

$$\textbf{GlobalImprovementFactor} = \frac{\textbf{4.200} \times \textbf{AverageCost} + \textbf{4.206.240}}{\textbf{315} \times \textbf{AverageCost} + \textbf{Act\_ProcSGMC}} \quad \text{(expressed in FMULT)} \tag{3}$$

The graphic representation of expression **(3)** is shown in Fig. 6, taking into account different possible values for the Tracking and Coherence factors. For common behavioral management values, with **AverageCost** < 100 FMULTs, the improvement factor ranges between 50 and 120, and in the worst cases, for unusually high behavior management, the improvement factor would be 13. Details of the calculation can be found in the Appendix.

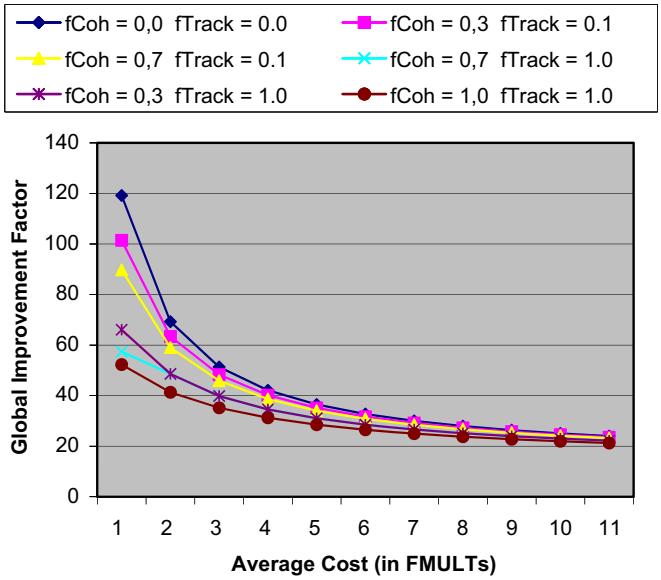

Fig. 6. Global improvement factor according to the average cost of one skeleton node behavioral management, considering the values of the tracking and the coherence factors.

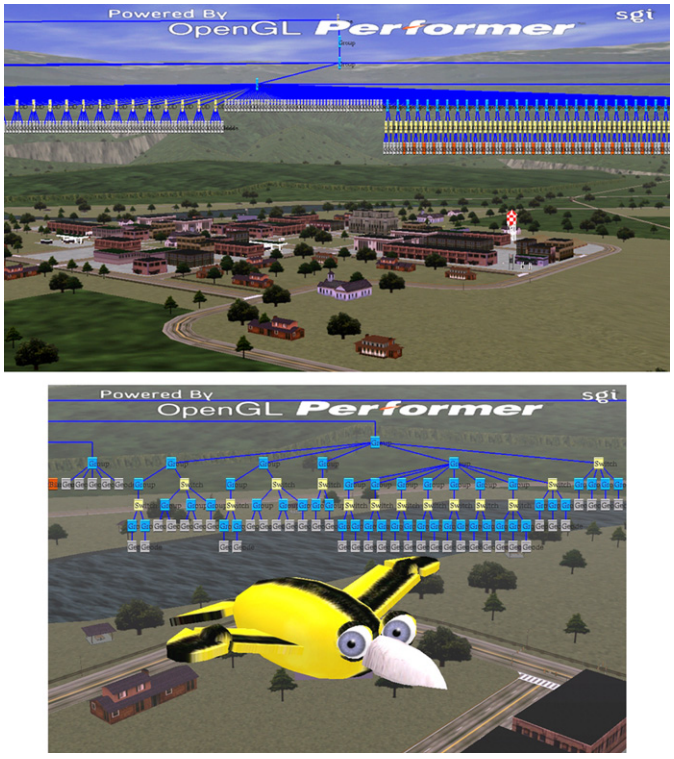

Fig. 7. Sideview of the "*Performer Town*" scene and its scenegraph (above). Representation of a virtual actor and its scenegraph (below).

7. Application example

7.1. Implementation in performer

In order to demonstrate the practical usefulness of the results of this paper, a library equipped with 134 functions that allows access to the structures and methods described has been developed. The library, written with an API in C, allows access to the 10 classes used in the model of objects; it has been used to create an extended

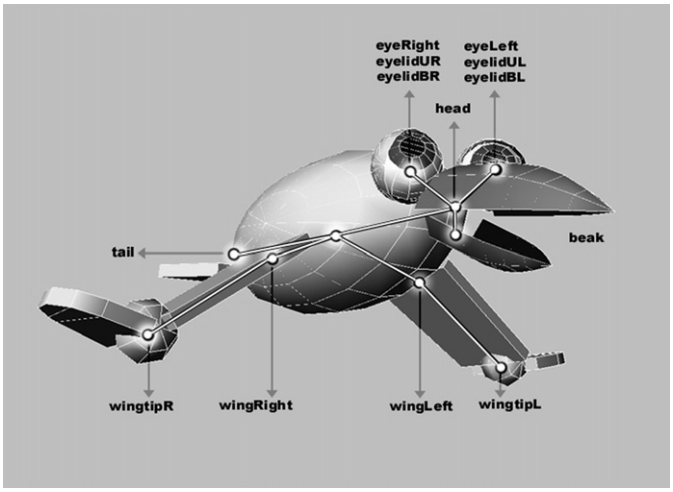

Fig. 8. Actor's topological structure.

version of the "perfly" application of OpenGL Performer [23], which incorporates and allows the management of multiple virtual actors.

The simulation scene in which virtual actors have been integrated was the model of a small town and its surroundings, provided as an example of the use of the OpenGL Performer library. This database, known as "Performer Town", is shown above, in Fig. 7, where a fragment of the scene graph is also represented. The virtual actors perform their activities in the sky of the aforementioned town, and their sub-scenegraphs are integrated in the global scenegraph (Fig. 7 below). For details on the integration of nodes in Performer, see [1].

A method for quick cloning of actors has been developed; there is no need to make copies of the nodes with geometrical information nodes, as the actors share the same geometry nodes. In order to make the method a little more sophisticated, we added the possibility of each actor having a different texture to this basic pattern. This is achieved by using callback routines assigned to the Actor nodes, which are responsible for applying the texture with which their geometries will be designed. Regardless, the behavior of each actor is completely autonomous and each of them may have its own personality. Each actor consists of 13 articulation points (see Fig. 8) and 19 degrees of freedom, and it supports 5 levels of detail. The implemented control mechanisms are: flight control, gaze control, facial expression control, blinking control and phrase control.

In Fig. 9, we see a scene in which several virtual actors are located at different distances from the camera and therefore have different levels of detail. The fact that virtual actors that are further away have fewer polygons is not easily appreciable at first sight: distances at which levels of detail change are adjusted so that the numerical loss of triangles does not affect image quality. Management of level of detail also affects the topology of actors, reducing the number of articulation points used according to distance and behavioral management, as well as simplifying calculations or even blocking certain mechanisms.

Fig. 10 shows how the culling of virtual actors operates in parallel with standard scenegraph culling. The control editor of the culling pyramid supplied by the "perfly" application was used to make the culling pyramid smaller than the vision frustum. This way we can see if culling is taking place appropriately. Two images are shown: the first one contains the original scene and the second one shows the scene with forced culling. The rectangle of the second image indicates the margins of the culling pyramid; we can see that certain actors are not drawn, and the same thing has happened with certain objects within "Performer Town".

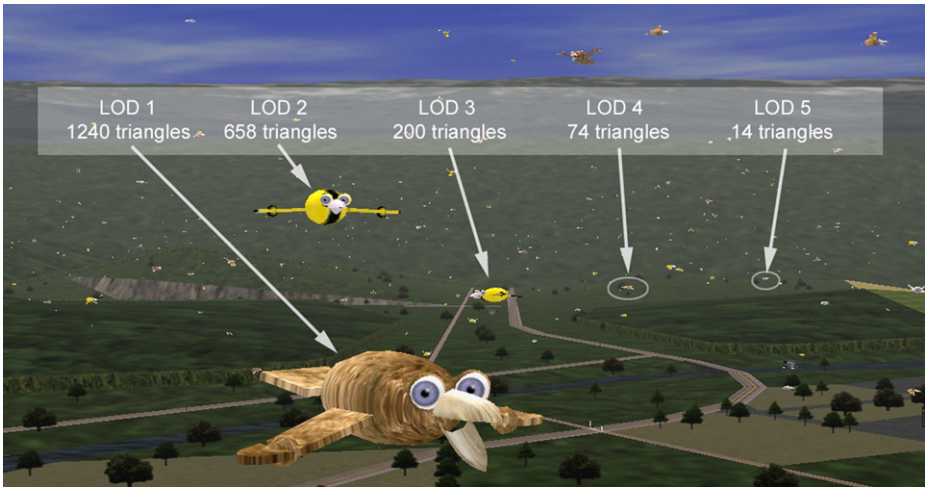

Fig. 9. Automatic selection of the level of detail according to distance.

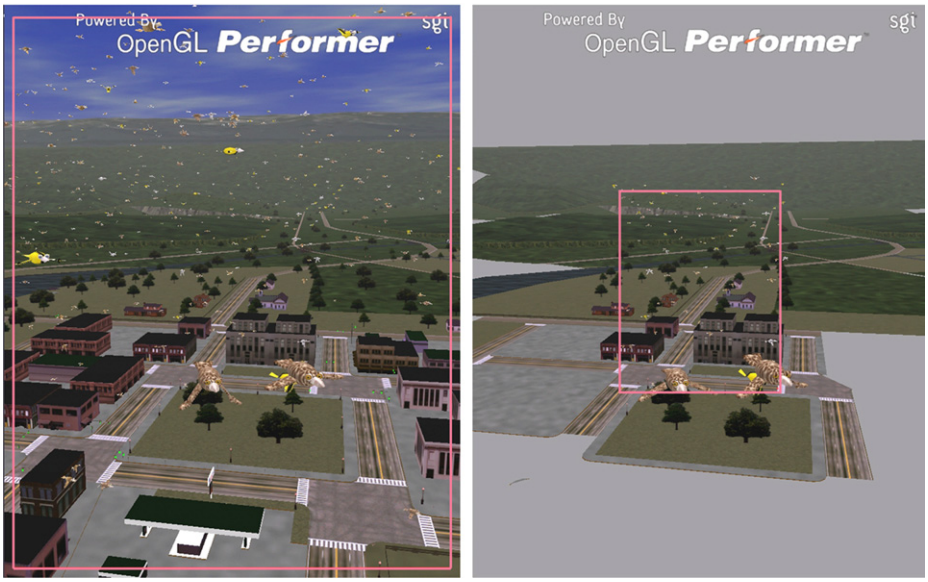

Fig. 10. Integration between actor culling and scenegraph culling.

7.2. Performance

The computer used was a PC based on a CPU Intel P4 at 3 GHz, as well as a graphic card based on an Nvidia FX5600 processor. The operating system was the 7.2 version of Linux RetHat and the library was the 2.5.2 version of *OpenGL Performer*. The resolution of the graphic window used for the simulations was 1024×768 pixels.

Regarding the performance of the system, we have proven that—using a domestic PC—it is possible to integrate 2000 actors in the "perfly+Performer Town" application with a refreshment rate of 50 images per second, or 6000 actors with a refreshment rate of 25 images per second (see Fig. 11).

These results prove not only the computational efficiency of the system, but also its capacity for integration in an existing scenegraph and its adequacy for the implementation of motor and behavioral models, as well as its potential for extension. Unfortunately, comparing these results with previous techniques is not easy, since not much research on the integration of virtual actors in simulation applications has been carried out. There are many real developments in videogame engines [29], but all of them are very vertical solutions, and have not been structured to allow any kind of generic articulated actor, neither their integration in existing scenegraphs.

8. Conclusions and future research

This paper focuses on the adaptation of several aspects of scenegraph processing to real-time applications that deal with multiple articulated virtual actors. In particular, a bounding volume hierarchy approach to efficient culling and LOD has been proposed. The main idea behind this work is the integration of geometry, connectivity and behavior to accelerate 3D rendering. Moreover, the methods proposed are highly applicable to the rendering of large game-scapes, crowd scapes and behaviorally realistic populated ecological landscapes.

This approach has several clear advantages in relation to other works:

- It shows the importance of managing the LOD at a topological and behavioral level instead of just focusing on the traditional geometric level of detail. In fact, it operates in an integrated fashion with the three types of LODs.
- Unlike the particular structures normally used in videogame motors [29], the proposed structure is generic and may be used to represent any kind of articulated actor, regardless of whether it has a human shape or not.

Fig. 11. Simulation scene with 2000 (above) and 6000 virtual actors (below).

- The structure is designed to be integrated in present day scenegraphs, such as Performer or OpenSceneGraph.
- The computational improvement introduced by the use of the described structures and methods has been theoretically assessed. The analysis shows that the actors processing is between 50 and 120 times faster than with the use of a traditional scenegraph.

However, the proposed system has certain limitations:

- The structure was designed for a fixed pipeline. Nowadays, with GLSL Shaders [30], alternative methods could be developed.
- The model of an actor based in rigid interconnected geometries works well at medium or far distances, but when the actor is very close it is necessary to work with skinning based models that support facial animation, hair simulation, etc. It should be noticed that all these techniques are very time consuming and, therefore, difficult to be included in the simulation of scenes with hundreds of actors. This is why we have not considered them for the moment.

Skinning is, therefore, one of the features to be added in the future. Also, presently, the system includes three different levels of bounding spheres, but to expand them would be a simple matter. In particular, it would be interesting to include a new type of bounding sphere for a group of actors suitable for crowd simulations.

Other possible lines of future research might be the design of configuration modules of the actor's library for scenegraphs other than OpenGL Performer (for instance VTK, OpenSG, osg, etc.) and the search for a method to add semantics to the scenegraph and facilitates the perception of their surroundings by the autonomous virtual actors. A utility program capable of acting as a testing ground for the behavior of virtual actors in a phase prior to their integration in the final application will also be very useful.

Acknowledgments

The authors would like to thank the reviewers for their suggestions that have contributed to the improvement of the content of this paper.

This work was partially supported by the Spanish "Dirección General de Investigación" No. TIN2007-63025 and by the Aragon Government through the IAF No. 2008/0574 and the CyT No. 2008/0486 agreements.

Appendix

This appendix details the notation and the data used for the calculation of the global improvement factor of the test scene. For complete details, see [28].

All the factors that determine the number of operations required for processing the scene are described in Table 4.

Table 5 presents an estimation of the number of elemental operations needed for processing a scene with a traditional method (Traditional Processing) and with the method proposed in this paper (Actor Processing). Columns 2 and 3 present the qualitative estimation for a generic scene while columns 4 and 5 show the quantitative values for the scene selected as an example.

It can be seen in Table 5 that the Actor Processing considerably reduces the number of operations required in each phase since the *fRefpointBsphsIn*, *fSklsrootBsphsOut*, *fSklsrootBsphsIn*, *fSklsrootBsphsIsect*, *fCheckIntBsphs* and *fLODtopo* factors all have values lower than 1. Also, in the computation of NO_O4b the value of *nBsphsPerActor* is divided by 2 since the *culling* check is done in a hierarchical way.

Table 6 presents the computational cost of each elemental operation for the Traditional Processing and for the Actor Processing

Table 4
Factors that determine the number of operations required to process the scene.

Factor	Description	Number	Ratio
nActors	Number of actors	100	
fRefpointBshsIn	Actors with *Refpoint Bsphere* within frustum	30	0.3
fSklsrootBsphsOut	Actors with *Sklsroot Bsphere* totally outside the frustum	20	0.2
fSklsrootBsphsIn	Actors with *Sklsroot Bsphere* within frustum	20	0.2
fSklsrootBsphsIsect	Actors with *Sklsroot Bsphere* intersecting the frustum edge (*Internal Bshperes* are totally outside or within the frustum)	8	0.08
fCheckIntBsphs	Actors which need culling with *Internal Bspheres*	2	0.02
nSkelsPerActor	Number of Skeleton nodes per actor	40	
nBsphsPerActor	Number of *Internal Bspheres* per actor	80	
fLODtopo	Estimated average reduction of number of Skeleton nodes per actor after applying our skeletal LOD management method	50%	0.5

Table 5
Number of elemental operations involved in the management of the scene, according to the type of processing: Traditional Processing (Trad) and Actor Processing (Actor).

Notation	Qualitative data		Quantitative data	
	Traditional processing	Actor processing	Trad	Actor
NO_O1	nActors x nSkelsPerActor	(fSklsrootBsphsIn × nActors) × (nSkelsPerActor × fLODtopo)	4000	400
NO_O2	nActors	nActors	100	100
NO_O3	nActors	nActors × fRefpointBsphsIn	100	30
NO_O4a	nActors x (1+1+nBsphsPerActor)	nActors × CheckIntBsphs × nBsphsPerActor	8200	160
NO_O4b	nActors x (1+fSklsrootBsphsOut+fSklsrootBsphsIn +fSklsrootBsphsIsect × (nBsphsPerActor/2))	nActors × (1+fSklsrootBsphsOut+fSklsrootBsphsIn +fCheckIntBsphs × (nBsphsPerActor/2))	460	200
NO_O5	nActors × (1+nSkelsPerActor)	nActors	4100	100

Table 6
Costs of each elemental operation for Traditional and Actor Processing (expressed in FMULT).

Notation	Traditional processing cost	Actor processing cost
CO_O1a	AverageCost	AverageCost/2
CO_O1b	908 FMULT	fTrack∗fCoh∗29+fTrack∗53+fCoh∗19+10 (1 node Skeleton with 2 DOF)
CO_O2a	AverageCost	AverageCost
CO_O2b	454 FMULT	fCoh∗22+63
CO_O3a	AverageCost	AverageCost/2
CO_O3b	566 FMULT	fCoh∗85+63
CO_O4a	50 FMULT	50 FMULT
CO_O4b	64 FMULT	64 FMULT
O5	8 FMULT	8 FMULT

(expressed in FMULT). The costs of generation of the transformation matrix (CO_O1b, CO_O2b, CO_O3b) are considerably reduced in the case of Actor Processing since they can take advantage of tracking and temporal coherence.

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version at doi:10.1016/j.cag.2010.07.006.

References

[1] Seron FJ, Rodriguez R, Cerezo E, Pina. A. Adding support for high-level skeletal animation. IEEE Transactions on Visualization and Computer Graphics 2002;8(4):360–72.
[2] Cohen-Or D, Chrysanthou YL, Silva CT, Durand. F. A survey of visibility for walkthrough applications. IEEE Transactions on Visualization and Computer Graphics 2003;9(3):412–31.
[3] Möller T, Haines E, Hoffman. N. Real time rendering. 3rd ed. A. K. Peters Ltd.; 2008.
[4] Sudarsky O, Gotsman. C. Dynamic scene occlusion culling. IEEE Transactions on Visualization and Computer Graphics 1999;5(1):13–29.
[5] Chenney S, Arikan O, Forsyth D Proxy simulations for efficient dynamics. In: Proceedings of Eurographics, Short Presentations, 2001.
[6] Yoon S., Salomon B., Manocha D. Interactive view-dependent rendering with conservative occlusion culling in complex environments. In: Proceedings of the 14th IEEE Visualization 2003 (VIS'03), 2003. p. 163–70.
[7] Tecchia F, Loscos C, Chrysanthou Y. Visualizing crowds in real-time. Computer Graphics Forum 2002;21(4):753–65.
[8] Aubel A, Boulic R, Thalmann D, Animated impostors for real-time display of numerous virtual humans, Lecture Notes in Computer Science, vol. 1434. Proceedings of the first international conference on virtual worlds, 1998. p. 14–28.
[9] Ulicny B, de Heras Ciechomski P, Thalmann D Crowdbrush: interactive authoring of real-time crowd Scenes. In: Proceedings of the ACM SIGGRAPH symposium on computer animation, 2004.
[10] De Heras Ciechomski P., Ulicny B, Cetre R, Thalmann.D A case study of a virtual audience in a reconstruction of an ancient roman odeon in aphrodisiac. In: Proceedings of the fifth international symposium on virtual reality, archaeology and cultural heritage (VAST), 2004.
[11] Luebke D, Reddy M, Cohen J, Varshney A, Watson B, Huebner. R. Level of detail for 3D graphics. Morgan Kaufmann; 2002.
[12] Ahn J, Wohn K. Motion level-of-detail: a simplification method on crowd scene. Computer Animation and Social Agents (CASA 2004) 2004:129–37. 7.
[13] Pratt DR, Pratt SM, Barham Paul T, Barker RE, Waldrop MS, Ehlert JF, Chrislip CA. Humans in large-scale, networked virtual environments. Presence 1997;6(5):547–64.
[14] Dobbyn S, Hamill J, O'Conor K, O'Sullivan C. Geopostors: a real-time geometry/impostor crowd rendering system. In: Proceedings of the 2005 symposium on interactive 3D graphics and games. ACM, 2005. p. 95–102.
[15] Cozot R, Multon F, Valton B, Arnaldi B, Animation levels of detail design for real-time virtual human. In: Proceedings of the Eurographics workshop on animation and simulation '99, 1999. p. 35–44.
[16] Carlson DA, Hodgins JK, Simulation levels of detail for real-time animation. In: Proceedings of the Graphics Interface '97, 1997. p. 1–8.
[17] Brogan D, Hodgins J, Simulation level of detail for multiagent control. In: Proceedings of the international joint conference on autonomous agents and multiagent systems (AAMAS), 2002. p. 199–206.
[18] O'Sullivan C, Cassell J, Vilhjalmsson H, Dingliana J, Dobbyn S, McNamee B, Peters C, Giang T. Levels of detail for crowds and groups. Computer Graphics Forum 2002;21(4):733–42.
[19] Horswill ID, Zubek R Robot architectures for believable game agents. In: Proceedings of the 1000 AAAI Spring Symposium on Artificial Intelligence and Computer Games. AAAI Technical Report SS-99-02, 1999.
[20] De Heras Ciechomski P, Thalmann D Populating virtual environments with crowds: rendering pipeline optimizations. In: EG 2006 course on populating virtual environments with crowds, 2006.
[21] Humanoid Animation Working Group, 〈http://www.h-anim.org/〉.
[22] Koenen R., editor. Overview of the MPEG-4 standard, 2002. 〈http://mpeg.chiariglione.org/standards/mpeg-4/mpeg-4.htm〉.
[23] Performer: 〈http://www.sgi.com/products/software/performer/〉.
[24] Open Scene Graph, 〈http://www.openscenegraph.org/〉.
[25] SNHC Face/Body Ad Hoc Group. Face and body definition and animation parameters. Document No. MPEG96/N1365, Chicago Meeting of ISO/IEC JTC1/SC29/WG11, Octubre 1996.
[26] Bar-Zeev A Scenegraphs: past, present and future, 〈http://www.realityprime.com/articles/scenegraphs-past-present-and-future〉.
[27] Kessler GD. Virtual environments models. In: Stanney K, editor. Handbook of virtual environments. Lawrence Erlbaum Associates, CRC Press; 2002.
[28] Rodríguez R. Ph.D. Thesis, Actores Sintéticos en Tiempo Real: Nuevas Estructuras de Datos y Métodos para su Integración en Aplicaciones de Simulación. Servei de Publicacions Universitat de Valencia, 2004 [in Spanish].
[29] Eberly D. 3D game engine design: a practical approach to real-time computer graphics. Morgan Kauffman.
[30] Rost Randi J. OpenGL shading language. 1st ed.. Pearson Education, Inc; 2004.