Reviving the Roman Odeon of Aphrodisias: Dynamic Animation and Variety Control of Crowds in Virtual Heritage

Pablo de Heras Ciechomski, Sébastien Schertenleib, Jonathan Maïm and Daniel Thalmann
EPFL, VRLab, CH-1015, Lausanne, Switzerland

Abstract. In this paper we propose a new method for rendering crowds of virtual humans with dynamically deformed skeletons with levels of detail using two simple caching schemes for animations and geometry. We show how the virtual heritage project ERATO pushed for these innovations as we did not find any off-the-shelf products for the purpose. We also show how to create a large variety in appearance using graphics algorithm techniques that run on graphics cards with a fixed function pipeline support like the one of OpenGL 1.1.

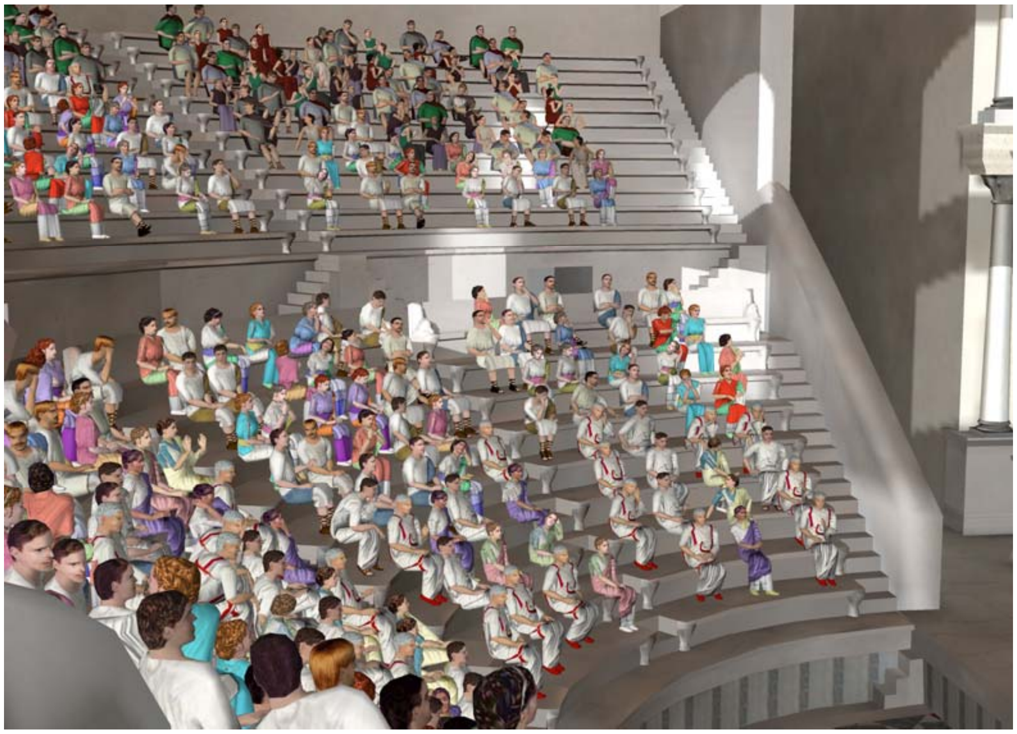
Figure 1: View of run-time engine

1. Introduction

In this paper we describe the improvements to a system that is used to visualize and simulate virtual crowds in a reconstruction of a Roman Odeon in Aphrodisias in Turkey. The work is part of the European project called ERATO [2] where we reconstruct an Odeon populated with a virtual crowd experiencing a theater play. With the help of archeologists, architects, musicians and historians we try together to give an experience as authentic as possible. The Odeon of Aphrodisias constructed in 2 AD, once the capital of Lydia in the Roman Empire, is located in the district of Karacasu south of Nazilli in southern Turkey. It is part of one of the most interesting archeological sites of the Aegean region. For more information on the Odeon see [5].

The purpose of our work is to provide a system architecture where a crowd of virtual humans can interact individually, helping researchers and historians to visualize and reconstruct heritage events. The virtual restitution is a combination of 3D virtual and acoustic environments. The work presented in this paper focus on the assigned task of reproducing an historical visualization of ancient heritage.

Some of the biggest challenges we are facing include providing tools for prototyping unique scenarios efficiently. Every such use case has its own constraints on data level or simulation events. We expect the system to be able to reconstruct any theater performance taking place in ancient theaters in different locations where rites and habits are different. To overcome the development costs involved in creating unique individual characters, we have to investigate alternative solutions. As shown in [1], we used a limited number of mesh templates from where we create different instances by changing their textures, animations, animation speeds and scaling properties as can be seen in Figure 1. By reusing the same geometry we reduce the system memory usage as well as the number of 3D models to be designed.

However, the variety is still limited by the number of textures available. Thus, we have extended this model by introducing color modifications on the different body parts which compose a virtual character using an original approach based on alpha map channels, which will be described in Section 3. The originality lies in how we choose to optimally render the triangles but not on the alpha channel approach in itself as it has been already used for billboards [10,11]. This allows creating an infinite variety of characters by changing their clothing colors. However, the related freedom in the color palette selection goes against the realism of their historical counterpart. Within the EU project ERATO [2], we were able to organize different panels with archaeologist and historians, which gave us invaluable information allowing to restrain the color for the different body parts based on historical context.

2 Prototyping

To elaborate the different scenarios which can feature more than a few hundred distinct virtual humans, we need to provide tools and scripting capabilities to unleash the potential for simulation designers. Our system offers different levels of interaction for off-line and on-line simulation adjustments. Therefore, our applications are able to parse scenarios configuration files written in the scripting language Python [3]. Most of our use-case scenarios were based on reconstructing life in ancient theaters. One of the most important differences with today’s theaters comes to the repartition of the audience seating on stone stairs. This provides more freedom to setup the crowds especially when we expect to place the virtual humans in a non uniform manner. To do so, we are taking benefit of the historical circular geometry architecture of our models. From a higher perspective, every layer of seats becomes a circular shape with different angles and radius. We have developed some routines to distribute the seats among those levels. To avoid that every character faces the center of the theater, we have introduced Perlin noise [4] in their facing direction, providing a more believable and realistic setup.

2.1 Managing the simulation workflow

Crowd simulations have highly intensive resources requirements. As we expect to provide good interactivity levels with the simulation, we must assure a high refresh frequency for the graphic component. As our simulation goes beyond crowd rendering, we need to extend the design architecture by separating the 3D rendering and the AI processing into two different flows of controls as described in Figure 2. By keeping theses two elements as distinct as possible, we are able to constrain the data synchronization issues coming with a multi-threading design scheme to a small subset of shared information. Recent development in CPU technology have expanded the install base of computers capable of running several hardware threads simultaneously, like the Intel Pentium IV processor with Hyper Threading technology [12] or multi-core architecture. Overall performance can be considerably improved by keeping every hardware thread active at all times. This is particularly true with I/O operations where the CPU is waiting for some acknowledgment like blocking OpenGL commands for instance. Since we are decoupling the graphics components and the AI, we can generate more complex behavior without sacrificing the overall level of interactivity.

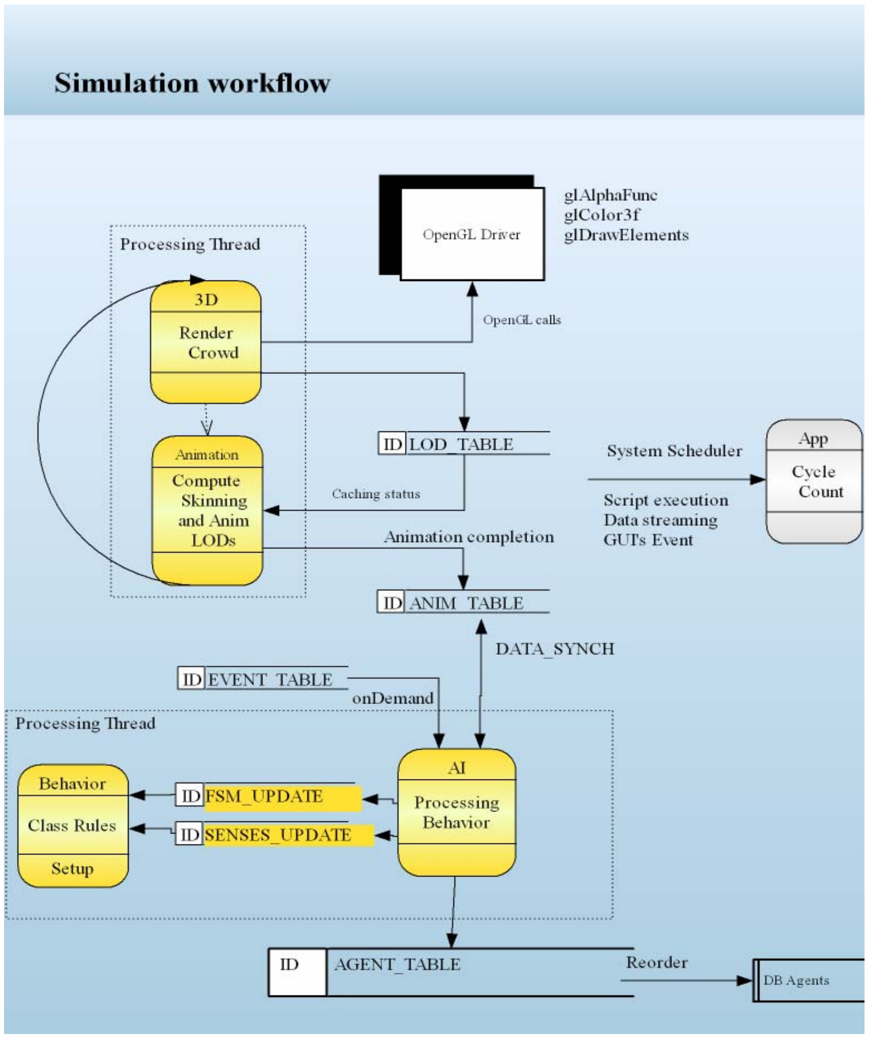

Figure 2 : Simulation workflow that shows the separation between rendering and artificial intelligence execution threads.

2.2 High-Level Optimizations

The crowd rendering is able to benefit from different level of details for each template. However, additional techniques taking advantages of the LOD selection have been elaborated. Based on this information, we are able to generate levels of detail for both the animation and the behavioral stages. Thus, we are updating the characters animation at distinct rates among their LODs. For instance, at the lowest LOD, we use an update rate of 4 fps. We are combining this asynchronous animation update with a caching system allowing us to reduce the number of skinning operations to be applied on a per frame basis.

Considering the AI processing, our approach is taking benefit of the fast computation of hierarchical finite state machines, which keep the current historical state of every agent. However, updating all those states at every frame would be inefficient and a waste of useful resources. Therefore, we have introduced a dedicated scheduler which classifies the agent's tasks into different priority groups. For instance, instead of checking on a frame basis if an animation is completed, the system will be only notified once. The addition of all thoses described optimizations has improved significantly the system responsiveness while allowing increasingly more details to the simulation. More information about the performance improvements will be discussed in the Results Section.

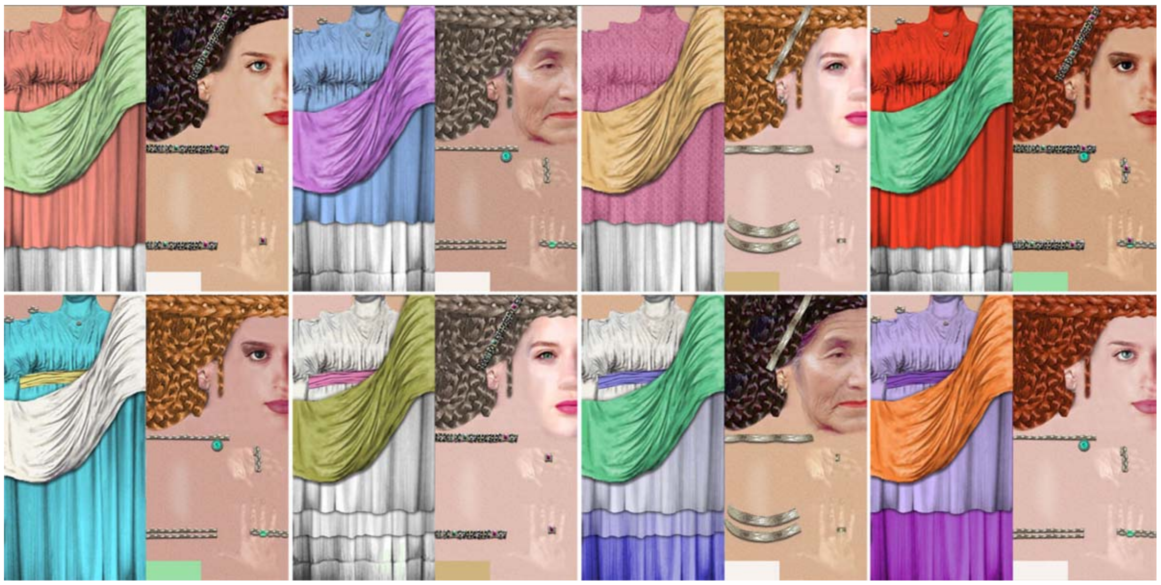
Figure 3 : Variety textures

3. Variety in Appearance

In the beginning of the project, the only variety we could have in appearance for each human was to create a completely new and specific texture. For example if we wanted to change the color of the dress, a new texture with the color of the dress was made as can be seen in Figure 3. We can notice that the dress stays the same but only parts of it change color and in our new approach we would like to color the belt in one way and the toga in another without creating a specific texture.

If we want to color the characters using one base texture that they all share, it should only carry the gray scale lighting information as can be seen in Figure 4 on the left. We can also note that the skin is preserving its color and that is due to the fact that we have no good way of modulating the skin color of our character as of yet, but it works fine for the garments. If we take a closer look at the problem we can find two cases where one is trivial to solve and the other one a bit more complicated. Let’s start with the trivial one which is a triangle that is fully within the belt region or toga region for example. Such a triangle would be one located close to the neck region for example. If we want to modulate the color of this triangle we simply tell OpenGL that we want another base color and we render it. How about the more problematic case? Let’s have a look at Figure 5.

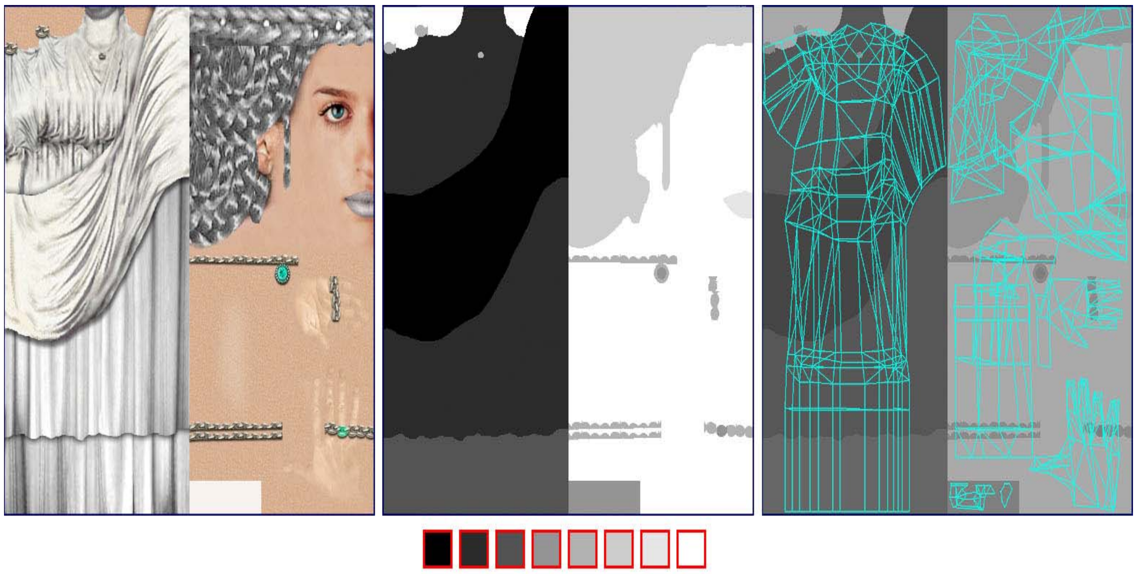

Figure 4: The base variety texture displayed along with the alpha map in the middle and the triangles laid out on the right. The 8 small squares beneath the alpha map show the alpha colors.

We have three regions a, b and c that need to be colored separately. One way to solve it is to relax the triangle into the simple problem, which means to subdivide it until we have enough triangles that cover the boundaries. This however goes against our wish to have low demands on triangle counts and will not work for low detail models. The solution is to draw the triangle three times and to make sure that only the part of the triangle that is within a specific region is drawn when we issue the variety command.

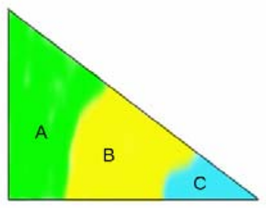

Figure 5: The problematic triangle

It is accomplished using the alpha channel and telling OpenGL to only draw the pixels having a specific alpha value of that part of the region, which is accomplished by using alpha function checks. In the worst case they have to be drawn as many times as there are alpha map areas, so it's better to handle the dirty triangles separately. A triangle is dirty if it covers more than one distinct alpha value which we check by rasterizing the triangle in software onto the alpha layer of the texture. We separate clean triangles into bins necessary for particular alpha keys. In Figure 4 we have 8 different such key maps so that amounts to 8 passes of normal GL rendering, where the skin is set to have white color variance (no change) and the last 8 passes are done for the dirty triangles using alpha tests. The result of the basic texture with different variety in colors is shown in Figure 6.

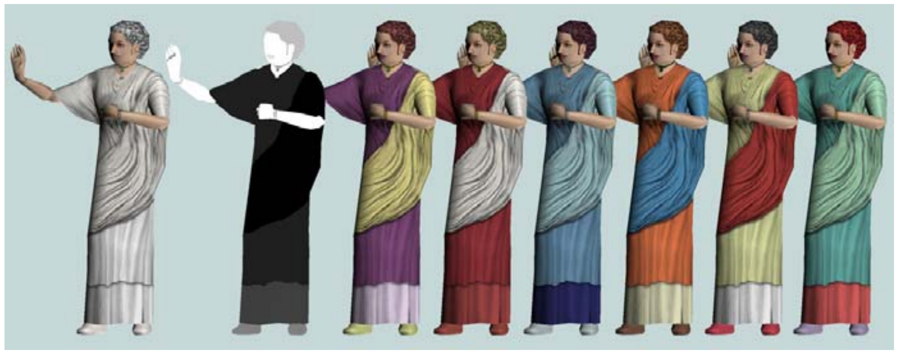
Figure 6 : Base variety texture (leftmost) and the alpha maps (next to it) and some combinations of colors.

4. Dynamic Animation and Geometry caching

Our partners in the ERATO project noticed the humans were limited in the number of animations they could perform, which was in fact due to the way the humans were rendered. In our previous approach full meshes were used as snapshots of the geometry which could then have a single texture applied [5]. When we had around 10 templates and each one had about 10 animations, we were already hitting the system memory limit of 2 GB per process under Windows or any similar 32-bit operative system. We had to find a way of creating more animations and not having to use such amounts of memory. To ameliorate this we took the basic skinning algorithm [14] used to compute the meshes and tweaked it to perform better, which resulted in a consumption of around 1000 times less memory. Since we do the deformation of vertices and normals in software this can be re-used in a caching scheme that would not be possible with a hardware-based solution. The reason for this cache is also that is gives us prohibitive update frequencies without it.

We can make the observation that the farther a virtual human is from the viewer the less detailed it has to be in geometry and in animation quality. Since we already have LODs we need the same kind of detail levels for animation. This is achieved using an animation cache and a geometric cache. A skeletal matrix palette is stored after each execution of the animation update, whose frequency is set per individual and increases with the closeness of the character to the viewer with a maximum update frequency of 50 Hz and a minimum of 4 Hz for far away individuals. The update time is linearly dependent of the squared distance of the camera.

The animation update also forces a geometrical update of vertex positions and normals which are stored in two separate arrays and are big enough to hold the maximum of vertices needed for the drawing calls, including extra space for duplicated vertices [5], which ensures that no extra copying has to be done to a specific rendering buffer. Updating the animation is independent of the geometrical update since characters can change between LOD areas with a greater frequency than the animation update when for example the camera sweeps over the crowd.

One could argue that a character could use the information from one LOD to another if the animation is not updated between such a shifts in LOD. A complete update can be avoided by copying the vertices that don't move from one LOD to another, where going from a more detailed LOD to a less detailed LOD could mean that no computation would be done. However since it happens rarely that a character moves from one LOD to another and does not update his animation in-between, this is regarded as a non-issue and not even considered for future work. From a caching perspective we are dealing with characters that constantly update anyhow and it would only make sense for immovable objects or objects with very slow animations.

6. Geometric Levels of Detail

When virtual humans are far away from the observer and map to very few pixels on the screen, it is possible to reduce the number of polygons rendered at each frame without decreasing the visual quality of the scene. To exploit this fact, we use geometrical LODs to hierarchically render simpler polygonal representations of our virtual humans [6, 7].

Progressive meshes [8] are a recent work that allows representing an arbitrary mesh as a continuous sequence of geometrical LOD approximations. This process uses edge collapses for simplification and vertex splitting to reverse the transformation. No new vertices are introduced that were not present in the original mesh. We use a subset of this work, along with a tool present in the ogre3d distribution [9]. The tool of ogre3d uses progressive meshes to build geometrical LODs and accepts as input three parameters: the number of desired geometrical levels, the depth increment from the observer and the polygon percentage reduction at each level. We adapt our file format to the ogre3d xml file format to create for each virtual human a bank of geometrical LODs. We finally import the resulting geometrical LODs in our system by converting back the ogre3d xml in our file format. The resulting geometrical LODs for a particular model are shown in Figure 7.

Some visual artefacts appear on the left arm of the model of Figure 7(b,c,d), and on the neck of Figure 7(c,d). These artefacts are due to the ogre3d progressive mesh implementation, which does not take enough into account the texture seams of the model (see Figure 4).

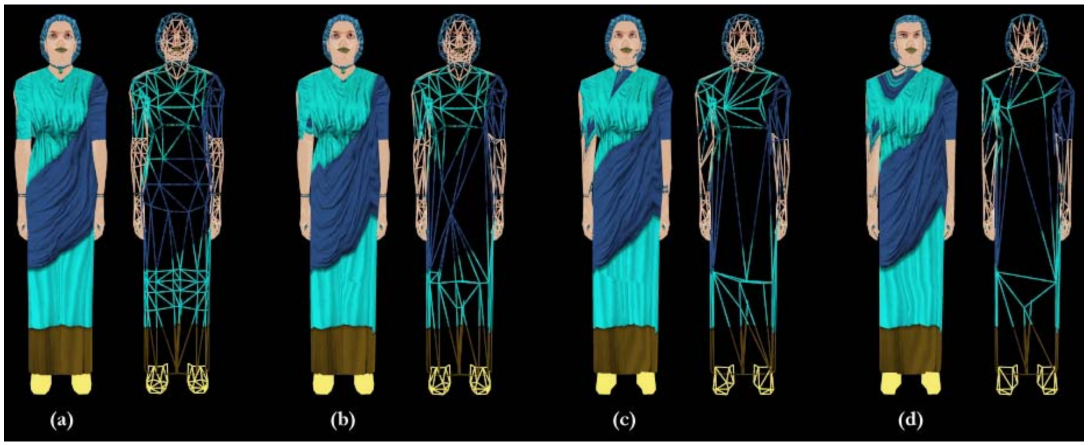

Figure 7: Different LODs for a model, from left to right : lod0 with 1042 tris (a), lod1 with 824 tris (b), lod2 with 652 tris (c), lod3 with 519 tris (d).

7 Results

The different techniques that we have developed have been profiled using a specific use-case scenario, helping us to understand the real-time performance improvements. All the different tests were done on a Intel Pentium IV running at 3.0Ghz with 1GB of main memory and an NVidia Geforce 6600 with 256MB of inboard DDR2 memory. We have developed a use-case scenario featuring a high detail 3D model of a roman theatre. It is filled with 340 instances of each three template humans. In order to reproduce similar condition for every setting, we have recorded a camera path. The path is divided in four different sections, each of them showing an average specific number of characters displayed on the screen.

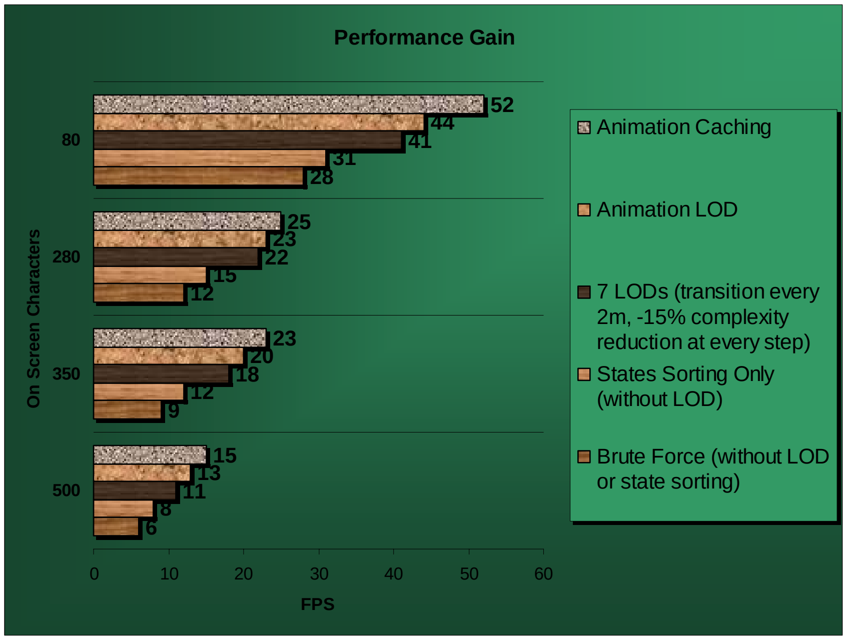

Figure 8: Performance gains from the different optimizations

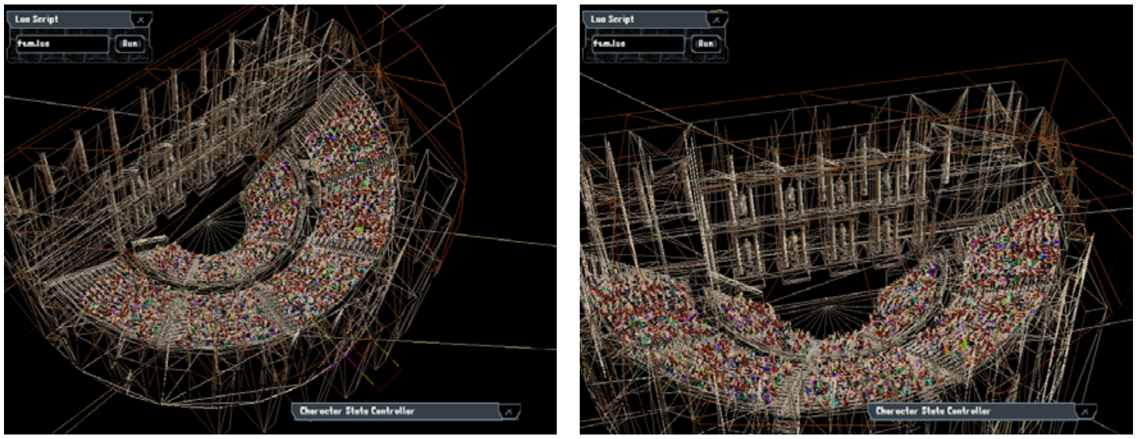

Figure 9 : Two views of the use case scenario.

From the graph in Figure 8 we can see how the caching techniques we have described are absolutely crucial with the use of LODs. When rendering 500 people caching amounts to almost 300% of the original speed. When we have fewer people in the theatre we see that the animation caching is still responsible for 200% of the original speed. These results speak well for further explorations into animation LODs.

8 Conclusions and Future Work

We have shown how dynamic skeletal animation can be accelerated using a simple caching scheme for animations and geometry. Using this technique each human can fully utilize our database of 150 animations without hitting any memory restriction problems. We have also shown how variety can applied using a simple alpha channel trick. What we can see though is that drawing performance is just barely acceptable for scenes with 500 humans with 15 fps minimum, when it should be 25 fps. Some animation restrictions can still be applied as well as with other restrictions for example for the variety layers and how many passes are applied for more far-away humans. The presented work have improved our crowd rendering framework as presented in [1], The benefits include more flexible and dynamic animations system as well as improved content creation pipeline for creating virtual characters. Variety in colors still needs to be ameliorated for artists to use it in a more relaxed manner such as a GUI application where they can immediately see the result. On our wish list is billboarding for larger crowds, shadows, per-pixel lighting, better LODs with texture seams fixed, better materials, animation transitioning, and further optimizations of animations and finally multi-texturing of faces for even more variety as in The Sims [13].

9 Acknowledgements

The presented work is supported by the Swiss Federal Office for Education and Science and the EC FIFTH FRAMEWORK INCO-MED PROGRAMME, in frame of the EU ERATO project, number ICA3-CT-2002-10031. We also want to thank Barbara Yersin and Mireille Clavien for their help in designing and editing the materials present in this paper.

References

[1] Daniel Thalmann et al, "Creating a Virtual Audience for the Heritage of Ancient Theatres and Odea," VSMM, 2004.
[2] ERATO, "ERATO EU Project."
[3] Python, *Python Programming Language*.
[4] Ken Perlin, "An Image Synthesizer." Computer Graphics, 1985.
[5] Pablo de Heras Cieschomski et al, "A case study of a virtual audience in a reconstruction of an ancient Roman Odeon in Aphrodisias," VAST, 2004.
[6] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," vol. v19, n10. ACM, 1976, pp. 547-554.
[7] T. Funkhouser, A., "Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments," C. H. Séquim, Ed. 20th annual conference on Computer graphics and interactive techniques, 1993.
[8] H. Hoppe, "Efficient implementation of progressive meshes," vol. 22. Computers & Graphics: Elsevier, 1998, pp. 27-36.
[9] Ogre3D, "Open Source Graphics Engine."
[10] F. Tecchia , C. Loscos, Y. Chrysantou : "Image-based crowd rendering" *IEEE computer Graphics and Applications 22*, 2 (March-April 2002), 36–43
[11] Simon Dobbyn, John Hamill, Keith O'Conor and Carol O'Sullivan, "Geopostors: A Real-Time Geometry/Impostor Crowd Rendering System", i3D 2005
[12] Intel, Intel Corporation and Hyperthreading are copyrighted trademarks, www.intel.com
[13] The Sims, Electronic Arts, http://thesims.ea.com
[14] Jeff Lander, ”Skin Them Bones: Game Programming for the Web Generation”, Game Developer, Miller Freeman, pp. 11-14, May 1998