Continuous Level of Detail for Large Scale Rendering of 3D Animated Polygonal Models

Conference Paper · July 2012

DOI: 10.1007/978-3-642-31567-1_19

3 authors:

Jose Francisco Ramos
Universitat Jaume I
69 PUBLICATIONS **453** CITATIONS
SEE PROFILE

Oscar Ripolles
CA Technologies
63 PUBLICATIONS **682** CITATIONS
SEE PROFILE

Miguel Chover
Universitat Jaume I
180 PUBLICATIONS **820** CITATIONS
SEE PROFILE

Continuous Level of Detail for Large Scale Rendering of 3D Animated Polygonal Models

Francisco Ramos, Oscar Ripolles, and Miguel Chover

Institute of New Imaging Technologies
Universitat Jaume I
Av. Vicent Sos Baynat s/n, Castellón, Spain
{Francisco.Ramos,chover}@uji.es, oripolles@ai2.upv.es

Abstract. Current simulation applications are mainly focused on the efficient management of scenarios with static objects. However, managing dynamic objects, such as animated characters, is very different and requires more specific processing methods which tend to have a high computational cost. Recent advances in graphics hardware offer more ways to improve the performance of these scenes. In this paper, we introduce a new method for rendering large crowds of animated characters at interactive frame rates. Our method is a fully-GPU hybrid combination of mesh instancing, continuous level of detail and hardware palette skinning. Thus, we take advantage of mesh instancing to render multiple instances of a given mesh belonging to a continuous level of detail model, avoiding the typical popping artifacts existing on previous approaches. We finally obtained a low storage cost, performance improvements when applying level of detail and mesh instancing techniques and, moreover, a minimization of the overhead produced by animating.

Keywords: crowd models, GPU, level of detail, animated character.

1 Introduction

Nowadays, simulation applications such as video games require more and more realism in the 3D scenes they represent. However, despite the increasing power of graphics hardware, there exists a bandwidth bottleneck between applications and graphics hardware in such a way that real-time rendering becomes an expensive task that limits the size of the geometric objects that can be processed, thus affecting the realism of the scene. These limitations are even worse in those scenes containing dynamic objects, such as animated characters, as the computational cost of the behavioural management of these dynamic objects tends to be a very time-consuming task. This is the reason why rendering articulated characters has usually been limited by using few characters, by selecting low-detailed meshes or by resorting to impostors or poor animation. Thus, at present time, reducing the number of calculations and polygons sent to the graphics hardware to achieve interactive rates in scenes with multiple animated characters is still a challenge.

The capabilities of recent graphics hardware have led to great improvements in rendering, and animated characters can also be favoured when they are rendered directly in the programmable pipeline. With the current architecture of GPUs, it is possible to use GPU-friendly level of detail techniques and perform skinning in the GPU in order to generate animations, which would greatly increase performance. This increase could be greater if instancing was applied.

Regarding multiresolution modeling, it becomes essential in crowded scenarios as a large number of characters will be rendered at different distances from the viewer. In this sense, geometry management becomes fundamental to achieve a balance between visual accuracy and the amount of time required for geometry processing. It is possible to use discrete solutions, which contain a fixed and small set of approximations, or continuous models, which offer a continuous spectrum of approximations. In the literature, most approaches that manage dynamic objects use discrete models, which entail some limitations as these models present some artifacts when switching among the different approximations and also a high storage cost. In contrast, continuous level of detail offers a better granularity avoiding popping effects while offering a better storage cost as information is not duplicated.

From a different perspective, performing skinning in the GPU boosts the performance of the 3D applications. Instead of storing the animation frames in shader constants, hardware palette skinning offers a nice codification of all the animation information on a texture. This technique permits using the same skinning information for many meshes which can be in different frames within the animation.

In general, rendering of a 3D scene composed of polygonal objects involves drawing these objects by calling some *draw* functions. Moreover, if the polygons differ in applied textures or shaders, an overhead is introduced as different API calls are needed. In this context, instancing[1] allows applications to render a mesh multiple times in different positions and different parameters with a single draw call. This feature considerably reduces CPU overhead and enables us to efficiently exploit graphics hardware.

In this paper, we present a new method to enable large scale rendering of animated characters by performing an efficient use of GPU capabilities. Our method provides a specific combination of the three aforementioned techniques: mesh instancing, hardware palette skinning and continuous level of detail for scenes with dynamic objects. Moreover, we also offer low memory usage and we are able to render thousands of dynamic objects at interactive frame rates. Our method is also easy to be implemented in applications that contain a large number of animated characters, as it only requires a vertex shader program and two textures.

2 Related Work

In this section we will firstly present the latest works on GPU-based level-of-detail modeling, as this technique is key for our framework. Then, we describe and characterize recent techniques that have been developed to manage large crowded scenarios.

Table 1. Characterization of approaches that apply level of detail techniques for rendering large scenes of animated characters

Work	LOD System	Use Impostors	GPU Shaders	Mesh Instancing
[2]	Discrete	No	No	No
[3]	Discrete	Yes	Yes	No
[4]	Discrete	No	Yes	Yes
[5]	Discrete	No	No	No
[6]	Discrete	No	CUDA	No

2.1 Level of Detail

Level of detail is a technique often used to reduce complexity of animated characters. Most-utilized criterion is the distance from the viewer in a per frame decision, although there exist other criteria[7, 8].

In this state of the art we will focus on the multiresolution frameworks which make use of graphics hardware to perform their calculations. A comprehensive characterization of previous multiresolution models can be found in [7].

Discrete multiresolution modeling was lately re-oriented to offer new methods which upload the different approximations to the GPU and avoid the *popping* artifacts by applying geo-morphing [9, 10], blending [11] or hardware-related techniques[12] by means of *shaders*.

It is possible to find in the literature several approaches which maintain the fidelity to the geometry of the input mesh while performing the LOD calculations in the GPU. Ji et al. [13] encode the geometry in a quadtree based on a LOD atlas texture, with a high memory cost and a complex extraction process. Ripolles et al. proposed a fully-GPU solution which could manage continuous [14] and view-dependent resolutions [15]. More recently, and with the aim of developing a fully-GPU adaptive framework, in [16] the authors proposed a multiresolution model which required 3 rendering passes and, moreover, entailed a very high memory cost.

From a different perspective, it is worth mentioning the development of some solutions which propose uploading to the GPU a coarse mesh and using refining patterns to refine each face [17–19]. These works offer interesting results although they are not capable of retrieving the original mesh geometry.

2.2 Animated Characters

Some authors have previously addressed the efficient rendering of these scenes [20–22]. In table 1, we show a comparison of approaches that make use of level of detail techniques for rendering animated characters. This table takes into account several aspects, such as the level of detail system used to manage geometry, the use of impostors mixing geometry, the exploitation of GPU shaders to accelerate rendering and also mesh instancing. In general, works exploiting GPU obtain better results in terms of computational costs that other ones. However, all these solutions resort to discrete multiresolution techniques, which suppose a limitation in visual quality.

3 Our Method

As we can see in Table 1, all the frameworks that manage crowds by means of LOD techniques resort to discrete solutions, which are easier to implement although less accurate. In this sense, we have previously commented that continuous models offer a better granularity as the level of detail can be precisely specified, adjusting the number of visualized polygons to the requirements of the application. Moreover, exploiting graphics hardware becomes nowadays almost essential to design 3D solutions that offer competitive frame rates. In this paper, we propose developing a new method to render large scale crowded scenes with animated characters by using a GPU-suitable level of detail model. Many of the solutions presented in the previous section were developed to efficiently manage static models and we must select a fully-GPU multiresolution model which offers mesh instancing, continuous level detail and skinning. Among the possible solutions, we have selected Sliding-Tris [14].

The workflow architecture used by our approach is presented in figure 1. Our method produces a Sliding-Tris model [14] for an input animated character. More precisely, as the models are usually divided into several submeshes, we build a continuous level of detail model in the GPU for each submesh. In this sense, although in the current implementation of our framework we use the same level of detail for the entire model, it would be possible to use different levels of detail for each instance. Finally, due to the fact that we use hardware palette skinning, it is necessary to upload a texture to the GPU that contains all frames of all animations.

3.1 Sliding-Tris

As we have previously commented, the work we are presenting is based on the selection of an appropriate level-of-detail solution that meets the requirements. We have decided to base our framework on the work proposed in [14]. This level-of-detail technique has mainly two features:

– It uses a simplification process based on vertex collapses and stores the collapse hierarchy in a texture.
– It includes a level-of-detail update routine which alters the contents of the vertex array instead of the indices array. This update algorithm is capable of modifying the level of detail in a single rendering pass. The geometry to render is defined by means of a sliding-window which restricts from CPU the number of triangles to process and visualize. This way, only the vertices of those triangles that will be visualized will be modified on GPU.

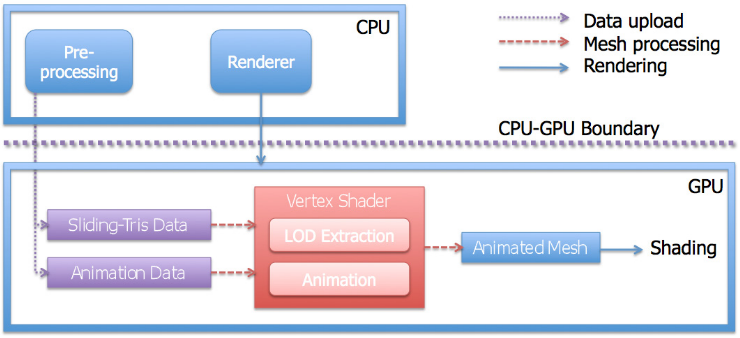


Fig. 1. Workflow of our method

Thus, a key aspect of Sliding-Tris is its easy integration, as a texture and a vertex shader are sufficient. Moreover, this vertex shader can be combined with the skinning technique so that all the calculations required for visualizing the mesh can be performed in the same rendering pass. Another important aspect is the fact that most of its data structures are static and can be shared by all the instances.

3.2 Pre-processing Step

The first part of pre-processing entails building a continuous level of detail model for each submesh that composes the animated character. This process makes use of Sliding-Tris to build the underlying model. This task is performed once for all as the resulting data are stored in a plain text file.

An application that uses our framework must prepare, before rendering the first frame, the necessary data structures in GPU. In our case these structures consist only of two textures per submesh: one to store the Sliding-Tris information and one to store the animation data. We must stress that these static data can be shared among all the instances and that these textures must only be uploaded once for each execution of the application.

Finally, in this pre-processing step we must also create the structure to manage the instances. In other words, each character contains an instance of the pieces or submeshes it is composed of.

3.3 Rendering

At runtime, as shown in algorithm 1, we iterate over the instances of each submesh in each character. Thus, for each instance, we determine its level of detail, and render the appropriate approximation. The decision about what level of detail to use is taken per frame on a per-instance basis. In order to adjust the level of detail, we used the distance to the camera, although other criteria could be applied. It is important to underline that we share the same level of detail for the whole character. However, our method is prepared to accept different levels of detail for each instance, and therefore to the different parts of the character providing a possible view-dependent framework.

Once instanced data are sent to the graphics pipeline, *vertex shader* units perform some fundamental tasks. First, the level of detail extraction process is performed to obtain the suitable geometry. Once completed, texture containing animation data is processed and the corresponding information about bone matrices for the current frame recovered. Thus, we finally send the resulting skinned and approximated mesh through the next stages in the pipeline. It is worth reminding that both tasks are performed in the same *vertex shader* so that the whole framework can work in one single pass.

Algorithm 1. CPU-side rendering loop

```
for $i = 1$ to $numCharacters$ do
  SubMeshes=AnimatedCharacter[$i$].SubMeshes
  for $j = 1$ to SubMeshes.$numInstances$ do
    SubMeshes[$j$].Instance.determineLOD()
    SubMeshes[$j$].Instance.renderInstanced()
  end for
end for
```

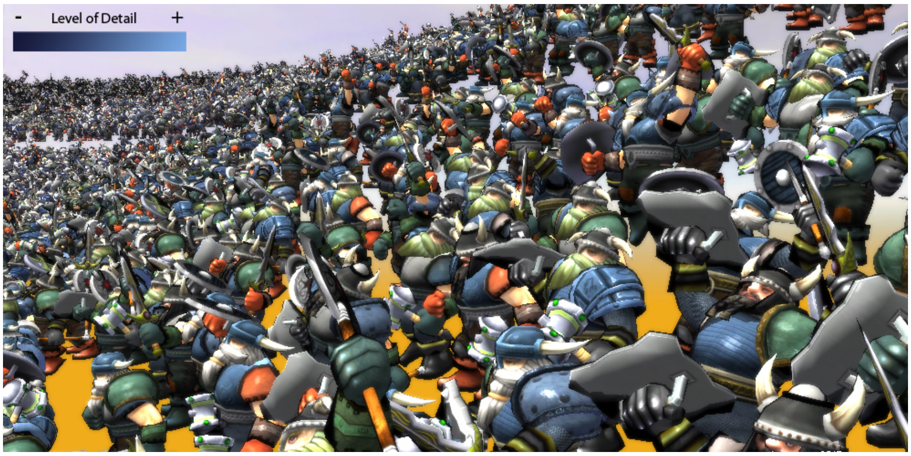


Fig. 2. Rendering of a large crowded scenes with 9,000 animated characters at around 30 FPS using an underlying continuous level of detail method

4 Results

In this section we present a set of measures taken with our method. On the one hand, we describe the results obtained from an analysis of the spatial cost. On the other hand, we also show results about the overall performance. All of our tests were performed using a Mac Pro 2.26 GHz with 8 GB of RAM and an NVidia Geforce GT 120 video card with 512 MB of video memory at a view port of 1024x768. Implementation was performed in HLSL as shader programming language; Direct3D was used as supporting graphics library.

4.1 Memory Cost

The memory cost is a key feature of an adequate level-of-detail solution. Table 2 compares the memory cost of Sliding-Tris [14] with two solutions: a discrete framework which uses 5 levels of detail [5] and a continuous approximation [16]. The original cost of the mesh in triangles is also presented. It can be seen how Sliding-Tris offers the best memory cost, reducing to 60% the memory needs if compared with the other fully-GPU solution.

Table 2. Memory cost study (in bytes)

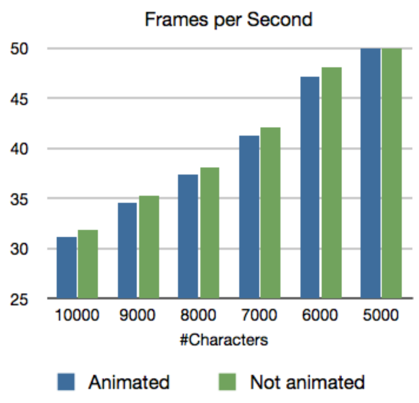
LOD solution	Memory cost
Original model (in triangles)	60v
Ripolles et al. 2009 [14]	80v
Rodriguez et al. 2010 [5]	105v
Hu et al. 2010 [16]	125v

4.2 Performance

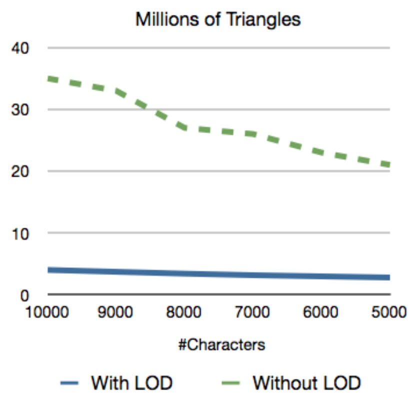
In order to analyse performance of our method we created a scene with 9,000 animated characters (see Figure 2).

In Figure 3a, we can observe the existing difference, in terms of frames per second, when comparing the resulting overhead produced by animating a large number of characters. Normally, animating characters is usually a heavy computational cost, but by performing this task at a GPU level this cost remains almost unaffected. Our results show improvements of around 3% of the total time required to compute animation.

Using level of detail is also an essential technique to take into consideration in our method. We tested our method by comparing the triangle throughput when enabling and disabling the underlying continuous level of detail model, while maintaining visual quality of the scene. As shown in figure 3b, geometry sent to through the graphics pipeline noticeably varies. Although the obtained visual quality is sufficient, we could improve the quality of the approximations by using



(a) Comparison of overhead produced by animating the characters by applying hardware palette skinning.



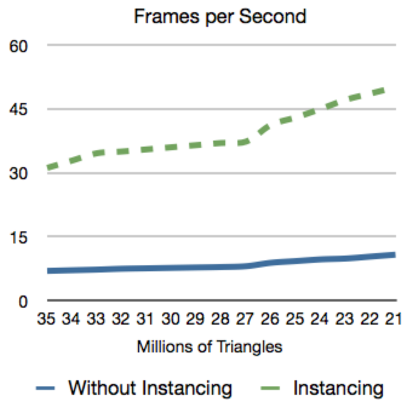
(b) Millions of triangles per number of characters in a scene by enabling or disabling the continuous level-of-detail system.

Fig. 3. Analysis of Performance with level of detail

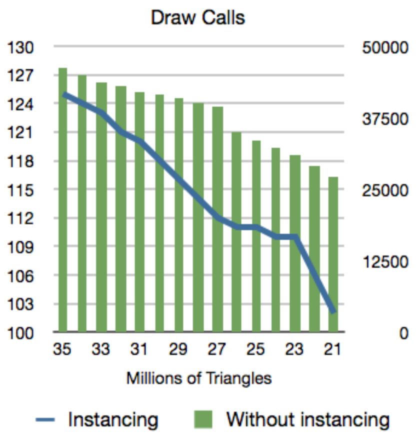
a more accurate level of detail, although performance would be affected. Thus, depending on the final application, adjustments are necessaries for the criterion applied to select the different levels of detail.

Instancing becomes very important when considered on top of the level of detail model. Figure 4a shows great improvements in frame rate when instancing is active. If we focus on a scene with 10,000 animated characters with continuous level of detail, it renders around 30 frames per second when instancing, and around 7 frames per second when not using instantiation. This difference is enough for an application to be interactive or not. It is important to underline that using instantiation also offers improvements on the CPU side, that is, we free the CPU to employ processing time for other operations such as physics or artificial intelligence. Thus, the importance of our method is proportional to the CPU load of the final application where it is implemented. Moreover, continuous level of detail becomes fundamental in scenes with many animated characters as discrete solutions can produce visual disturbances while a continuous multiresolution system provides enough granularity to offer a balance between performance and perceived visual quality.

Finally, Figure 4b shows a draw calls comparison when using instancing in our method. This figure needs two Y-axis as the difference in draw calls is very noticeable. In general, each draw call removed allows a reduction in CPU overhead, and also a possible increase in performance. We obtained 123 draw calls to render around 10,000 characters. In contrast, if instancing is not active, 46,100 draw calls are necessary. Thus, there exists a significant difference in final frame rate as Figure 4a showed.



(a) Frames per second per millions of triangles by using instancing and not using it. Animation and level of detail are active in this test.



(b) Draw calls produced in our method by using instancing and no instancing. Animation and level of detail are active in this test.

Fig. 4. Analysis of Performance with instancing

5 Conclusions and Future Work

We have presented a method for rendering large crowded scenes with animated characters at interactive frame rates. We successfully combined hardware palette skinning, mesh instancing and continuous level of detail. It is important to mention that our framework offers continuous approximations, while all the previous works use discrete solutions which are less accurate. Our results show low storage cost, a minimization of the overhead produced by animations and also competitive rendering times.

Some lines of future research might be supporting view-dependent capabilities across the different submeshes that compose the character and introducing some variations in submeshes to obtain more homogeneousness in the scene.

Acknowledgements. This work was supported by the Spanish Ministry of Science and Technology (Project TIN2010-21089-C03-03), Bancaixa (Project P1.1B2010-08) and Generalitat Valenciana (Project PROMETEO/2010/028).

References

1. Dudash, B.: Mesh instancing. Technical Report 00000-001-v00, NVIDIA (2004)
2. Pratt, D.R., Pratt, S.M., Barham, P., Barker, R.E., Waldrop, M.S., Ehlert, J.F., Chrislip, C.A.: Humans in large-scale networked virtual environments. Presence 6(5), 547–564 (1997)

3. Dobbyn, S., Hamill, J., OConor, K., OSullivan, C.: Geopostors: a real-time geometry impostor crowd rendering system. In: Proc. of Interactive 3D Graphics and Games, pp. 95–102 (2005)
4. Dudash, B.: Animated crowd rendering. GPU Gems 3, 39–52 (2004)
5. Rodriguez, R., Cerezo, E., Baldassarri, S., Seron, F.J.: New approaches to culling and lod methods for scenes with multiple virtual actors. Computers & Graphics 34(6), 729–741 (2010)
6. Feng, W.-W., Kim, B.-U., Yu, Y., Peng, L., Hart, J.: Feature-preserving triangular geometry images for level-of-detail representation of static and skinned meshes. ACM Trans. Graph. 29, 11:1–11:13 (2010)
7. Luebke, D.: Level of detail for 3D graphics, vol. 1. Elsevier Science Inc. (2002)
8. Ahn, J., Oh, S., Wohn, K.: Optimized motion simplification for crowd animation. Comput. Animat. Virtual Worlds 17, 155–165 (2006)
9. Sander, P.V., Mitchell, J.L.: Progressive buffers: View-dependent geometry and texture for lod rendering. In: Symposium on Geometry Processing, pp. 129–138 (2005)
10. Borgeat, L., Godin, G., Blais, F., Massicotte, P., Lahanier, C.: Gold: interactive display of huge colored and textured models. Trans. Graph. 24(3), 869–877 (2005)
11. Southern, R., Gain, J.E.: Creation and control of real-time continuous level of detail on programmable graphics hardware. Comput. Graph. Forum 22(1), 35–48 (2003)
12. Giegl, M., Wimmer, M.: Unpopping: Solving the image-space blend problem for smooth discrete lod transitions. Computer Graphics Forum 26(1), 46–49 (2007)
13. Ji, J., Wu, E., Li, S., Liu, X.: Dynamic lod on GPU. In: Proceedings of the Computer Graphics International, pp. 108–114 (2005)
14. Ripolles, O., Ramos, F., Chover, M.: Sliding-tris: A sliding window level-of-detail scheme. In: CGGM 2008 (2008)
15. Ripolles, O., Gumbau, J., Chover, M., Ramos, F., Puig-Centelles, A.: View-dependent multiresolution modeling on the GPU. In: WSCG (2009)
16. Hu, L., Sander, P.V., Hoppe, H.: Parallel view-dependent level-of-detail control. IEEE Transactions on Visualization and Computer Graphics 16, 718–728 (2010)
17. Lorenz, H., Döllner, J.: Dynamic mesh refinement on GPU using geometry shaders. In: WSCG (February 2008)
18. Schwarz, M., Stamminger, M.: Fast GPU-based adaptive tessellation with CUDA. Computer Graphics Forum 28(2), 365–374 (2009)
19. Dyken, C., Reimers, M., Seland, J.: Real-time GPU silhouette refinement using adaptively blended bézier patches. Computer Graphics Forum 27(1), 1–12 (2008)
20. Savoye, Y., Meyer, A.: Multi-layer level of detail for character animation (2008)
21. Pilgrim, S., Steed, A., Aguado, A.: Progressive skinning for character animation. Comput. Animat. Virtual Worlds 18(4-5), 473–481 (2007)
22. Pettre, J., de Heras Ciechomski, P., Maim, J., Yersin, B., Laumond, J.-P., Thalmann, D.: Real-time navigating crowds: scalable simulation and rendering. Computer Animation and Virtual Worlds 17(3-4), 445–455 (2006)