Continuous Level of detail on Graphics Hardware

Francisco Ramos, Oscar Ripollés , Miguel Chover

Universitat Jaume I, Dept. de Lenguajes y Sistemas Informáticos, Castellón de la Plana 12071 Spain

Abstract

Recent advances in graphics hardware provides new possibilities to succesfully integrate and improve multiresolution models. In this paper, we present a new continuous multiresolution model that maintains its geometry, based on triangle strips, in high-performance memory on the GPU. This model manages level of detail perfoming fast strips updating operations. We show how this approach takes advantage of new GPU's capabilities in an efficient manner.

Key words: multiresolution model, level of detail, triangle strips, real-time rendering, graphics hardware

1 Introduction

One of the main problems of graphic interactive applications such as computer games or virtual reality is the geometric complexity of the scenes they represent. In order to solve this problem, different modelling techniques by level of detail have been developed, trying to adapt the number of polygons of the objects to their importance inside the scene. The application of these techniques is common in standards such as X3D, graphic libraries such as OpenInventor, OSG, and even in game engines such as Torque, CryEngine, etc., where models with continuous levels of detail are introduced, based mainly on Progressive Meshes [1]. The tendency in the last years has been to improve the features of continuous models by using at their maximum the possibilities offered by graphic hardware, with the intention of competing with the discrete models

Email addresses: Francisco.Ramos@uji.es (Francisco Ramos), oripolle@sg.uji.es (Oscar Ripollés), chover@uji.es (Miguel Chover).

that, though more limited, are perfectly adapted to current graphics hardware. Specifically, they have worked on the representation of multiresolution models which use triangle strips to accelerate the visualization by means of vertex arrays located in the GPU. The fundamental problem of these techniques is the fact that a continuous model needs to make changes on the list of indexes of the primitives it draws, which causes graphic hardware to lower its performance when having to carry out this kind of operations.

1.1 Related work

Last years, multiresolution models have progressed substantially. In the beginning discrete models were employed in graphic applications, due mainly to the little implementation complexity they showed, which is the reason why still nowadays they keep being used in applications without great graphic requirements. Nevertheless, the increase in realism in graphic applications compels to use multiresolution models which are more exact in their approximations, which don't require high storage costs and which are faster in visualization. This has given way to continuous models, where two consecutive levels of detail only differ in few polygons and where, besides, the duplication of information is avoided considerably improving the spatial cost offered by the discrete ones.

The best known continuous multiresolution model is Progressive Meshes [1], included in Microsoft Corporation's graphic library DirectX. This model presents excellent results in visualization in real time, although it is based on triangle primitives.

Advances have been made in the use of new graphic primitives which minimize the data transfer between the CPU and the GPU, apart from trying to make use of the connectivity information given by a polygonal mesh. With this purpose graphic primitives with implicit connectivity, such as triangle strips and triangle fans, have been developed. Many continuous models based on this type of primitives have been recently developed [2-7].

In these years, graphics hardware performance has evolved outstandingly, giving rise to new techniques which permit to accelerate even more the continuous models. The use of stripification algorithms which try to take the maximum advantage of the GPU cache, and the new extensions of graphic libraries which allow to visualize a whole mesh with few instructions, are examples of these new techniques.

Nowadays GPUs offer new capacities that, exploited to the maximum, can offer very good results in several aspects. One of them involves storing information directly on the high speed memory located in the GPU. This characteristic allows managing the information in the GPU avoiding data transfer between

CPU and GPU and taking the maximum advantage of the proximity of the memory and the graphic processor. There are some related works which make use of the new capacities of the current GPUs, such as [8], which implements a discrete model manager which puts geomorphing into practice by using vertex shaders; another work is [9] which creates different shaders depending on the level of detail.

1.2 Motivation

In general, the main problem of continuous models lies in the high cost of extracting the level of detail, which usually takes about 20% of the total visualization cost. Apart from the extraction, the use of AGP buses poses the problem of being much more optimized to upload data than to download it, favoring the use of the memory of the graphic card to store static objects that don't change their geometry. But the appearance of the PCI-Express bus makes it possible to use a symmetric bus, which allows uploading and downloading data to the GPU at the same speed, so that it is possible to work with the GPU memory in a reliable way and without penalizations in data download.

1.3 Contributions

In this article we present a new multiresolution model integrated into the graphics hardware. This model makes use of the present GPUs capacities to store its data structures inside. The fundamental idea on which the model is based is creating data structures which are efficient for its integration into the GPU, and that at the same time offer an optimum performance regarding both visualization and spatial cost. The model works directly with the GPU memory, obtaining appreciable improvements as shown in the results section.

This way, what this model offers is a complete integration into the graphic hardware, a low cost of extraction of the level of detail, exploiting the coherence between levels of detail, and a low spatial cost.

The implemented model features different characteristics:

- Wholly based on triangle strips.
- Simplification based on progressive edge collapses.
- Static stripification. Triangle strips are only generated once, at the highest level of detail, using a method that takes advantage of GPU's cache.
- Geometric information of the model is maintained and stored in the GPU.

- Level of detail management is performed by a data structure, LOD-Manager, which allows fast strips updating and removing degenerated triangles from.

2 Fundamentals

Multiresolution models

To construct a continuous multiresolution model based on primitives of implicit connectivity, as triangle strips, it is necessary to fulfil certain requirements. On one hand, a mesh made up of this kind of primitive must be available and, on the other hand, the simplification method that should be employed in order to generate the different levels of detail must be selected.

There are several mesh simplification methods [10][11], but one of the most important in the progressive mesh simplification is [1]. This method is based on iterative edge contractions, and it is the one employed on well-known multiresolution models such as [2-7].

It is possible to find in literature many works where the problem of converting a polygonal mesh made up of triangles into triangle strips is solved [12][14]. This process is commonly called stripification, and it can be carried out in a dynamic or static way. Dynamic stripification involves generating the triangle strips in real time, that is, for each level of detail new strips are generated. On the other hand, static stripification entails creating initially triangle strips and working later with versions of the original strips. There are several models that use dynamic stripification [3][4], especially variable resolution models. For their part, other models as [2][5-7] use static stripification techniques.

The main problem of static stripification models can be observed in Figure 1. As model reaches lower levels of detail, it presents vertex repetitions that do not add any information to the final scene but involve higher data traffic between the CPU and the GPU. Models as [2][7] solve this problem applying filters to eliminate degenerated triangles. The first employs filters in visualization, avoiding sending those vertices at the moment of de rendering, and the second executes a preprocess that detects them initially, storing that information and eliminating them from the strips before visualizing them.

Given the architecture of present GPUs, it is preferable to employ static stripification techniques since we avoid strip creation and destruction on the GPU, that would imply an additional cost which would make the model much less competitive. Furthermore, there is an additional cost entailed by calculating the new triangle strips at each level of detail, which also penalizes the use of these techniques. Moreover, it is preferable to eliminate degenerated triangles before the visualization, which permits to accelerate it considerably by resizing strips, apart from allowing a better implementation of the model on the GPU, avoiding creating a specific code for the filters. Nowadays, varied acceleration techniques have appeared, which integrated into a multiresolution model would also become key to improve its performance. Basically, we can notice stripification techniques oriented to exploit vertex caches [12] and hardware acceleration techniques by means of graphics library extensions [13].

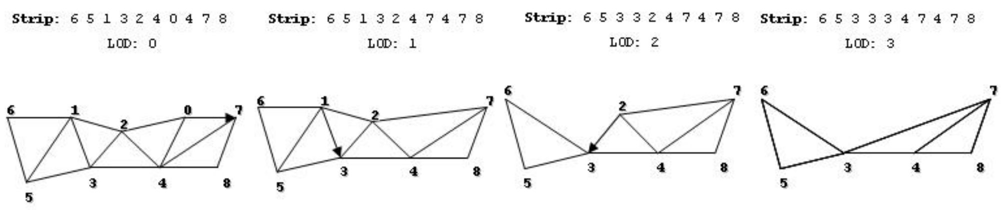

Strip: 6 5 1 3 2 4 0 4 7 8
LOD: 0
Strip: 6 5 1 3 2 4 7 4 7 8
LOD: 1
Strip: 6 5 3 3 2 4 7 4 7 8
LOD: 2
Strip: 6 5 3 3 3 4 7 4 7 8
LOD: 3

Fig. 1. Multirresolution triangle strips

High-perfomance memory on GPUs

A vertex buffer object is a feature that enables us to store data in high-performance memory on the GPU. The basic idea is to provide some buffers, which will be available through identifiers. There are different ways to interact with buffers:

- Bind a buffer: it activates the buffer in order to be used by the application.
- Put and get data: it allows us copying data between a client's area and a buffer object in the GPU.
- Map a buffer: you can get a pointer to a buffer object in the client's area, but it can produce that the driver waits for the GPU to finish their operations.

There are two kinds of vertex buffer objects: array buffers and element array buffers. On one side, array buffers contain vertex attributes, such as vertex coordinates, texture coordinates data, per-vertex color data and normals. On the other side, element array buffers contain only indices to elements in array buffers. The ability to switch between various element buffers while keeping the same vertex array allows us to implement level of detail schemes by changing the elements buffer while working on the same array of vertices.

In order to implement the model on graphics hardware, we have used different functions which interact with buffer objects. Among them, we can highlight:

- glBindBufferARB: this function sets up internal parameters so that the next operations work on this current buffer object.

- glBufferDataARB: this function is an abstraction layer between the memory and the application. Basically, this function copies data from the client memory to the buffer object bound.
- glBufferSubDataARB and glGetBufferSubDataARB: its purpose consist of replacing or obtaining respectively, data from an existing buffer.

3 Implementacion details

3.1 General framework

A brief diagram of the model is shown in Figure 2. At the beginning, information about vertices and strips, at the highest level of detail, is uploaded into the GPU. Later, by means of LOD-Manager data structure, strips are updated in accordance with the current level of detail.

In our approach we first perform two essential tasks: generate triangle strips at the highest level of detail and calculate vertex-collapse simplification.

At runtime, we upload vertices and strips information into the GPU. Then, depending on application demands, we perform vertex-split or edge-collapse operations directly on the strips. This task is executed by the LOD-Manager. Concretely, when a level of detail transition is required, it downloads from the GPU, the strips affected by these changes. Later, it modifies and uploads the updated strips to the graphics system. Last, strips information in the GPU is then used for display.

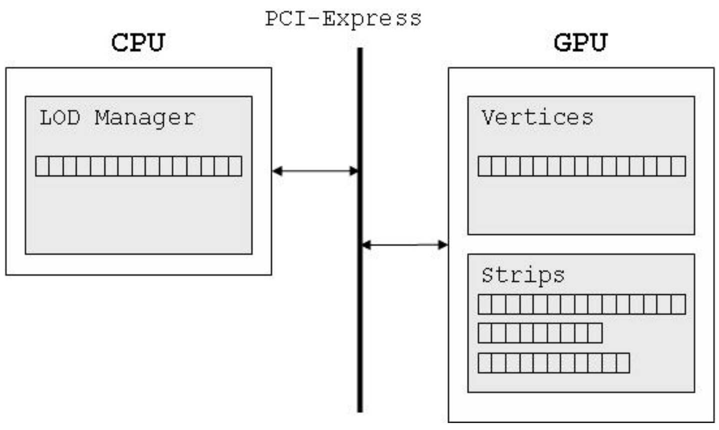


Fig. 2. Model architecture

3.2 LOD-Manager data structures

Main function of LOD-Manager consists in serving level of detail demands required by applications. It is able to quickly change the geometric information located in the GPU by applying a series of pre-calculated records. These records mainly stores two kind of information: simplifications and filters.

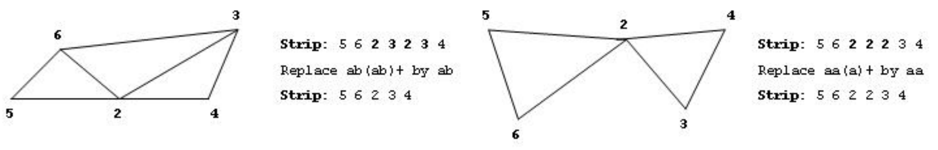


Fig. 3. Removed patterns

Simplification information contains which strips change for each level of detail, and where are located the vertices to split or collapse. It allows us to quickly locate information to be modified when we move from a level of detail to another. However, as model moves to coarse LODs, an accumulation of identical vertices is produced. Sending these vertex repetitions to the graphics hardware does not contribute at all to the final scene, because it is equivalent to send degenerated triangles, as is shown in Figure 1. We have checked that most vertex repetitions can be removed, following patterns like aa(a)+ or ab(ab)+. Patterns aa(a)+ are replaced by aa, and ab(ab)+ by ab. Figure 3 shows an example for each kind of pattern, we can observe that final geometry of strips do not change after removing these patterns.

3.3 GPU data structures

Two essential data structures for the model performance are stored in the GPU: vertices and strips, which compose the polygonal mesh. On the one hand, vertices are stored in a vertex array buffer. On the other hand, we might allocate each strip in an element buffer. However, we have checked that creating as many buffers as strips implies noticeably decreases in performance due to bind operations. A solution to this problem, with optimum results, consist of creating a single element buffer, where is located every strip to be rendered. In this manner, we avoid continuous bind operations to assign an element buffer for each strip.

3.4 Controlling level of detail

In continuous multiresolution models, level of detail management, carries two fundamental tasks: level of detail extraction required by applications and visualization of resultant geometry.

Level of detail extraction

At a high level, pseudo algorithm for moving from LOD n to LOD n+1 would consist in downloading, from the GPU, the chunks of memory corresponding to the strips affected by the change of the level of detail. After that, we replace vertex n by the vertex where it collapses to, in every strip where it appears. Later, derived vertex repetitions must be removed. Finally, the strip is uploaded to the GPU for visualization.

```
for LOD = currentLOD to demandedLOD
  for Strip = StripsAffected(LOD).Begin() to StripsAffected(LOD).End()
    auxStrip=DownloadFromGPU(Strip);
    CollapseOrSplit(auxStrip,LOD);
    UploadToGPU(auxStrip);
  end for
end for
```

Fig. 4. Level of detail extraction from a LOD to a coarse one.

Visualization

Figure 5 corresponds to the visualizationaalgorithm. This algorithm takes advantage of new GPU capacities. It directly stores and manages strips to visualize from graphics hardware memory.

```
for IndexStrip = 0 to NumberOfStrips - 1
  glDrawRangeElements (
    GL_TRIANGLE_STRIP,
    currentLOD,
    NumberOfVertices - 1,
    StripBufferManager(IndexStrip).size(),
    GL_UNSIGNED_INT,
    (const void*)(StripBufferManager(IndexStrip).Offset()*sizeof(EnteroUn)),
end for
```

Fig. 5. Visualization algorithm.

4 Results

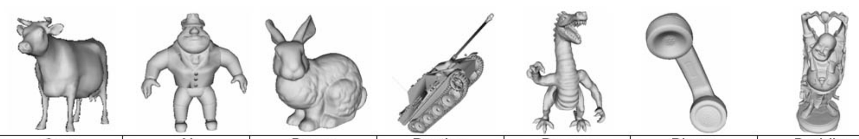
Figure 6 shows a spatial cost comparative. On average, the presented model fits in 1.5 times the original mesh in triangles and 2.3 times in triangle strips.

Two well-known utilities to generate strips have been tested in this multiresolution model: Stripe Utility [14] and NVTriStrip Library [12]. Triangle strips for different objects have been generated from both utilities. Model generated from the NVTriStrip Library shows better frame-per-second rates than the Stripe object when the level of detail is higher; this behaviour is shown in Figure 7b.

Results of visualization are shown in Figure 7a, where it is compared our approach to other models. It is possible to observe that our model offers the best visualization times due to its hardware integration.

5 Conclusions

We have presented a uniform resolution model that noticeably improves existing models, in terms of storage and visualization cost. This model features: a total graphics hardware integration with implementation on high-perfomance memory, optimized hardware primitives, vertex cache exploitation and low spatial cost.



	Cow	Al	Bunny	Panther	Dragon	Phone	Buddha
Vertices	2904	3618	34834	38911	54294	83044	543699
Faces	5804	7124	69451	69397	108588	165963	1085634
Size Tris kb	113.4	140.0	1358.2	1421.2	2120.9	3242.5	21217.6
Size Strips kb	73.5	91.4	867.6	971.1	1387.6	1999.5	14107.9
Model Cost Mb	0.16	0.20	2.07	2.00	3.59	4.71	33.53
Ratio Triangles	1.5	1.5	1.6	1.4	1.7	1.5	1.6
Ratio Strips	2.3	2.2	2.4	2.1	2.7	2.4	2.4

Fig. 6. Spatial cost comparison

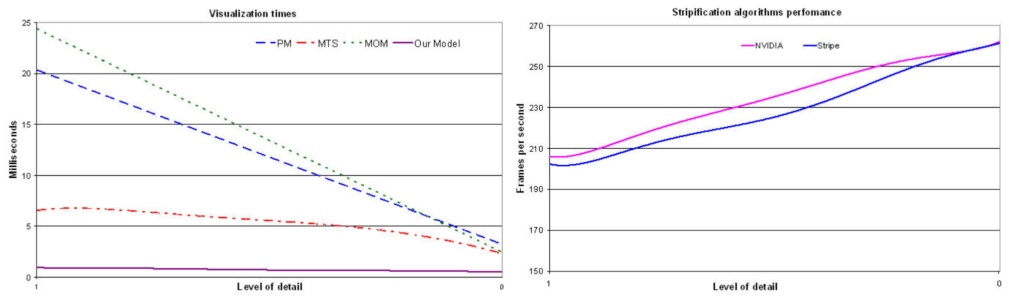


Fig. 7. (a) Multiresolution models comparison and (b) stripification techniques perfomance in our approach

References

[1] Hoppe H. Progressive Meshes. Computer Graphics (SIGGRAPH), 30:99-108, 1996.

[2] El-Sana J, Azanli E, Varshney A. Skip strips: maintaining triangle strips for view-dependent rendering. In: Proceedings of Visualization 99, 1999. p.131-137.

[3] Michael Shafae, Renato Pajarola. DStrips: Dynamic Triangle Strips for Real-Time Mesh Simplification and Rendering. Proceedings Pacific Graphics Conference, 2003.

[4] A. James Stewart: Tunneling for Triangle Strips in Continuous Level-of-Detail Meshes. Graphics Interface 2001: 91-100.

[5] O. Belmonte, I. Remolar, J. Ribelles, M. Chover, M. Fernndez. Efficient Use Connectivity In-formation between Triangles in a Mesh for Real-Time Rendering, Future Generation Computer Systems, Special issue on Computer Graphics and Geometric Modeling, 2003. ISSN 0167-739X.

[6] J. Ribelles, A. Lpez, I. Remolar, O. Belmonte, M. Chover. Multiresolution Modeling of Polygonal Surface Meshes Using Triangle Fans. Proc. of 9th DGCI 2000, 431-442, 2000. ISBN 3-540-41396-0.

[7] J. F. Ramos, M. Chover, LodStrips, Lecture notes in Computer Science, Proc. of Computational Science ICCS 2004, Springer, ISBN/ISSN 3-540-22129-8, Krakow (Poland), vol. 3039, pp. 107-114, June, 2004.

[8] Olano, Marc, Bob Kuehne and Maryann Simmons, Automatic Shader Level of Detail. Proceedings of Graphics Hardware 2003, Eurographics/ACM SIGGRAPH, July 2003.

[9] Creation and Control of Real-time Continuous Level of Detail on Programmable Graphics Hardware James Gain, Richard Southern; Computer Graphics Forum, March 2003

[10] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In Proceedings of SIGGRAPH '97 (Los Angeles, CA), Computer Graphics Proceedings, Annual Conference Series, pages 209 - 216. ACM SIGGRAPH, ACM Press, August 1997.

[11] A Developer's Survey of Polygonal Simplification Algorithms, David P. Lueke IEEE CG A, June, 2001

[12] NvTriStrip Library, NVIDIA Corporation (2002). Available in Internet at following URL http://developer.nvidia.com/object/nvtristrip_library.html.

[13] ARB_vertex_buffer_object Specification. http://oss.sgi.com/projects/ogl-sample/registry/ARB/vertex_buffer_object.txt

[14] F. Evans, S. Skiena and A. Varshney, Optimising Triangle Strips for Fast Rendering, IEEE Visualization '96, 319-326, 1996. http://www.cs.sunysb.edu/ stripe