An approach to improve strip-based multiresolution schemes

J. F. Ramos, M. Chover, O. Belmonte, C. Rebollo
Departamento de Lenguajes y Sistemas Informáticos
Universitat Jaume I, Campus de Riu Sec
12071, Castellón, Spain
{jromero, chover, belfern, rebollo} @uji.es

ABSTRACT

Triangle strips have been widely used for static mesh representation because they are optimal for rendering. This primitive reduces the number of vertices sent to the graphics pipeline and the storage costs. We present an approach to improve multiresolution models that takes this connectivity property into account. Our model uses strips both in the data structure and in the rendering stage. It also offers the following features: it is easily implemented, and is efficient and fast.

Keywords

Keywords: multiresolution model, level of detail, triangle meshes, triangle strips.

1. INTRODUCTION

A common approach to solving the problem of complexity in large scenes involves level-of-detail (LOD) or multiresolution modelling. Multiresolution models can be classified into two large groups: discrete multiresolution models, i.e. those that contain various representations of the same object with different levels of detail, and continuous multiresolution models, which are those that represent a vast range of approximations to a continuous object.

Discrete models typically store between five and ten LODs and tend to suffer from popping artifacts when a LOD is changed. Some graphics standards, like X3D or OpenInventor, use such models. In continuous multiresolution models, two consecutive LODs differ by only a few triangles. These small changes introduce minimal visual artifacts. On the other hand, the size of the model becomes smaller than that of a discrete model, because no duplicate information is stored. Currently, Progressive Meshes by Hugges Hoppe [Hop96a] is the best known continuous multiresolution model. It is included in the graphics library DirectX from Microsoft Corporation.

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG 2004, February 2-6, 2003, Plzen, Czech Republic.

Although these models have shown excellent results in interactive visualisation, they work with triangles. Recently, these models have been improved to include connectivity information by storing the models as triangle strips or triangle fans that reduce the amount of information sent to the graphics pipeline and increase the rendering frame rate [Rib00a][Ste01a][Sha03a].

In short, modelling a mesh as a collection of strips or fans of triangles avoids the need to store and send a large amount of redundant information to the graphics system; it also gives rise to better visualisation times and lower storage costs.

Section 2 begins with a brief introduction to multiresolution models and then we review the state-of-the-art in multiresolution models. In section 3 a new model is presented, along with its data structure and algorithms. There is also an explanation of its main characteristics and a description of how to construct it, with internal details. Later, in section 4, we present the results of comparing this model with PM [Hop96a] and MTS [Bel03a]. Finally, in section 5, conclusions and future work are presented.

2. RELATED WORK

Multiresolution Models

Garland [Gar99a] defines a multiresolution model as a model representation that captures a wide range of approximations of an object and which can be used to reconstruct any one of them on demand.

The common criteria used to determine the most suitable LOD are the distance from the object to the viewer, the projected area of the object on the display, the eccentricity of this object on the display, and the intrinsic importance of the object.

Ribelles et al. [Rib02a] present a characterisation of multiresolution models. This work classifies the models taking into account certain other criteria. Some other important works on this subject are [Pup97a][Gar99a].

In this section, multiresolution models that take advantage of connectivity are reviewed.

The VDPM model by Hoppe [Hop97a] first determines the set of triangles to visualise, and then triangle strips are searched on-the-fly over the mesh. This is a time-consuming task but the final visualisation time is reduced because triangle strips are rendered quickly.

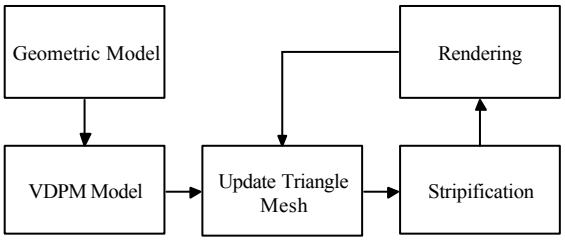

Figure 1. VDPM, Hoppe 97

The El-Sana et al. model [Els99a] first determines a set of triangle strips over the polygonal model. These triangle strips are stored in a data structure called a Skip List. After determining which triangles to render, the skip lists are updated and then rendered. This model passes the display triangle strips through filters for removing vertex repetitions.

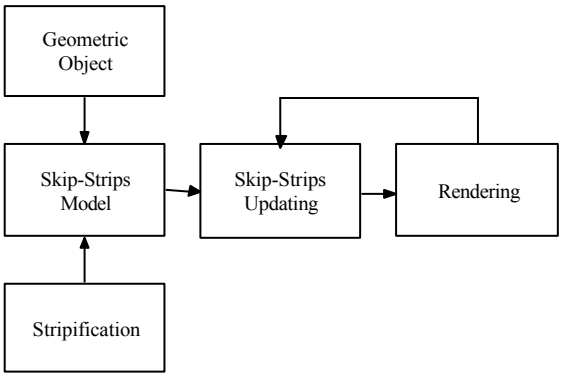

Figure 2. Skip-Strips, El-Sana 99

The first multiresolution model to use the triangle fan primitive in storage and in the rendering stage, taking advantage of the connectivity information between the triangles in a mesh, is the model by Ribelles et al. called MOM-Fan [Rib00a]. This model uses the triangle fan primitive both in the data structure and in the rendering stage. The main drawback of this model is the high number of degenerate triangles used in representation, although they are purged out before the rendering stage. Another drawback of the model is that the average number of triangles in each triangle fan is small.

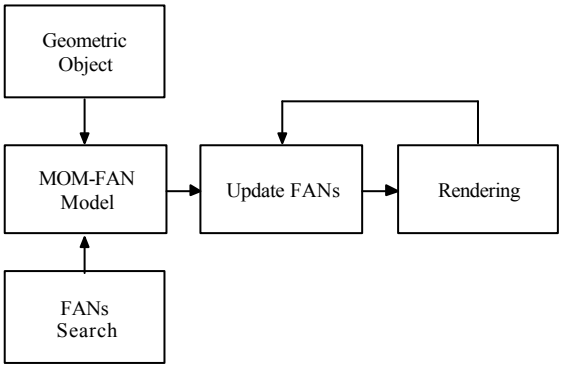

Figure 3. MOM-Fan, Ribelles 2000

As regards strips, the first multiresolution model to take advantage of the triangle strip primitive in storage and in the rendering stage is that by Belmonte et al., called MTS [Bel03a]. This model uses the triangle strip primitive both in the data structure and in the rendering stage. The model consists of a collection of multiresolution strips, each of which represents a triangle strip at every LOD, and this is coded as a graph. Only the strips that are modified between two consecutive LOD extractions are updated before visualisation.

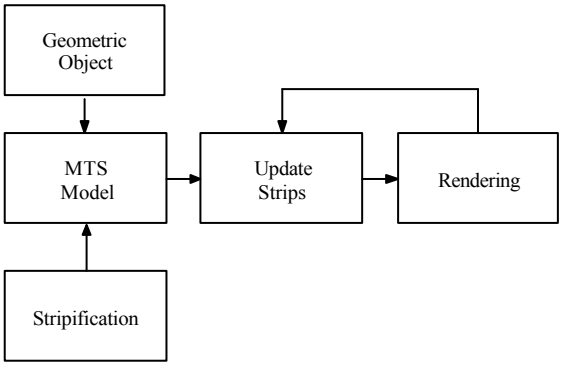

Figure 4. MTS, Belmonte 2003

Recently, some works have been presented that are based on the triangle strip primitive. These focus on dynamically simplifying the triangle strips for each demanded LOD. The model by Shafae et al, called DStrips [Sha03], manages the triangle strips in such a

way that only the triangle strips that are being modified are actually processed, while the rest of the triangle strips in the model remain unchanged. This update mechanism reduces the extraction time, although results published in this work still show a high extraction time.

Another approach to the use of triangle strips in a multiresolution model is the work by A. James Stewart [Ste01a], extended by Porcu [Por03a]. This work uses a tunnelling algorithm in order to connect isolated triangle strips, thus obtaining triangle strips with large numbers of triangles while reducing the number of triangle strips in the model as it is simplified. Again, its main drawback is the time consumed by the stripification algorithm.

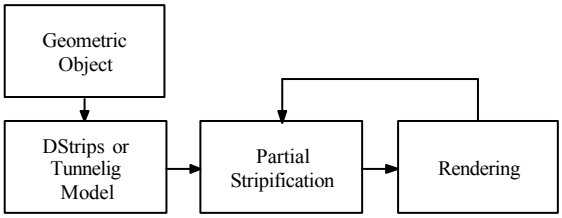

Figure 5. Dynamic Stripification Models

3. OUR APPROACH

Introduction

The proposed model here represents a mesh as a set of multiresolution triangle strips. The model maintains the strips both in the data structure and in the rendering stage.

Our model offers the following features:

- Easy to implement: the data structures and algorithms are simple and clear.
- Efficiency: with low LOD recovery[1] time, we obtain better results than MTS [Bel03a], the first model wholly based on triangle strips.
- Speed: using triangle strips, rendering is faster than PM [Hop96a], which is based on triangles.

In the following sections we present the data structure, algorithms and the construction process of the model presented.

Data Structures

We store a list of multiresolution strips where each is represented by an array of pointers to vertices. This structure is defined in such a way as enable those vertex sequences to be utilised directly for rendering by using the strip graphics primitive. Therefore, we have a list of arrays to represent the strips, *lStrips*, and a vertex array to store the geometric coordinates, *lVerts*. Furthermore, we have another data structure, *lChanges*, which allows the swift recovery of the strip vertex positions that change between LODs.

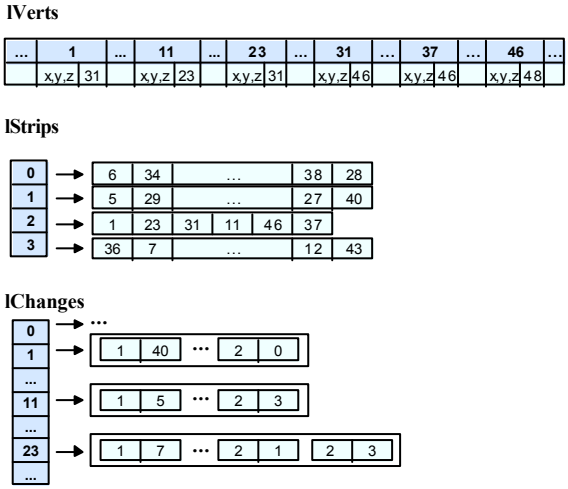

Figure 6. Data structure example

3.2.1 lVerts

This structure stores the 3D coordinates of each vertex in the mesh. In addition, each record has a field *next* that stores a pointer to the vertex to be collapsed. When a vertex is removed due to the change in level of detail, that vertex is substituted by the vertex found in its *next* field. Another important feature of this structure is that *lVerts* stores the vertex according to the order of simplification. This enables the easy identification the vertices which are not valid for one LOD. If we change from LOD 0 to LOD 1, vertex 0 will be replaced with the vertex in its *next* field in all occurrences of that vertex in the *lStrips* structure. In Figure 6 we can see an example of these data structures.

In our model, each vertex can only be collapsed with its neighbours on the external edge; to eliminate internal edges it is necessary to collapse their two vertices. For example, in Figure 7 we can see that in LOD 0, vertex 31 could only be collapsed with vertex 46 or vertex 1.

3.2.2 lStrips

This structure uses a list to maintain the multiresolution strips. Each strip is represented by means of an array of pointers to vertices that provide the vertex sequence to draw the strip in the drawing stage. Therefore, taking strip 2, in Figure 6, as a reference, it can be seen that the vertices 1,23,31,11,46,37 define the strip with the highest LOD.

[1] Rendering time = recovery time + drawing time

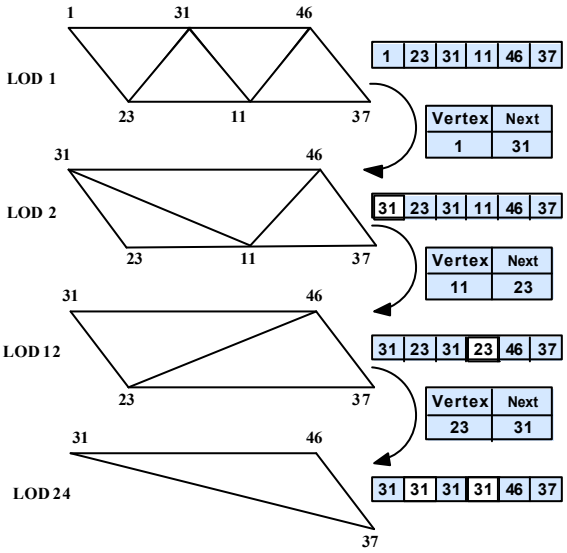

Figure 7. Level-of-Detail Transition

3.2.3 lChanges

lChanges is a list that allows us to quickly recover the positions of the vertices that are changed, when the model switches between LODs. As we can see in Figure 6, *lChanges* is ordered by the number of the LOD. As the example shows, when the model changes from LOD 1 to LOD 2, it is necessary to change position 40 in strip 1, position 0 in strip 2, and so on. Only in those positions can we find vertex 1, a vertex that will be changed in all of its occurrences in *lStrips* with vertex 31 (the vertex pointed to in their *next* field). This means that vertex 1 will never appear in LOD 2 because all the vertices in a LOD must have and index lower than or equal to the current LOD number.

3.2.4 Construction

Construction is a preprocess that fills the data structures of the model. These structures store the vertices, the strips that represent the mesh and the changes that take place on the strips when the LOD changes.

The phases of this preprocess can be summarised in the following steps:

- **Stripification**. Using the STRIPE algorithm [eva96a], we build the vertex array with the vertex coordinates data and the strip list with the highest level of detail.
- **Simplification**. By means of the QSLIM algorithm [Gar97a], we select the vertex simplification order.
- **Arrangement**. Once we have the simplification order, we change all the structures to store the vertices in that order. We need to arrange the vertex array and all the occurrences of the vertices in the strips.
- **Collapse**. We fill the *next* field of each vertex taking into account the results of the simplification process and the restriction that all vertices to be collapsed must be neighbours on the external edge.

With this information the model is able to change between LODs quickly and efficiently using the algorithms described below.

Algorithms

3.2.5 Level of detail recovery algorithm

The level of detail recovery algorithm works by changing the vertices of all strips by simplifying them. As shown in Figure 8, at the beginning it is important to detect whether the new level of detail is higher than the current one, or vice versa, in order to traverse strips in ascending or descending order. The vertices involved are then changed in each strip, in a sequential and ordered way, until a new level of detail is reached.

```
DetermineStep(s);
FOR currentLOD TO newLOD STEP s
    WHILE NOT EndChangesLOD
        SplitOrCollapseVerticesInStrips;
    END WHILE
END FOR
```

Figure 8. Level-of-Detail Recovery Algorithm

In Figure 7 we show how this algorithm operates. Starting at LOD 1, to move to LOD 2 it is necessary to traverse the lChanges data structure for LOD 1, as shown in Figure 6, and each change in the strip and position indicated is applied. In this case, then, it is necessary to change the vertex located in strip 2, position 0, that is, vertex 1 is changed by its next: 31, and so on. In the following section it will be shown how the drawing algorithm detects the repeated vertices, which are not sent to the graphics system.

3.2.6 Drawing algorithm

The aim of the drawing algorithm is to traverse all strips, reducing the vertices sent to the graphics system by detecting repetitions. In Figure 7 it can be seen how, in the strip of LOD 12, there is a vertex repetition: 31 23. The drawing algorithm is able to detect these situations and then only sends these vertices to the graphics system once. Figure 9 shows the pseudocode of this algorithm.

```
FOR EACH Strip;
        SendVertex;
        IF FoundRepetitions THEN
                JumpToNextVertex;
        ENDIF
END WHILE
```

Figure 9. Drawing Algorithm

4. RESULTS

Tests designed to compare multiresolution models follow the ones introduced by [Rib99a]. The tests carried out were:

- Linear Test: consists in extracting the LODs of the model in a linear and proportionately increasing or decreasing way.
- Exponential Test: consists in extracting LODs in an exponential way, that is, at the beginning very distant levels of detail are extracted and then the closer levels.

The spatial cost of the model is also presented here.

All these tests were compared with [Hop96a][Bel03a], the first of which has been and still is a reference model in the multiresolution world, while the second one is the most recent multiresolution model to make full use of multiresolution strips.

To pass the tests three well-known meshes were taken as a reference, from the Stanford 3D Scanning Repository, so as to make it easy to compare this model with other well-developed models.

Tests were carried out on a PC with an Intel Xeon 2.4 Ghz processor and 512 Mb RAM, and with an ATI Fire GL E1 64 Mb graphics card.

Spatial cost

In Table 1, it can be seen how the model presented here, has a similar spatial cost than the other models in the comparison.

Mesh	Vertices	PM	MTS	Our Model
Cow	2904	0.272 Mb	0.252 Mb	0.388 Mb
Bunny	34834	3.282 Mb	2.963 Mb	3.556 Mb
Phone	83044	7.863 Mb	6.765 Mb	8.099 Mb

Table 1. Spatial cost comparison

Tests

The following tables show the results of applying linear and exponential test to models PM [Hopp96a], MTS [Bel03a] and the one presented here: Model.

As shown in Table 2, which corresponds to the linear test, and in Table 3, corresponding to the exponential test, the total visualisation time is shown first, while the lower part shows the percentage of this time used to extract the level of detail and the percentage used to draw the resulting mesh.

It should be pointed out that, comparing this model with PM [Hop96a] and MTS [Bel03a], it was necessary to add the time spent on discarding repeated vertices to the recovery percentage and to deduct it from the drawing percentage; the total time for rendering is not affected by this calculation.

	PM		**MTS**		**Our Model**	
	Render (ms)		Render (ms)		Render (ms)	
	% rec	*% drw*	*% rec*	*% drw*	*% rec*	*% drw*
Cow	0.986715		1.006055		0.573033	
	7.88%	92.12%	38.00%	62.00%	16.56%	83.44%
Bunny	11.29813		6.169898		5.684376	
	0.76%	99.24%	22.32%	77.68%	8.93%	91.07%
Phone	32.983562		14.250778		14.296618	
	0.24%	99.76%	16.83%	83.17%	8.33%	91.67%

Table 2. Linear test

	PM		**MTS**		**Our Model**	
	Render (ms)		Render (ms)		Render (ms)	
	% rec	*% drw*	*% rec*	*% drw*	*% rec*	*% drw*
Cow	1.1235591		1.113182		0.62946	
	6.52%	93.48%	27.42%	72.58%	16.24%	83.76%
Bunny	16.55663		6.901847		7.837922	
	0.50%	99.50%	17.95%	82.05%	8.70%	91.30%
Phone	48.922801		16.46929		19.722395	
	0.17%	99.83%	12.98%	87.02%	8.24%	91.76%

Table 3. Exponential test

As we can see in both tests, the bigger the mesh is, the worse the results of the model presented here will

be, as compared to MTS [Bel03a]. This is because the drawing algorithm must discard repeated vertices, which involves an extra cost that is proportional to the number of vertices in the mesh to be processed.

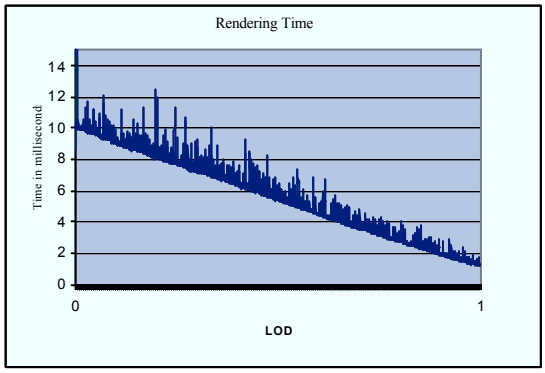

Table 4. Visualisation time for Standford bunny model in a linear test.

5. CONCLUSIONS & FUTURE WORK

The model presented here offers the following features: it is easily implemented, and is efficient and fast.

In summary, it has a similar cost that is lower than the models it was compared with. Moreover, its level of detail recovery cost is very small compared to strip-based models like MTS [Bel03a] and it is similar to that of triangle-based models such as PM [Hop96a]. Finally, rendering times are quite good, considering that the time spent discarding repeated vertices increased this time. Thus, the bigger the meshes are, the more this cost also increases. This is an item to be improved in this model which would increase its performance to a notable degree.

Another characteristic to be improved would be length of strips, which is now static and ought to be dynamic, thus discarding repeated vertices directly in the data structure instead of using the drawing algorithm would improve drawing, decreasing storage cost too.

6. ACKNOWLEDGMENTS

This work was supported by the Spanish Ministry of Science and Technology grants TIC2001-2416-C03-02 and TIC2002-04166-C03-02, and FEDER funds.

7. REFERENCES

[Bel03a] O. Belmonte, I. Remolar, J. Ribelles, M. Chover, M. Fernández. Efficient Use Connectivity Information 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.

[Els97a] El-Sana J, Azanli E, Varshney A. Skip strips: maintaining triangle strips for view-dependent rendering. In: Proceedings of Visualization 99, 1999. p.131-7.

[Eva96a] 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

[Gar97a] M. Garland, and P. Heckbert, Surface Simplification Using Quadratic Error Metrics, Proceeding of SIGGRAPH '97, 209-216, 1997.

[Gar99a] : M. Garland, Multiresolution Modelling: Survey & Future Opportunities. State of the Art Reports of EUROGRAPHICS '99, 111-131, 1999.

[Hop96a] : H. Hoppe. Progressive Meshes. Computer Graphics (SIGGRAPH), 30:99-108, 1996.

[Hop97a] Hoppe H. View-dependent refinement of progressive meshes. In: Procedeeding of SIGGRAPH 97, 1997. P.189-98.

[Por03a] Massimiliano B. Porcu, Riccardo Scateni. An Iterative Stripification Algorithm Based on Dual Graph Operations. EUROGRAPHICS 03.

[Pup97a] Puppo E, Scopigno R. Simplification, LOD and multiresolution-principles and applications. Tutorial notes of EUROGRAPHICS 99, vol. 16, no. 3, 1997.

[Rib99a] J. Ribelles , M. Chover, A. Lopez and J. Huerta. A First Step to Evaluate and Compare Multirresolution Models, Short Papers and Demos EUROGRAPHICS '99, 230-232, 1999.

[Rib00a] J. Ribelles, A. López, I. Remolar, O. Belmonte, M. Chover. Multiresolution Modelling of Polygonal Surface Meshes Using Triangle Fans. Proc.of 9th DGCI 2000, 431-442, 2000. ISBN 3-540-41396-0.

[Rib02a] J. Ribelles, A. López, Ó. Belmonte, I. Remolar, M. Chover, Multiresolution modeling of arbitrary polygonal surfaces: a characterization, Computers & Graphics, ISBN/ISSN 0097-8493, vol. 26, num. 3, pp. 449-462, 2002.

[Sha03a] Michael Shafae, Renato Pajarola. DStrips: Dynamic Triangle Strips for Real-Time Mesh Simplification and Rendering. Proceedings Pacific Graphics Conference, pp. -, 2003

[Sha03b] Pajarola publications web http://www.ics.uci.edu/~pajarola/posters/DStrips.pdf. (PDF Format).

[Ste01a] A. James Stewart: Tunneling for Triangle Strips in Continuous Level--of--Detail Meshes. Graphics Interface 2001: 91-100.