QUALITY STRIPS FOR MODELS WITH LEVEL OF DETAIL

Oscar E. Ripollés
Dept. Lenguajes y Sistemas Informáticos
Universitat Jaume I
12071 Castellón (Spain)
oripolle@sg.uji.es

Miguel Chover
Dept. Lenguajes y Sistemas Informáticos
Universitat Jaume I
12071 Castellón (Spain)
chover@uji.es

Francisco Ramos
Dept. Lenguajes y Sistemas Informáticos
Universitat Jaume I
12071 Castellón (Spain)
jromero@uji.es

ABSTRACT
Multiresolution models are widely employed in computer graphics applications in order to reduce the traffic of information between the CPU and the GPU. The present tendency towards the usage of triangle strips in these models is based on its low cost and high rendering speed. But using this primitive poses the problem of the degeneration of the strips as the level of detail changes. Degenerated triangles are those that have no mathematical area and imply sending information for triangles that will not be rendered. We present a strip generation algorithm to solve this problem, where strips are constructed in such a way as to maintain their quality through all levels of detail.

KEY WORDS
Stripification, simplification, multiresolution modeling.

1. Introduction

A common way to deal with the problem of working with the current polygonal models, which have a high and growing complexity, is the use of multiresolution modeling techniques. According to Garland [1], a multiresolution model represents an object through a set of approximations at a different level of detail and allows us to recover any of them on demand. The first multiresolution models that were developed were based on a relatively small number of approximations (normally between 5 and 10) [2] and were known as discrete multiresolution models. Later, continuous multiresolution models appeared with the aim of improving discrete models, offering a wide range of different approximations to represent the original object. Continuous multiresolution models are widely used because they are capable of solving the problems of interactive visualization, progressive transmission, geometric compression and variable resolution. A comprehensive description of multiresolution models can be found in [3].

Multiresolution models allow us to reduce the amount of geometry information sent to the graphics system, which results in an improvement in performance. The use of triangle strips in these models offers further improvement, since it adds a compact representation of the connectivity existing in a triangle mesh and enables faster rendering.

The main problem of multiresolution models based on strips arises when, starting from a set of strips representing the initial mesh at maximum detail and applying the successive simplifications, the strips start to include a large quantity of degenerated triangles, repeated vertices and unnecessary edges. An example of these low-quality strips can be observed in Figure 1, where the strip in the middle is collapsed after two simplification steps, where edges *0, 3* and *1, 2* are also collapsed.

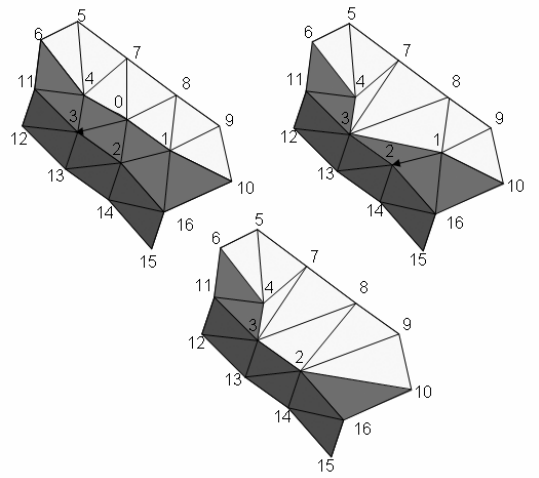

Figure 1. Collapse of a strip.

One possible way to overcome this problem is to use strips which are dynamically generated for each level of detail. Research has been conducted on this approach, and it is possible to find methods of building and maintaining a good set of triangle strips like the one proposed by Stewart [4], and also multiresolution models based on dynamic strips [5][6]. But the additional cost involved in generating the strips for every level of detail is high; therefore the use of static strips, despite their limitations, can turn out to be more suitable.

The work presented in this article proposes an algorithm for building triangle strips for static models which avoids working with low quality strips. It constructs the strips from the minimum to the maximum level of detail following the simplification sequence, while maintaining the original appearance of the 3D model.

The article presents the following structure. Section 2 contains a study of the work previously carried out about strip search and multiresolution models based on this primitive. In section 3 the proposed method is detailed. Section 4 offers a comparative study of our algorithm against other possible solutions. Lastly, in section 5 the results obtained are commented and future lines of work are outlined.

2. Previous work

2.1. Stripification methods

The use of the triangle strip primitive allows us to greatly accelerate the visualization of geometry.

Although finding an optimum set of strips from a given triangulation is an NP-complete problem [7], there are different solutions which, though not optimum, maximize its performance following diverse criteria.

Among the many studies carried out we can highlight the methods introduced in [8][9][10] to generate strips in a static way, as well as the one proposed by Stewart [4] for the dynamic generation of strips. The suggested algorithms show differences in generation and rendering speed, in the use of memory or in the number of strips generated, which make them more suitable for a specific use. It is also important to comment on the studies which make optimum use of the vertex cache. In this regard, methods such as Hoppe's [11] or the one devised by Nvidia [12] have appeared in recent years. The company referenced before has created its own library in order to find strips that derive the maximum benefit from vertex caches and from the spatial locality of vertex buffers.

Finally, mention should be made of the algorithm proposed by Belmonte *et al.* [13], which, as the method presented in this article, also considers the generation of strips following a simplification criterion. But in this case, as in the rest of the algorithms, the generation starting from the maximum level of detail and its subsequent simplification causes degeneration of the strips obtained at levels of low detail.

2.2. Multiresolution methods based on strips

One of the first models to benefit from the triangle strip primitive was the one presented by Hoppe [5], known as Progressive Meshes and included in Microsoft's DirectX library. This model uses triangles during the change of the level of detail but it constructs the strips before the visualization. Later, El-Sana *et al.* [14] presented Skip Strips model, which was the first model to maintain a data structure to store the strips that avoided the need to calculate them in real time. But this model still uses triangles to adjust the geometry at each level of detail.

The MTS model [15] uses triangle strips both as the storage and the visualization primitive. It consists of a set of multiresolution strips, each of which represents a triangle strip and all its levels of detail; only the ones that are modified when changing the level of detail are updated before being rendered. Some time later Dstrips [6] appeared, which is a method that tries to maintain the strips initially calculated, modifying the existing ones and searching for new strips only when a specific zone of the model requires it. Recently Lodstrips [16] has been also presented, which uses triangle strips both in the data structure and the visualization process. It is easy to implement and it is efficient and fast in extracting the level of detail, which enables soft transitions between the different levels of detail to be achieved.

3. Our approach

The objective of the algorithm we present is to find triangle strips that are optimum for multiresolution models. With the method we propose, we intend to avoid the strips to be cut when simplified.

With this intention, we start out with the mesh simplified to the minimum level of detail, which means we may start with just a few triangles. For every step of our algorithm we will need to know the vertices that split, the two new triangles that appear and the set of existing triangles that must be modified. It is possible to collect all this information during the simplification process of the original mesh.

We can consider two main cases. The first of them, shown in Figure 2, represents a refinement along a border edge between two strips. In this situation, two triangles will be modified and a total of four new vertices for two new triangles will be inserted, as each strip will need a *swap* operation.

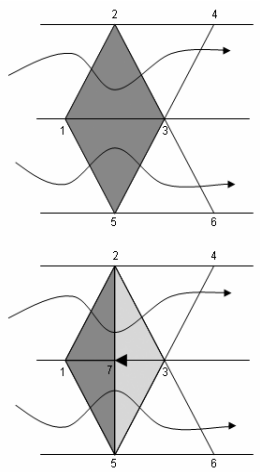

Figure 2. Vertex split along a border edge between two strips.

In this way, following the example in Figure 2, strips will be initially made up of vertices *1, 2, 3, 4* and *1, 5, 3, 6*. After the split, they will be made up of vertices *1, 2, 7, 2, 3, 4* and *1, 5, 7, 5, 3, 6*. We should mention that it might be possible to find this case with just one strip, this only being possible if the strip includes a fan.

The second case can be observed in Figure 3. It shows a split along an edge which is not a border. This makes it impossible to add the two new triangles into the existing strip without resorting to degenerated triangles that would increase the number of vertices required. In this case it is necessary to create a new strip, since we will be able to insert only one of the new triangles into the existing strip. The insertion of these two new triangles will involve a rise of five units in the total number of vertices. Thus, the strip that was initially made up of vertices *1, 2, 3, 4, 5, 6* will now contain edges *1, 2, 3, 2, 7, 4, 5, 6* and a new strip will appear with vertices *3, 7, 5*.

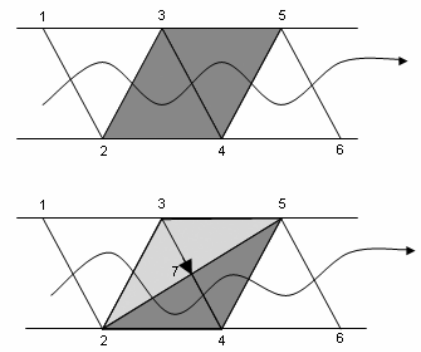

Figure 3. Edge expansion along a non-border edge inside a strip.

With the two general cases introduced, the method to build strips presented in this article will be similar to the one presented in Algorithm 1. The two new triangles share an edge with one of the triangles that will be modified, and it will therefore be through this edge that we will locate the triangle or triangles where we will be able to insert them. Once the edge is found, we will simply insert the new triangle taking care to choose the right side of the edge. If we do not find that edge, we will be obliged to create a new strip. Finally, we will always have to check that all the modified triangles have been changed correctly. We must respect all the changes implied in each step, since otherwise we will not obtain the correct polygonal model when we reach the maximum level of detail.

3.1. Optimizations

With the algorithm proposed, most steps involve the insertion of four new vertices for the two new triangles. These four insertions allow us to obtain a 30% saving with respect to the three vertices per triangle that would be.

```
if find_edge() do
    choose_side();
    insert_triangle();
else do
    create_strip();
check_modifications();
```

Algorithm 1. Strip generation algorithm.

necessary if we represented the model with a triangle mesh To improve these results, the algorithm has been extended so that, in each step, no repeated vertices or edges or unnecessary edges are inserted. Furthermore, it is possible to improve the results if, each time we must create a new strip, we try to insert the new triangle at the end of an already existing strip.

On many occasions we have no choice but to insert a new triangle as a new strip. In successive iterations we may have to add a new triangle next to this one. But, depending on how we have inserted it, we will be able to do the new insertion or not. This is due to the fact that, when inserting a new triangle, one of its three edges will not be explicitly reflected on the strip and then we will not be able to find it in the search for edges in our algorithm. In order to avoid this situation as many times as possible, we have developed a function that predicts the usefulness of the three edges. This prediction is carried out by following their evolution throughout the remaining refinement steps. With this information we will be able to decide which of the three edges is less useful when it comes to inserting the new triangle as a new strip. At this point, we have to choose whether it is better to eliminate the edge that will be used sooner, or the one that will be used later. Our experiments have proven that penalizing the edge we will use sooner offers better results, since it allows the new triangles to be inserted into strips in the last steps of the process.

4. Results

In order to analyze the strips generated as a result of the algorithm presented here, we have conducted a study of the vertices sent to the graphics card for different levels of detail. These data have been compared with those obtained using a simple triangle mesh for each level of detail and with a multiresolution model based on strips that uses a simplification method involving edge collapse, such as Skip-Strips [14], MTS [15] or LodStrips [16]. The experiments were carried out using Windows XP on a Dell PC with a processor at 2.8 Ghz, 1 GB RAM and an Nvidia GeForce 6600 graphics card with 256MB RAM.

	Triangles	SMM	Strips
Cow (5804 triangles)	25.272.930 (100%)	15.752.211 (62.3%)	13.777.800 (54.5%)
Al Capone (7124 triangles)	38.127.687 (100%)	23.158.015 (60.7%)	20.883.500 (54.7%)
Bunny (69451 triangles)	3.619.981.407 (100%)	2.163.219.828 (59.7%)	1.976.880.000 (54.6%)

Table 1. Results in total number of vertices sent going from the minimum to the maximum level of detail.

Figures 4, 5 and 6 offer the obtained results for three different polygonal models. They show the number of vertices sent to the graphics processor for each level of detail, considering 100% as the maximum detail and 0% as the minimum, although in the tests 10% was taken as the minimum since a lower level of detail would entail complete loss of the original shape of the 3D model. It should be pointed out that the information marked in the figures with the name *Triangles* refers to a model that uses triangle meshes, *SMM* is an abbreviation of strip-based multiresolution model and *Strips* are the result of the algorithm proposed in this article.

As we expected, this algorithm sends fewer vertices than a triangle mesh. It can be observed that for high levels of detail the example multiresolution model sends fewer vertices. But if we consider the total number of vertices necessary to go through all the levels of detail, from the minimum to the maximum, our method involves less information traffic, as shown in Table 1. In this way, for more than 60% of the levels the algorithm presented sends less geometric information to the GPU.

In Figure 7 the resulting stripification of the cow model using our algorithm is presented. In addition to the strips representing the model at maximum level of detail, two more images taken during the process are also offered, for a 33% and a 66% of the total detail.

5. Conclusions and future work

We have presented a new method for strips generation in which we obtain a set of triangle strips that will maintain its quality throughout the simplification process. We improve on the results offered by previous stripification algorithms, since all of them offer low quality for levels of coarser detail. In contrast, our algorithm needs a larger number of triangle strips at levels of high detail. But, in general, the total number of vertices covering from the minimum to the maximum level of detail is about 15% lower than the multiresolution model ours was compared with and this means a saving of around 50% with respect to the original triangle mesh.

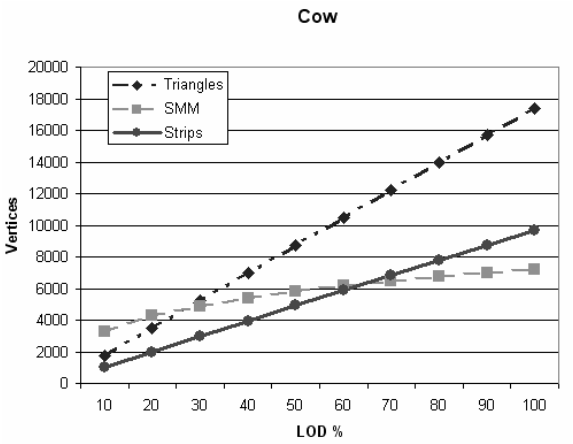

Figure 4. Results for the cow model.

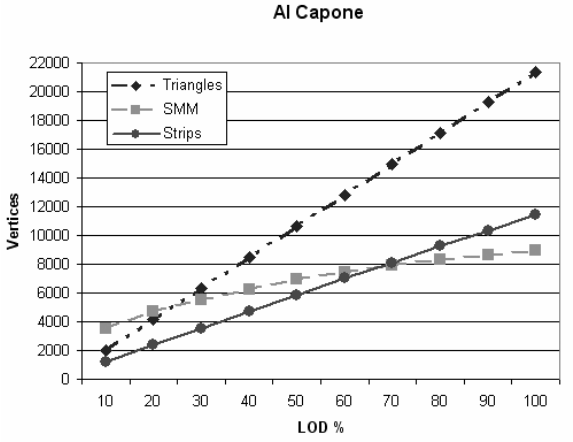

Figure 5. Results for the Al Capone model.

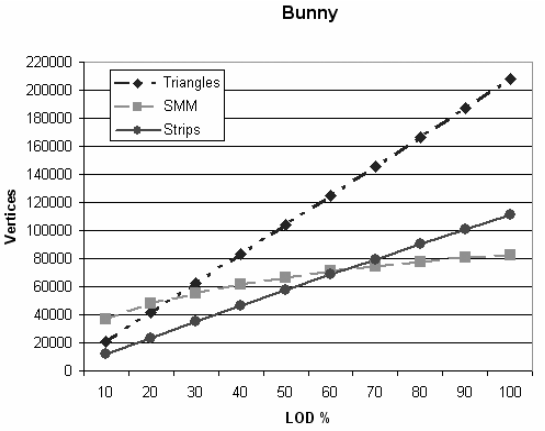

Figure 6. Results for the bunny model.

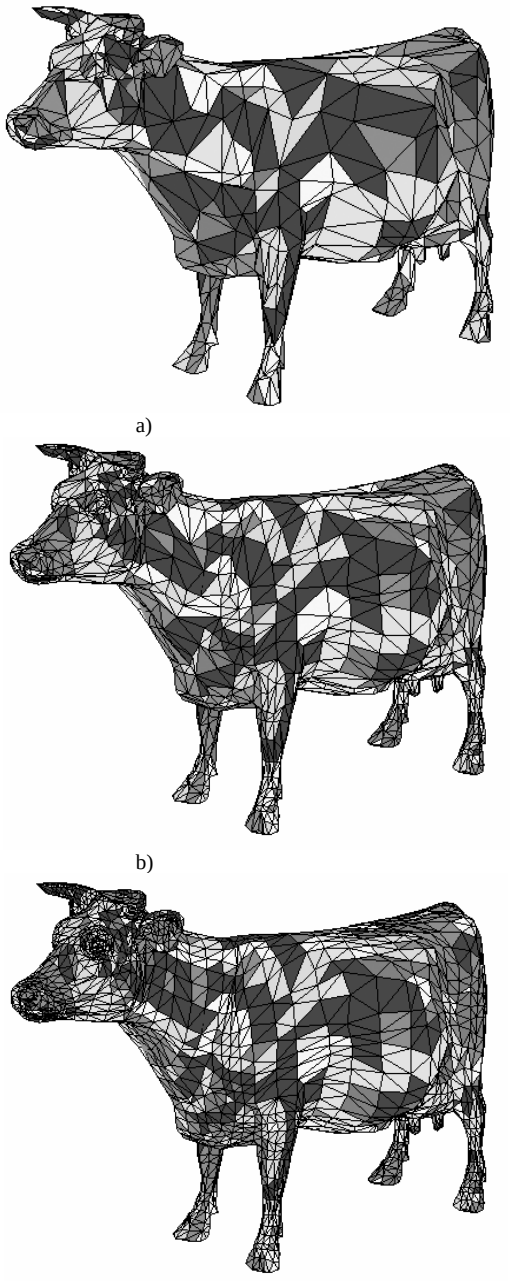

Figure 7. Stripification of the cow model for a) 33% b) 66% and c) 100% of the maximum detail.

Through the results it can be observed how, as the level of detail is reduced, the strips used by the multiresolution model worsen, reaching a point near 20% of detail, where it is even better to use the triangle mesh instead of the strips the model offers. This lends further support to the idea that has encouraged the investigation of this new algorithm, which avoids working with low quality strips.

From the results obtained we can also infer that our design loses quality as the models increase in size. In many applications, such as games, it is usual to work with models which are not as complex as the ones analyzed, where it is easy to offset this low polygonal complexity with a correct treatment of illumination or other aspects of the visualization of geometry.

As a guideline in our future work, it would be interesting to focus a subsequent study on decreasing the strips generated for high levels of detail. In any case, we must also take into account that the longest strips are not always better, as was proved by Hoppe [11]. Thus, the correct management of the vertex cache is a very important issue for possible improvements in the algorithm. In the same way, we consider that a significant improvement can be achieved by utilizing a prediction system that better fits the evolution of simplification, since this simple prediction method already offers a 5% improvement in the number of vertices sent. Finally, it would be interesting to combine the optimization of both high and low levels of detail in the same search method. With this aim, in future studies it would be advisable to consider the search for strips starting from an intermediate level of detail.

6. Acknowledgements

This work has been supported by the Spanish Ministry of Science and Technology (TIN2004-07451-C03-03 and FIT-350101-2004-15), the European Union (IST-2-004363) and FEDER funds.

References:

[1] M. Garland, Multiresolution modeling: Survey &Future opportunities, *State of the Art Reports of EUROGRAPHICS'99*, 1999, 111-131.

[2] Funkhauser et al., Management of large amounts of data in interactive building walkthroughs, *Proceedings of the 1992 symposium on Interactive 3D graphics*, 11-20.

[3] J. Ribelles, A. López, O. Belmonte, I. Remolar, M. Chover, Multiresolution modeling of arbitrary polygonal surfaces: a characterization, *Computers & Graphics, 26*(3), ISSN 0097-8493, 2002, 449-462.

[4] J. Stewart, Tunneling for Triangle Strips in Continuous Level-of-Detail Meshes, *Graphics Interface*, 2001, 91-100.

[5] H. Hoppe, Progressive messhes, *ACM SIGGRAPH 1996*, 99-108.

[6] M. Shafae, R. Pajarola, Dstrips: Dynamic Triangle Strips for Real-Time Mesh Simplification and Rendering, *Proceedings Pacific Graphics Conference 2003*.

[7] F. Evans, S. Skiena, A. Varshney, Efficiently generating triangle strips for fast rendering, *Technical report, Department of Computer Science, State University of New York at Stony Brook*, Stony Brook, NY, USA, 1997 11794-4400.

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

[9] K. Akeley, P. Haeberli, D. Burns, *tomesh.c: C Program on SGI Developer's Toolbox CD*, 1990.

[10] X. Xiang, M. Held, P. Mitchell, Fast and Effective Stripification of Polygonal Surface Models, *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[11] H. Hoppe, Optimization of Mesh Locality for Transparent Vertex Caching, *ACM SIGGRAPH 1999*, 269-276.

[12] C. Beeson, J. Demer, *Nvtristrip, library version*, Software available via Internet website. http://developer.nvidia.com/view.asp?IO=nvtristrip_library. January 2002.

[13] O. Belmonte, J. Ribelles, I. Remolar, M. Chover, Búsqueda de tiras de triángulos guiadas por un criterio de simplificación, *Actas del X Congreso Español de Informática Gráfica (CEIG 2000)*, ISBN/ISSN 84-8021-314-0, Spain, 51-64.

[14] J. El-sana et al., Skip Strips: Maintaining Triangle Strips for View-dependent Rendering, *Proceedings of Visualization 99*, 131-137.

[15] O. Belmonte, I. Remolar, J. Ribelles, M. Chover, M. Fernández, C. Rebollo, Multiresolution Triangle Strips. *Proc. of Visualization, Imaging and Image Processing (VIIP 2001)*, ISBN/ISSN 0-88986-309-1, Spain, 182-187.

[16] 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, 107-114.