LodStrips: Level of Detail Strips

J.F. Ramos and M. Chover

Departamento de Lenguajes y Sistemas Informáticos
Universitat Jaume I, Campus de Riu Sec, 12071, Castellón, Spain
{jromero,chover}@uji.es

Abstract. Meshes representation at different levels of detail is an important tool in the rendering of complex geometric environments. Most works have been addressed to the multiresolution model representation by means of triangle meshes. Nowadays, models that exploit connectivity have been developed, in this paper a multiresolution model that uses triangle strips as primitive is presented. This primitive is used both in the data structure and in the rendering stage, decreasing the storage cost and accelerating the rendering time. Model efficiency is measured by means of a set of tests and results compared to Progressive Meshes and Multiresolution Triangle Strips multiresolution models, obtaining better rendering times and spatial cost.

1 Introduction

One of the main problems in graphics applications is the bottlenecks that take place in the graphics pipeline. These bottlenecks reduce the performance of the application and can vary even from frame to frame. The identification and elimination of these bottlenecks will be fundamental for the optimization of the application.

In each stage of the rendering process from the CPU to the GPU, there are different locations where these problems can appear. If the problem is analyzed from the point of view of geometry, the basic drawback is how to draw a huge number of triangles per frame. In this case, the problem is the number of vertices that are sent to the GPU. The traditional solution to this problem has been to use discrete level of detail (LOD), in an attempt to avoid overloading the CPU. Nevertheless, the use of discrete LODs has the disadvantage of popping and the need to use blending techniques.

In this paper, we present a continuous multiresolution model, called LodStrips, which has the following characteristics:

- *Continuity*. Transitions between levels of detail are smooth. The changes mean eliminating or adding one vertex.
- *Connectivity exploitation*. The model is based on the use of triangle strips. This leads to reduction in the storage and rendering costs.
- *Fast extraction*. It avoids the intensive use of the CPU that usually takes place with the continuous multiresolution models.
- *Cache use*. The use of strips means having at least one cache of two vertices.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp. 107–114, 2004.

2 Previous Work

Continuous multiresolution models are widely used because they are able to solve the problems of interactive rendering, progressive transmission, geometric compression and variable resolution. These models have been developed to represent, chiefly, triangle meshes. A characterization of these models can be found in [10]. Nevertheless, at the present time, some of the models presented exploit connectivity information using primitives like triangle strips or triangle fans. Fast rendering and a smaller storage cost can be achieved using these primitives. The rendering time decreases when fewer vertices are sent to the GPU and the connectivity of the mesh is stored implicitly.

One of the first models to use triangle strips is VDPM by Hoppe [6] After calculating the set of triangles to be rendered, this model performs an on-the-fly determination of the strips to be rendered. This is a time-consuming task but the final rendering time is reduced because triangle strips are faster than triangles.

Later, El-Sana et al. introduces the Skip-Strips model [2]. This is the first model to maintain a data structure to store strips, thus avoiding the need to calculate them on-the-fly.

Ribelles et al. introduced the MOM-Fan[9] This is the first model that no longer uses triangles, but instead another primitive that exploits connectivity. 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 degenerated triangles used in the representation. Another drawback to the model is that the average number of triangles in each triangle fan is small.

Following this approach, MTS by Belmonte et al. appeared. This is a model that uses the strip primitive in the storage and in the rendering stage [1]. The model is made up of a collection of multiresolution strips. Each multiresolution strip 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 rendering.

Recently, some works based on the triangle strip primitive have been presented. These focus on the dynamic simplification of the triangle strips for each demanded LOD. The model by Shafae et al. called DStrips [11] manages the triangle strips in such a way that only those triangle strips that are being modified are processed, while the rest of the triangle strips in the model remain unmodified. This updating mechanism reduces the extraction time. However, results published from this work still show a high extraction time.

Another approach to the use of triangle strips in a multiresolution model is the work carried out by A. James Stewart [12], and extended by Porcu [7]. This work uses a tunneling algorithm to connect isolated triangle strips, thus obtaining triangle strips with high 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.

Improvements of multiresolution models are applied in many ways. In [15] vertex connectivity exploitation is applied to implement a multiresolution scheme and in [14] a method is applied to maximize vertex reuse.

3 The LodStrips Model

The LodStrips model represents a mesh as a set of multiresolution strips. We denote a triangle strip mesh M as a tuple (V;S), where V is a set of vertices v_i with positions $v_i \in R^3$, and S is a collection of sub-triangulations $s^1,\ldots,s^m$, so each $s^i \in S$ is an ordered vertex sequence (1) also called a strip

$$\left\{s_1^i \cdots s_q^i\right\} \qquad S = \left\{\begin{array}{c} s_1^1 \cdots s_k^1 \\ \vdots \\ s_1^m \cdots s_r^m \end{array}\right\} \qquad V = \left\{v_1 \ldots v_n\right\} \tag{1}$$

Each row inside the S matrix represents a triangle strip. After some modifications, this matrix will be adapted to become a multiresolution triangle strip data structure in order to be used in our model. In this way, this data structure will change during level of detail transitions, as described in the sections below.

The model has been built in order to optimize data access as well as the vertices sent to the graphics pipeline. In this way, it manages the triangle strips both in the data structure and in the rendering stage.

A set of vertices with their 3D coordinates and a set of multiresolution strips are needed to support the multiresolution model. Moreover, an auxiliary structure is used to improve level of detail transitions.

3.1 Data Structures

The purpose of the data structure is to store all the information necessary to recover every level of detail on demand. Three data structures are used: lVerts, lStrips and lChanges.

We denote a lVerts structure as a set V which contains an ordered vertex sequence and where each $v_i \in V$ consists of four items (2). The first three items are vertex positions: $(x_i, y_i, z_i) \in R^3$ and the last one, v_{k_i}, is the vertex into which v_i collapses, where $k > i$.

$$V = \left\{v_1, \ldots, v_n\right\} \qquad v_i = \left(x_i, y_i, z_i, v_{k_i}\right) \tag{2}$$

Thus, the lVerts data structure stores 3D coordinates and information about vertex simplification of each vertex in the mesh.

In order to collapse a vertex onto another, two kinds of simplification can be applied: external and internal edge collapses. External edge collapses consist of a vertex simplification where the destination vertex is on the external edge of the strip. Internal edge collapses are applied when the destination vertex is on the opposite edge. Simplification is achieved by means of two external vertex collapses.

Transitions between levels of detail involve vertex collapses. This information is stored in the lVerts data structure and when a vertex v_i has to be collapsed, it is replaced by v_{k_i} in every strip where it appears.

V is ordered according to its simplification order, that is, v_0 will be the first vertex to collapse, v_1 will be the second, and so on. Assuming that a vertex simplification is a level of detail change, when a transition from LOD i to LOD i+1 is demanded by the application, vertex v_i is replaced by v_{k_i} in all occurrences of that vertex and in every multiresolution triangle strip, in other words, in the data structure lStrips. Thus, an initial mesh M^1 can be simplified into a coarser M^n by applying a sequence of successive vertex collapse operations.

The sequence of vertex collapses is selected from simplification algorithms, since it determines the quality of the approximating meshes.

The multiresolution strip set is stored by the lStrips data structure. It consists of a collection L, where each $L^i \in L$ is an ordered vertex sequence, which denotes a multiresolution strip.

$$L = \begin{Bmatrix} v_1^1 & \dots & v_r^1 \\ \vdots & \vdots & \vdots \\ v_1^m & \dots & v_t^m \end{Bmatrix} \tag{3}$$

Each row $L^i \in L$, or each strip in the L collection, changes dynamically with vertex collapses and with strip resizing.

Vertex collapses are performed by replacing vertices in the data structure L by others that simplify them. These collapses can give rise to situations where repeated sequences appear in the data structure and these repetitions have to be removed, which involves resizing some strips.

Thus, as vertex collapses are applied, the lStrips data structure will become smaller, which allows us to have a very simple data structure for coarse levels of detail.

The lodStrips model incorporates a new data structure, lChanges, which allows us to quickly recover the positions of the vertices that are changed in each level of detail transition; this also allows the quick removal of consecutive vertex repetitions.

We denote a lChanges data structure as a level of detail ordered set, C, where each tuple c_j^i has the structure (l_j^i, k_j^i), where $l_j^i \in L$, which represents a position in L and k_j^i is another scalar that determines whether l_j^i values are for collapsing a vertex or for removing a set of consecutive vertices.

$$C = \begin{Bmatrix} c_1^1 & \dots & c_s^1 \\ \vdots & \vdots & \vdots \\ c_1^m & \dots & c_t^m \end{Bmatrix} \tag{4}$$

This data structure increases model performance because it allows us to quickly apply level of detail changes between transitions. Without this data structure it would be very expensive to apply these changes.

3.2 Algorithms

Multiresolution models need algorithms to be able to support multiresolution capabilities. The LodStrips model and most multiresolution models have two main algorithms to do these tasks, i.e. a level of detail recovery algorithm and a drawing algorithm. We assume the rendering stage to be a stage that contains these two algorithms, which are applied in a sequential order, first extraction and then drawing.

The level of detail recovery algorithm goes into action when a level of detail change is induced by the application. Then, data structure C is traversed from $C^{currentLOD}$ to C^{newLOD}, applying changes stored in each tuple Cij ∈ C, where i is in the interval [currentLOD , newLOD]. It is important to notice that, depending on whether the level of detail is bigger or smaller than the current one, splits or collapses will be applied to the lStrips data structure, although the information stored in c^i_j referring to collapses is also used to perform splits. The pseudo-code is shown below.

Level of detail recovery algorithm.

```
for lod=currentLOD to newLOD
    if newLOD>currentLOD   //To a more coarse mesh
      for change=lChanges.Begin(lod) to lChanges.End(lod)
        if (change.isCollapse()) then
          lStrips.Collapse(lod,change);
        else
          lStrips.Resize(lod,change);
    else  //To a more detailed mesh
      for change=lChanges.Begin(lod) to lChanges.End(lod)
        if (change.isSplit()) then
          lStrips.Split(lod,change);
        else
          lStrips.Resize(lod,change);
```

After the level of detail recovery algorithm has processed multiresolution strips the drawing algorithm takes over, traversing each strip to obtain their vertices in order to send them to the graphics system.

3.3 Model Performance

LodStrips model consists of data structures, with a pre-process that fills them, and algorithms for online extraction of the level of detail demanded:

- *Stripification*. Using the STRIPE algorithm [3] the lVerts data structure is built and lStrips filled with the highest level of detail.
- *Simplification*. We get the vertex collapsing order by means of the QSLIM algorithm [4].
- *Arrangement*. Once we have the vertex collapsing order, data structures must be adapted to the simplification order obtained from QSLIM.

- *Collapse*. For each v_i E V, v_{k_i} is calculated, as described in section 3. This item represents the vertex v_i to be collapsed. It is calculated taking into account the results of the simplification process.

4 Results

The LodStrips model has been submitted to several tests, all of which are aimed at evaluating the rendering time in a real time application.

All these tests were carried out to PM [5], MTS [1] and LodStrips, and results were compared. The first model has been and still is a reference model in the multiresolution world. The second is the most recent multiresolution model that makes complete use of multiresolution strips.

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

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

Table 1 shows triangle mesh costs of the objects used in the tests and the sizes of the three models compared. It can be seen how the model presented here has a spatial cost that is lower than the rest of the models compared. This allows more objects to be placed in memory if necessary.

Table 1. Spatial cost comparison in Mb.

Mesh	Tr. Mesh	PM	MTS	LS	Ratio PM	Ratio MTS	Ratio LS
Cow	0.100	0.272	0.252	0.186	2.7	2.5	1.9
Bunny	1.193	3.282	2.963	2.111	2.8	2.5	1.8
Phone	2.850	7.863	6.765	4.844	2.8	2.4	1.7

Tests designed to compare multiresolution models follow the ones introduced by [8]. The tests carried out are the linear test and exponential test.

The linear test consists of extracting the LODs in a linear and proportionately increasing or decreasing way. The Exponential test consists of extracting LODs in an exponential way, that is, in the beginning it extracts very distant levels of detail and, later, it extracts closer levels.

The following tables show the results of applying the linear and exponential tests to models PM [5], MTS [1] and the one presented here, LodStrips.

As can be seen in Table 2, corresponding to the linear and exponential tests, the total rendering time is shown first. The lower part of the table, shows the percentage of this time used in extracting the level of detail and in drawing the resultant mesh.

Table 2. Linear and exponential tests

	LINEAR TEST						EXPONENTIAL TEST					
	PM		MTS		LodStrips		PM		MTS		LodStrips	
	Render (ms)		Render (ms)		Render (ms)		Render (ms)		Render (ms)		Render (ms)	
	% rec	% drw	% rec	% drw	% rec	% drw	% rec	% drw	% rec	% drw	% rec	% drw
Cow	0.917916		0.934682		0.231398		1.234464		0.53519		0.298161	
	6.43	93.57	24.36	57.38	24.36	75.64	5.88	94.12	37.6	62.4	20.73	79.27
Bunny	10.792452		6.304261		3.077063		16.164691		6.998482		4.129842	
	0.59	99.41	21.85	78.15	2.57	97.43	0.49	99.51	17.54	82.46	1.89	98.11
Phone	32.983562		14.812924		8.301228		48.922801		16.735283		11.756625	
	0.24	99.76	16.29	83.71	1.65	98.35	0.17	99.83	12.81	87.19	1.15	98.85

As we can see in both tests, the LodStrips model offers better rendering times than MTS and PM. LodStrips spends a small percentage of time on extracting the level of detail, which leads to good rendering times. In the opposite case, MTS spends quite a lot of time on extraction, and this slows down the total rendering time for this model.

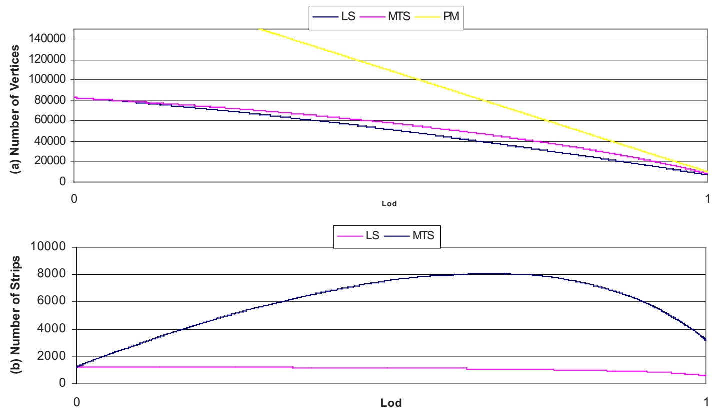

Fig. 1. a) Vertices sent to the graphics system by the bunny object in a linear test, b) Strips sent by MTS and LodStrips model in a linear test for the bunny object.

Vertices sent to the graphics system are directly related to the rendering time. The number of vertices sent by the model can be seen in Figure 1 a).

Strips are a way for organizing vertices that allows us to have a virtual two-vertex cache. As we can see in Figure 2 b), the LodStrips model has a low variation of strips sent, whereas the MTS model has a high variation of them in the progression of levels of detail. In any case, it seems that there is no relation between the vertex sent and the number of strips sent.

5 Conclusions

The LodStrips model offers many advantages and it should be underlined that it is a model with only three simple data structures and it is easy to implement. Moreover, it offers a fast LOD extraction which allows us to obtain smooth transitions between LODs, as well as very good rendering times because extraction is usually an important part of the total rendering time. This model is wholly based on the triangle strips, which leads to an important reduction in storage and rendering costs.

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.

References

1. 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, 2003.
2. El-Sana J, Azanli E, Varshney A. Skip strips: maintaining triangle strips for view-dependent rendering. In: Proceedings of Visualization 99, 1999. p.131-7.
3. 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
4. M. Garland, P. Heckbert, Surface Simplification Using Quadratic Error Metrics, SIGGRAPH'97, 209-216, 1997.
5. Hoppe H. Progressive Meshes. Computer Graphics (SIGGRAPH), 30:99-108, 1996.
6. Hoppe H. View-dependent refinement of progressive meshes. SIGGRAPH, 1997.
7. Massimiliano B. Porcu, Riccardo Scateni. An Iterative Stripification Algorithm Based on Dual Graph Operations. EUROGRAPHICS 03.
8. 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.
9. 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.
10. J. Ribelles, A. López, Ó. Belmonte, I. Remolar, M. Chover, Multiresolution modeling of arbitrary polygonal surfaces: a characterization, Computers & Graphics, vol. 26, n.3 2002.
11. Michael Shafae, Renato Pajarola. DStrips: Dynamic Triangle Strips for Real-Time Mesh Simplification and Rendering. Proceedings Pacific Graphics Conference, 2003.
12. A. James Stewart: Tunneling for Triangle Strips in Continuous Level of Detail Meshes. Graphics Interface 2001: 91-100.
13. L. Velho, L.H. de Figueiredo, and J. Gomes.: Hierarchical Generalized Triangle Strips. The Visual Computer, 15(1):21-35, 1999.
14. A. Bogomjakov, C. Gostman.: Universal Rendering Sequences for Transparent Vertex Caching of Progressive Meshes. Proceedings of Graphics Interface 2001.
15. Leif P. Kobbelt, Thilo Bareuther. Hans-Peter Seidel.: Multiresolution Shape Deformations for Meshes with Dynamic Vertex Connectivity. Computer Graphics Forum. vol. 19, 2000.