Adaptive visualization of 3D meshes using localized triangular strips

M-H MOUSA M-K HUSSEIN
Faculty of Computers & Informatics
Suez Canal University
Ismailia, Egypt.
{mohamed_mousa, m_khamiss}@ci.suez.edu.eg

Abstract:- 3D meshes are the principal representation of 3D objects. They are very powerful in manipulation and easy to visualize. However, they often require a huge amount of data for storage and/or transmission. In this paper, we present an effective technique to stream triangular meshes using enhanced mesh stripificatio algorithm. In fact, stripificatio algorithms are used to speed up the rendering of geometric models because they reduce the number of vertices sent to the graphics pipeline by exploiting the fact that adjacent triangles share an edge. This enables to use our technique to adaptively visualize the 3D models during their transmission. The firs step of the proposed technique is based on storing 3D objects as a set of strips. This encodes the geometry and the connectivity of the input model in a robust fashion. The stripificatio algorithm achieves compression ratios above $61 : 1$ over ASCII encoded formats. Second, the strips are directly sent to the rendering pipeline in accordance with the viewpoint direction resulting in faster transmission and rendering of complex graphical objects. Binary space partitioning, kd-trees, is used to enhance our stripificatio algorithm. Some examples are given to demonstrate the effectiveness of our technique.

Keyworks:- triangle strips, adaptive visualization, kd-tree partitioning, mesh compression.

1 Introduction

Graphics data are widely used in various applications such as video gaming, engineering design, virtual reality and scientifi visualization. Complex models are used to add additional levels of realism to the scene. These models are obtained from various sources such as 3D scanning and modeling software. In order to generate the necessary geometry for real time rendering of these models, complete topology information is necessary for a given mesh of input polygons. Over the years a number of mesh representations have been developed that provide the required topology information, given an arbitrary input mesh. The triangular meshes are considered as the native representation for such 3D models. They provide an effective structure to represent the geometry and the adjacency of the vertices composing the model. Most of these mesh representations are based on the winged-edge representation introduced by Bruce Baumgart [3].

Most applications require compact storage, fast transmission, and efficien processing of 3D meshes. Therefore, many algorithms have been proposed to compress 3D meshes efficientl [12]. In fact, 3D meshes can be composed of hundreds of millions of polygons. The transmission of these polygons over the network is very slow, time consuming and increasing the traffi over the network. There are several compression schemes for triangle meshes that encode either the geometry or the connectivity [18, 17, 13, 6]. They encode the mesh through a compact representation of a vertex-spanning tree and its dual graph. However, they are not adequate for progressive visualization because it is necessary to decode all the geometry firs in order to visualize the model based on its connectivity. On the other hand, for interactive visualization not only the speed at which a triangle mesh can be received is important but also the speed at which it can be displayed. Hence, the bottleneck is the rate at which the data can be sent to the rendering engine. Each triangle of the mesh can be rendered individually by sending its three vertices to the graphics hardware. Then every mesh vertex is processed about six times, which involves passing its three coordinates and optional normal, color, and texture information from the memory to and through the graphics pipeline. One popular approach for fast visualization of these types of models is the conversion of the model into a set of triangles fans. These triangles fans are one of the primitive polygons that are widely supported by graphical hardware and software [14]. A triangle fan describes a set of connected triangles that share one central vertex. This saves storage and processing time since graphics pipeline can take advantage by only performing the viewing transformations and lighting calculations once per vertex. Another more general and powerful approach for fast visualization is the conversion of the 3D model into a triangle strip. A triangle strip is a series of connected triangles, sharing vertices, allowing for faster rendering and more efficien memory usage for computer graphics. They are optimized on most graphics cards, making them the most efficien way of describing an object. There are two primary reasons to use triangle strips:

1. Triangle strips increase code efficien y. After the firs triangle is define using three vertices, each new triangle can be define by only one additional vertex, sharing the last two vertices define for the previous triangle.

2. Triangle strips reduce the amount of data needed to create a series of triangles. The number of vertices stored in memory is reduced from $3N$ to $N + 2$, where N is the number of triangles to be drawn. This allows for less use of disk space, as well as making them faster to load into the RAM.

Such triangle strips [10, 14, 8, 11] are widely supported by today's graphics hardware. The firs algorithm proposed for creating triangle strips is the SGI [2]. For rendering purposes, an optimal stripificatio covers the mesh with as few strips and swaps as possible [10]. Computing an optimal set of triangle strips is an NP-complete problem [9]. Various heuristics for generating good triangle strips have been proposed by Evans et al. [10], Speckmann and Snoeyink [16], and Xiang et al. [21]. Deering [7] reduced the number of strips by introducing the concept of generalized triangular mesh. Vanecek and Kolingerová [19] produced much more lower number of triangle strip. For more details about several triangle strips algorithms, see [20].

Another important issue of triangle stripificatio is mesh compression. Chow [5] presented a mesh compression scheme based on triangle stripificatio to enhance object rendering. This achieves a compression ratio below 37:1 over ASCII encoded formats. Silva and Yadav [15] enhanced the compression ratio to achieve about 40:1 over ASCII encoded formats.

Contribution

In this paper we present a technique to convert a given triangular mesh into a set of triangle strips. Our algorithm can control the number of produced triangle strips. This is done by thresholding the number of triangles in every strip. In additoin, we propose a localization of the triangle strips by using a kd-tree with a splitting condition that ensures the coherence of the local neighborhood visibility of each triangle strip. On the other hand, we associate with every triangle strip S_i a normal vector n_i, where n_i is the average normal vector over all the triangles contained in S_i. The set of strips $\{S_i\}$ in combination with their corresponding n_i's can accelerate the visualization by preventing the strips that conflic with the viewpoint direction from being sent to the graphics pipeline. Our technique achieves a compression ratio over 60:1 over ASCII encoded formats.

The rest of the paper is organized as follows. Section 2 presents the general steps for encoding a given triangular mesh into a single strip file The corresponding reconstruction algorithm is given in Section 3. We explain in Section 4 the generation of multiple triangle strips for a given triangular mesh. Moreover, we show how to use multiple stripificatio in order to accelerate the visualization of the mesh. Some experimental results that demonstrate our approach are given in Section 5. Finally, we conclude in Section 6.

2 Generation of triangle strip

Given a triangular mesh **M**, this mesh can be identifie by the triple (V, E, T), where V is the set of vertices, E is the set of edges and T is the set of triangles. For each vertex $v \in V$, we defin an attribute **deg**(v) as the number of edges incident on v and **N**(v) as the set of vertices in the firs neighborhood around v. Similarly, for each edge $e \in E$, we defin **status**(e)

as the number of triangles sharing the edge e. In the same manner, we defin **status**(f) as the number of triangles that share an edge with f and **neighbor**(f, e) as the neighboring face of f that shares the edge e. The proposed algorithm is summarized in Algorithm 1.

Algorithm 1 Strip fil generation algorithm.

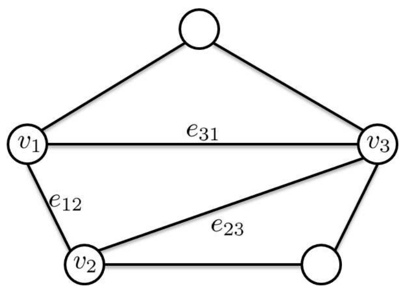
```
Input: Triangular mesh M
Output: Compressed strip fil
1: First vertex selection $v_1$
2: Intermediate vertex selection $v_2$
3: Final vertex selection $v_3$
4: Check edges status of the current selected triangle
5: Check vertex degree of the three vertices $(v_1, v_2, v_3)$
6: Check vertex degree of all vertices that are already presented in the compressed fil
```

The algorithm starts by selecting a candidate vertex v_1. The vertex v_1 should satisfy the following condition:

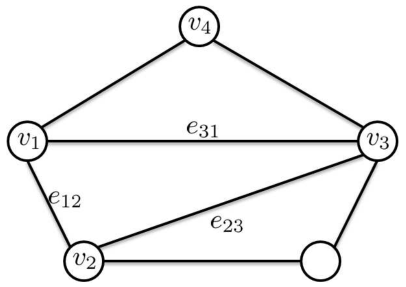
$$\mathbf{deg}(v_1) = \min_{v \in V}\{\mathbf{deg}(v)\} \quad \& \quad \mathbf{deg}(v_1) \neq 0. \quad (1)$$

If there are more than one vertex satisfying the previous condition, we arbitrarily select any one of them and write the coordinates of the selected vertex v_1 inside the strip file Once v_1 is selected, we navigate around $\mathbf{N}(v_1)$ to choose the second vertex v_2 such that **deg**(v_2) is minimum and similarly insert the vertex's coordinates into the strip file Now consider the edge e_{12} which has v_1 and v_2 as endpoints. If e_{12} is a border edge then this edge identifie exactly one face, otherwise the edge e_{12} identifie two faces. In case e_{12} is shared by two faces, we select the face that has minimum **status** and v_3 is selected as the third vertex in that face. On the other hand, in case e_{12} is a border edge, we select v_3 as the third vertex from the face identifie by e_{12}. In both cases, once the vertex v_3 is selected, v_3 is inserted in the strip file After the selection of the firs triangle consisting of v_1, v_2 and v_3, the value of the following attributes are decreased by 1: **deg**(v_1), **deg**(v_2), **deg**(v_3), **status**(e_{12}), **status**(e_{23}) and **status**(e_{31}). Next, we will show how to generate the strip of triangles around the selected triangle.

Without loss of generality, consider firs that the vertices v_1, v_2 and v_3 are as shown in Figure 1(a). We



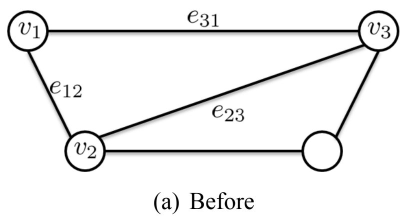
(a) Before



(b) After

Fig. 1: (a) **status**$(e_{23}) > 0$ and **status**$(e_{31}) = 1$. (b) v_4 will be the third vertex of the auxiliary triangle sharing e_{31}.

check firs the value of **status**(e_{23}). If **status**$(e_{23}) > 0$ then we check next if **status**$(e_{31}) > 0$. If **status**(e_{31}) is equal to 1, remember that **status**(e_{31}) is already decreased by 1, then we will cover the triangle adjacent to this edge, see Figure 1(a). Such triangle is called auxiliary triangle or side triangle. This case will be marked using the linking agent '#'. Let v_4 be the third vertex of this auxiliary triangle, see Figure 1(b). This vertex will be inserted into the strip file The status of the edges and the degree of the vertices will be updated and control go back to step 4. If **status**$(e_{31}) = 0$ then there are no more triangles uncovered adjacent to edge e_{31}, see Figure 2. Thus the control will go to step 3 with vertex v_3 renamed to v_2 and vertex v_2 renamed to v_1. Now if the edge status of e_{23} is equal to 0, see Figure 3, then there are no more triangles uncovered adjacent to this edge. We jump then to the edge e_{13}. If the edge status of e_{13} is equal to 1, then we will cover the triangle adjacent to this edge. In fact, the strip creation process will have to change its direction in terms of a leading edge. The leading edge was e_{23} but now it will be e_{31}. This change of direction is marked inside the strip fil by the linking agent '!' and the control goes again to



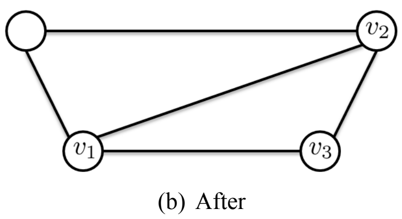
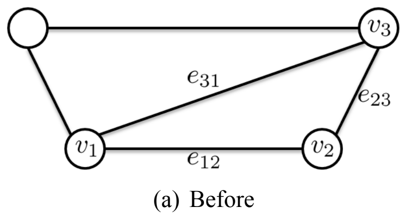


Fig. 2: (a) **status**$(e_{23}) > 0$ and **status**$(e_{31}) = 0$. (b) vertex v_3 renamed to v_2 and vertex v_2 renamed to v_1.



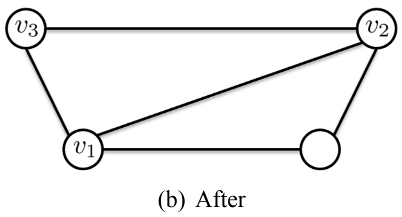
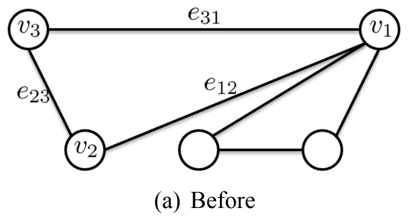


Fig. 3: (a) **status**$(e_{23}) = 0$ and **status**$(e_{31}) = 1$. (b) The leading edge will be e_{31} and its adjacent triangle will be covered.

step three to fin a new vertex v_3.

In case the edge e_{13} has a status equal to 0 then there are no more triangles uncovered adjacent to edge e_{13}. Here the values of **status** for all of the edges e_{12}, e_{31} and e_{23} are equal to 0. In this case, we search for a vertex with a non-zero degree, **deg**, among the vertices v_1, v_2 and v_3. This is done in the following order:

1. If **deg**$(v_1) > 0$ then a linking agent 'A' will be inserted in the strip fil and the algorithm goes to step two with selected vertex as v_1 to fin the new vertices v_2 and v_3 , as shown in Figure 4.



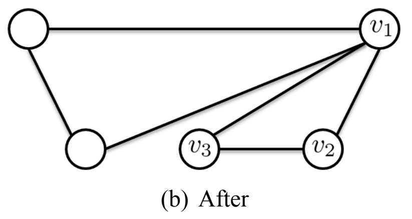
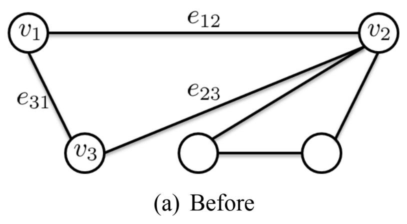


Fig. 4: (a) **status**(e_{23}) = **status**$(e_{31}) = 0$ and **deg**$(v_1) > 0$. (b) v_1 becomes the leading vertex and the control searches for v_2 and v_3.

2. If **deg**$(v_2) > 0$, see Figure 5, then we use the linking agent 'B' and the vertex v_2 is renamed to v_1. The control, as in the previous case, goes to step two to fin the new vertices v_2 and v_3.



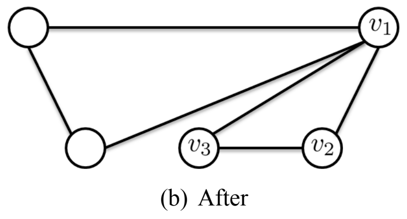
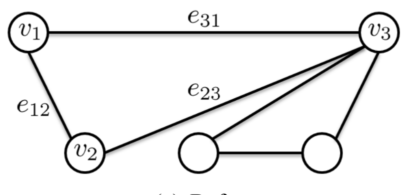


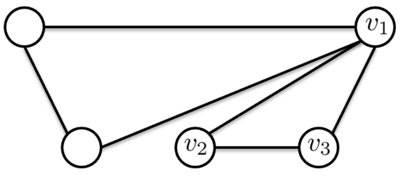
Fig. 5: (a) **status**(e_{23}) = **status**$(e_{31}) = 0$ and **deg**$(v_2) > 0$. (b) v_2 becomes the leading vertex and is renamed to v_1 and the control searches for v_2 and v_3.

3. If **deg**$(v_3) > 0$, see Figure 6, then this case is marked using the linking agent 'C'. The vertex v_3 is renamed to v_1 and the control goes to step two to fin the new vertices v_2 and v_3.

4. Otherwise If **deg**(v_1) = **deg**(v_2) = **deg**$(v_3) =$ 0, then we search for a vertex with non-zero degree among all the vertices that are already appeared previously in the strip file In this case, the found non-zero degree vertex is renamed to v_1 and the linking agent '?' is inserted into the



(a) Before



(b) After

Fig. 6: (a) **status**(e_{23}) $=$ **status**(e_{31}) $= 0$ and **deg**$(v_3) > 0$. (b) v_3 becomes the leading vertex and is renamed to v_1 and the control searches for v_2 and v_3.

strip fil followed by the reference of new vertex v_1. The control, therefore, goes to step two to fin new vertex v_2.

Note that, when writing a vertex in the strip file we write its coordinates if it is the firs time to be appeared in the file Otherwise we write its reference preceded by the linking agent '<'.

For example, consider the tetrahedron which consists of the set of vertices {(0,0,0),(0,0,1),(0,1,0),(1,0,0)} and the set of faces {(1,2,3),(1,3,4),(2,3,4),(2,4,1)}. The application of our technique yields the following strip "0 0 0 0 0 1 0 1 0 #1 0 0 <4 !<1".

3 Reconstruction algorithm

The reconstruction (decompression) algorithm is just the reverse process of the stripificatio algorithm and is summarized in Algorithm 2.

The decompression algorithm starts by initializing the list of vertices with the firs three vertices - v_1, v_2 and v_3 - that appear at the beginning of the file These vertices defin the firs triangle by which the list of faces is initialized. Next, we read the rest of the fil character by character searching for a linking agent or a new vertex.

- If the character is the linking agent '#', this means that we found an auxiliary triangle. In this case, the next token in the fil will be a vertex, say v_4. If vertex v_4 does not exist in the list of vertices then v_4 will be added to the list. The new vertex in combination with v_1 and v_3 reconstruct a new triangle to be inserted to the list of faces.

- If the linking agent is the character '!', this implies that there is a new vertex to be read from the fil and a change in the order between v_3 and v_2, such that v_3 becomes v_2. The newly retrieved vertex will be v_3; v_1, v_2 and v_3 reform the next triangle to be inserted to the list of faces.

- If the linking agent is the character 'A', this implies that there are new two vertices to be read from the fil v_2 and v_3. v_1, v_2 and v_3 reform again the next triangle to be inserted to the list of faces.

- If the linking agent is the character 'B', this means that v_2 becomes v_1, and the next entries in the compressed fil will be two vertices v_2 and v_3. The vertices v_1, v_2 and v_3 form the next triangle to be inserted to the list of faces.

- If the linking agent is the character 'C', this means that v_3 becomes v_1, and the next entries in the compressed fil will be the two vertices v_2 and v_3. The vertices v_1, v_2 and v_3 form the next triangle to be inserted to the list of faces.

- If the linking agent is the character '?' this means that the next entry is a vertex identifie . This vertex becomes v_1 and the next entries in the compressed fil will be two vertices v_2 and v_3. The vertices v_1, v_2 and v_3 form the next triangle to be inserted to the list of faces.

Algorithm 2

```
Input: A strip fil
Output: The corresponding OFF fil
  Initialize the list of vertices by the firs three vertices,
  while not end of the strip fil do
    Read a new character $c$ from the string
    if c is a linking agent then
      see details below
    else
      it is a vertex
    end if
  end while
```

- If the linking agent is equal to '$<$' then we will read a vertex identifie , which is already loaded in the the buffer. The linking agent '$<$' is followed by the index of a vertex that have been already represented in the list of vertices.

- Otherwise this implies a change in the order of vertices v_1, v_2 and v_3, such that v_2 becomes v_1 and v_3 becomes v_2. Next, we will read the coordinates of a new vertex v_3. Vertices v_1, v_2 and v_3 form the next triangle to be inserted to the list of faces.

Note that each time we reed a vertex there are two cases. In the firs case, the vertex is already loaded to the buffer. We therefore, will fin a linking agent '$<$' followed by the index of this vertex inside the buffer. In the other case, the vertex is new. In this case we read its coordinates directly from the file

For example, consider the simple strip given in Section 2, "0 0 0 0 0 1 0 1 0 #1 0 0 <4 !<1". The application of our decompression algorithm decodes this strip fil to original list of vertices={(0,0,0),(0,0,1),(0,1,0),(1,0,0)} and list of faces={(1,2,3), (1,3,4), (2,3,4), (2,4,1)}.

4 Single vs. multiple stripification

The visualization process of a model in any standard format, such as OFF, has to wait until receiving all information about the geometry first This leads to a lot of latency in the visualization process. On the other hand, our algorithm encodes and decodes the geometry and the connectivity of the 3D models in an interwoven fashion. The geometry and the connectivity information are received together which means that the visualization process will start as soon as the reception of the firs three vertices from the strip fil without any latency. Figure 7 shows an example of the progressive visualization of Triceratops model. The model is stripifie as a single strip. The visualization starts immediately once the firs three vertices, at the begining of the strip file are received to the graphics pipeline. Figures 7(a), 7(b), 7(c), 7(d) and 7(e) correspond to the transmission of 20%, 40%, 60%, 80% and 100% respectively of the strip to the graphics pipeline.

The proposed strip fil generation presented in Section 2 generates a single strip for each input model. However, we can modify this algorithm to generate two or more strips for each model. We modify our algorithm, Algorithm 1, by introducing a threshold that control the number of triangles contained in the strip. For example, to generate two strips for a given model, we set the threshold value to $\frac{N_f}{2}$ where N_f is the total number of faces in the model.

Our modifie algorithm generates two types of strips: "independent" and "dependent" strips. Independent strips means that each strip has its own list of vertices. While, on the other hand, dependent strips means that there is only one list of vertices shared for all strips. The difference between dependant and independent strips is that independent strips generates fil with sizes greater than that of dependent strips. However, working with independent strips gives the ability to use the traditional techniques of parallelism to accelerate the reconstruction process. Parallel strip fil generation become more suitable when we manipulate large scale meshes. Carrying out multiple operations or tasks simultaneously enables us to navigate the list of vertices and faces of large scale meshes very quickly. Figure 8 shows an example of the creation of multiple strips for the Gargoyle model. In this example, we use the standard space paritioning using the Kd-tree data strucutre [4] to localize each strip in a single leaf of the kd-tree. The division step is perforfmed as the follolwing. Starting from the initial bounding box of the set of points of the mesh, each cell C_i of the kd-tree hierarchy is split, in addition to thresholding the number of triangles in C_i, if the following condition is not satisfied

$$\forall p \in C_i \rightarrow \vec{n}_p \cdot \vec{n}_i > 0.2 \qquad (2)$$

where $\vec{n}_p$ is the surface normal at the point p and $\vec{n}_i$ is the average normal vector in C_i. This condition ensures that there is no folding inside the cell C_i. Therefore projecting the local points of the cell C_i on the local sphere will keep the coherency of local neighborhood visibility of surface patch contained in C_i. The use the kd-tree with this splitting condition prevents the strips from the conversion into thin curves along the object surface. In addition, this kind of division can guide the visualization process by removing

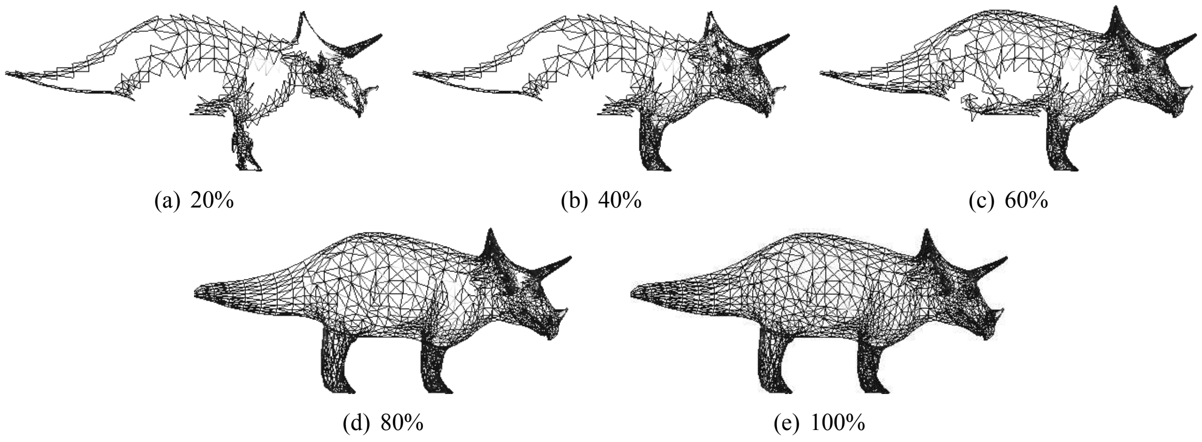

(a) 20% (b) 40% (c) 60% (d) 80% (e) 100%

Fig. 7: Progressive visualization of Triceratops model.

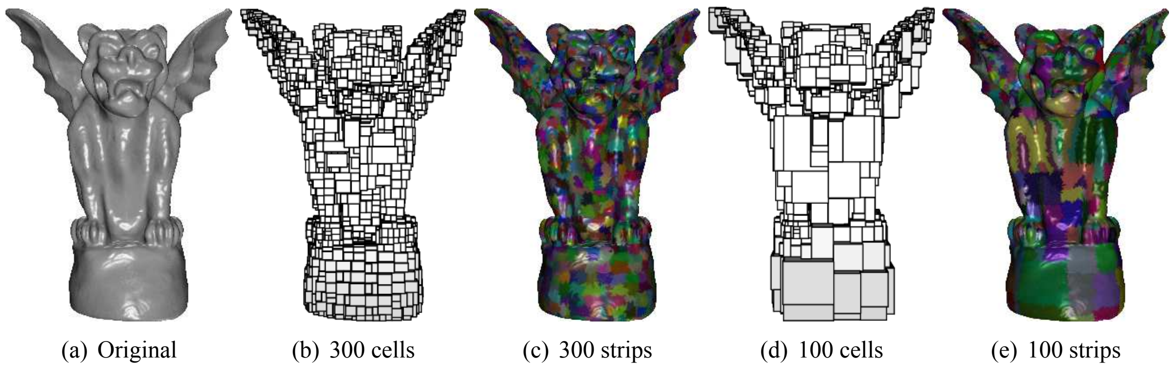

(a) Original (b) 300 cells (c) 300 strips (d) 100 cells (e) 100 strips

Fig. 8: Generation of multiple strips for the Gargoyle model. (a) the original model. (b), (c) the division of the model into 300 strips. (d), (e) the division into 100 strips.

hidden patches, cells, from the visualization pipeline. Figures 8(b) and 8(d) represent the space partitioning into 300 and 100 cells, respectively, using the kd-tree. Figures 8(c) and 8(e) show the corresponding strips of the previous two trees.

Table 1 shows a comparison between generating a single strip or two independent strips for a set of models. The second column presents the initial model size. The third column gives the size of the encoding of the models as a single strip S. Finally, the fourth and the fift columns present the size of strip fil when dividing the given models into two independent strips S_1 and S_2 respectively. Table 1 reflect that the sum of the sizes of the independent strips is not so far from the size of the single strip.

Generating more strips for each model enables to visualize the objects efficientl by hiding strips that are not compatible with the viewing direction. This makes the visualization more adaptable to the viewing direction, $\vec{u}$. For each strip S, we compute the average normal vector of this strip. To do this, we compute the average normal vector for each face contained in S and averaging these unit normals:

$$\vec{n}_S = \frac{\sum_{i=1}^{N_f^S} \vec{n}_i}{N_f^S} \tag{3}$$

where N_f^S is the total number of faces in the strip S, $\vec{n}_i$ is the unit normal vector for the face $f_i \in S$. Using the information of the average normal vector $\vec{n}_S$, we can predict whether S is visible or not. The is performed by evaluating the inner product $(\vec{n}_S \cdot \vec{u})$. If this value is greater than zero then the strip S is not visible.

5 Examples and comparisons

Our algorithms are implemented in C++. The used data structures are based on the polyhedral surface and the Halfedge templates [1] to manipulate the input models. In addition, we use the kd-tree data structure to space partitioning the bounding box containing the objects. The input file are given as OFF files The results presented in this paper are obtained on a 2.6 GHz Pentium Dual-Core CPU with 2.0GB RAM.

Table 2 contains some detailed information about the application of the proposed stripificatio algorithm on a set of models. The firs and second column gives the input object and its original size. The third column show the size of the obtained strip. A comparison between the original object size and the compressed size is presented in the fourth column. The ratio here is computed over the sizes of the original ASCII file and the generated strip files The compression ratio is evaluated as follows:

$$Ratio = \frac{f_i - f_c}{f_i}, \tag{4}$$

where f_i is the input fil size and f_c is the compressed fil size. The fift and the sixth columns represent the time in second elapsed to compress the corresponding fil and the decompression into the original OFF format respectively. As shown in Table 2, our stripificatio algorithm results in a good compression ratios. On the other hand, the decompression process does not required a large amount of time to decode the given strip file Table 1 shows the generation of multiple independent strips, here two strips, for the same set of models. The generation of multiple strips for the same model preserves the compression aspect of the method. On the other hand, Figure 8 shows the use of the multiple stripificatio in adaptive visualization. The space partitioning using the kd-tree and computing the average normal for each strip does not take more than 0.3 seconds for an object containing 200k of triangles.

6 Conclusion and limitations

Thanks to the recent advances in 3D aquisition techniques, 3D meshes have become more available and contain a tremendous number of sample points and faces. Streaming or visualizing these kind of objects is the bottleneck of the computer graphics domain. In this paper, we present a technique that handle this problem. Our technique is based on converting the input mesh into a triangular strip. We enhance our stripificatio by introducing the usage of the well known kd-tree space partitioning to localize the strip in a compact region on the surface of the mesh. This prevents the algorithm from spreading thin strips along the mesh surface. However, our technique suffers from some limitations such as:

Model	Input size(KB)	S	S_1(KB)	S_2(KB)
Chinese Dragon	763	334	187	206
Bunny	2337	1376	772	848
Holes	260	149	85	92
Mannequin	818	440	244	269
Triceratops	172	95	54k	59
Gargoyle	6624	3729	2102	2312
Cow	188	107	66	72
Eros	35978	14631	7352	7694
Isidore-horse	18068	8303	4178	4267
Lagomaggiore	50695	18677	9939	9698
Camel	2448	1269	672	711
Neptunel	165001	38319	74088	39411

Table 1: Generating two independent sub-strip file instead of a single one. Second column represents the initial fil size (input model), third column gives the single strip size and the fourth and the fift columns represent each sub-strip size.

Model	O- file(KB	C- file(KB	Ratio	C-time(sec)	D-time(sec)
Chinese Dragon	763	334	0.56	13	1
Bunny	2337	1376	0.41	50	4
Holes	260	149	0.43	6	1
Mannequin	818	440	0.46	17	1
Triceratops	172	95	0.45	3	1
Gargoyle	6624	3729	0.44	103	8
Cow	188	107	0.43	4	1
Eros	35978	14631	0.59	487	42
Isidore-horse	18068	8303	0.54	314	21
Lagomaggiore	50695	18677	0.61	800	68
Camel	2448	1269	0.48	44	3
Neptunel	165001	74088	0.55	3830	184

Table 2: Comparison between the size of the original ASCII file and the obtained strip files fourth column. Second and third columns represent the input 3D model and the obtained strip fil sizes respectively. Compression and decompression times (in seconds) are given in the fift and sixth columns respectively.

1. Although our technique generates triangle strips with as few swaps as possible, there is no rule to guarantee the generation of strips with minimum number of swaps for a given mesh.

2. The generated number of local strips is not adaptive to the local geometry of the surface. In fact, the number of strips is based on a user parameter.

References

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] K. Akeley, P. Haeberli, and D. Burns. *tomesh.c, a C-program on the SGI Developer's Toolbox CD*, 1990.

[3] B. G. Baumgart. Winged edge polyhedron representation. Technical report, Stanford, CA, USA, 1972.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975.

[5] M. M. Chow. Optimized geometry compression for real-time rendering. In *IEEE Visualization 97*, pages 347–354, 1997.

[6] C. Courbet and C. Hudelot. Random accessible hierarchical mesh compression for interactive visualization. In *Proceedings of the Symposium on Geometry Processing*, SGP '09, pages 1311–1318, 2009.

[7] M. Deering. Geometry compression. *Computer Graphics Forum*, 29:13–20, 1995.

[8] Q. Deng, M. Zhou, and J. Zhang. Real-time rendering of large scale scenes based on multiresolutional strip models. In *ICALIP'10: International Conference on Audio Language and Image Processing*, pages 234–238, 2010.

[9] F. Evans, S. S. Skiena, and A. Varshney. Completing sequential triangulations is hard. Technical report, 1996.

[10] F. Evans, S. S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. In *IEEE Visualization*, pages 319–326, 1996.

[11] T. Gurung, M. Luffel, P. Lindstrom, and J. Rossignac. Lr: compact connectivity representation for triangle meshes. *ACM Transaction on Graphics*, 30(67):67:1–67:8, 2011.

[12] J. Peng, C.-S. Kim, and C. J. Kuo. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image*, 16(6):688–733, 2005.

[13] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[14] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide*. 7 edition, 2009.

[15] F. G. M. Silva and P. Yadav. Compression and progressive visualization of geometric models. In *WSCG'08: International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 71–78, 2008.

[16] B. Speckmann and J. Snoeyink. Easy triangle strips for TIN terrain models. In *Proceedings of 9th CCCG*, pages 239–244, 1997.

[17] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, 1998.

[18] C. Touma and C. Gotsman. Triangle mesh compression. In *Proceedings of Graphics Interface*, pages 26–34, 1998.

[19] P. Vanecek and I. Kolingerová. Multi-path algorithm for triangle strips. In *Computer Graphics International*, pages 2–9, 2004.

[20] P. Vanecek and I. Kolingerová. Comparison of triangle strips algorithms. *Computers & Graphics*, 31(1):100–118, jan 2007.

[21] X. Xiang, M. Held, and J. Mitchell. Fast and efficien stripificatio of polygonal surface models. In *I3DG*, pages 71–78, 1999.