Compression and Progressive Visualization of Geometric Models

Frutuoso G. M. Silva
IT - Networks and Multimedia Group
University of Beira Interior, Portugal
fsilva@di.ubi.pt

Pranjul Yadav
Indian Institute of Technology, Guwahati, India
pranjul@iitg.ernet.in

ABSTRACT

The maximum compression, efficient transmission and fast rendering of geometric models is a complex problem for many reasons, thereby gaining a lot of attention from several areas, like compression and rendering of geometric models. Normally, the stripification 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.

In this paper, we present a new compression algorithm based on stripification of geometric models that enable us a progressive visualization of the models during its transmission. It occurs because our algorithm encodes and decodes the geometry and the connectivity of the model in an interwoven fashion.

The main purpose is the storage of object files as strips files in server computer, which enables faster transmission and display of the models at client side. In fact, our compression algorithm achieves compression ratios above 40:1 over ASCII encoded formats and the triangle strips improve rendering performance.

Keywords
Compression algorithms, Triangle strips, Progressive visualization, Mesh compression.

1 INTRODUCTION

The tremendous growth of computer graphics and entertainment (e.g. online games and virtual environments) over the Internet (LAN and WAN) has increased the necessity of rapid transmission of 3D models.

The increase in this area is fueled by the emerging demand for interactive visualization of 3D models in a network environment using several standard formats, as VRML [Vrm97] and X3D [X3d04]. But bandwidth over a network is a limited resource. Thus, we need compression and decompression algorithms for easy transmission and visualization of 3D models over the Internet, i.e. it is desired to compress 3D models in order to reduce storage and transmission time requirements. This is because these 3D models can be composed of millions of polygons, which take a long time to transmit over the network (e.g. Internet).

Besides, for interactive visualization not only the speed at which a model can be received is important, but also the speed at which it is displayed. One popular approach for fast visualization of these types of models is the conversion of a triangulated model into strips of triangles. Such triangle strips are widely supported by graphical hardware and software [Woo96].

A sequential strip is a sequence of $n+2$ vertices that represent n triangles where each triangle shares a common edge. Sequential strips allow us to reduce the transmit cost by a factor of three, from $3.n$ to $n+2$ vertices. But finding the minimum number of strips to cover a given model is a NP-hard problem. However, several heuristics have been proposed to generate good triangle strips.

In this paper we present a new compression algorithm based on stripification of geometric models that enable us a progressive visualization of the models during its transmission because our algorithms encodes and decodes the geometry and the connectivity of the model in an interwoven fashion. On the contrary, most of the compression methods proposed in the literature compress the geometry and the connectivity separately. This in fact, does not allow a progressive visualization of the models during its transmission without decoding its geometry.

Section 2 presents a short overview of compression and stripification algorithms. The compression and decompression algorithms are described in Section 3. Section 4 presents some comparisons and results. Finally, Section 5 presents some conclusions and future work.

2 RELATED WORK

There are several efficient compression schemes for triangle meshes that encode the geometry and connectivity separately [Tau98, Tou98, Ros99]. 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 first, to visualize the model based on its connectivity.

Bajaj *et al.* presented a compression algorithm that also encodes the geometry, the connectivity and the attributes separately. Besides he uses four geometric primitives to encode the mesh that are: contours, branching points, triangle strips and triangle fans. On the contrary, we only use triangle strips and encode the geometry and connectivity in an interwoven fashion.

These compression methods are useful for encode files, as for example VRML or X3D ASCII file formats, which have the geometry and connectivity separately.

For fast visualization of geometric objects the use of triangle strips is common because the graphical hardware widely supports it. The first algorithm proposed for creating triangle strips is the SGI algorithm [Ake90]. But other good algorithms proposed so far are the STRIPE [Eva96] and the FTSG [Xia99].

Triangle stripification algorithms can be categorized in several different ways. For example, based on the type of input data and the type of optimization. In terms of optimization we are normally interested in the minimization of number of strips and vertices. Thus, Deering [Dee95] introduced the concept of generalized triangle strips in which the main objective was to reduce the number of strips. Based on Deering's work, Chow [Cho97] proposed a mesh compression scheme optimized for real-time rendering. Their algorithms achieved compression ratios below 37:1 over ASCII encoded formats. But, we will see in later sections that our algorithm achieves compression ratios above 40:1 over ASCII encoded formats.

Furthermore it is also possible to encode strips files and further reduce the size of these files. But normally, the available connectivity compression techniques do not support the encoding of stripified models. However, more recently Isenberg [Ise01] proposed a simple and efficient scheme for coding the connectivity and the stripification of a triangle model. For example, the application of Isenberg's scheme in our case it is possible but only to compress the strip files, i.e. without connectivity.

Recently, Kim *et al.* [Kim06] presented a compression method that enables a progressive decompression of an arbitrary portion of a model without decoding other non-interesting regions. This method is accomplished by adapting selective refinement of a multiresolution model to the model compression domain.

For more details about several triangle strips algorithms see the recent work of Vanecek and Kolingerová [Van07] that presents a comparison of triangle strips algorithms and also the survey on 3D mesh compression presented by Peng *et al.* [Pen05].

3 COMPRESSION AND DECOMPRESSION ALGORITHMS

To better understand the compression and decompression algorithms we will first introduce the AIF data structure that supports their development.

3.1 AIF Data Structure

The data structure used in the compression algorithm is a version of AIF data structure [Sil03]. Figure 1 presents the four relations explicitly represented in original AIF data structure.

Basically, a 2-dimensional mesh in the AIF data structure is defined by the triple M={V,E,F}, where V is a finite set of vertices, E is a finite set of edges, and F is a finite set of simply connected faces.

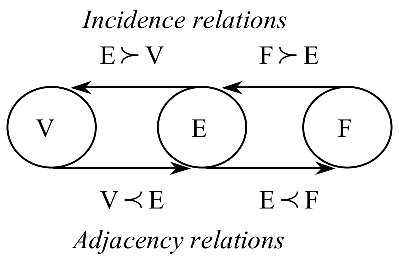

Figure 1. AIF Data Structure Scheme.

We create a new version of AIF data structure that represents 5 out of 9 relations between cells of a 2-dimensional cell complex, namely three basic adjacency relations, $V \prec E$, $V \prec F$ and $E \prec F$, and two inverse relations (or incidence relations) $E \succ V$ and $F \succ E$. These five basic relations can be combined to form the nine-adjacency and incidence relations. Though this work can also be done using only the four relations presented in original AIF data structure, but the fifth one is used to speed up the stripification process.

Besides, the AIF data structure has a companion query operator for fast retrieval of adjacency and incidence information (i.e. topological information of a mesh). Note that the query operator does not handle all the data structure constituents at once. It handles a mesh locally by using the adjacency and incidence relations stored in the data structure. Thus, its time performance is independently of the mesh size. This is very important for handling large meshes efficiently.

3.2 Compression Algorithm

Our compression algorithm is based on the principle of converting a model into strips. The compressed model is not exactly a composition of many strips but a collection of many linking agents and vertices. Linking agents are special symbols, which will be used to make long strips. The entire algorithm has been divided into five elementary steps. These steps have been designed to cover models either with or without holes. The compression algorithm is the following:

```
Algorithm 1
  INPUT: A mesh in AIF data structure
  OUTPUT: A strip file
Begin
  While there are vertices with a degree greater than zero
  1. First Vertex Selection
  2. Intermediate Vertex Selection
  3. Final Vertex Selection
  4. Look Back
  5. Dead Lock
End
```

In first step of Algorithm 1 we will select the vertex (v_1) with the minimum vertex-degree in the mesh. The vertex-degree is the number of incident edges on a vertex. However, only the edges that have an edge-status greater or equal to one are considered. The edge-status is the number of incident faces on it. For example, for a manifold mesh an edge has normally two incident faces.

Note that our algorithm modifies the edge-status during the creation of strips. Whenever a triangle is included in a strip then the edge-status of its edges will decrease by one. Thus an edge-status equal to zero means that the strip already covers the edge.

If there exist several vertices having a minimum vertex-degree then we will select any one of these. We select a vertex that has a minimum vertex-degree and not the maximum vertex-degree because if we start with vertex having less degree then the number of strips formed will be less. Otherwise we have to create separate strips for isolated triangles that will appear if we select the vertex with maximum degree.

In second step of the Algorithm 1, we will select a vertex (v_2) that belongs to an edge (e) incident on v_1, such that the status of e is different of zero. Once again, we will select the vertex that has the minimum degree (i.e. the same strategy used in step 1).

In the step three, we will select the third vertex (v_3) that forms the triangle defined by vertices v_1, v_2, v_3. Thus the triangles that can share the edge e (i.e. the edge defined by the vertices v_1 and v_2) are two or one.

If there exist only a triangle that share the edge e then the third vertex is automatically founded. In this case, the status of the selected triangle will be set to zero, so that it cannot be selected again. The status of a triangle is defined based on the status of its bounding edges. Thus the status will be decreased for all edges adjacent to triangle.

If there exist more than one triangle adjacent to edge e we will select the triangle which has the minimum degree. If the triangles have the same degree we select one triangle arbitrarily.

After selecting the third vertex (v_3) it will be inserted in the compressed file (i.e. in strip). In this case it is necessary to check if v_3 is already present in the compressed file or not. If not then the coordinates of the vertex will be placed in the compressed file. Otherwise, we will use the notion of dummy buffer by only placing the order number of the first occurrence of this vertex in the compressed file. The order number will be preceded by the linking agent '<'. This linking agent denotes a vertex identifier.

Finally, the total number of triangles uncovered in the mesh will decrease by one.

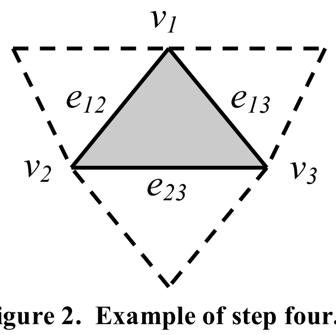

Figure 2. Example of step four.

In the step four of the algorithm we shall look back for the triangle selected by previous step. Thus assuming that the triangle is defined by the vertices v_1, v_2, v_3 and they appear in this order in compressed file (see the example in Figure 2).

We will first check the edge status of the edge (e_{23}) defined by vertices v_2 and v_3. If the edge e_{23} has a status greater than zero then we will check the edge status of edges e_{13}, formed by vertices v_1 and v_3.

If edge e_{13} has a status equal to one then we will cover the triangle adjacent to this edge. Such triangles are called auxiliary triangles or side triangles because the front of the strip is the edge e_{23}. This case will be marked using the linking agent '#'. Then the third vertex of this auxiliary triangle will be found (v_4) and will be inserted in the compressed file. The status, degree of edges and vertices will be updated respectively.

If the edge e_{13} has an edge-status equal to zero then there is no more triangles uncovered adjacent to edge e_{13}. Thus the control flow will go to step three with vertex v_3 becoming as v_2 and vertex v_2 becoming as v_1 to find a new vertex v_3, because the edge e_{23} has an edge-status greater than zero.

If the edge-status of e_{23} is equal to zero then there are no more triangles uncovered adjacent to edge e_{23}.

Now if the edge-status of e_{13} is equal to one, then we will cover the triangle adjacent to this edge. In fact, our strip making process will have to change its direction in terms of leading edge. The leading edge was e_{23} but now it will be e_{13} or e_{31}. Note that the edges e_{13} and e_{13} are different for creating a strip (i.e. the order of the vertices is important).

If we consider e_{13} as the new leading edge then we mark this by linking agent '!' and the algorithm goes to step three to find a new vertex v_3. But if we consider e_{31} as the new leading edge then we mark this by linking agent '@' and the algorithm goes to step three again.

If the edge e_{13} has a status equal to zero then there are no more triangles uncovered adjacent to edge e_{13}. Thus the edge-status of all the three edges (e_{12}, e_{13} and e_{23}) is equal to zero.

But we will now check the vertex-status of our three vertices (v_1, v_2, v_3). If there is any one vertex has a vertex-degree greater that zero then the algorithm goes to step two with this vertex as v_1 to find a new vertex v_2. Thus depending whether v_1, v_2, v_3 is chosen as the new v_1 a linking agent 'A', 'B' or 'C' will be inserted, respectively.

Otherwise, the control flow will go to the last step, called dead lock. Entry in this step implies that our current strip is concluded and it is necessary to start a new strip. Then the control flow goes to step 1 to find a new vertex that has a degree greater than zero.

Figure 3 shows a simple mesh with a hole that exemplifies the use of linking agents 'C' and 'B'. This mesh is encoded by our compression algorithm by the following sequence: 6 5 4 B 1 2 C 4 3

Some other linking agents are also used in the compression algorithm but they will be presented in the context of the decompressed algorithm.

Our compression algorithm encodes a mesh file in a strip file reducing its size by 40% in average, as we will see in next Section.

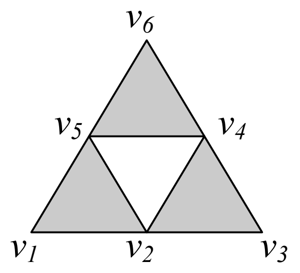

Figure 3. A mesh with an hole.

3.3 Decompression Algorithm

The decompression algorithm is just the reverse of the compression algorithm. However in the compression algorithm we need to make queries to find the adequate vertices to construct the strips. On the contrary, the decompression algorithm only decodes the strips, which contains only vertices and some linking agents. Thus the time consumed by the decompression algorithm is very small in comparison with the compression algorithm.

The decompression algorithm is the following:

```
Algorithm 2
  INPUT: A strip file
  OUTPUT: A SMF file
Begin
  1. Read the lines of strip file to a string
  2. Create a list of vertices and put the first three
  vertices there
  3. While not reach the end of the string
    (a) Read a new character 'c' from the string
    (b) switch ('c')
          case: '/'
          case: '#'
          case: '!'
          case: '@'
          case: 'A'
          case: 'B'
          case: 'C'
          case: '<'
          default - it is a vertex
  4. Create the SMF file
End
```

The decompression algorithm starts by reading the strip file for a string. Then it creates the list of vertices and inserts the first three vertices that appear always in the beginning of the strip file. After this we have a loop until the end of the string is reached. Then we read a character ('c') from the string for each iteration and process it. But the character 'c' can only be a linking agent or a vertex.

If the character 'c' is the linking agent '/' this means that the current strip is concluded. Then it is necessary to start a new strip. But we have to be careful in reading the vertices because there are two ways of representing a vertex, i.e. by their three coordinates or by an identifier.

If the character 'c' is the linking agent '#' this means that we found an auxiliary triangle. In this case the next entry (character) in compressed file will be a vertex. Note that the concept of auxiliary triangle is already defined in the compression algorithm.

The linking agent '!' implies a change in the order of the last three vertices to continue the strip. Thus the shared edge will be the edge defined by the vertices v_1 and v_3 and not the edge defined by the vertices v_2 and v_3 as normally. In this case the next entry in the compressed file will be the third vertex.

The linking agent '@' also implies also a change in the order of the last three vertices to continue the strip. Thus the shared edge will be the edge defined by the vertices v_3 and v_1 and not the edge defined by the vertices v_2 and v_3 as normally. In this case also the next entry in the compressed file will be the third vertex.

The linking agent 'A' is used for the case where the strip will continue based on first vertex only (v_1) and not based on an edge. Then the next entries in compressed file will be two vertices.

The linking agent 'B' denotes the case where the strip will continue based on the vertex v_2 and the next entries in the compressed file will also be two vertices.

The linking agent 'C' is used for the case where the strip will continue based on last vertex (v_3) and the next two entries in compressed file will be vertices.

If the linking agent is equal to '<' then we will read a identifier of a vertex, which is already in memory (i.e. in the buffer).

Finally, if the character 'c' is not a linking agent then we are reading a vertex (i.e. the coordinates that define the vertex).

At the end of the loop we create the decompressed file in the SMF file format.

4 RESULTS AND COMPARISONS

Table 1 presents a comparison between the size of the original ASCII files and compression files for seven different models pictured in Figure 4. It provides also the compression ratio achieved by our algorithm.

Table 1 contains the model name, the original file size (Server), the compressed file size (Client) and the compression ratio. Note that, the compression ratio is calculated as follows

Ratio = (initial file size - compressed file size) / initial file size

The initial files are in SMF file format, which have a similar structure to, that the OBJ file format. The wireframe models pictured in Figure 4 was converted to X3D file format for visualization purposes at client side.

As we can see, the compression ratio of our algorithm is about 0.40 in average.

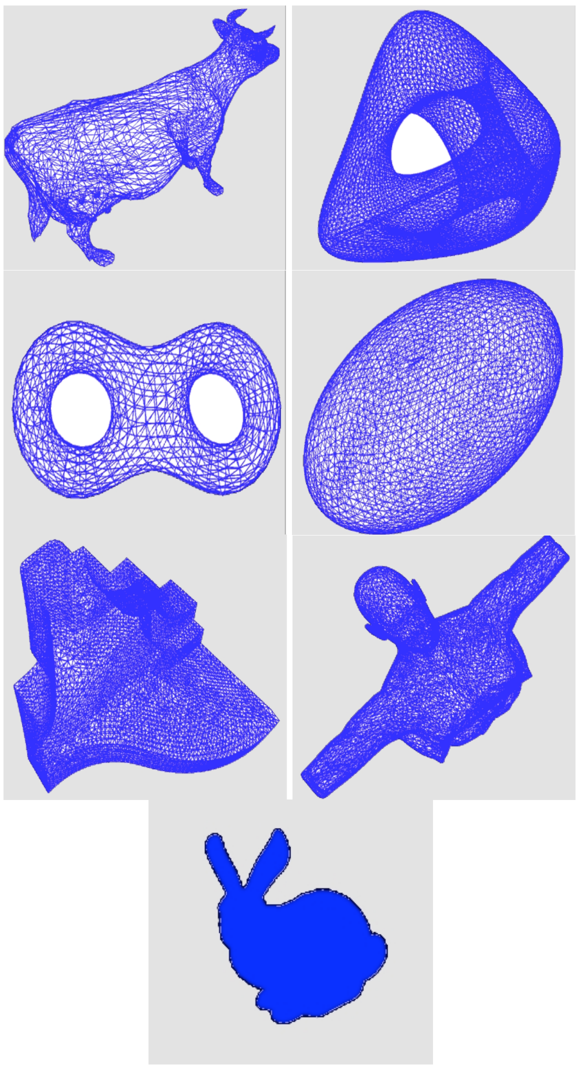

Figure 4. Models used for compression and decompression. From top-left: Cow, Genus3, Eight, Ellipsoid, Fandisk, Torso and Bunny.

However, another layer of compression is possible if the server zipping the compressed file before transmitting it. This layer of compression (i.e. zipping) will further increase the compression factor. For example, the size of the compressed file for

Bunny model is 1413 KB and the size of the compressed zipped file is 518 KB.

Table 2 presents a comparison between the size of the original SMF files and the corresponding decompressed files for the several models used. Note that, the decompressed files are smaller than original files, but both files represent the same geometric model. This means that the simple fact that compressing and decompressing an object file reduces its size, because some unnecessary symbols are removed, for example, some blank spaces.

Model	**Original File**	**Compressed File**	**Ratio**
Cow	183 KB	106 KB	0.42
Bunny	2507 KB	1413 KB	0.43
Eight	47 KB	29 KB	0.38
Ellipsoid	212 KB	118 KB	0.44
Fandisk	493 KB	311 KB	0.37
Genus3	438 KB	253 KB	0.42
Torso	469 KB	268 KB	0.43

Table 1. Compression Results.

Table 3 presents a comparison between the number of vertices in strips and the number of vertices per triangle. The number of vertices in strips determines the amount of data sent to graphical pipeline. In this case, our algorithm achieved good results comparing with the results presented by Vanecek and Kolingerová [Van07] for several stripification algorithms. For example, for the Cow model we achieved 6637 vertices in strips while all other algorithms achieved results between 7000 and 8000. For Bunny model we achieved 79920 vertices in strips while all other algorithms achieved results between 82000 and 102000 vertices.

Model	Original File	Decompressed
Cow	183 KB	177 KB
Bunny	2507 KB	2438 KB
Eight	47 KB	44.8 KB
Ellipsoid	212 KB	204 KB
Fandisk	493 KB	475 KB
Genus3	438 KB	424 KB
Torso	469 KB	454 KB

Table 2. Compression and Decompression Results.

The number of vertices per triangle shows the efficiency of the algorithm because it determines the ratio of number of vertices per triangle. In this case, once again, our algorithm achieved better results than all other algorithms according to Vanecek and Kolingerová [Van07] results (i.e. for common models Cow and Bunny).

The number of strips and the average length of strips created by our algorithm are presented in Table 4. In terms of the number of strips our algorithm produces more strips than the other algorithms. Thus the average length of strips is normally inferior to the average length of strips produced by other algorithms (see Vanecek and Kolingerová results [Van07]). However, the number of vertices per triangle in strips for our algorithm is smaller (see Table 3) than the number of vertices per triangle in strips for other algorithms. This means that other algorithms produce fewer strips but with more number of vertices per triangle, i.e. they produce strips with more repeated vertices than our algorithm.

Model	**# Triangles**	**Vert. in Strips**	**Vert. per △**
Cow	5804	6637	1.14
Bunny	69473	79920	1.15
Eight	1535	1699	1.11
Ellipsoid	6923	7793	1.12
Fandisk	12946	15042	1.16
Genus3	13311	15117	1.13
Torso	14451	16389	1.13

Table 3. Number of vertices in strips.

Therefore our algorithm is better to compression and visualization purposes.

Model	Number of strips	Average length of strips
Cow	231	28.71
Bunny	2645	30.21
Eight	42	40.45
Ellipsoid	246	31.67
Fandisk	544	27.65
Genus3	469	32.23
Torso	617	26.56

Table 4. Number of strips achieved by our algorithm.

4.1 Progressive Visualization

When a client asks the server for a particular file, the server will send the compressed model in packets of small bits. In other compression algorithms these initial packets consist of vertex position information only. Information about connectivity generally appears in later packets. Hence the application client has to wait until it receives all information about vertices. This is a very worse situation leading a lot of latency because the user cannot have early information about the model.

Whereas the compression of geometry and connectivity in our algorithm is made by interwoven fashion, thus each strip is self-sufficient in terms of visualization. If the client uses our algorithms then as soon as it will receive the first packet or rather the first full strip from the server the visualization process can be started. This early visualization is advantageous because it give the user early information about the model.

Figure 5. From top-left: Cow in 20, 40, 60, 80 and 100 percent of its size.Figure 5 and Figure 6 show two examples of progressive visualization effect for Cow and Ellipsoid models. The compressed files have been divided into small fractions of their total file size to simulate their transmission, namely 20, 40, 60 and 80 percent of its size.

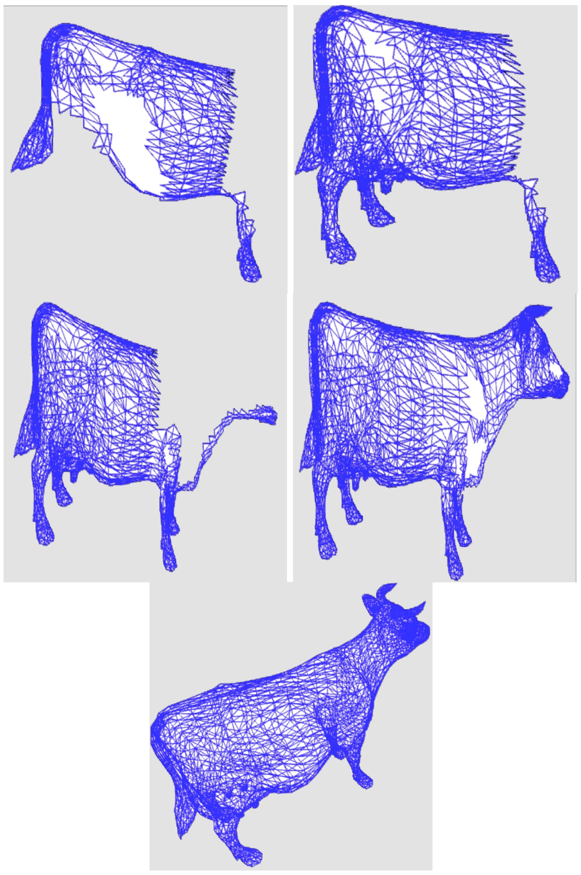
Figure 5. From top-left: Cow in 20, 40, 60, 80 and 100 percent of its size.

Note that each fraction of the strip file can be visualized separately and it gives an idea to user of the shape of the model. This allows the user to cancel a transfer before whole object is received. For example, when the object is not what the user expected.

5 CONCLUSIONS

Our compression algorithm was developed for compression and visualization purposes. Thus, it produces normally more strips than other algorithms but with less number of vertices per triangle. Besides, it allows a progressive visualization of models during its transmission over the network. This means that segments of the compressed file are self-sufficient in terms of visualization.

Our algorithm uses an efficient technique to compress a file. In most of the compression strategies, compressed file consists of geometry (i.e. vertices) followed by connectivity. In our case the compressed file, consist of combinations of vertices and linking agents, i.e. the geometry and connectivity are compressed in an interwoven fashion. This combination helps us to achieve a compression factor above 40 percent over ASCII encoded formats.

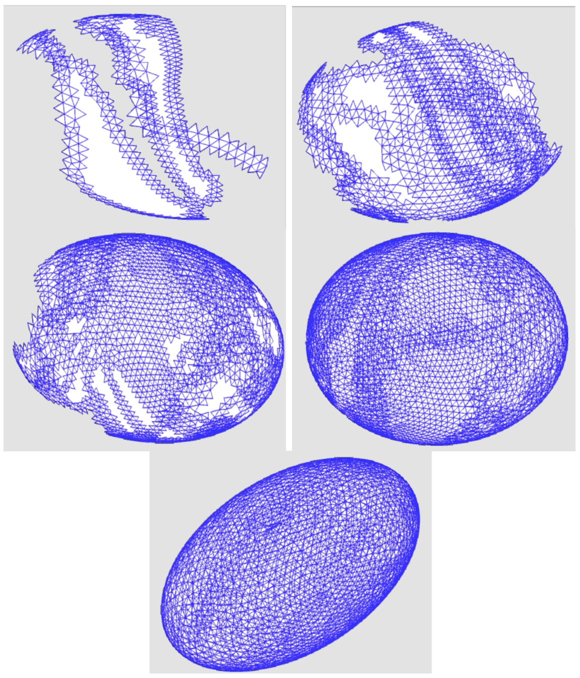
Figure 6. From top-left: Ellipsoid in 20, 40, 60, 80 and 100 percent of its size.

The future work will concentrate on improving the strip making process. The main idea is to reduce the number of strips created by our algorithm but maintaining the number of vertices per triangle.

6 ACKNOWLEDGMENTS

The second author wants to thank to Prof. Abel Gomes and the Institute of Telecommunications (IT) by the grant that supported his work.

7 REFERENCES

[Ake90] K. Akeley, P. Haeberli, and D. Burns. tomesh.c: C program on sgi developer's toolbox cd, 1990.

[Baj99] C. L. Bajaj, V. Pascucci and G. Zhuang. Single resolution compression of arbitrary triangular meshes with properties, Computational Geometry, Vol. 14(1), pages 167–186, 1999.

[Cho97] M. M. Chow. Optimized geometry compression for real-time rendering. In Proceedings of Visualization '97, pages 347–354, 1997.

[Dee95] Michael Deering. Geometry compression. Computer Graphics Forum, Vol. 29, pages13–20, 1995.

[Eva96] Francine Evans, Steven S. Skiena, and Amitabh Varshney. Optimizing triangle strips for fast rendering. IEEE Visualization '96, pages 319–326, 1996.

[Ise01] Martin Isenburg. Triangle strip compression. Computer Graphics Forum, 20(2), 2001.

[Kim06] Junho Kim, Sungyul Choe, and Seungyong Lee. Multiresoluion random accessible mesh compression. Computer Graphics Forum, Vol. 25(3), pages 323–332, 2006.

[Pen05] Jingliang Peng, Chang-Su Kim, and C. Jay Kuo. Technologies for 3d mesh compression: A survey. Journal of Visual Communication and Image Representation, Vol. 16(6), pages 688–733, 2005.

[Ros99] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. IEEE Transactions on Visualization and Computer Graphics, Vol. 5(1), pages 47–61,1999.

[Sil03] Frutuoso G. M. Silva and Abel J. P. Gomes. AIF - A data structure for polygonal meshes. Lecture Notes in Computer Science, Vol. 2669, Part III, pages 478–487, 2003.

[Tau98] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. ACM Transactions on Graphics, Vol.17(2), pages 84–115, 1998.

[Tou98] Costa Touma and Craig Gotsman. Triangle mesh compression. Proceedings of Graphics Interface, pages 26–34, 1998.

[Van07] Petr Vanecek and Ivana Kolingerov. Comparison of triangle strips algorithms. Computer & Graphics, Vol. 31(1), pages100–118, 2007.

[Vrm97] VRML. The virtual reality modeling language (vrml) - iso/iec 14772-1, 1997. http://www.web3d.org/x3d/specifications/vrml/

[Woo96] M. Woo, J. Neider, and T. Davis. OpenGL Programming Guide. Addison-Wesley, 1996.

[X3d04] X3D. X3d international standards - iso/iec 19775, 2004. http://www.web3d.org/x3d/specifications/

[Xia99] Xinyu Xiang, Martin Held, and Joseph S. B. Mitchell. Fast and effective stripification of polygonal surface models. In ACM Symposium on Interactive 3D Graphics, pages 71–78, 1999.