AN EVOLUTIONARY MESH COMPRESSION ALGORITHM FOR VIDEO GAMES

Eduardo Lago Aguilar
Igr Alexánder Fernández Saúco
~
Virtual Games Group, University of Informatics Sciences (UCi)
Havana City, Cuba
E-mail: {lago|alexanderfs}@uci.cu

ABSTRACT

This paper presents an efficient algorithm for generating triangle strips from triangulated meshes, providing a compact representation suitable for transmission and rendering of graphical 3D models used in games. The method is based on a simple heuristic that reduces the number of vertices used to describe the triangulated models. It also presents an evolutionary variation that produces better results every time the game is executed.

KEYWORDS

Triangle strip, mesh, mesh compression, rendering, randomization, evolutionary.

INTRODUCTION

There is an abundant amount of geometry data in modern graphical 3D applications, especially in games. Geometry data is usually present as triangulated meshes due to the capabilities of video cards to manage this basic geometry form. However, video cards have memory limitations particularly with real-time games where thousands of triangles are needed to make surfaces look smooth. A common codification schema to handle this problem is called *triangle strip* (TS) [5], which enumerates a sequence of adjacent triangles to avoid repeating the vertexes shared by the same edge. This graphic primitive is supported by all video cards and is very fast and economic.

This document discusses an algorithm that considerably reduces the amount of memory and time required to store and renders the geometry data, by converting triangular meshes into TS. The algorithm is based on the method presented in [3]. It also includes a new simple greedy heuristic and introduces an evolutionary variation with a cheaper pruning criterion to avoid unnecessary calculations.

OVERVIEW

After setting the first three vertices of the first triangle in a mesh, each new triangle is formed by joining a new vertex with the last two. For a four-triangle mesh, a triangle strip representation would be *{ABCDEF}* (6 vertices) instead of the expensive representation *{ABC, CBD, CDE, EDF}* (12 vertices) normally used as can be appreciated in Figure 1.

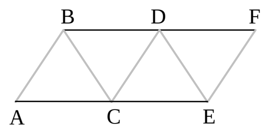


Fig. 1: Four triangle mesh covered by one TS = *{ABCDEF}*

The swapping

To represent certain meshes in TS form, it is necessary to insert *swaps*. A swap is nothing but re-sending a previously enumerated vertex into the sequence. For instance, in the mesh of Figure 2, the vertices *D* and *E* can be swapped by inserting vertex *D* again into the sequence, thus in *{ABCDE**D**FG}* the re-sending of vertex *D* represent the swap. It's worth to note that the swap introduces a new null (zero area) triangle but leaves a complete mesh representation. How many swaps can be introduced without affecting the performance in a significant way? Isn't it better to split certain meshes in many TS instead of introducing null triangles? Is it possible to find an optimal representation of a mesh by cutting it into several TS? If not, how much can be improved with the compression rate of a mesh represented by triangle strips?

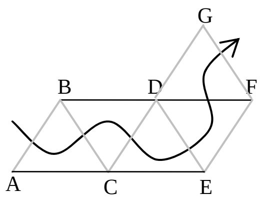


Fig. 2: A swap is introduced to cover the mesh in one TS = *{ABCDEDFG}*

PROBLEM

Let's propose a formal definition of the TS problem using a known concept. A *dual graph* $G'$of a given planar graph G has a vertex for each plane region of G, and an edge for each edge joining two neighboring regions [1], see Figure 3. The TS problem is analogous to find a Hamiltonian path for the dual graph of the mesh [3].

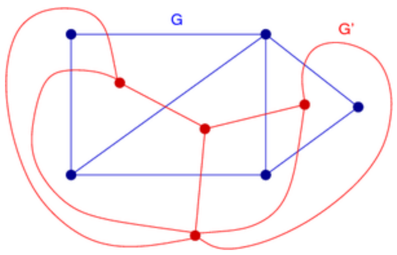


Fig. 3: G′ is the dual graph of G

Let $M = \{t_1, t_2, ..., t_n\}$ a set of triangles that define a mesh with t_i = $\{v_1, v_2, v_3 \mid v_i \in V\}$ where V is the set of all vertices of the mesh. Let's call formally *S,* triangle strip of *M,* to a sequence of vertexes:

$$S = \left\{ \begin{matrix} v_1, v_2, ..., v_k \mid t = \{v_j, v_{j+1}, v_{j+2}\} \in M \vee \\ t = swapping, j \le k-2, v_j \in V, k \ge 3 \end{matrix} \right\}$$

Then, the problem would be to find a set *H* of triangle strips that completely covers the mesh and minimizes the total number of vertices, thus:

$$H = \left\{ S_1, S_2, ..., S_m \mid \bigcup_{i=1}^{m} T(S_i) = M \wedge \sum_{i=1}^{m} |S_i| \text{ is minimun} \right\}$$

where $T(S_i)$ is the set of triangles of the strip S_i.

The compression

A triangulated mesh $M = \{t_1, t_2, ..., t_n\}$ can be described in the best case by one triangle strip *S* of length $n + 2$ without swaps; this doesn't mean that it is always possible to attain it. The theoretical compression rate of planar meshes into TS can be defined based on the number of vertices that describe a triangulated mesh ($3n$) and the number of vertices necessary for the best case $n + 2$, thus:

$$\lim_{n \to \infty} \frac{3n - (n+2)}{3n} = \frac{2}{3} \approx 0{,}6667$$

In general, the compression rate for an arbitrary partition *H* of a mesh as triangle strips can be defined as:

$$\mu = 1 - \frac{n + p + 2m}{3n}$$

where *p* is the total number of swaps for all strips in *H* and *m* is the total number of generated strips.

The worst case is produced when there is a strip for each triangle (when $m = n$, $p = 0$) thus:

$$0 \le \mu \le 2/3$$

Where did the previous formula come from? When a swap is introduced into a strip it penalizes memory and speed with one vertex, unlike a strip whose penalty is two vertices.

SOLUTION

Let's introduce two necessary concepts used by the algorithm: a triangle is referred to as *free* if it doesn't belong to any strip, and the *degree* of a triangle is the number of free triangles adjacent to it and can change at each step of the algorithm.

The algorithm starts by creating an empty set of strips *H* and follows as: set *S* as an empty set of vertices, select the lowest degree free triangle from the mesh and append its vertices to *S*. With the last two vertices of *S*, try to form a new triangle *T* with a third vertex *V* of the mesh; if *T* cannot be formed or it's not free, then roll the last three vertices of *S* and try again. If after two rolls this triangle doesn't exist then *S* is closed and appended to *H*, start forming a new *S* again.

If the *T* is found then *V* is added to *S* and starts the second heuristic: using the last edge, formed by the last two vertices of *S,* and the third vertex *V* of the mesh*,* try to form a new free triangle *T* and append *V* to *S*, do this until *T* cannot be formed. Check whether there is another triangle *T* adjacent to the edge formed by the last and the one before; if it exists, then the swap is introduced by extracting the last vertex of *S*, re-sending the currently penultimate (previously the one before) and appending the extracted one. If even this triangle *T* doesn't exist then close *S*, append it to *H* and start forming a new *S* again while there are still free triangles remaining. The figure below shows the result of the algorithm in a Quake 3 Arena game level.

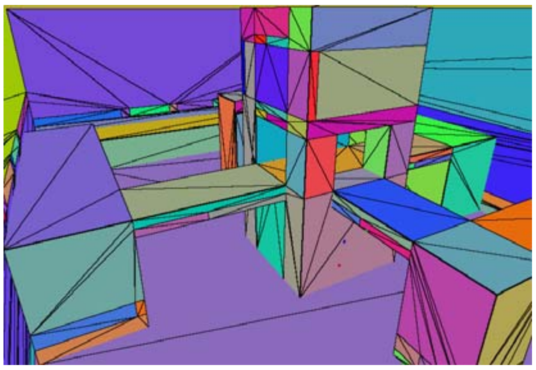


Fig. 4: Q3A game level with μ = 41.19%

The algorithm takes into account the existence of an efficient proven data structure that models the dual graph with the adjacency information and supplies basic operations like finding the first free lowest degree triangle of the mesh; get the degree of a triangle, etc. The structure is updated on each step of the algorithm and keeps the list of triangles sorted by degree. The details of this structure are not relevant at this point.

The pseudo-code is broken in two procedures: (1) a top level build for selecting the first triangle and strip direction, (2) a sub routine to sequence the strip with possible introduction of swaps based on the decision made in (1):

```
Step 0: let $H = ø$ the strip partition for the
mesh, $G$ the dual graph, $S = ø$ the current strip
Step 1: while exist free triangles in $G$ do
 Step 1.1: $T$ = first free triangle available in $G$,
 append $T$ to $S$
 Step 1.2: while not exist a free adjacency
 triangle to $T$ in $G$ via its last edge do
 Step 1.2.1: roll the three vertices of $S$
 Step 1.2.2: if after two rolls the while
 condition (1.2)  is still true, restore $S$ to its
initial status and go to Step 1.5
 Step 1.3: let $A$ the free adjacency triangle to $T$
 in $G$ encountered in previous iteration. Include
```

```
  third vertex of $A$ in $S$
  Step 1.4: call $sequence(S, A, G)$
  Step 1.5: append $S$ to $H$, go to step 1
Step 2: return $H$
```

pseudo-code for *build()*: first triangle selection

```
Step 0: let $S$ the current strip, $T$ the current triangle in $S$, $G$ the dual graph, and $fails = 0$ the number of fails seeking for adjacency triangles
Step 1: check for a free adjacency triangle $A$ to $T$ in $G$ via its last edge
  Step 1.1: if it exists then include third vertex of $A$ in $S$, set $T = A$, $fails = 0$
  Step 1.2: if it doesn't exist: if fails = 0 then do swapping else restore the strip from the last swap. Set $fails = fails + 1$
Step 2: if $fails < 2$ then go to step 1
Step 3: return $S$
```

pseudo-code for *sequence()*: strip sequencing

RANDOMIZATION

The algorithm described previously starts to sequence a strip for the "first lowest degree triangle". However, it can happen that more than one triangle is the lowest degree free; the choice of the starting triangle can produce a different solution. The Figure 5 depicts this situation obtaining three possible solutions

$H_1 = \{(1, 2, 3, 4, 5, 4, 6, 4, 7, 8), (11, 9, 10, 3, 5)\}$

$H_2 = \{(8, 7, 4, 6, 4, 5, 3, 10, 9, 11), (1, 2, 3, 4)\}$

$H_3 = \{(11, 9, 10, 3, 5, 4, 6, 4, 7, 8), (1, 2, 3, 4)\}$

The solutions H_2 and H_3 are the better ones but not necessarily reached using a deterministic algorithm. The solution will be the same in every algorithm execution and the remaining ones will never be explored.

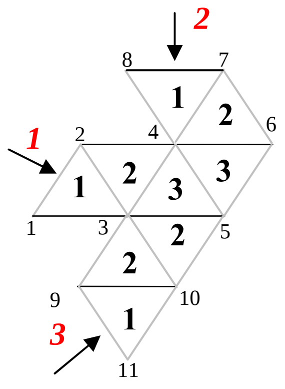


Fig. 5: More than one triangle is the lowest degree free triangle

The results of randomization of the first lowest degree free triangle selection, after several experiments show the existence of multiple better and worse solutions, as can be seen on the annexed histograms For instance, the game level displayed in Figure 4 reached over a 42% compression rate in the random version of the algorithm.

The bell-like shapes of the samples that resulted from multiple program executions suggest a standard distribution (e.g. Gauss, Exponential) for the compression coefficient. But the tests did not produce such result in several cases. Therefore, we decided to ignore any possible standard distribution and always treat it as a sample distribution directly related to the complex topology of the mesh.

EVOLUTION

At this time, we focused on the task of designing an algorithm to determine the optimal stripping, without noting that we had the best scenario for making continuous experiments for computing it: video games.

Our graphics engine for games allows for compressed triangulated surfaces, which composes our game levels/models into triangle strips. It is possible to get one compressed solution for each surface of the level, but it is practically impossible to compute the optimized one for one of those meshes in a reasonable time.

Why trying to compute the optimized stripping in a short time interval if we have all the time of live of our games to do it? While someone executes our games for playing, the engine can explore a different solution using randomized selection and improving its performance on each execution. Therefore all that is needed is a cheaper pruning criterion.

Suppose that at least one solution was computed for an arbitrary mesh *M* with compression level μ* calculated from *p** and *m**. In the next game execution, on each new strip creation step, the state of the algorithm is characterized by *p* and *m* and a remaining set of free triangles. At this point the best case can be presented on which just one new strip without swaps is formed with all free triangles, leading to the new algorithm state *(p, m + 1)*. If under such conditions the new compression level μ of the mesh is worst than the current best one μ* then the algorithm execution is stopped and the current solution is used, thus:

$$\mu \leq \mu *$$

$$1-\frac{n+p+2(m+1)}{3n} \leq 1-\frac{n+p*+2m*}{3n}$$

$$p+2(m+1) \geq p*+2m*$$

If the previous condition is never satisfied and the algorithm reaches to the end then a new better solution is found and stored. The additional storage space needed is very small in comparison with the size of the mesh, currently *p*, m** and the list of indexes of the first selected triangles for each strip.

The described algorithm is formally categorized as a random-restart hill climbing, a meta-algorithm that turns out that it’s often better to spend CPU time to explore the space, rather than optimizing from an initial condition.

This evolutionary solution can also be used in multiplayer games and work in cooperative mode. Each time a multiplayer game session is created all players can share their best solution indicators *p* and *m* through the network. All these solutions are compared and the best one selected if there is one. The process of network replication of the best solution is not expensive at all in contrast with mesh sizes and can be done during game level loading time, with practically idle networking activity, which is also proportional to mesh sizes. The info needed to be transmitted is just the list of indexes of the first selected triangles for each strip.

A new question arises: why don’t preprocess all game level meshes to compress it with optimal rates before the game release?

Time to play

Video game evolution recently brought the era of Massive Multiplayer Online Games (MMOGs). MMOGs exclusively emphasize multiplayer game play with thousands/millions of simultaneous players, huge scenes and worlds composed by millions of triangles. On the other hand, some MMOGs have no end condition that includes awarding a winner based on player’s behavior with the purpose of the player never stops to play.

The preprocessing of one MMOG world with the intention of reach the optimal striping compression will require a long time and very expensive power of computing. Therefore a MMOG is the best scenario for execute this algorithm using the power of computing of those millions of users as long as be used in a correct manner without saturate the network and clients processors.

CONCLUSIONS

Both versions of the described algorithm were implemented in C++ language using the G3D library [2]. The results produced for the deterministic and evolutionary algorithm, after several iterations, are shown in the annexes.

The use of both algorithms in video games can really improve the overall performance of games containing extensive geometric data. The memory overhead can be reduced in a significant way increasing the chance of including smoother surfaces and obtaining higher details on 3D models and game levels. The evolutionary version can be executed during model/level loading, ensuring that all processed data yield compression rates very close to the theoretical limit.

A multiplayer game engine programmed with the evolutionary aspect, can achieve high levels of performance that grows in time as the game is played especially in MMOGs resulting in a drastically reduction of geometry data, load and render time.

ACKNOWLEDGEMENTS

The authors would like to thank the Stanford 3D Scanning Repository, United States Geological Survey, Georgia Institute of Technology, and Nasa’s Planetary Data System for models *bunny* and *roseburg*. We also are in debt with Oliver Matias van Kaick for the support and with Morgan McGuire for the G3D library. Thanks to idSoftware for making public the Q3A source code.

Thanks to all members of the Virtual Games Group for the overall wisdom, help and friendship and especially to the student Hassan Lombera for the direct collaboration in the brief, English translation, implementation issues, data and experiment’s results collection. Also we would like to thank Professor Juan G. Consuegra from UCi and José H. Cifuentes Urquiza from IBM™ for helping with the English correction of the final draft.

REFERENCES

[1] H. Whitney, Non-separable and planar graphs, Trans. American Mathematical Society. 34 (1932), 339–362.

[2] McGuire, M., G3D 6.09 3D Engine, http://g3d-cpp.sourceforge.net

[3] O. Matias van Kaick, et al, Efficient Generation of Triangle Strips from Triangulated Meshes. Department of Computer Science Federal University of Parana 81531-990 Curitiba-PR, Brazil, 2004

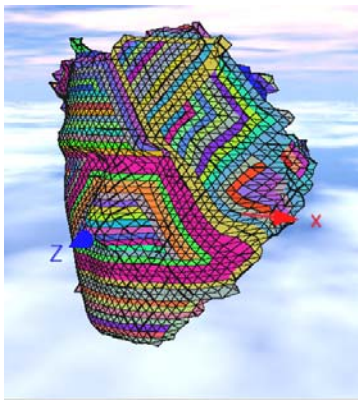
[4] Stuart Russell, Peter Norvig, Chapter 4, Artificial Intelligence: A Modern Approach Prentice-Hall, (1995), ISBN 0-13-103805-2

[5] Triangle strip, Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Triangle_Strip

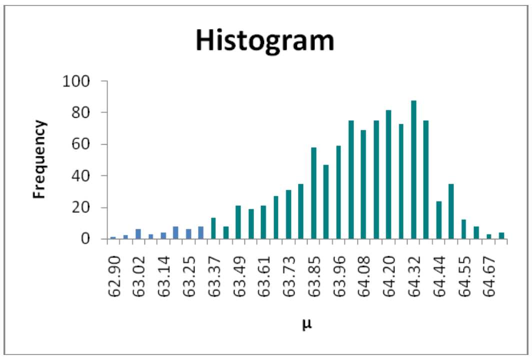
ANNEXES

Model	Triangles	Vertices	Deterministic μ	Evolutionary μ
face	3186	1683	63.30%	**64.73%**
grundy	12412	6208	61.70%	**62.29%**
bunny	69491	34834	60.79%	**60.91%**
roseburg	80423	40343	56.85%	**56.93%**

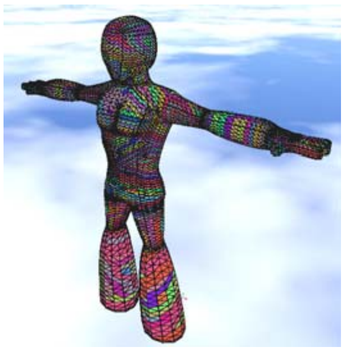
Table 1: Compression levels for 4 models



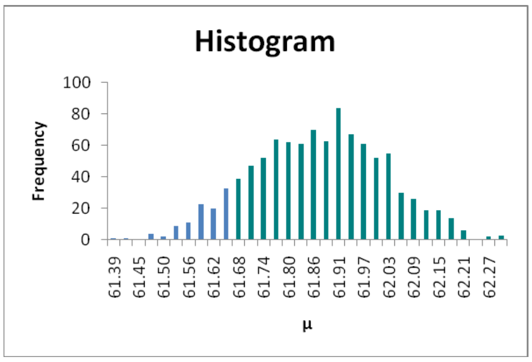
a) face



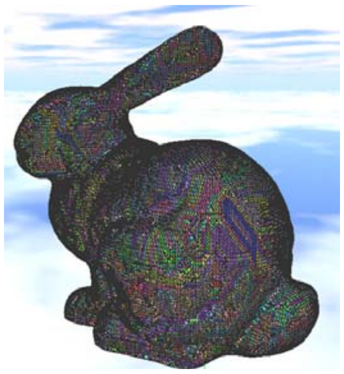
a') face histogram resulted after 1000 experiments



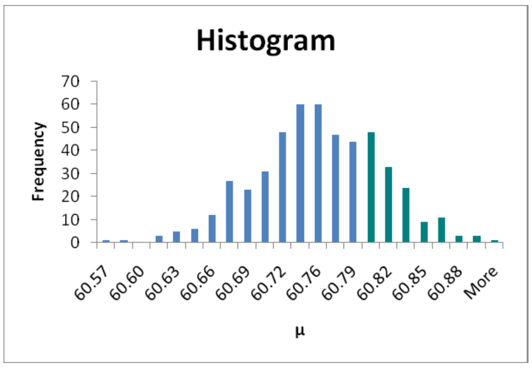
b) grundy



b') grundy histogram resulted after 1000 experiments



c) bunny



c') bunny histogram resulted after 500 experiments