3^{rd} Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2006)
C. Mendoza, I. Navazo (Editors)

Partitioning Meshes into Strips using the Enhanced Tunnelling Algorithm (ETA)

Massimiliano B. Porcu, and Riccardo Scateni

Dipartimento di Matematica e Informatica, University of Cagliari, Cagliari, Italy

Abstract

Triangle meshes are the most used representations for three-dimensional objects, and triangle strips are the organization of triangles mostly used for efficient rendering. Since the problem of optimal strip decomposition of a given mesh is NP-complete, many different heuristics have been proposed; the quality of the stripification is usually evaluated using standard indicators as the total number of strips, the number of isolated triangles, the cache coherence, the number of swap vertices.

In this paper we present the Enhanced Tunnelling Algorithm (ETA), a stripification method working on the dual graph of a mesh. The method uses a sophisticated mechanism of dynamical update of identifiers, guided by a localization procedure. The algorithm adopts a modified search approach in the dual graph that accelerated the convergence speed of the algorithm.

The ETA results efficient and robust, able to deal with datasets of any dimension. The quality of the stripification is remarkable: very few strips (not seldom just one), no isolated triangles, good cache coherence (ACMR value), good number of vertex per triangle.

1. Introduction

A strip of triangles is a set of connected triangles characterized by the property that a new vertex in the list of vertices inserts a new triangle in the structure. Triangle strips are widely used for efficient rendering of solid objects represented by triangle meshes: in pre-processing, the mesh is partitioned into a set of triangle strips (possibly composed of one isolated triangle) and then each strip is passed to the Graphics Processing Unit (GPU) for rendering. The two opposite results of the stripification process are: at worst, the collection of all the isolated triangles defining the original mesh needing $3n$ (where n is the number of triangles in the mesh) vertices for rendering, at best, a single strip representing the whole mesh, with all the edges turning the right way one after each other, needing just $n+2$ vertices for rendering.

It is worth reminding that there are proofs [GJT76, AHMS96] that a problem equivalent to searching a single strip on the mesh (finding a Hamiltonian path on the dual graph) is an NP-complete problem, thus the stripification process is based on global or local heuristics.

The total number of strips and isolated triangles are not the only parameters to measure the quality of a stripification produced by a given heuristics. Another tight bound in the process is represented by the hardware architecture of current GPUs. Since they are capable of storing in a local cache a number k of vertices sent down the pipeline (values of k, even if steadily increasing are several order of magnitude less than the number of vertices in a mesh), is far more convenient, building the strip, to reference vertices in the cache instead of sending them back down the pipeline. Normally, each vertex is processed more than once, since each vertex belongs to six triangles on average. Counting how many vertices are not found in the cache is thus a crucial parameter to evaluate the efficiency of the stripification. This indicator of efficiency is called Average Cache Miss Ratio (ACMR) [Hop99] and is the ratio between the number of cache misses during rendering and the number of triangles. This value can theoretically range from an upper bound of 3.0 (corresponding to an empty cache) to a lower bound of 0.5 when all the vertices are found in the cache (all the vertices have to be sent at least once) [BG02].

For heuristics that produces *generalized triangle strip* [Dee95], even the number of swaps per strip is an important quality parameter. We can express it as *vertices per triangle* *ratio* (number of vertices of the stripification divided by the number of triangle in the mesh); the minimum value for this ratio is $(n+2)/n$.

Our approach to find a stripification encoding of a mesh, operates on the dual graph (see sections 3 and 4) using a single topological operator, known as the *tunnelling operator*, introduced first by Stewart [Ste01] and subsequently deeply improved by the authors [PS03]. Our improved version of the algorithm relies on a single relevant parameter, the *tunnel length*, which influences both the time spent to stripify the mesh and the final set of strips obtained (number and mean length). We already showed in [PS03] and [PSS05] how to overcome the problem of the loops in the dual graph, introducing dynamic management of the node identifiers (ID). Nevertheless, this enhancement led to increasing the execution time, especially using high tunnel length. This made the method not completely feasible when applied to large meshes.

In this paper, we present an *Enhanced Tunnelling Algorithm* (ETA), that appears to be an effective, robust and efficient stripification method, able to deal with dataset of any dimension and generating strips of good quality.

In this picture, the major contributions of this work are:

Dynamic ID management: we adopt a novel dynamic ID management technique, introducing a *localization procedure*; this drastically reduces the execution time of the algorithm, in a way that is quantitatively dependent from the mesh dimension and the tunnel length used. On the other hand, given a fixed time, it is now possible to use longer tunnel than in the previous version. This is particularly important for large datasets.

Seeking in the dual graph: we adopt a new search procedure for tunnels in the dual graph, passing from breadth-first to depth-first search. A breadth-first compared to a depth-first approach, slow down the convergence of the algorithm. Using this approach we are able to reach more quickly results near the global minimum (single strip).

Single strip encode: Running the algorithm with high tunnel length and avoiding to get stuck in local minima makes the production of a single strip possible for the method. This is quite frequent for *closed* (watertight) datasets, independently from the dimension and the genus.

Stripification quality: even if a single encode is not obtained, the total number of strips is often really low, down to few ones. Considering the built-in feature of the method, that is able to produce strips with high cache coherence, it appears that the quality of the encoding produced by the method is remarkable, according to the standard benchmarking tools.

The rest of this work is organized as follows: in section 2 we briefly go over the previous work done in stripification; we then show, in section 3, the relations existing between the triangle mesh and its dual graph and explain how the ETA works; sections 4 and 5 are dedicated to show the details of the algorithm and its implementation; in section 6 we show the results obtained using our algorithm on several meshes widely used in literature for benchmarking; finally, in section 7 we draw our conclusions and describe the future evolutions of this work.

2. Previous Work

A mesh is usually stored as a quadruple $M = (K,V,D,S)$ where K is a simplicial complex describing the connectivity of the mesh, $V = v_1,\dots,v_m$ is the set of positions defining the mesh, D are the discrete and S the scalar attributes. Using graphics API, like OpenGL of DirectX, K becomes implicitly defined by the order in which the vertices V_i are sent to the GPU. This is why arranging well the triangles in strips is important for efficient rendering. We can split the field of research in two: stripification as pure pre-processing, stripification while rendering the mesh.

In the former field, Evans et al. [ESV96] were the first to work on improving the SGI algorithm distributed with the first implementation of OpenGL; Chow [Cho97] proposed a stripification method working on a *generalized triangle mesh* [Dee95] allowing thus to define strips of triangles where the order of vertices inside the triangle can change from one to another; Speckmann et al. [SS97] defined a special stripification optimized for terrain models (2.5D models); Xiang et al. [XHM99] work on the spanning tree of the dual graph to obtain as few strips as possible; Hoppe [Hop99] is the first to propose the usage of a simple greedy algorithm to take advantage of the caching strategy of the graphics boards; Isenburg [Ise01] relies on an existing stripification code for pointing out how much can be saved using a compression scheme for strips; Estkowski et al. [EMX02] instead produces theoretical results regarding the optimal decomposition of a given mesh in strips.

In the latter field, Hoppe [Hop97] determines the triangle strips on-the-fly after the identification of the visible triangles extracted from a progressive mesh; El-Sana et al. [ESAV99, ESEK*00] uses a special data structure, called *skip-strip* to allow efficient selection of triangles and construction of strips while changing the view-point; Stewart [Ste01] first proposes the usage of the tunneling operator on CLOD (Continuous Level Of detail) suggesting to use it whenever the level of detail changes; Shaefe et al. [SP03] introduces a data structure called *DStrips* with algorithms able to preserve pre-computed triangle strips through changes in a CLOD mesh; Ramos et al. [RCBR04] suggest the usage of a data structure called *LodStrips* to keep information on which strips use at different level of details; [PSS05] propose a mixed approach for strip repairing while changing level of detail in a CLOD based on pure topological consideration and the use of tunnelling.

In this context, it is worth mentioning the work reported

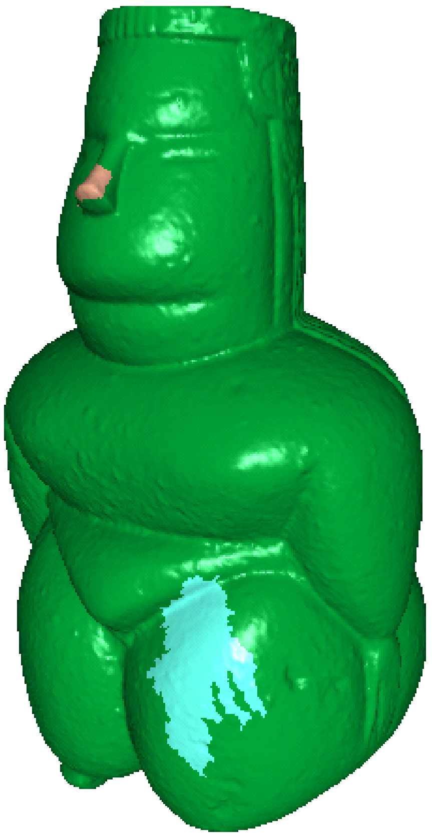


Figure 1: *An image of the triangle mesh representation of the* Dea Madre*, a statue exposed at the National Archeological Museum of Cagliari; the mesh has* 571,806 *triangles, the rendering shown here used three strips obtained in 6,217 sec.*

in [GE04] and [DGBGP06], since it follows an approach similar to ours. The goal is building a Hamiltonian path on the dual graph of the mesh, thus resulting in a single strip in the triangulation. To do so, they first build a perfect matching in the graph, and then use topological operations on the graph inserting, if needed, new triangles. Their results are in a way more complete, ensuring always a single strip; on the other hand, our approach shows that, in several cases, is possible to obtain a single strip encode without changing M, that is, with no extra *ad-hoc* triangles addition.

3. Operating on the Dual Graph

The usage of the tunnelling operator requires that we refer to the mesh explicitly in term of its adjacency graph, usually called its *dual graph*. Each node of the graph is associated with a triangle of the mesh and an edge represents an adjacency relation. Each node has, at most, three incident arcs. Moreover, if the original mesh is closed (i.e.: it has no borders) each node has exactly three incident arcs (see figure 2).

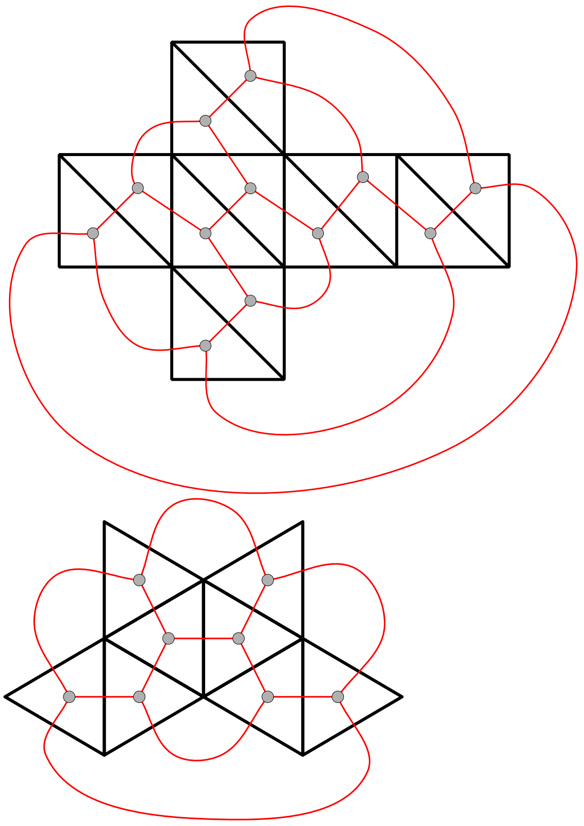


Figure 2: *From left to right, an unfolded triangulated cube and an unfolded octahedron, both with the dual graph of the triangle mesh.*

3.1. The Tunnelling operator

The tunnelling algorithm [Ste01, PS03, PSS05] stripifies the mesh using just a single operation on its dual graph.

To explain how it works we need to define how a graph edge can be *colored*. An edge in the graph has two possible colors (see figure 3):

black linking nodes associated to triangles in the same strip;
red linking nodes associated to triangles not belonging to the same strip.

In every node there are, at most, two incident black edges. The nodes with only one incident black edge are *terminal nodes* (they correspond to the initial and final triangle of the strip). Nodes with three incident red nodes are associated to isolated triangles.

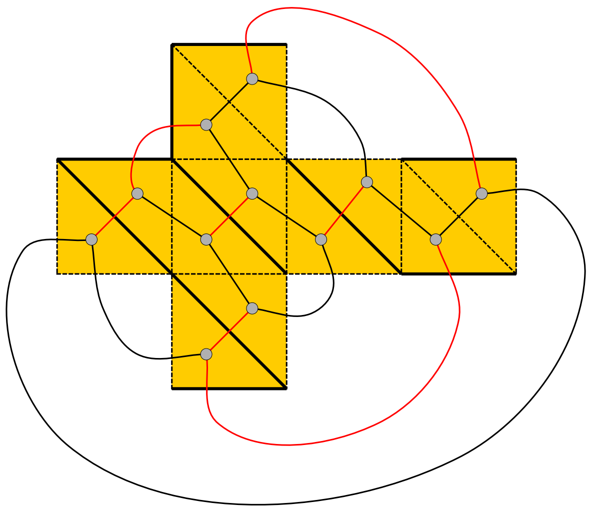


Figure 3: *The mesh of figure 2 represented as a single strip and its dual graph colored accordingly.*

The only operation performed on the mesh is searching a path in the graph called *tunnel*. A tunnel is an alternating sequence of black and red edges, starting and ending with a red edge, connecting two terminal nodes. Its length is always odd and we denote by k-tunnel a tunnel of length k.

Finding tunnels in the graph, is possible to increase the length of the strips while reducing their number. When a tunnel is found, in fact, we *complement* the path, that is, changing each black edge in a red one and vice-versa. With every complement operation we decrease by a unit the number of black paths (strips in the triangulation) on the graph. See figure 4 for an example.

The main weakness of the original version of the algorithm [Ste01], is the creation of loops in dual graph during tunnel search. The set of *no-loop rules* proposed by the author is not enough to avoid this problem. In [PS03] we extended this set of rules, avoiding the generation of loops for any graph topology. The price to pay for imposing these extended *no-loop rules* is keeping track of every different strip in the graph. This is done tagging each node of the graph (triangle) with an identifier (ID) corresponding to the strip it belongs to. During the tunnel search is necessary to operate a *run-time update* of these identifiers, taking into account that each time a red edge is switched to a black one, two strips are merged and have to be retagged accordingly.

3.2. Tunnelling on CLOD

The tunnelling technique as exposed here can be easily used to keep a mesh stripified while changing level of detail navigating in a CLOD structure. A comprehensive explanation on how this can be accomplished is in [PSS05].

These results can be compared to the ones described in [DGGP05]. The difference between the two approaches is mainly in the initial strategy adopted to find the stripification.

4. The Enhanced Tunnelling Algorithm (ETA)

Dynamical update of IDs is the main complication for the implementation of the algorithm. When switching a black edge into a red one, a strip is split in two parts; those parts can be merged to different strips, switching edges from red to black. This mechanism can also affect portion of the *same strip*, several times, in deep nested way [PS03]. Moreover, changes have to be discarded if the search does not reach the goal.

The simplest way to perform this task is to follow the strip, updating its ID node by node; it is also possible to use some advanced data structure, but in our experience this is a further complication of the code, that does not correspond to a substantial improvement in time performance.

Updating ID node by node, the efficiency of the operation obviously decreases as the average strip length grows. This kind of approach gives to the method some kind of *non local behaviour*. This is definitively a problem since much of the propagations attempts are unfruitful and then discarded. In fact, as the strips get longer, the probability that searching for tunnels ends in finding loops increases and then the dynamic changes of IDs are discarded.

The original tunnelling method [Ste01] was inherently local: the operator was limited, in traversing the graph, by the length of the searched tunnels. Nevertheless, the introduction of dynamic ID management dramatically increased the overall performance of the algorithm, since the discard-backtrack procedure was far more efficient and the maximum tunnel length drastically decreased on average to reach comparable results.

In this paper, we propose a solution to the problem, the ETA, that keeps the tunnelling operation local.

4.1. Localization

Our solution relies on a mechanism that fixes an upper bound for the propagation. In other words we don't change the IDs of the edges external to the *k-star* of the terminal node from which the search starts from. We call k-star the boundary of the set of triangles k steps away from the initial one (see figure 5)

The localized algorithm performs the following steps:

1. Fix k, the maximal tunnel length;
2. Select a terminal node in the graph, where to start the tunnel search from;
3. Find its k-star;

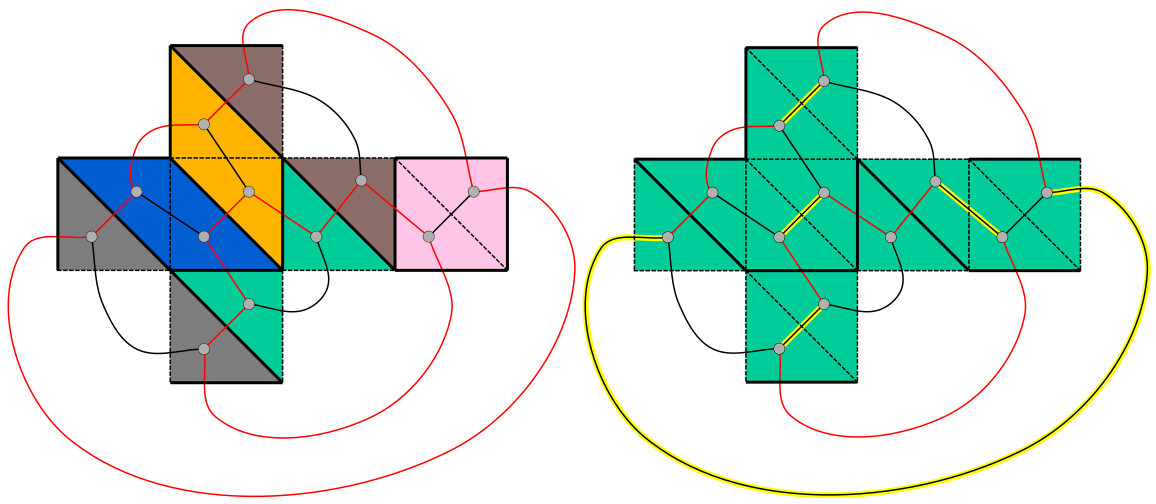


Figure 4: *An example of tunnelling. On the left we can see all the black edges dropped by the first stripification phase, notice that, at this stage, all the red edges are* 1*-tunnels. On the right we evidence the complemented edges: since each complement operation decreases by one the number of strips, after five such operations we pass from the initial six strips to one, and thus complete the stripification.*

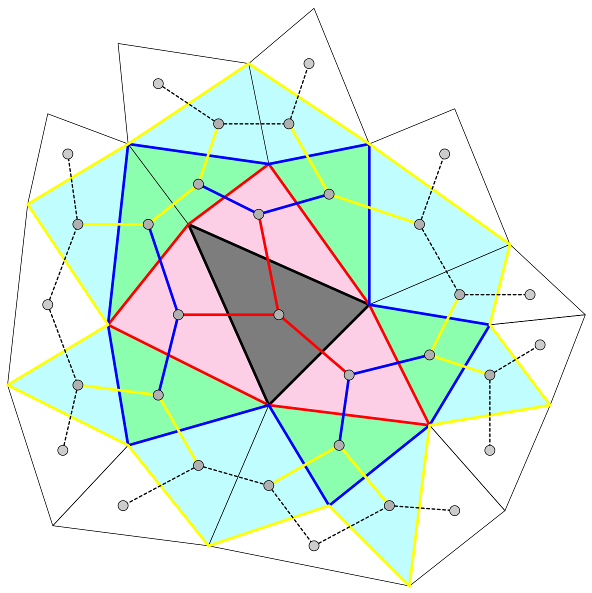


Figure 5: *The first three k-star of the triangle marked dark grey: the 0-star is the triangle itself, the 1-star is marked light red, the 2-star light green and the 3-star light blue. Notice that k-tunnels are completely inside the k-star where also longer tunnels can be found.*

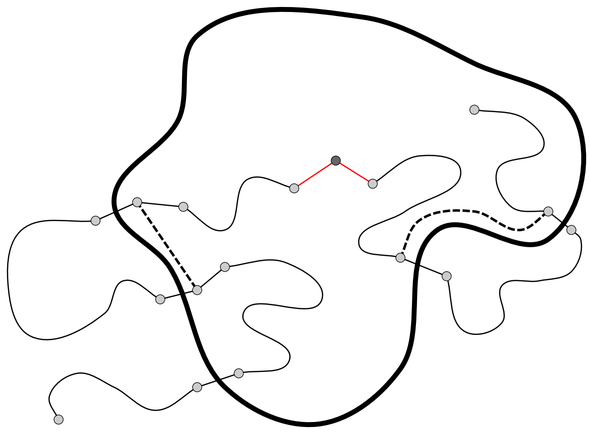


Figure 6: *The dark node is the selected terminal, the dark boundary is its k-star; the edges crossing the boundary are temporarily removed from the graph and the new, dashed ones, replace them.*

4. For any strip crossing the boundary of the k-star count the number of intersection, say n and, while $n > 1$, take a consecutive pair of edges of the strip crossing the boundary, create a temporary fake edge connecting them and decrease n by 2 (see figure 6);
5. Launch the tunnelling;
6. Eliminate the temporary edges and restore the strips modified at step 4. changing their IDs only if a tunnel was found;
7. Change terminal node and restart from step 2.

This procedure is guided from the value of k that is the only parameter the user is asked to set.

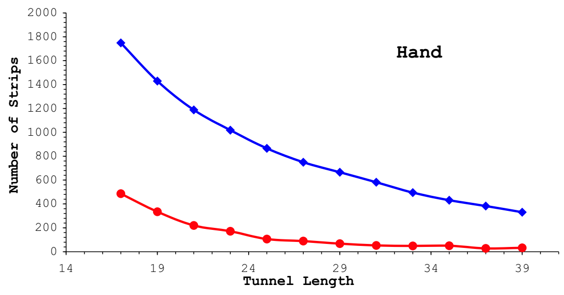
4.2. Forecast

Another enhancement, allowing to further improve the performance, is the introduction of an *access threshold* in performing the tunnelling operation. Searching operation from a terminal node stops when a tunnel is found (positive result) or when all possible paths up to a fixed maximum length k has been analyzed (negative result). What we found in our experiments is that the most part of the time spent by the program is due to a small subset of searches. For high values of k, the proportion is around $90-10$ (90% of the execution time used in 10% of the searches) with peaks up to $95-5$. Moreover, a high fraction of the *expensive* researches has negative result. Using an appropriate time threshold for a search task from a single seed, we have high probability to avoid *expensive* + *negative* researches. The threshold, dependent from the data set, is practically fixed limiting the maximum number of accesses to the nodes that the searching algorithm can perform.

5. Searching approach

The overall performance of ETA, in terms of total number of strips, is strictly related to the searching approach in the dual graph. In [Ste01, PS03], a breadth-first approach is proposed: starting from length $k=1$, search all k-tunnels and increase k only if no k-tunnels are any longer present. A different result is obtainable using a depth-first approach, that is set a maximum value of k and accept every tunnel found, independently from its length.

Our results show that the convergence of the search is much faster when using the latter approach. As one can see in Fig. 7 where we show data obtained with two datasets over which we measured the behavior using the alternative searching strategies, the difference is quite relevant. We can notice that to reach comparable stripifications using the two different approaches it is necessary to set a tunnel length much longer in the breadth-first approach. Since the time needed to perform the two different search is comparable, we can conclude that the overall performance of the algorithm benefit very much from this choice.



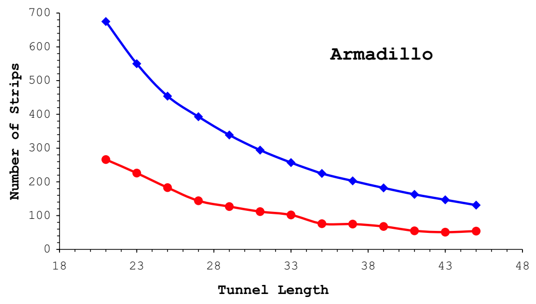


Figure 7: *The blue line shows the convergence of the breadth-first approach, the red line the convergence of the depth-first approach.*

6. Results and Discussion

We applied our method to a large variety of meshes, ranging in size and with different characteristics. In table 2 we list the features of each mesh and the time used to obtain the results listed in table 3.

The last three datasets are very irregular, composed by several unconnected portions; the number of portions is thus the lower limit for the obtainable number of strips. Moreover, some of these portions consist of a single triangle (e.g.: in the Stanford Dragon dataset there are 88 isolated triangles, and 22 two-triangles components) and this influences also another parameter we list in table 3, the number of isolated triangles.

We obtained all the reported results on a PC with an AMD Athlon 1.6 GHz CPU, with 1 GB of RAM.

6.1. Time Performance

The features of the ETA induce us to switch the focus from *execution time* concept to *time budget* concept. The algorithm perform a systematic search on the dual graph, taking into account every path of length less or equal a fixed value k. The number of possible paths grows exponentially with the value of k; thus, it is practically impossible to test every possibility. In this situation, we can not say that the program *terminates* its own task or *exhausts* its seeks. If the mesh has not being encoded in a single strip, it will be always possible to try to improve the result, searching for other tunnels with higher value of k. The quality of the final result, in terms of number of strips, depends on the maximum tunnel length and *the amount of time the user accepts to wait*.

The enhancements we described in section 4, *substantially* increases the efficiency of the algorithm, allow us to operate on very big meshes with high tunnel length: this was not possible with the previous version of the algorithm [PSS05]. This means practically that we are able to obtain better results from the "total number of strips" point of view.

Table 1: *Number of strips obtained using different time budgets on the same dataset.*

	200 Secs	400 Secs	1000 Secs	2500 Secs
Horse	66	13	8	6
Dea Madre	172	78	29	7
Hand	86	51	33	23
Buddha	373	161	115	49

Nevertheless, it is not easy to exactly quantify the amount of the benefit; it is strictly dependent from the mesh size and the depth of the search. For instance: to obtain 17 strips on the *Happy Buddha* mesh, we operate with tunnel length 200 and used almost 10 hours of CPU; with the same parameters, the previous version did not terminate after 1 week of CPU!

In table 1 we summarize the results obtained for four large data sets: *Hand*, *Armadillo*, *Dea Madre* and *Happy Buddha*, allocating different time budget. As we can see, it is possible to obtain good results with small time budgets; moreover, the result is always subject to improvement (unless we get to the minimal number of strips as stated before) when we decide to use larger budgets. It is worth to point out that the stripification is obtained in a pre-processing step, so it is in general possible to use large (but reasonable) time budgets.

In table 3 we listed our *best results*, obtained allocating a time budget up to a maximum of $\sim$ 10 hours. That was necessary especially for big meshes as *Hand*, *Armadillo*, *Dea Madre* or *Happy Buddha*. Because of the method characteristics, we are confident that the results for these four meshes could still be improved, searching for longer tunnels and thus using more time.

6.2. Single strip encode

Using the ETA is possible, in many cases, to find a single strip encoding of a mesh. This result has a double importance, because it shows two different facts completely new: several meshes *admit a single strip encoding* and *is possible to find it*. For what we known, no other methods can produce the same result. Also in the approach of Gopi and Eppstein [GE04], to obtain a single strip encoding is often necessary to modify the base mesh, adding extra triangles selected *ad-hoc*.

In table 3 are listed the number of strips and isolated triangles obtained for each dataset. We produced a single strip encode for *Goblet*, *Heart*, *Fandisk*, *Oilpump*, *Horse*, *Sculpture* (*Goblet*, *Fandisk*, *Horse* and *Sculpture* appear also in [GE04]). Dataset *Cessna* is composed of 11 unconnected parts; we obtained 13 strips: 9 parts are single strip encoded.

All the above mentioned meshes are *closed* meshes. This seems to be the crucial feature to obtain a single strip encoding. Other characteristics, as dimension or genus, appear less important; dataset *Sculpture*, for instance, has genus three.

Table 2: *Time in hour:minute:seconds to obtain the best stripification we got from the listed datasets; each dataset is defined by the number of vertices and triangles of the mesh, and the number of components not connected, resulting in not connected graphs.*

Data Set	Triangles	Vertices	Parts	Time
Goblet	1,000	520	1	0.08
Heart	1,717	861	1	0.78
Fandisk	12,946	6,475	1	1.08
Cessna	13,546	6,795	11	54
Oilpump	20,544	10,274	1	1:46
Sculpture	57,780	25,386	1	1:14:01
Bunny	69,451	35,947	1	17:12
Horse	96,966	48,485	1	9:03:03
Shoe	156,474	78,239	1	36:43
Armadillo	345,944	172,974	1	1:52:22
Dea Madre	571,806	290,449	1	1:43:37
Hand	654,666	327,323	1	47:46
Buddha	1,087,716	543,652	1	9:53:04
David	99,995	51,601	26	55:24
Dragon	871,414	437,645	151	2:00:35
Guyver	1,394,874	701,392	65	1:54:25

For open meshes, the algorithm is not in general able to obtain a single strip encoding. The number of strips grows with the number of *border triangles*; in very irregular meshes with a high fraction of border triangles, as *David*, *Dragon* and *Guyver*, the final number of strips can be quite distant from one.

6.3. Stripification Quality

Even in situations where we are not able to produce a single strip encode, the total number of strips is very low, up to few units for regular meshes. This is true also for big datasets as for instance, *Dea Madre* (570*k* triangles, 3 strips) or *Hand* (650*k* triangles, 14 strips).

In addition of number of strips and isolated triangles, the quality of a stripification is evaluated considering cache coherence and total number of swaps. As already stated in section 1, the ACMR is widely used to measure how the triangle strips are compliant with modern GPU architecture. In table 3, we list the ACMR value computed for caches of 32 entries; as we can see, the value is almost independent from the characteristics of the dataset we start from; moreover, cache coherence is a *built-in* feature of the method: strips are produced with a good ACMR value automatically, and do not require to be processed in any way for that.

The average ACMR value for a 32 entries cache is $\sim$ 0.70; this is a good, and is in general obtained after a preprocess-

Table 3: *For each dataset present in table 2 we list: in column (a) the number of strips, in column (b) the number of isolated triangles, in column (c) the ACMR for a cache of size 32 and in column (d) the vertices per triangle ratio.*

Dataset	(a)	(b)	(c)	(d)
Goblet	1	0	0.91	1.18
Heart	1	0	0.74	1.32
Fandisk	1	0	0.71	1.44
Cessna	13	0	0.65	1.56
Oilpump	1	0	0.70	1.52
Sculpture	1	0	0.69	1.46
Bunny	14	0	0.72	1.47
Horse	1	0	0.70	1.46
Shoe	5	0	0.70	1.51
Armadillo	18	0	0.70	1.46
Dea Madre	3	0	0.70	1.46
Hand	14	0	0.70	1.46
Buddha	17	0	0.69	1.45
David	242	17	0.71	1.45
Dragon	477	96	0.70	1.46
Guyver	725	18	0.71	1.45

ing algorithmic step, as in [BG02], or imposing some constrain in strip generation as in [GE04]. For instance, our results for *Fandisk, Horse* and *Buddha* are practically coincident with the result listed in table 1 of [DGBGP05], respectively ACMR = {0.71, 0.71, 0.68}.

Furthermore, our studies show that, for caches with 64 entries, the average ACMR value is 0.65.

To take into account the number of swaps (remind we are generating a generalized strip), we list in table 3 the number of vertices per triangle ratio. Our results can be considered in full agreement to the values obtained with standard real space methods, as the various SGI implementations [Van02].

For irregular datasets as *David*, *Dragon* and *Guyver*, even if the number of strips remains high, the cache coherence is good, with an ACMR value of ~ 0.70, as well as the vertex per triangle ratio.

7. Conclusions and Future Work

We described a stripification algorithm based on a simple topological operation on the dual graph of the triangle mesh that is robust and easy to use. The only two parameters needed to drive its execution are the maximal tunnel length and the predictor time budget. The localization strategy imposed to the tunnelling brings to a major performance and quality improvement over the previous non-localized version. The results we presented are a demonstration of the good quality of the stripification the algorithm can produce, evaluated using ACMR and vertices per triangle ratio.

We plan to investigate the limits of the stripification algorithm when applied to huge meshes, elaborating subdivision strategy allowing to stripify every single portion of the mesh in-core. This would result in an application of the algorithm to meshes of any size.

Acknowledgements

The *Dea madre* dataset was obtained from tridimensional scans of manufacts exposed at the *Museo Archeologico Nazionale* in Cagliari. We are indebted to its director, Carlo Tronchetti, for letting us use these digital data and to the VCG of the ISTI-CNR in Pisa for the hardware and software used in the acquisition and re-construction.

We thank Daniele Vacca for his work on the visualization tool.

References

[AHMS96] Arkin E. M., Held M., Mitchell J. S. B., Skiena S. S.: Hamiltonian triangulations for fast rendering. *The Visual Computer 12*, 9 (1996), 429–444. 61

[BG02] Bogomjakov A., Gotsman C.: Universal rendering sequences for transparent vertex caching of progressive meshes. *Computer Graphics Forum 21*, 2 (2002), 137–148. 61, 68

[Cho97] Chow M. M.: Optimized geometry compression for real-time rendering. In *IEEE Visualization '97* (Nov. 1997), pp. 346–354. 62

[Dee95] Deering M. F.: Geometry compression. In *Proceedings of SIGGRAPH 95* (Aug. 1995), Computer Graphics Proceedings, Annual Conference Series, pp. 13–20. 61, 62

[DGBGP05] Diaz-Gutierrez P., Bhushan A., Gopi M., Pajarola R.: Constrained strip generation and management for efficient interactive 3d rendering. In *Proceedings of Computer Graphics International 2005* (June 2005), pp. 115–121. 68

[DGBGP06] Diaz-Gutierrez P., Bhushan A., Gopi M., Pajarola R.: Single strips for fast interactive rendering. *The Visual Computer 22*, 6 (June 2006), 372–386. 63

[DGGP05] Diaz-Gutierrez P., Gopi M., Pajarola R.: Hierarchyless simplification, stripification and compression of triangulated two-manifolds. *Computer Graphics Forum 24*, 3 (Sept. 2005), 457–467. 64

[EMX02] Estkowski R., Mitchell J. S. B., Xiang X.: Optimal decomposition of polygonal models into triangle strips. In *Proceedings of the eighteenth annual symposium on Computational geometry* (2002), ACM Press, pp. 254–263. 62

[ESAV99] El-Sana J. A., Azanli E., Varshney

A.: Skip strips: Maintaining triangle strips for view-dependent rendering. In *IEEE Visualization '99* (Oct. 1999), pp. 131–138. 62

[ESEK*00] EL-SANA J., EVANS F., KALAIAH A., VARSHNEY A., SKIENA S., AZANLI E.: Efficiently computing and updating triangle strips for real-time rendering. *Computer-Aided Design 32*, 13 (Oct. 2000), 753–772. 62

[ESV96] EVANS F., SKIENA S. S., VARSHNEY A.: Optimizing triangle strips for fast rendering. In *IEEE Visualization '96* (Oct. 1996), pp. 319–326. 62

[GE04] GOPI M., EPPSTEIN D.: Single-strip triangulation of manifolds with arbitrary topology. *Computer Graphics Forum 23*, 3 (Sept. 2004), 371–379. 63, 67, 68

[GJT76] GAREY M. R., JOHNSON D. S., TARJAN R. E.: The planar hamiltonian circuit problem is NP-complete. *SIAM Journal of Computing 5*, 4 (Dec 1976), 704–714. 61

[Hop97] HOPPE H.: View-dependent refinement of progressive meshes. In *Proceedings of SIGGRAPH 97* (Aug. 1997), Computer Graphics Proceedings, Annual Conference Series, pp. 189–198. 62

[Hop99] HOPPE H.: Optimization of mesh locality for transparent vertex caching. In *Proceedings of SIGGRAPH 99* (Aug. 1999), Computer Graphics Proceedings, Annual Conference Series, pp. 269–276. 61, 62

[Ise01] ISENBURG M.: Triangle strip compression. *Computer Graphics Forum 20*, 2 (2001), 91–101. 62

[PS03] PORCU M. B., SCATENI R.: An iterative stripification algorithm based on dual graph operations. In *Proceedings of the EuroGraphics conference 2003 (short presentations)* (2003), The Eurographics Association, pp. 69–75. 62, 63, 64, 66

[PSS05] PORCU M. B., SANNA N., SCATENI R.: Efficiently keeping an optimal stripification over a clod mesh. *Journal of WSCG 13*, 2 (feb 2005), 73–80. 62, 63, 64, 66

[RCBR04] RAMOS F., CHOVER M., BELMONTE O., REBOLLO C.: An approach to improve strip-based multiresolution schemes. *Journal of WSCG 12*, 1 (Feb. 2004), 349–354. 62

[SP03] SHAFAE M., PAJAROLA R.: Dstrips: Dynamic triangle strips for real-time mesh simplification and rendering. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications* (2003), pp. 271–280. 62

[SS97] SPECKMANN B., SNOEYINK. J.: Easy triangle strips for TIN terrain models. In *Canadian Conference on Computational Geometry* (1997), pp. 239–244. 62

[Ste01] STEWART A. J.: Tunneling for triangle strips in continuous level-of-detail meshes. In *Graphics Interface 2001* (June 2001), pp. 91–100. 62, 63, 64, 66

[Van02] VANECEK P.: Comparison of stripification techniques. In *Proceedings of 6th Central European Seminar on Computer Graphics CESCG'02* (2002), pp. 65–74. 68

[XHM99] XIANG X., HELD M., MITCHELL J. S. B.: Fast and effective stripification of polygonal surface models. In *Proceedings of the 1999 symposium on Interactive 3D graphics* (1999), ACM Press, pp. 71–78. 62

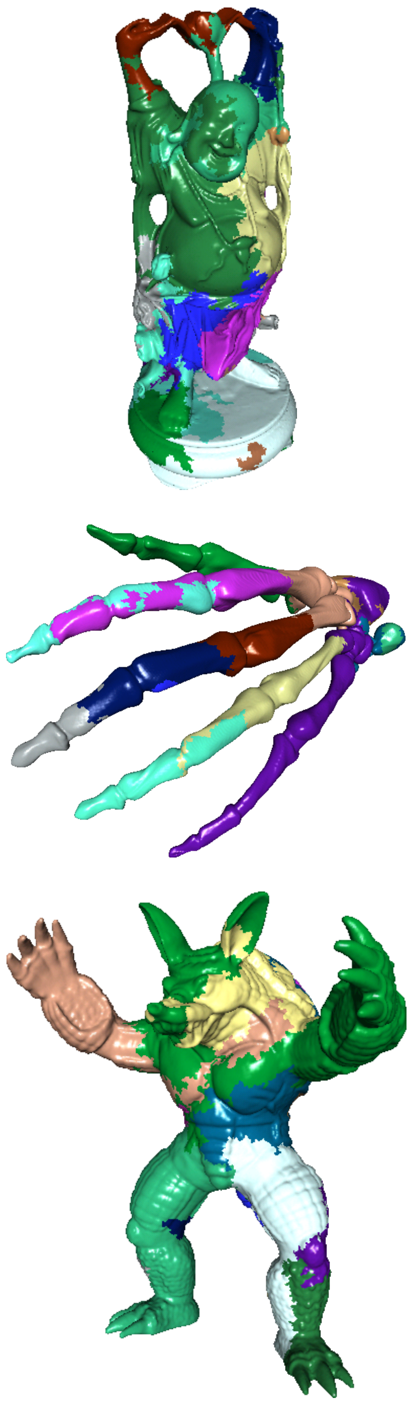


Figure 8: *From top to bottom: images of the* Happy Buddha *(1,087,716 triangles), the* Hand *(654,666 triangles) and the* Armadillo *(345,944 triangles) datasets; they are, respectively, rendered with 19, 14, and 18 strips.*