**3D Mesh Processing: Triangle Mesh Stripification**

**Real-Time Computer Graphics**

**Jak Boulton**

**10026037**

|  |
| --- |
| *Abstract*  *This report discusses a small subset of the available stripification algorithms. Each of the algorithms is reviewed with regard to a specific criteria paying extra attention to efficiency of strip generation. An ensemble of algorithm authors and reviewers have been selected and referenced throughout. The preferred algorithm is discussed with regard to the reasons for selection and more specific algorithm details are provided.* |

**1. Introduction**

A vast array of applications rely on rendering a large number of primitives at as high frame rate as physically possible to offer the user the best experience and usability. In areas such as 3D printing and scientific rendering the efficient storage, manipulation and rendering of large volumes of data is crucial. Among others, polygonal surfaces are one of the most widely used representations for geometric models. This is most likely due to the flexibility they bring with compression techniques and portability with popular rendering packages such as 3D Studio Max, Maya and Blender (Kaick, 2004).

This report focuses on a single method for minimising the data sent to the graphics card by converting polygonal models to a series of *triangle strips.* This is a common encoding scheme which works by lower the number of vertices sent to the GPU by repeating previous vertices to render triangles. Triangle strips are widely recognised and supported by many graphics libraries such as OpenGL and DirectX.

In this report a small subset of *stripification* algorithms are discussed in relative detail. A preferred algorithm is chosen with a discussion as to why it was chosen over the other mentioned techniques before delving deeper into how the algorithm works. A conclusion follows containing tabled data detailing the efficiency of the chosen method compared to others.

**2. Algorithms**

There are many algorithms which have been designed and developed over recent years starting with the most popular, SGI’s *tomesh.* This is the both the most publically available and the most used, it has even been used within many other techniques and has been extended and improved by the most authors (Vanecek, 2002). Although it was the first, and has been developed over many years, SGI’s *tomesh* is still able to stripify meshes efficiently.

|  |  |
| --- | --- |
|  |  |
| *Figure 1: A Sequential Strip* | *Figure 2: A Generalised Strip* |

All triangle stripification algorithms contain two main types of triangle strips. As shown in figure 1, a sequential strip is a simple strip of triangles with alternating winding. Figure 2 shows a generalised strip (1, 2, 3, 4, 5, 6) which would create an error triangle (as shown by the red dotted line). This problem is solved by adding a ‘swap’ vertex, altering the index sequence to (1, 2, 3, 4, 3, 5, 6). The swap costs an additional vertex and so is generally avoided by most algorithms. Sequential strips are the most efficient as they send the least number of vertices to the graphics pipeline. To compare to standard rendering of polygonal surfaces (by sending a list of triangles to the graphics pipeline, three points at a time) which costs points, a sequential strip costs a mere . A generalised strip (as shown in figure 2) offers a cost reduction not quite as good as a sequential strip due to the need for swaps () (Vaněček, 2005) (Kim, 2005).

The main problem with SGI’s *tomesh*  is that generalised strips are not avoided and there is no preference as to which strip to choose, it is a simple *greedy* heuristic used to decide which triangle is added to the strip next.

Step-by-step breakdown of algorithm:

1. If there are no more triangles in the triangulation then exit.
2. Find the triangle t with the least number of neighbours (if more than one exists, choose arbitrary).
3. Start a new strip.
4. Insert the triangle t to the strip and remove it from the triangulation.
5. If there is no neighbouring triangle t, then go to 1.
6. Choose new triangle t’, neighbouring triangle t, with the least number of neighbours. If there is more than one triangle t’ with the same least number of neighbours, look one level ahead. If there is a tie again, choose t’ arbitrary.
7. T <- t’. Go to 4.

The next algorithm is another heavily used and available method known as *STRIPE.* This is based heavily on SGI’s *tomesh* but has been expanded to be able to convert meshes from other polygonal types down to triangle meshes before stripification takes place (Vanecek, 2002).

|  |  |
| --- | --- |
| http://cs.queensu.ca/~jstewart/strips/algorithm/bunnySGISmall.png | http://cs.queensu.ca/~jstewart/strips/algorithm/bunnyTunnelSmall.png |
| *Figure 3: The Stanford Bunny, rendered using SGI’s tomesh technique* | *Figure 4: The Stanford Bunny, rendered using the Tunneling algorithm* |

Tunneling is another popular algorithm aimed at producing the smallest number of triangle strips for a given polygonal surface. This is one of the most efficient algorithms available since it is generally recognised that the lower the number of triangle strips, the more efficient the algorithm is. This is due to the reduction in vertices sent to the GPU. Figures 3 and 4 show a comparison between SGI’s *tomesh* and the tunnelling algorithm run on the Stanford bunny model. *Tomesh* generated 705 strips from 69,451 vertices where the tunnelling algorithm generated only 158 strips (Stewart, 2001) (Porcu, 2003).

The most basic way of visualising how tunnelling works is by viewing it as a group of searches over the mesh surface known as tunnels. If tunnels are created and managed correctly, triangle strips can be merged and elongated to such an extent that minimises the number of strips significantly (Porcu, 2005).

Widely known as the most efficient publically available stripification algorithm is FTSG, originally designed by Xinyu Xiang (Xiang, 1999). Xiang, in his paper discusses the comparison between FTSG and other techniques providing tables when required to prove the performance of his algorithm against others. The algorithm contains a wide selection of different algorithms to offer a number of features, breadth-first and depth-first searches of the dual-graph data structure are used to get the best results.

The most basic view of the algorithm is as follows:

1. Compute a triangulation of faces of the model that are not already triangles.
2. Construct a spanning tree, t, in the dual-graph, g, of the triangulation.
3. Partition t, into a set of paths corresponding to Hamiltonian triangulations[[1]](#footnote-1)
4. Greedily decomposing the corresponding Hamiltonian strips into sequential or fan tristrips.
5. Concatinate short tristrips into longer tristrips, using a set of postprocessing heuristics applied to the result of Step 4.

The next section explains the algorithm chosen and the reasons for the choice.

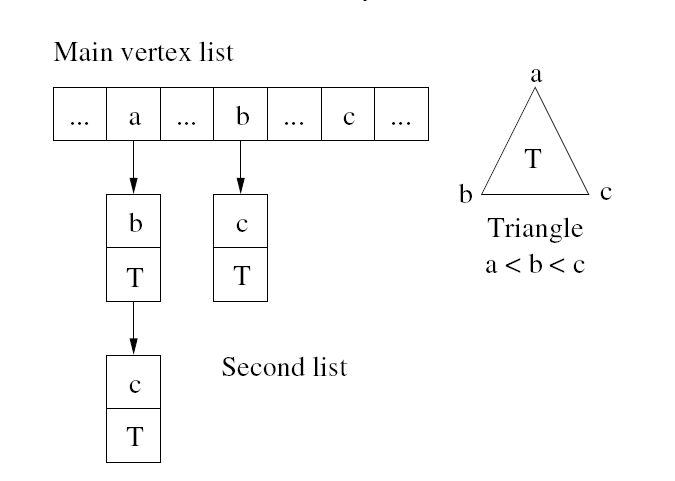
|  |  |  |
| --- | --- | --- |
| Method | Avg. V/T | CPU (ms) |
| -dfs –alt –DP –zero | 1.21 | 0.014 |
| -dfs –alt –seq –zero | 1.23 | 0.009 |
| -dfs –alt –DP | 1.24 | 0.013 |
| Tomesh | 1.36 | 0.027 |
| STRIPE(Convex Faces) | 1.36 | 0.263 |
| SRIPE (Fully Triangulated) | 1.39 | 0.071 |

*Table 1: Performance of the different stripification codes: average vertices per triangle (“v/t”) and CPU time (in ms).*

**3. Strip**

The selected algorithm is known as *Strip* and was originally developed by Kaick in his paper “Efficient Generation of Triangle Strips from Triangulated Meshes” (Kaick, 2004). In his paper, Kaick compares his algorithm with many of the popular algorithms, namely, Skip Strip, FTSG and Stripe and claims – with evidence that *Strip* is the most efficient algorithm.

*Strip* not only beats most, if not all publically available stripification algorithms, it is also very easy to understand and so beats many of the algorithms in its simplistic design. It works much like the standard *tomesh* algorithm, but changes certain heuristics along with the data structure used to optimise creation of the triangle strips.



*Figure 5: Dual Graph Construction*

The *Strip*  algorithm is built around the ability to traverse the surface of a model with ease, with the ability to check triangles and their neighbours. The dual-graph data structure (shown in figure 5) contains all the information required and so is used alongside the algorithm to gain the best possible performance. There are two main stages to the algorithms execution, the first of which is the setup. This is where the data structure is filled with the appropriate information such as a *free* setting denoting that the triangle is not in a strip and a *degree* denoting the number of free neighbouring triangles.

The second phase is the construction of triangle strips. This is when the algorithm searches through the polygonal surface (using the dual-graph) checking neighbours and adding triangles to the strips. The first step is to select a starting triangle, this is done by getting the triangle with the lowest degree – the triangle with the least number of (free) neighbours, and this ensures that isolated triangles are chosen first. The algorithm then goes on to search neighbours (again choosing the neighbour with the lowest degree) before added the favoured neighbour to the strip. This continues until the strip has no neighbours which can be added to the strip. The algorithm uses heuristic cases to ensure efficiency. If a free neighbour has a degree of zero, the neighbour is instantly added to the strip – decreasing execution time. Avoiding swaps is also important for *Strip* since (as explained above) this is inefficient due to the addition cost of vertices sent to the GPU.



*Figure 6: The Stanford Bunny, rendered using Strip.*

The *Strip* algorithm was chosen due to its power, and simplicity. While the algorithm is simplistic on the surface, it is however complex in the way that it functions, a great deal of that complexity is based on the construction of the data structure that allows for the algorithm to run at its best. When researching, many data structures were found, namely, dual-graphs, half-edged and doubly connected edge lists. Each of the data structures found offer similar benefits to the dual-graph data structure in terms of linking each of the triangles to their respective neighbours for ease of traversal.

**4. Conclusion**

This report began with an overview of polygonal surfaces and optimisations such as *triangle strips* before moving onto discussions of a group of the most well-known, efficient and publicly available stripification algorithms. The eye was then placed on the chosen algorithm; *Strip,* its benefits were outlined along with an overview of its processes. Overall, *Strip* was the chosen algorithm due to its simplicity and efficiency.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Number of Strips | | Number of Vertices | |
| FTSG | Strip | FTSG | Strip |
| Buddha | 25567 | 19640 | 1398464 | 1421420 |
| Bunny | 618 | 563 | 81412 | 81908 |
| Canyon | 2297 | 1738 | 120884 | 123152 |
| Champlain | 4357 | 3339 | 255236 | 260369 |
| Crater | 4568 | 3468 | 278565 | 283208 |
| Dragon | 20571 | 15943 | 1121151 | 1140173 |
| Emory Peak | 1744 | 1325 | 93403 | 95060 |
| Hand | 10394 | 8493 | 806855 | 816202 |
| Mars | 462 | 369 | 23010 | 23383 |
| Rice Lake | 9668 | 7322 | 514734 | 523862 |
| Roseburg | 1802 | 1400 | 102920 | 105108 |

The below table compares FTSG with Strip to show which algorithm is the most efficient:



*Figure 7: ‘The Chair’ – rendered using Strip.*

# 

# 5. Bibliography

Kaick, O. M. v., 2004. Efficient Generation of Triangle Strips from Triangulated Meshes. *Journal of WSCG,* 12(1 - 3).

Kim, B.-U., 2005. A Simple and Efficient Triangle Strip Filtering Algorithm. *Journal of information science and engineering,* Volume 21, pp. 1277-1288.

Mirzaian, A., 90. *Hamiltonian Triangulations and Circumscribing Polygons of Disjoint Line Segments,* Ontario: Natural Sciences and Engineering Research Council of Canada.

Porcu, M. B., 2003. *An Iterative Stripification Algorithm Based on Dual Graph Operations.* s.l., EUROGRAPHICS 2003.

Porcu, M. B., 2005. Efficiently Keeping an Optimal Stripification Over a CLOD Mesh. *The Journal of WSCG,* Volume 13, pp. 1213 - 6964.

Stewart, A. J., 2001. Tunneling for Triangle Strips in Continuous Level-of-Detail Meshes. In: *Graphics Interface.* s.l.:s.n., pp. 91 - 100.

Vanecek, P., 2002. *Comparison of Stripification Techniques,* Pilsen, Czech Rep.: Centre of Computer Graphics and Data Visualisation.

Vaněček, P., 2005. *Triangle Strips for Fast Rendering,* Západočeská: Západočeská univerzita v Plzni.

Xiang, X., 1999. *Fast and Effective Stripification of Polygonal Surface Models,* s.l.: s.n.

Botsch, M., n.d. *OpenMesh – a generic and efficient polygon mesh data structure,* s.l.: Computergraphik und Multimedia.

Diaz-Gutierrez, P., 2005. Hierarchyless Simplification, Stripification and Compression of Triangulated Two-Manifolds. *EUROGRAPHICS 2005,* 24(3).

El-Sana, J., n.d. *Efficiently Computing and Updating Triangle Strips for Real-Time,* s.l.: s.n.

Evans, F., 1996. *Optimizing Triangle Strips for Fast Rendering.* s.l., IEEE.

Porcu, M. B., 2003. *Iterative Stripification of a Triangle Mesh: Focus on Data Structures.* Plzen, Science Press.

Shafae, M., n.d. *DStrips: Dynamic Triangle Strips for Real-Time Mesh Simplification and Rendering,* s.l.: ACM.

1. Any hamiltonian cycle in a hamiltonian triangulation of Σ is a spanning polygon of Σ (Mirzaian, 90). [↑](#footnote-ref-1)