# Hamiltonian Triangulations and Triangle Strips

An Overview

Peter Palfrader

February 2011

# Contents

**Abstract**

This work presents triangle strips, a useful concept in computer graphics, and describes existing algorithms in the strongly related category of Hamiltonian triangulations and algorithms to create triangle strips from given triangulations.

# 1 Introduction

In order to do work on an abstract object, to display it, transform it or operate on it in other ways, a computer needs a model of that entity, a concrete representation that describes specific properties of the object. Choosing a suitable representation is of course a very domain specific problem.

In computer graphics triangles are a common primitive for creating such models, for instance for approximating a three-dimensional body's surface with a set of triangles.

When it is time to display an image of this body, the set of triangles is loaded and either processed by the computer's host CPU or sent to the graphics hardware for processing by specialized hardware. The responsible unit then works on each vertex and transforms it as needed, clips lines where necessary, etc.

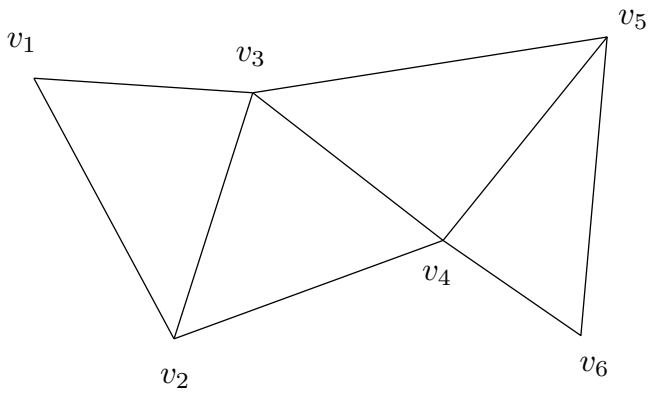


Figure 1: A simple strip of triangles.

Consider the set of triangles in figure 1 which we may want to transfer from the host's main memory to the graphics hardware. A naïve system

would send four triangles with three vertices each, viz. $(v_1, v_2, v_3)$, $(v_2, v_4, v_3)$, $(v_4, v_5, v_3)$, and $(v_4, v_6, v_5)$.

## 1.1 Triangle Strips

One obvious optimization is to not process three vertices per triangle each but to utilize the fact that often triangles are adjacent and as such share two vertices.

Looking at the set of triangles from figure 1 once more, it appears that the list $(v_1, v_2, v_3, v_4, v_5, v_6)$ can also describe these four triangles: The first three elements of that list specify the first triangle, then elements two, three and four make up the second triangle, $v_3, v_4, v_5$ the third and finally the last three list elements represent the fourth. More generally, such a set of $n$ triangles can be represented by a list of elements $(v_i)_{i=1}^{n+2}$, where the $i$th triangle has vertices $v_i$, $v_{i+1}$ and $v_{i+2}$.

Not all groups of triangles can be represented as a *sequential* triangle strip however. Consider the five triangles in figure 2. If, as in the previous example, we simply start at one end and begin listing vertices we would and produce the list $(v_1, v_2, v_3, v_4, v_5)$ for the first three triangles. Unfortunately now we no longer can continue this strip as the final two vertices, $v_4$ and $v_5$ are not one side of the next triangle. Even if we start differently, with $(v_1, v_3, v_2, \ldots)$ or from the right, we eventually end up in a situation like this.

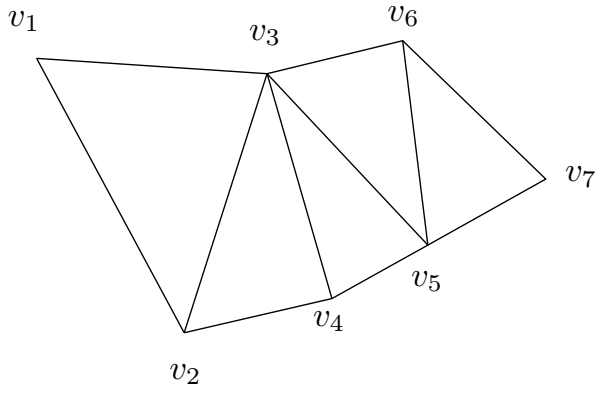


Figure 2: Non-sequential triangles.

Apart from simply starting a new strip whenever we get into such a situation

there are two approaches to handle this kind of triangle set. The first, chosen for the IRIS GL programming interface by SGI, introduces a special *swap* command. This code, as the name suggests, swaps the two most recent elements in the list. With such a command the triangles in figure 2 can be strippified as $(v_1, v_2, v_3, v_4, \mathit{swap}, v_5, v_6, v_7)$. The second approach, of which OpenGL is a representative, does not provide a swap-command. Instead one emulates it by repeating a previous vertex, creating a triangle with an empty area: $(v_1, v_2, v_3, v_4, \mathbf{v_3}, v_5, v_6, v_7)$.

## 1.2 Triangle Fans

A construct similar to triangle strips is the triangle fan. In a triangle fan a single vertex is a point of each triangle.

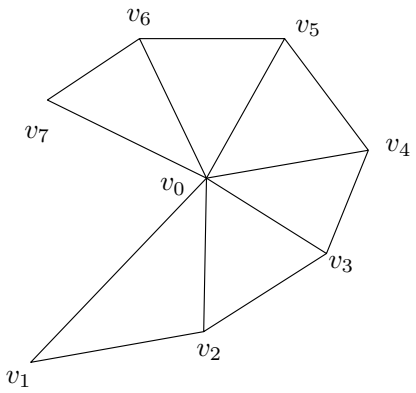


Figure 3: A triangle fan.

One representation of the fan depicted in figure 3 is $(\mathbf{v_0}, v_1, v_2, v_3, v_4, v_5, v_6, v_7)$. The vertex $v_0$ is part of all triangles. Together with $v_1$ and $v_2$ it makes up the first triangle. The second one is built from $v_0$ again, then $v_2$ once more and the new vertex $v_3$. The third triangle is $(v_0, v_3, v_4)$, then two more until the last is finally $(v_0, v_6, v_7)$. Like in triangle strips, $n+2$ vertices are needed to encode a fan of $n$ triangles.

It should be obvious that every convex polygon can trivially be turned into both, sequential triangle strips and triangle fans.

## 1.3 Visibility

In certain applications the orientation of triangles is used for backface culling. In such scenarios it is important to not destroy that information when creating strips from a set of triangles.

In strips the orientation of triangles of course alternates. As such it is a useful convention that for the purpose of visibility the orientation of the first triangle shall be the deciding one.


Figure 4: Two triangle strips: The left strip starts from the left and is $(v_1, v_2, v_3, v_4, v_5, v_6)$, the right strip starts from the right and is $(v_6, v_5, v_4, v_3, v_2, v_1)$.

Figure 4 shows two sequential triangle strips. It can be seen that independent of starting position (left or right) the first triangle (drawn bold) in these strips is always oriented counter-clockwise. There is no simple, sequential strip of these triangles that has a different orientation. If such is wanted the strip can either be split in two or, maybe the preferable option, a swap operation can be used. One possible strip of these triangles that starts with a clockwise triangle is $(v_1, v_3, v_2, \mathit{swap}, v_4, v_5, v_6)$. Again, if swap operations are not available, they can be simulated using zero-area triangles.

# 2 Hamiltonian and Sequential Triangulations

In the previous section we have explained what triangle strips and fans are. We have so far however assumed that the set of triangles are given already. This section will briefly introduce triangulations and their dual graph, then present algorithms for testing whether a given triangulation is Hamiltonian or even sequential and for producing such triangulations if possible.

## 2.1 Triangulations

A triangulation of a polygon is a decomposition of its polygonal area into a set of triangles, such that the triangles do not overlap, do cover the area of the polygon completely, but nothing more, with the additional constraint that any vertex of a triangle already be a vertex of the polygon. Figure 5 shows a simple polygon (i.e. one that is not self-intersecting and has no holes) and one of its triangulations.

Figure 5: A simple polygon and one possible triangulation of its area.

There are a lot of different algorithms to compute a triangulation given a simple polygon, with a significant spread in computational cost but also in implementation complexity.

One approach is ear-clipping, described in *Polygons have ears* [Meisters 1975]. An ear is a three consecutive vertices $(v_1, v_2, v_3)$ of a polygon where $v_1$ and $v_3$ are visible to each other, i.e. the chord joining these vertices lies entirely within the polygon. The algorithm continues to identify ears and removes these from the polygon until only one triangle remains of the original polygon (Meisters shows that there always are at least two non-overlapping ears in each polygon with more than three vertices). The set of removed ears together with this final triangle makes up the triangulation. While this algorithm is easy to understand it unfortunately has a runtime complexity of $\mathcal{O}(n^2)$ in the number of polygonal vertices.

For certain sets of simple polygons, like convex polygons, straight forward linear algorithms exist. Bernard Chazell showed in 1991 that in fact all simple polygons can be triangulated in linear time [Chazelle 1991]. While this algorithm is optimal in runtime, it is unfortunately also so complex that it is considered infeasible to implement [Bhattacharya et al., 2001], [Held lecture 2010].

For polygons with holes it has been shown that there is a lower bound of $\Omega(n \log(n)$ [Computational Geometry, 2nd ed, p. 59].

## 2.2 Dual Graph of a Triangulation

Each triangulation implicitly defines a corresponding graph, its *dual graph*. Introducing this concept allows us to easily re-use definitions and results from graph theory.

A graph $\mathcal{G}$ is said to be the *dual graph* of a triangulation $\mathcal{T}$ if each vertex of $\mathcal{G}$ corresponds to exactly one triangle of $\mathcal{T}$, and two vertices are connected if and only if their corresponding triangles share an edge.

Figure 6: A triangulation with its dual graph.

Figure 6 shows the same triangulation as previously together with its dual graph.

## 2.3 Hamiltonian Triangulation

Graph theory defines a *Hamiltonian path* as a path that contains each vertex of a graph exactly once [Clark, Holton, 1991, p. 99]. Using that, we can define a *Hamiltonian triangulation* as a triangulation whose dual graph contains a Hamiltonian path [Arkin et al., 1994].

Figure 7 shows two different triangulations of our example polygon. While the triangulation on the left does not contain a Hamiltonian path, the one on the right does.

Figure 7: Two triangulations of the same polygon; one is Hamiltonian, the other is not.

One could ask if each polygon allows for a Hamiltonian triangulation. Fortunately this question is easily answered, albeit in the negative. Figure 8 shows a polygon with a unique triangulation which is however not Hamiltonian.

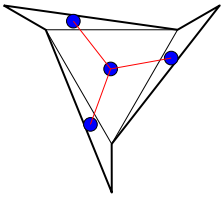

Figure 8: A polygon that is not allowing a Hamiltonian triangulation.

The fact that there are apparently polygons that do admit Hamiltonian triangulations and polygons that do not creates the decision problem of whether a given polygon contains such a triangulation.

In 1994 Arkin et alii presented an algorithm that determines if a polygon has a Hamiltonian triangulation [Arkin et al., 1994]. This algorithm runs in linear time in the size of the polygon's visibility graph, so worst case in $\mathcal{O}(n^2)$ in the number of polygon vertices.

The authors use a dynamic programming approach, defining a property $D[i, j]$ as *the subpolygon left of the chord* $(i, j)$ *contains a Hamiltonian triangulation ending with* $(i, j)$. Initially all $D[i, j]$ values are computed for pairs $i$ and $j$ where the difference between $i$ and $j$ equals 2, i.e. they identify ears. Then they compute more values with ever increasing difference of $i$ and $j$. The polygon contains a Hamiltonian triangulation if in the end there exists a pair $i$ and $j$ such that both $D[i, j]$ and $D[j, i]$.

## 2.4 Two Guard Problem

Akin, Held, Mitchell and Skiena's approach suggests that the question of whether a polygon allows for a Hamiltonian triangulation is related somehow to visibility.

A polygon $P$ is said to be *walkable*, if two guards can walk along different paths from a point $s$ on the polygon to a point $t$ on the polygon, without ever losing sight of each other.

Consider the polygon in figure 9. Two guards, Alice and Bob, starting in $s$ can reach $t$ without ever losing sight of one another by following for instance these steps:

i) Alice walks to the point labeled $a$; ii) Bob walks to 1 and continues on to 2; iii) While Bob slowly walks from 2 to 3, Alice backtracks to $b$; iv) Bob continues to go to 4 and further on to 5; v) Alice walks to $t$; vi) Bob meets Alice at $t$.

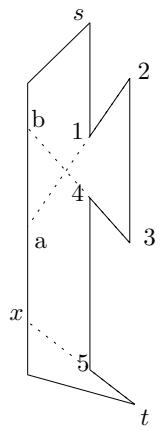


Figure 9: This polygon allows for a walk from $s$ to $t$.

A stricter version of this property is that of *straight walkability*. In a straight walk there is one additional requirement, namely that the guards must not backtrack. The polygon in figure 9 is also straight walkable, however not from $s$ to $t$. The walk has to either start or end in the wedge on the right hand side (between, including, 1 and 4) with the opposite end-point being somewhere in the bottom part between $x$ and 5.

An even more restrictive version is *discrete straight walkability*. A walk is discrete if at any time only one guard is moving while the other rests at a vertex. The polygon from figure 9 is discretely straight walkable as well.

## 2.5 Creating Hamiltonian Triangulations

Arkin et alii note that it is easy to see that every discretely straight walkable polygon has a Hamiltonian triangulation. Starting with all the edges of the polygon in a set $\mathcal{T}$, both guards begin their discrete straight walk along the polygonal edge. Whenever a guard reaches a vertex the line between the

two guards gets added to $\mathcal{T}$. Once both two guards meet at the opposite side, $\mathcal{T}$ will be the triangulation of the polygon.

In 2001, Bhattacharya, Mukhopadhyay and Narasimhan presented an algorithm that determines if a polygon is walkable, straight walkable and discretely straight walkable, each in optimal, linear time. Furthermore their method not only answers the decision problem, it even determines all pairs of points of a polygon that allow for a particular walkability property [Bhattacharya et al., 2001].

An earlier work by Narasimhan described a linear time algorithm for constructing a Hamiltonian triangulation when given a vertex pair allowing a discrete straight walk [Narasimhan 1995]. Combining these two results allows for creating a Hamiltonian triangulation of a polygon in linear time if such a triangulation exists.

Unfortunately these algorithms work only for simple polygons. Arkin and her colleagues showed in [Arkin et al., 1994] that for polygons with holes the decision problem is $\mathcal{NP}$-complete. They did this by reducing from the known to be $\mathcal{NP}$-complete problem of determining whether a planar, cubic graph is Hamiltonian.

## 2.6 Sequential Hamiltonian Triangulations

A subset of Hamiltonian triangulations is also *sequential*. Ostensively, a sequence of triangles is sequential if it takes alternating left and right turns. Arkin et alii define a triangulation as sequential if the dual graph contains a Hamiltonian path such that no three triangulation edges consecutively crossed by the path share a vertex. Figure 10 shows both a sequential and a non-sequential triangulation. Note how in the non-sequential image the path crosses more than two, five in fact, lines that share the vertex $v$.

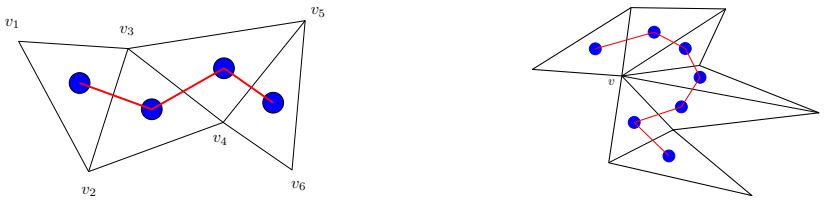


Figure 10: Sequential and non-sequential triangulations.

The advantage of sequential lists of triangles is that their representation as

a strip will not require the use of any swap commands or zero-area triangles.

Akin, Held, Mitchell and Skiena note that it can be determined in linear time whether a given triangulation is a sequential Hamiltonian one [Arkin et al., 1994]. The main observation is that in any such triangulation a triangle can be present in only one of six ways (three possible "entry" edges, each case with two "exit" edges), and that from that the entire strip follows. The approach is then to simply pick an arbitrary triangle and try to extend strips in both directions for all six cases. Then check if for any of these six strips the strip covers all triangles before it either ends or self-intersects. With appropriate data structures this can be determined using only linear time and space, both in the size of the triangulation (and thus also linear in the number of polygon points).

Robin Flatland described an algorithm for actually creating a sequential triangulation of a simple polygon in [Flatland 2004]. The algorithm determines all points that allow sequential walks (i.e. discrete straight walks with the additional constraint that a guard can move only to the immediate next vertex before the other guard has to make their move). It is trivial to create a sequential triangulation from a point that permits a sequential walk.

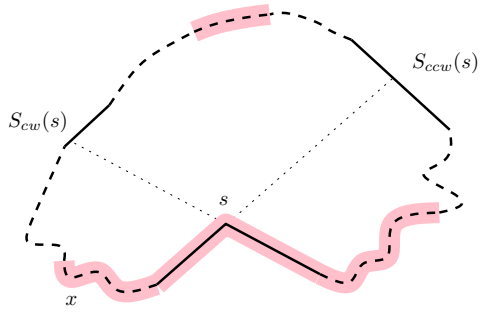


Figure 11: The reflex vertex $s$ rules out sequential walks starting in the highlighted regions.

She starts with labeling all vertices as potentially allowing a (right and left) sequential walk, and then goes on to rule out specific vertices. She notes that a reflex vertex ($s$ in the illustration in figure 11) can never be the starting point for a sequential walk as the two guards would lose sight of each other after both had made a step each. Even more, a reflex vertex rules out half the nodes between itself and the point where rays shot from $s$ hit the polygon on the other side – if both guards started in $x$ for instance (with

$x$ being closer to $s$ than to $S_{cw}(s)$) then eventually the right guard would become hidden behind $s$ before the left one passes $S_{cw}(s)$. She also argues that the opposite side of a reflex vertex $s$ cannot be the starting point of a sequential walk either, since the guards would not see each other at the end of their journey when they get near to s.

Her algorithm runs in overall $\mathcal{O}(n \log(n))$ time. This complexity is dominated by the need to do one ray shooting (in logarithmic time) for each reflex vertex. The number of such vertices can be linear in the number of total vertices.

# 3 Tristrip Decomposition

Not every triangle mesh can be encoded as a single triangle strip. This is quite easy to see since every polygon has a triangulation but not every polygon has a Hamiltonian triangulation as discussed in the previous section.

But even when a given set of triangles cannot be represented by one strip there is something to be gained by representing it by a small number of strips. If a mesh of $n$ triangles can be partitioned into $k$ strips then instead of $3 \cdot n$ vertices only $n+2 \cdot k$ vertices need to be stored, transmitted, transformed and/or otherwise processed by graphics hardware.

Unfortunately finding the smallest possible such $k$ has been shown to be $\mathcal{NP}$-hard [Estkowski et al., 2002] which is why much existing research has been about finding reasonably good approximations fast.

In this section we will briefly present a small selection of these heuristics.

## 3.1 SGI Stripping Algorithm

One of the earliest algorithms for this problem is the SGI Stripping Algorithm, described in [RTR, p. 460] and in some more detail in [Evans et al., 1996].

SGI's approach is a greedy algorithm, i.e. one that always picks what appears to be the best choice locally without respect to any global concerns.

The algorithm works as follows:

1. Pick a starting triangle.
2. Build three different strips, one for each edge of the triangle.
3. Extend these triangle strips in the opposite direction.
4. Choose the longest of these three, discard the others.
5. Repeat until all triangles are covered.

In step 1 the algorithm has to pick a triangle from the set of all triangles to start a new strip. The original algorithm preferably picks triangles with low degrees (few neighbors), but it has been argued that picking random triangles does not hurt significantly (Chow as cited in [RTR]).

Once a starting triangle has been selected, the algorithm tries to build strips starting on each of the three sides of the triangle (step 2). When extending a strip and there is a choice between two neighbors to extend to, the algorithm again prefers triangles with a lower degree. In the case of a tie the algorithm

looks ahead one step and if there is still a tie makes an arbitrary decision. The preference of lower-degree triangles minimizes the chances of isolated triangles being left over at the end that cannot be joined to any strips.

All three strips are then extended in step 3 to their opposite direction as well, making sure that each strip is as long as possible before selecting the longest of three in step 4 and discarding the others.

When no more triangles are left that aren't already part of a strip the algorithm terminates.

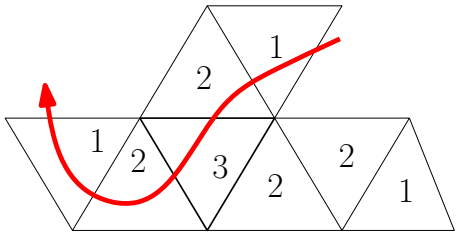


Figure 12: Tie breaking in the SGI stripping algorithm.

Figure 12 illustrates the selection and tie-breaking step. A strip that already consists of the top two triangles and gets extended to the triangle with degree three now has two potential paths to follow, both leading to a triangle of degree two. The next successor after the immediate neighbors on the left has a degree of 1 whereas on the right it has a degree of two, so the algorithm decides to continue the strip to the left side.

SGI's algorithm has no preference whatsoever for sequential strips over non-sequential strips. SGI's Iris GL support swap operations natively, so this seems understandable but it is a concern when using the method on other systems.

When using appropriate data structures this algorithm can be implemented to use linear time. The use of hash tables for storing adjacency information and the use of a priority queue to keep a list of good next starting positions has been suggested.

## 3.2 Fast and Effective Stripification of Polygonal Surface Models

The approach chosen by Xian, Held and Mitchell for their `FTSG` is quite different. It takes a global approach instead of being greedy. It operates in three main phases, viz. (1) computing a spanning tree in the triangulation's dual graph, (2) partitioning that tree into strips and (3) joining small scripts together to form longer chains.

If the input to `FTSG` is not already a complete triangulation but also contains non-triangle faces, the algorithm first triangulates those. Once only triangles are present the dual graph is computed and a spanning tree constructed using a hybrid between depth-first and breadth-first search. The hybrid approach chosen starts like depth-first, trying to build a long path. When that terminates, instead of backtracking along the path to find a position at which to fork off a new chain, the algorithm starts as close to the root as possible for the next path. This method's goal is to have fewer but long chains in the spanning tree, reducing the number of forks and thus eventually the number of different triangle strips needed.

Once the spanning tree has been constructed, the author's bottom-up *Path Peeling Algorithm* partitions the tree into distinct paths in phase 2 while guaranteeing that each (with the possible exception of one) path consists of at least three triangles. This was done since graphics hardware would only be able to profit from paths if they had at least things lengths; in shorter chains the set-up cost dominated any potential savings.

In the final phase each strip is greedily decomposed into either sequential strips or triangle fans.

The authors note that their algorithm runs in linear time, except for the triangulation step that may be required if non-triangle faces are present. They however also state that the triangulation step also usually runs in linear time in practice.

## 3.3 Efficient Generation of Triangle Strips from Triangulated Meshes

A newer greedy algorithm was published in 2004 by Kaick et alii [Kaick et al., 2004]. This algorithm appears to be quite similar to SGI's Stripping algorithm de-

scribed in 3.1 above. The authors use a different strategy for picking the next triangle when building strips, preferring neighbors with a higher adjacency degree where SGI's method preferred lower degrees. Their algorithm provides for two different tie breaking methods, one that sometimes allows for immediate inserts, cutting short further consideration, and one that tries to minimize required swap operations.

The authors also describes the data structures used to some detail and they appear to be remarkably simple.

According to the paper, their algorithm usually is a bit fast than `FTSG`, while creating comparable results.

## 3.4 Iterative Stripification Algorithm

This approach by Porcu and Scateni is a representative of iterative stripping algorithms [Porcu, Scateni 2003].

The algorithm works, as others do, on the dual graph of the triangle mesh. Remember that a graph edge in the dual graph represents two triangles from the mesh sharing an edge, i.e. being adjacent to one another. The algorithm defines a coloring of graph edges as follows: An edge representing two triangles that are consecutive in a strip is drawn *solid* while an edge that represents triangles that are not neighbors in an existing strip are painted *dashed*.

It also defines a so-called *tunneling* operator. This operator works on paths, tunnels, of the dual graph that meet certain requirements. The path has to consist of alternatingly dashed and solid edges, with the first and last edge always being dashed. The end points also have to terminate in a vertex that is itself the terminal of an existing triangle strip, i.e. at most one of the edges in that vertex is solid – if no solid edge is present at that vertex then that vertex is considered the terminal of a one-triangle strip.

When the tunneling operator gets applied to a tunnel it flips the coloring of each edge of the tunnel. From the requirements listed above it follows that each tunnel is of odd length with one more dashed edge than solid edges. After the operation this property is reversed and there is now one more solid edge in the graph, thus improving the stripification. Care must be taken not to create any loops with this operation and significant effort is expended on this.

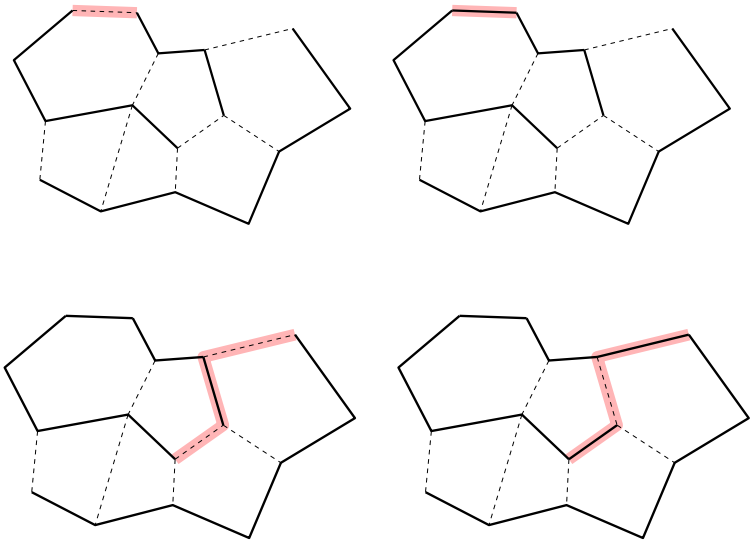

Figure 13: The tunneling operator improves a stripification.

Figure 13 shows the workings of the tunnel operator. Highlighted in red is the tunnel that is being operated upon. In the top line a tunnel of length one connects two short strips, reducing the total number of strips in that graph to two. Applied on the same graph once more in the bottom line it connects the two remaining strips into a single, long triangle strip covering the entire mesh.

# 4 Conclusion

Section 1 introduced the concepts of triangle fans, triangle strips and sequential triangle strips.

In section 2 we briefly covered triangulations and triangulation's dual graph, related the connection between walkability and Hamiltonian triangulations, and presented existing work on sequential triangulations.

We finished showing different approaches on how to decompose triangle meshes into a small number of strips in section 3.

Unfortunately there has been no opportunity to explore algorithms that are able to exploit larger vertex caches.

# References

[Meisters 1975] G.H. Meisters: "Polygons have ears", Amer. Math. Monthly, vol. 82, pp. 648-651, 1975.

[RTR] Tomas Akenine-Möller and Eric Haines: "Real-Time Rendering 2nd Edition.", A. K. Peters, Ltd., 2002.

[Arkin et al., 1994] E. Arkin, M. Held, JI Mitchell, and S. Skiena: "Hamiltonian triangulations for fast rendering", Second Annual European Symposium on Algorithms, volume 855, pages 36-47, Springer-Verlag Lecture Notes in Computer Science, 1994.

[Narasimhan 1995] Giri Narasimhan: "On Hamiltonian Triangulations in Simple Polygons (Extended Abstract)", in Proceedings of the Fifth MSI-Stony Brook Workshop on Computational Geometry, 1995.

[Flatland 2004] Robin Flatland: "On Sequential Triangulations of Simple Polygons", in Proceedings of the 16th Canadian Conference on Computational Geometry, 2004, pp 112–115.

[Bhattacharya et al., 2001] B. Bhattacharya, A. Mukhopadhyay, and G. Narasimhan: "Optimal Algorithms for Two-Guard Walkability of Simple Polygons", LNCS 2125, pages 438–449. 2001.

[Estkowski et al., 2002] Regina Estkowski, Joseph S. B. Mitchell and Xinyu Xiang: "Optimal decomposition of polygonal models into triangle strips", Symposium on Computational Geometry, 2002.

[Evans et al., 1996] Francine Evans, Steven Skiena, Amitabh Varshney: "Optimizing triangle strips for fast rendering", Proceedings IEEE Visualization 96, pp. 319–326. Computer Society Press, 1996.

[Xian et al., 1999] X. Xiang, M. Held, and J. S. B. Mitchell: "Fast and effective stripification of polygonal surface models", Proceedings Symposium on Interactive 3D Graphics, pp. 71–78. ACM Press, 1999.

[Gopi, Eppstein, 2004] M. Gopi, and David Eppstein: "Single-Strip Triangulation of Manifolds with Arbitrary Topology", EuroGraphics 2004.

[Kaick et al., 2004] Oliver Matias van Kaick, Murilo Vicente Gonçalves da Silva, Oliver Matias, Kaick Murilo, Vicente Gonçalves, Hélio Pedrini: "Efficient Generation of Triangle Strips from Triangulated Meshes", 2004

[Porcu, Scateni 2003] Massimiliano B. Porcu and Riccardo Scateni: "An Iterative Stripification Algorithm Based on Dual Graph Operations", In Proceedings of Eurographics 2003 (short presentations).

[Garey et al., 1976] M. R. Garey, David S. Johnson, Robert Endre Tarjan: "The Planar Hamiltonian Circuit Problem is NP-Complete", SIAM J. Comput., volume 5, number 4, 1976, pages 704–714.

[Chazelle 1991] B. Chazelle: "Triangulating a Simple Polygon in Linear Time", Discrete Comput. Geom. 6 (1991), 485–524.

[Das et al., 1993] Gautarn Das and Paul J. Heffernan and Girl Narasimhan: "Finding all weakly-visible chords of a polygon in linear time, Manuscript", 1993

[El-Sana et al. 2000] Jihad El-Sana, Francine Evans, Amitabh Varshney, Steven S. Skiena and Elvir Azanli: "Efficiently Computing and Updating Triangle Strips for Real-Time Rendering", Computer-Aided Design, 32, No. 13, pages 753–772, November 2000.

[Diaz-Gutierrez et al., 2006] Pablo Diaz-Gutierrez, Anusheel Bhushan, M. Gopi, Renato Pajarola: "Single Strips for Fast Interactive Rendering", The Visual Computer, 22(6), 372-386, June 2006.

[WP] Wikipedia, "Polygon triangulation — Wikipedia, The Free Encyclopedia", `https://secure.wikimedia.org/wikipedia/en/w/index.php?title=Polygon_triangulation&oldid=409799239`

[Held lecture 2010] Martin Held: "Algorithmische Geometrie: Triangulation", Lecture notes/slides, `http://www.cosy.sbg.ac.at/~held/teaching/compgeo/slides/triang_slides.pdf`, p. 11.

[Computational Geometry, 2nd ed] Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf: "Computational Geometry (2nd revised ed.)", Springer-Verlag, ISBN 3540656200, 2000.

[Clark, Holton, 1991] John Clark, Derek Allan Holton: "A first look at graph theory", World Scientific Publishing Co. Pte. Ltd., Singapore, ISBN 9810204892, 1991.