Tiling Triangular Meshes

Ming-Yee Iu

EPFL I&C

Abstract

1 Introduction

When modelling large graphics scenes, artists are not expected to model minute and repetitive features such as grass or sand with individual pieces of geometry or individual texture images because it is simply too time-consuming. Even if an artist were inclined to do such a thing, all the detail would quickly exceed the amount of memory available on graphics workstations, making the scene difficult to render. Instead, various techniques for using a small number of parameters or a small number of images to generate large textures for large surfaces are usually employed instead.

Tiling is one such technique, and it is commonly used on square meshes, in particular on terrain modelled used elevation maps. For a square grid, an artist need only design a single image where the top edge lines up with the bottom edge of the image, and the left edge lines up with the right edge. This image can then be pasted on top of each square in the grid to create an impression of a single large seamless texture as shown in Figure 1.

Tiling arbitrary triangular meshes is much more difficult though because the number of triangular tiles needed to tile the surface seamlessly can quickly grow to large numbers. The same problem actuallly occurs during the tiling of arbitrary quadrilateral meshes as well even though the specific case of tiling a grid of squares is quite simple. This paper examines approaches for reducing the number of tiles that an artist needs to design in order to tile arbitrary triangle meshes.

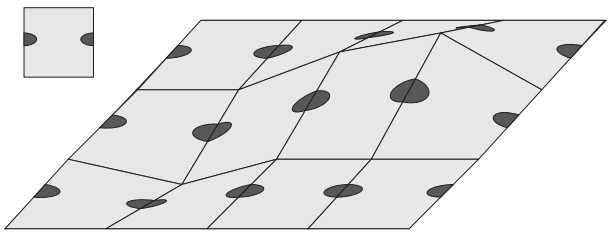

Figure 1: A grid of squares can be textured with a single tile.

2 Previous Work

There is a significant body of work concerning the field of automated texturing of large models. Lapped textures [4] are a scheme for taking arbitrarily shaped texture patches and repeatedly pasting them over the surface of a model until it is completely covered. Other schemes exist for taking a large texture image and searching over the image to find triangle pieces that can be used to cover triangles on the surface mesh in a seamless fashion [2]. And then, of course, there are the entire fields of texture synthesis and procedural textures that allow for the automatic generation of textures from a small number of parameters. *Note to self: maybe you should read some of those papers so that you can cite them here*

The primary graphics paper on tiling triangular meshes though is by Neyret and Cani [3]. It notes that the simplest way of tiling a triangular mesh is to use a single tile where each side of the tile is symmetrical and can be matched seamlessly with any other side of the triangle, such as in Figure 2. Although the tiling may end up being somewhat repetitive, it is possible to mitigate this problem by creating variations of the single tile with sides that are the same as in the original tile, but with different patterns in the center of the tile.

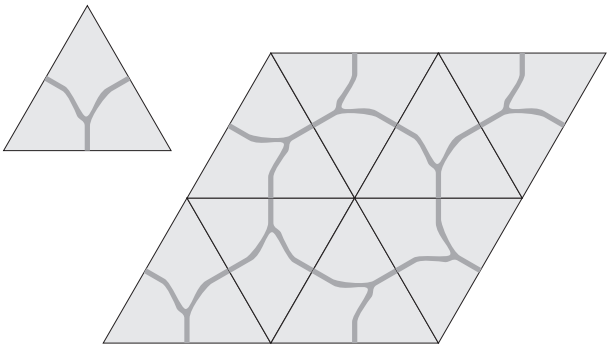

Figure 2: It is possible to tile an arbitrary triangle mesh using a single tile.

The problem with using a single tile is that it severely limits the design of the tile. The tile pattern must be isotropic, the boundaries must be symmetric, and it becomes difficult to use advanced techniques such as ani-

mated tiles. The ideal tileset used to tile a mesh should be one where each triangular tile can have three completely different boundaries on each of its sides and where each edge boundary may be *oriented*. Figure 3 shows an oriented edge boundary in which a particular tile edge must be matched with a corresponding complement edge in order to form a seamless pattern. Unfortunately, as the number of possible edge boundaries increase, the number of possible tiles that might appear in the mesh increases dramatically as well. For example, given n different edge boundaries (where an oriented edge boundary and its complement count as two), there are n ways of creating a tile where all the edge boundaries are the same, $n(n-1)$ ways of creating tiles consisting of two different edge boundaries, and $n(n-1)(n-2)/3$ ways of creating tiles consisting of three different edge boundaries. This means that there are a total of $n+n(n-1)+n(n-1)(n-2)/3$ possible texture tiles that might appear in a mesh.

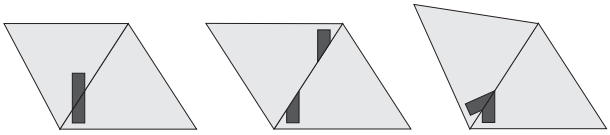

Figure 3: When oriented edges are used, one tile edge must be aligned with a corresponding edge to form a seamless pattern. Such an edge cannot seamlessly align with itself or reflections of itself.

So if a triangle tileset with only one oriented edge boundary were to be created (composed of an edge boundary a and its complement a′), then four different triangles are possible: (a, a, a), (a, a, a′), (a, a′, a′), and (a′, a′, a′). Each tuple describes what combinations of edge boundaries must appear on each side of a triangle tile. In a tileset where each tile side can be different and all edge boundaries are oriented, n is 6, meaning an artist might need to create 76 different tiles to successfully tile a triangle mesh.

Although Neyret and Cani describe these issues with tileset size, they then restrict themselves to considering tilings involving only a small number of edge boundaries. They also discuss ways of tiling an object with triangular tiles independent of the object's real geometry, and describe algorithms for automatically generating triangular tile sets.

3 Minimizing the Number of Tiles

Although the expression given by Neyret and Cani describes all possible tiles that can be formed by generating permutations of edge boundary types, the actual number of tiles needed in a tiling can be much less if the tiling is done carefully. For example, we previously stated that a triangle tileset with only one oriented edge boundary results in four different tiles. In reality, it is possible to tile an arbitrary triangle mesh using only two of those tiles—(a, a, a′) and (a, a′, a′)—not four as previously suggested. *Note to self: is it worthwhile including the proof of this?*

So is it possible to tile an arbitrary triangle mesh without putting an undue burden on tile artists? And is it possible to calculate in advance the minimum number of tiles that an artist needs to create?

To allow for the maximum flexibility in tile design, we want to allow for triangle tiles where each side can have different oriented edge boundaries. In fact, we want the stronger condition that all sides of each triangle tile *must* have different oriented edge boundaries; otherwise, it becomes trivially easy to minimize the number of tiles needed by restricting our tiling to only use a single oriented edge boundary and its complement, thereby allowing for a tiling using only two different tiles.

In order to gain insight into this problem, we can recast it as edge coloring [6]. First, consider the triangle mesh to be tiled as a graph. We can then take the dual of this graph where each triangle face becomes a vertex and edges are drawn between vertices corresponding to adjacent faces. Since each vertex in the dual comes from a triangle face in the original mesh, the maximum degree of vertices in the dual is three. An edge coloring of this dual is an assignment of colors to the edges of the dual such that for each vertex in the dual, all edges adjacent to it have different colors. Figure 4 shows an example of such an edge coloring. According to Vizing's theorem, a graph with maximum vertex degree three can be edge colored using four different colors. It might also be possible to edge color the graph using only three colors, but finding such a coloring is generally NP-complete, which, given the large size of the graph, means that finding such a coloring is essentially infeasible.

Unfortunately, the existence of a four edge coloring doesn't really help all that much. The most obvious way of mapping the edge coloring back to a particular tiling is to have each edge color correspond to a different oriented edge boundary and its complement. This means that each triangle tile will have a different edge boundary on each of its side, but since the edge coloring doesn't give any particular insight into the orientations of these edges, we have to assume the orientations are arbitrary. So given an edge coloring of the dual using four colors—A, B, C, and D—we can go back to the original mesh, and assign one of eight oriented edge boundaries—a, a′, b, b′, c, c′, d, and d′—to each tile side in a way that corresponds to the four edge coloring such as in Figure 5. This means that we might still require $(8\cdot6\cdot4)/3$ or 64 different tile types.

Interesting things happens though, if the dual is bipar-

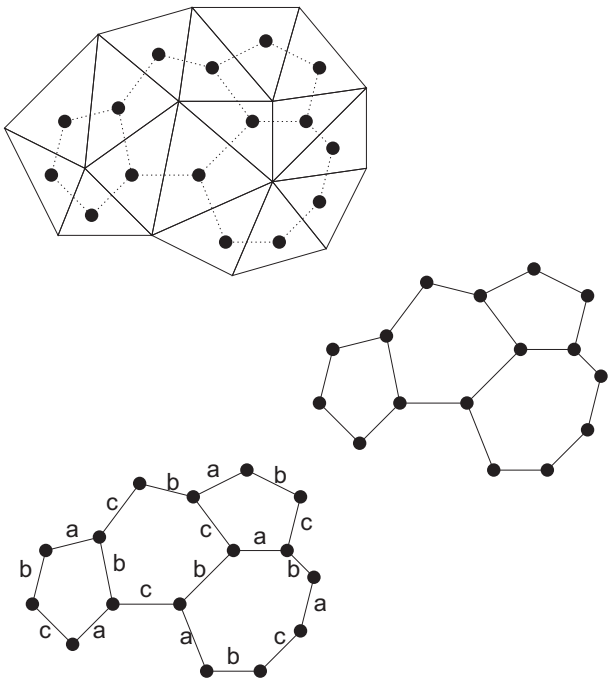

Figure 4: An edge coloring of the dual of a triangular mesh. Note to self: this would be cooler if you could find a mesh requiring a four edge coloring.

tite. Firstly, it becomes possible to edge color the dual using only three colors, and this coloring can be found in linear time [5, 1]. Secondly, because the triangles in the original mesh are bipartite, it becomes possible to have the edge boundaries of a given triangle all be oriented in only one direction or the other. If all orientations flip between adjacent triangles, then a consistent scheme of orienting the edge boundaries is possible. Figure 6 shows how the orientations of edge boundaries can be restricted using a bipartite dual. This means that a triangle mesh with a bipartite dual can be tiled in polynomial time using only four tiles: (a, b, c), (c, b, a), (a′, b′, c′), and (c′, b′, a′).

Requiring an artist to design four separate tile images in order to tile a triangular mesh is quite reasonable. In fact, in practice, only three tile image need to be designed because one pair of tiles can be created by starting with a square tile and cutting it in half. As noted before, oriented edge boundaries are allowed, and each side of a triangular tile is allowed to be different, meaning there is a lot of freedom in the design of the tileset. If tiling using only four tiles results in a repetitive pattern, it is easy to start with a three edge coloring of the dual and to add additional colors selectively. This way, the number of possible edge boundary types can be increased while controlling the number of permutations of these edge boundaries in tiles, thereby preventing the number of triangle tile permutations from exploding. Unfortunately, all of these great properties require that the original triangle mesh have a bipartite dual. But what does it mean for the dual of the triangular mesh to be bipartite? Does that inhibit the geometry of models in some way? How is an artist or application supposed to know how to create such a mesh?

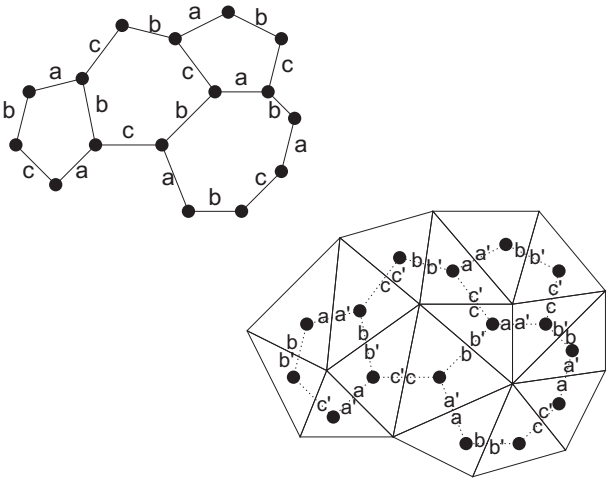

Figure 5: A mapping of an edge coloring to different tile types with arbitrary edge boundary orientations.

4 Triangle Meshes with Bipartite Duals

In fact, a triangle mesh with a bipartite dual is actually equivalent to a triangular mesh where all interior verices have an even degree. Here is the proof:

$\Longrightarrow$

Given a triangle mesh with a bipartite dual, assume there exists at least one interior vertex with an odd degree. There are an odd number of faces adjacent to an odd degree vertex, and if we traverse these faces, we end up with an odd cycle in the dual. But all cycles in a bipartite graph are even, resulting in a contradiction, so our assumption must be false. Therefore, all interior vertices in the triangular mesh must be even.

$\Longleftarrow$

This is another proof by contradiction. Given a triangle mesh where all interior vertices have an even degree, assume that the dual is not bipartite. Hence, there must be a cycle in the dual that is odd. Take this cycle from the dual and map it back to the original mesh. We now have a cycle traversing over triangles as demonstrated in Figure 7. This new cycle encloses a subset of the triangles in the triangle mesh. Since the original cycle in the dual is odd, the new cycle must cross over an odd number of edges. Each one of these edges is incident to exactly one vertex

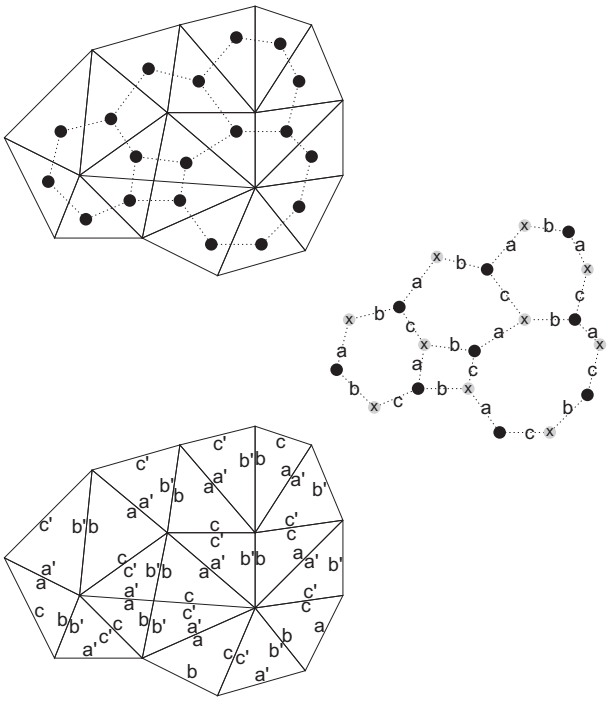

Figure 6: If a mesh has a bipartite dual, it can be tiled using only four different tiles.

from the enclosed area. Let us consider this enclosed area now. If we wanted to count the sum of the degrees of all the vertices from the enclosed area, we would first sum up the original degrees of all these vertices and then subtract the length of the dual cycle (because these correspond to edges that do not contribute to the enclosed area). Since the original degrees of each of the enclosed vertices are even, the sum of the original degrees of all these vertices is also even. But because the cycle length is odd, the total sum of the degrees of all the vertices from the enclosed area is odd. Every edge in the enclosed area though must be incident to two vertices from the enclosed area, meaning that the total sum of the degrees of all vertices from the enclosed area must be even. This is a contradiction, meaning the the original assumption must be false, hence the dual must be bipartite.

5 Transforming Meshes

When all the vertices in a triangular mesh have an even degree, it is extremely easy to tile them with a small number of triangular tiles. And although the concept of such a mesh is easily understood, it is not obvious how an artist can be expected to create 3d models satisfying such constraints. Here, we examine two techniques for automatically transforming triangular meshes to make each vertex have an even degree.

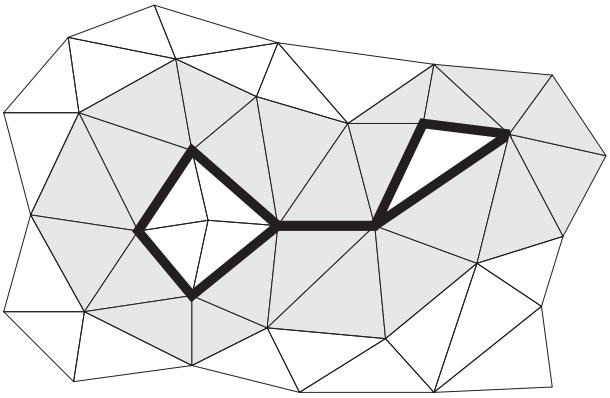

Figure 7: When an odd cycle is mapped back to the original triangle mesh (the triangles in the cycle are marked in grey), it encloses a subset of the mesh (marked with a thick black line) that can be analyzed.

For open surfaces, it is possible to simply add edges connecting odd interior vertices to boundary vertices and boundary edges (since we do not care about the degree of vertices on the boundary). It is also possible to takes pairs of odd vertices and join them with edges, thereby making them both even. Figure 8 illustrates these two operations. Unfortunately, not all pairs of odd vertices can be joined in this way; for example, adjacent odd vertices cannot be joined. And it is also not clear how to choose which vertices to join so as to minimize the modifications made to the mesh. This is important because applying these operations results in the splitting of triangles. Not only do split triangles result in increased rendering time because of the greater number of triangles, but split triangles may also be degenerate or of non-uniform size leading to inconsistent tiling or visual artifacts.

In any case, for closed surfaces where there is no boundary, the above operations may not be sufficient to eliminate all the odd vertices from the mesh, meaning another technique is needed.

Therese Biedl of the University of Waterloo came up with an interesting construction that eliminates all odd vertices in both open and closed surfaces. With this construction, an extra vertex is added to the center of every triangle in the mesh. These extra vertices are then connected to the three vertices of the original triangle and to the midpoints of the three edges of the original triangle. Afterwards, each vertex in the resulting mesh will have an even degree, and, provided that the original triangles were well-formed, the resulting triangles will generally have uniform sizes and shapes as well. Figure 9 demonstrates how the construction works.

Unfortunately, the resulting mesh has six times as

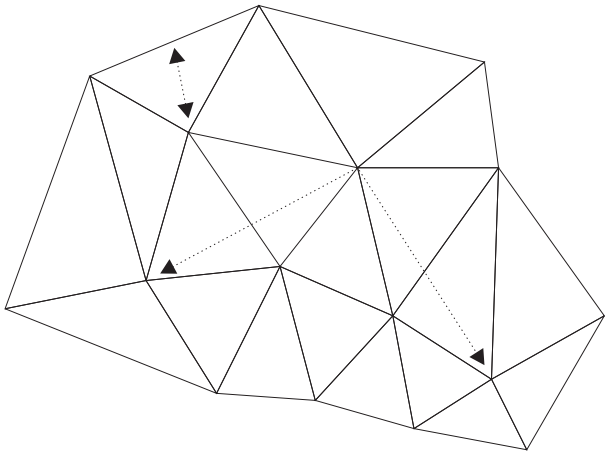

Figure 8: Adding edges between odd vertices and the boundary or between pairs of odd vertices will make those vertices even.

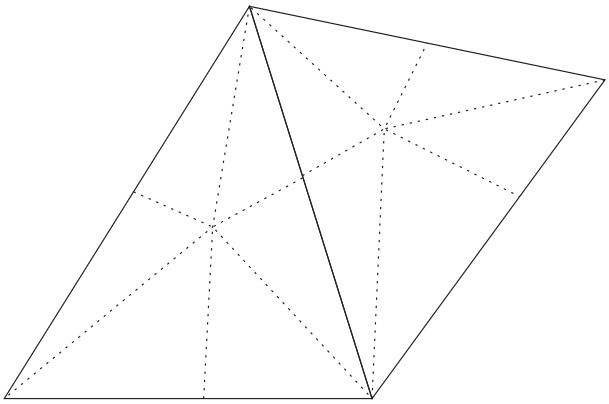

Figure 9: By adding edges and vertices to every triangle, it is possible to force every vertex to have an even degree

many triangles as before, meaning that it takes six times as long to render the scene. There are only 64 ways in which each subdivided triangle can be tiled though. And if each such tiling is stored as a separate texture, then it becomes possible to render the original non-divided mesh using this new set of texture tiles, meaning there will be no slowdown. If there is insufficient memory to store that many texture, another alternative is to tile each subdivided triangle in exactly same way, as shown in Figure 10. This single large texture is essentially a triangle tile with a single non-oriented edge boundary on all sides, and it can then be used to texture the entire original mesh. Then if memory allows, the tiling of the subdivided mesh can be selectively altered so that a limited number of additional subdivided triangle tilings appear in the mesh. In this way, one can control the exact number of generated texture tiles that will be needed to cover the original mesh. So starting with the four original tiles, it's possible to permute them to generate between 1 and 64 larger tiles that can be used to tile the mesh. Of course, on more modern hardware, one can simply combine the original four tiles on-the-fly to generate the larger tiles using a fragment program.

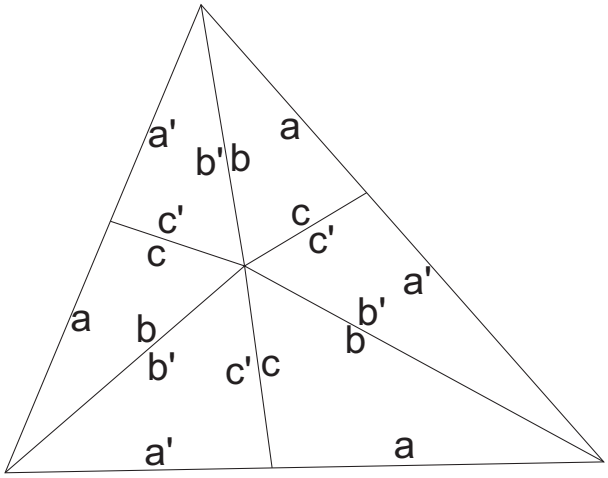

Figure 10: Tiling a subdivided triangle in this way, results in a single large triangular tile with identical non-oriented edge boundaries

6 Conclusions and Future Work

Although the tiling of simple grids of squares is quite easy, the tiling of arbitrary triangle meshes is actually quite hard because if done improperly, an artist might have to design a large number of tiles for the tiling. If the tiling process is done carefully, however, it is possible to tile the an arbitrary triangle mesh using only four tiles. An artist also has significant latitude in the design of these tiles.

Unfortunately, this paper only lays out the theoretical groundwork for these claims. To verify the practicality of these techniques, it will be necessary to build a few implementations. It is also unclear how these tiling techniques can be extended to handle levels of detail and to avoid aliasing artifacts. There are also theoretical issues that could be examined more deeply such as finding exact lower bounds on the number of tiles needed to tile triangle meshes with non-bipartite duals or looking at the tilings of arbitrary quadrilateral meshes, which might yield further good insights into the problem

References

[1] Olivier Bodini and Eric Rémila. Tilings with trichromatic colored-edges triangles. *Theor. Comput. Sci.*,

319(1-3):59–70, 2004.

[2] Sebastian Magda and David Kriegman. Fast texture synthesis on arbitrary meshes. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 82–89. Eurographics Association, 2003.

[3] Fabrice Neyret and Marie-Paule Cani. Pattern-based texturing revisited. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 235–242. ACM Press/Addison-Wesley Publishing Co., 1999.

[4] Emil Praun, Adam Finkelstein, and Hugues Hoppe. Lapped textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 465–470. ACM Press/Addison-Wesley Publishing Co., 2000.

[5] Alexander Schrijver. Bipartite edge coloring in $o(\delta m)$ time. *SIAM J. Comput.*, 28(3):841–846, 1999.

[6] Eric W. Weisstein. Edge chromatic number. *http://mathworld.wolfram.com/ EdgeChromaticNumber.html*.