The Visual Computer manuscript No.
(will be inserted by the editor)

Yu-Kun Lai · Shi-Min Hu · Ralph R. Martin

Surface Mosaics

Abstract This paper considers the problem of placing mosaic tiles on a surface to produce a *surface mosaic*. We assume that the user specifies a mesh model, the size of the tiles and the amount of grout, and optionally, a few control vectors at key locations on the surface indicating the preferred tile orientation at these points. From these inputs, we place equal-sized rectangular tiles over the mesh such as to almost cover it, with controlled orientation. The alignment of the tiles follows a vector field which is interpolated over the surface from the control vectors, and also forced into alignment with any sharp creases, open boundaries, and boundaries between regions of different colors. Our method efficiently solves the problem by posing it as one of globally optimizing a spring-like energy in the Manhattan metric, using overlapping local parameterizations. We demonstrate the effectiveness of our algorithm with various examples.

Keywords surface mosaics, particle optimization, Manhattan metric, overlapping local parameterizations

1 Introduction

Mosaics are an art form with a long history: many examples are known from Graeco-Roman times. The idea is simple: an image is formed using small colored square tiles which almost touch, so as to cover some area. Typically, this region is planar, and the tiles are oriented to emphasise the distinction between important objects in the foreground, and the background of the image.

In other cases, the underlying surface may be curved, e.g. a sculpture. In this case, even if all tiles are of the same color, careful placement of tiles is now necessary because of the curvature of the surface. Clearly, this problem is similar to the well-studied one of covering a surface with an orthogonal parameter net. However, it is also different: the size and shape of the tiles is *constant*, and, in addition, small *gaps* are allowed between them. Furthermore, it is desirable that locally, the tiling should follow 'natural' preferred directions in the surface. These may be determined by aesthetic choices made by an artist, or may e.g. be inferred from the principal directions, from creases, or from color boundaries. Earlier papers have considered *planar* mosaics, but as far as we are aware, no-one has yet considered algorithmic production of *surface* mosaics.

Yu-Kun Lai
Tsinghua University, Beijing, China
E-mail: laiyk03@mails.tsinghua.edu.cn

Shi-Min Hu
Tsinghua University, Beijing, China
E-mail: shimin@tsinghua.edu.cn

Ralph R. Martin
Cardiff University, Cardiff, UK
E-mail: ralph@cs.cf.ac.uk

1.1 Problem Statement

We assume the input given is: a mesh model M to be covered with tiles, possibly with open boundary; the fixed size of the tiles, which may be squares or rectangles: $l_x \times l_y$; a factor giving the fractional area g of the surface to be filled by grout between the tiles (we typically use $g = 0.1$); and a small number of vectors on the surface indicating the preferred orientation of the tiles at key locations on the surface. Alternatively, instead of specifying the tile size and grout factor, the user may specify the desired number N of tiles and the aspect ratio η of the tiles; we may convert between these quantities using $N = (1-g)A/l_x l_y$, $l_x = \eta l_y$ where A is the surface area. We wish to find N mesh sites at which to place N tiles, so that all tiles are disjoint, and cover the input surface, less the amount left for grout. In addition to following user-specified orientation vectors, the method should automatically align the tiles with any *feature lines* in the surface, corresponding to sharp creases, open boundaries, or color boundaries. For models without image texture information available, the color and other properties of the tiles can be specified by the user; typically a single value will be assigned to a connected component separated by features. For models with image texture available, the color of each tile could also be decided by averaging color values of the

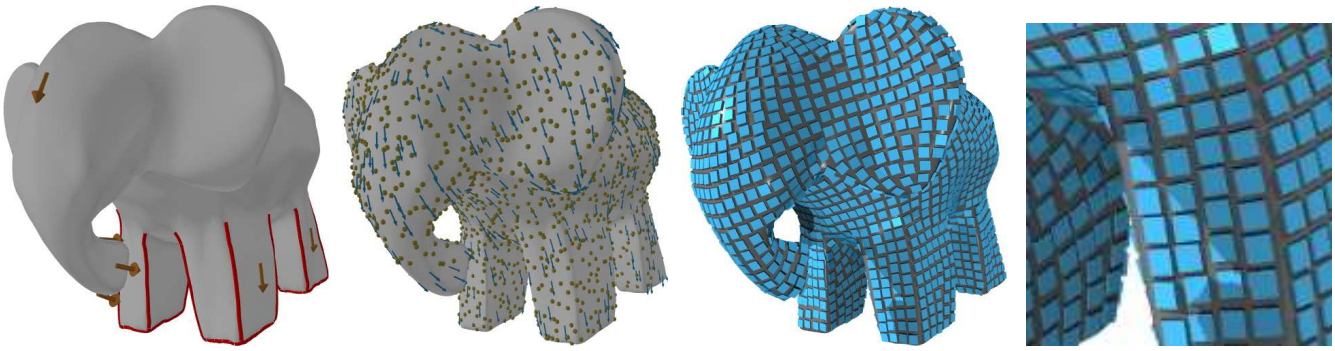

Fig. 1 Steps in surface mosaic generation: (a) control vectors and features, (b) vector field and initialization, (c) final result, (d) close-up.

part of the surface covered by the tile, producing richer results.

The given geometric problem cannot be solved by simply computing a parameterization and putting one fixed-size tile at each location of a regular parametric grid, because, for objects of arbitrary genus, equiareal orthogonal (i.e. isometric) global parameterizations do not in general exist. We solve the problem by posing it as a global optimization problem based on a spring-like energy in the Manhattan metric. Using the Manhattan metric naturally tends to give tile centers on a rectangular grid, as desired—using a Euclidean metric would lead to a hexagonal grid instead, and would be suitable for placing hexagonal tiles. However, we prefer square (or rectangular) tiles because (i) square tiles are more aesthetically pleasing, and (ii) are usually used in traditional mosaics; furthermore (iii) square tiles can be placed naturally with edges aligned along sharp curves or other boundaries, and hence can better draw attention to features and global geometric shapes.

To efficiently compute distances in the Manhattan metric, which needs to be done frequently, we use overlapping local parameterizations and local charts. This simplifies the distance computation and provides significantly improved performance and robustness compared to simply tracing over the mesh.

Related work is discussed in Section 2. Section 3 outlines our algorithm. The optimization method used for tile placement, and handling of singularities and feature lines, are discussed in Sections 4 and 5. Experimental results are presented in Section 6 and conclusions are given in Section 7.

2 Related Work

Various earlier work has considered production of mosaics for computer graphics; in some cases the definition of a mosaic is more general than the one we gave earlier. Kaplan [9] solves a particular kind of mosaicing problem called *Escherization*, in which a perfect tiling of the plane is achieved using repeating shapes with complicated boundaries. Given some input image, of say a head, it is distorted somewhat to produce the nearest such shape which can tile the plane. Other work, however [5,8], uses equal-sized tiles, with appropriate orientation, to emphasise feature edges in some input image, with the aim of producing similar results to real planar mosaics; they can also generalise the shape of the tiles.

Photomosaicing [6,15] is another related technique. It forms a target image using many small image tiles taken from a library. However, the tiles are always placed in a regular way, with fixed orientation. Kim [10] extends this idea to placement of image tiles to fill a region of arbitrary shape; for good results, however, small deformations of the tiles are necessary. Klein [11] extends photomosaics to videos. An input video is decomposed into a collection of small video tiles taken from a library; efficient retrieval of optimized video samples from the library is of most importance in this problem.

Work with more similar aims to ours concerns feature-aligned quad-dominated remeshing. Alliez [1] shows how to remesh a mesh model using quads locally aligned with the principal directions on the surface. Marinov [12] improves the approach by working directly on the 3D surface, so that models of arbitrary topology can be handled. The periodic global parameterization proposed in [14] can be used to generate a quad-dominated remeshing aligned with a specified vector field (e.g. the principal directions). By using a global parameterization, this approach can produce a more even placement of quads than greedy tracing approaches. Other parameterization methods [7], and methods for texture synthesis on surfaces [18,19] are also relevant.

Clearly, the requirement to tile a mesh model rather than a planar region leads to a different problem than is addressed by work on planar mosaics. Furthermore, none of the surface processing ideas above can be directly used to cover a surface with rectangles of a fixed size, with global optimisation of distribution of grout.

3 Algorithm Overview

The key problem in mosaicing is positioning of the tiles. For decorative mosaics, Hausner [8] uses a *centroidal Voronoi diagram* and the *Manhattan metric* to solve this problem. However, this approach cannot easily be extended to mesh

models of arbitrary topology, because general mesh models are not homeomorphic to a disk and so a 2-dimensional Voronoi diagram is generally not applicable. Instead, we base our approach on a particle optimization method proposed in [20], adapted to use the Manhattan metric. We use the particles to represent the tile centers. This considers non-oriented particles attached to underlying implicit surfaces, which leads to simple constraints. Oriented particles have also previously been used in [17], as a tool for surface modeling. This method can be adapted to mesh models, by projected gradient descent optimization of the energy functional; we must take into account the different metric. The optimization method spreads the particles evenly across the surface with respect to the Manhattan metric by means of repulsive forces between them.

The main steps of our algorithm are (see Fig. 1) vector field generation, initial tile placement, and tile placement optimization, as outlined next.

3.1 Vector field generation

A vector field is used to guide the local orientation of tiles. Thus, at each position of the mesh, we determine a local coordinate frame comprising the surface normal plus two orthogonal tangent directions in the surface. One way of doing this (except for planar and spherical regions) would be to use the principal directions plus the surface normal; for methods for estimating these, see [4,21]. Using this approach in practice generally requires a prior smoothing operation [1,12, 14], to reduce the number of umbilics (singularities) where the orthogonal net nature of the vector field breaks down. Local frames based on principal directions have the desirable property that they are naturally orthogonal to feature lines such as sharp edges or lines of symmetry.

However, both to provide user control, and to avoid having too many singularities, we propose as an alternative to allow the user to specify the vector field at various key points of the surface, which we then interpolate to provide a vector field over the whole surface, as in [18]. This is done by initially setting the vector field at all unspecified points to zero, and then iteratively updating the vector field at each point until convergence. The update is determined as a weighted difference between the current vector field at each point, and a weighted sum of the vector fields at its 1-ring neighbors. Thus, the updated value is given by

$$\mathbf{X}_0' = \mathbf{X}_0 + t \sum_{i=1}^{k} W_i(\hat{\mathbf{X}}_i - \mathbf{X}_0), \tag{1}$$

where $\mathbf{X}_0$ is the vector field at some point being considered, and $\hat{\mathbf{X}}_i$ are the values of the vector field in its 1-ring neighborhood, projected onto the tangent plane at the given point to ensure the updated vector field lies in the tangent plane. W_i is a weight, proportional to the reciprocal of the edge length and normalized to have sum one, so that vertices closer to the vertex being considered have larger weights. t is a step-size control; relatively small t is used to provide stable convergence, and it is set to 0.1 in all our experiments. A second vector field orthogonal to the first can be determined locally by taking the cross-product of the first vector field and the surface normal.

3.2 Initial distribution of tiles

We use an iterative global optimization method to find the final positions of tiles. Good initial tile placement is important, both to achieve results of high quality, and for efficiency. Ideally, each tile should occupy an identical area on the surface, so we use error diffusion initialization [2] to evenly distribute the N starting positions over the surface. Each triangle is assigned a real number indicating its allocated initial number of tiles according to its area. The tiles are added in a region-growing process: when considering a triangle, an integer number of tiles closest to the real number is placed inside the triangle, at random locations. The difference between the integer used and the real number desired (either positive or negative) represents an error which is diffused to its neighbors. Thus, each sufficiently large neighborhood contains a number of tiles proportional to its area.

3.3 Tile optimization

After initialization, the positions of the tiles are optimized using a spring-like energy defined in terms of the Manhattan metric. Details are given in Section 4 and special cases are considered in Section 5. After optimization, a tile of the desired size is placed at each position; its orientation is computed by averaging the vector field in its neighborhood. (Using such an average, rather than the vector field at the position itself, tends to give better results overall at locations where the vector field is changing rapidly.)

The next two Sections, together with the problem statement, contain the main new ideas in the paper.

4 Tile Position Optimization

We now discuss the optimization framework, and the key issue of how the necessary Manhattan distances within it are efficiently computed.

4.1 Optimization framework

The basic framework used is similar to that in [20], and is based on energy minimization. The energy leads to a repulsive force between tiles, which in equilibrium gives the final positions of the tiles. The energy between any two tiles T_i and T_j is defined as

$$E_{ij} = \exp(-d(T_i, T_j)^2 / 2\sigma^2), \tag{2}$$

where $d(T_i, T_j)$ measures the *Manhattan distance* between the centers of the tiles (as explained later), and σ is the *interaction radius* that controls the range of the repulsive force. Because Manhattan distances are generally larger than corresponding Euclidean distances, σ also needs to be larger; in practice we set $\sigma = 0.9\sqrt{A/N}$, where A is the surface area and N is the desired number of tiles. Note that the interaction radius σ only controls the the fall off of potential energy, and hence interactive forces, with increasing distance, and does not affect the final inter-tile distances in equilibrium.

The energy for a single tile T_i is the sum of the interaction energies with its neighbours: $E_i = \sum_{j=1}^{n} E_{ij}$. In principle, the negative gradient of this energy produces the repulsive force. However, the energy E is defined in terms of the Manhattan distance, and is only defined on the surface, and not in the embedding space, and thus ∇E itself must be computed by restricting it to the tangent plane. The computation is quite involved and does not have a simple closed-form solution. Thus, following [20], we define a suitable force on the i^{th} tile based on Euclidean distance, by analogy with spring-like forces:

$$\mathbf{F}_i = \sum_{j=1}^{N} (\mathbf{p}_i - \mathbf{p}_j) E_{ij}, \tag{3}$$

where $\mathbf{p}_i$ is the position of tile i; this approach is found to work well in practice.

Note that the sum above should be taken over all tile positions, in principle, but our energy function is designed to have almost local support, so in practice it is sufficient to consider the k-nearest neighbors of tile i. Typically we use $k = 20$ in our implementation. The required nearest neighbor queries can be accelerated using an approximate nearest neighbor algorithm [13].

Optimization is carried out using projected gradient descent. At each step, the new position $\mathbf{p}_i'$ for each tile is found by updating $\mathbf{p}_i$ using $\mathbf{p}_i' = \mathbf{p}_i + t\mathbf{F}_i$, and then projecting back the resulting position onto the mesh; t controls step size and is usually set to 1.

To prevent the optimization process becoming stuck in some local minimum, *teleportation* of tiles may be used. This can be done by waiting until optimisation has proceeded far enough, and then periodically checking for 'large' holes (i.e. of at least one tile size). Suppose a number h exist. We then also determine the h closest pairs of tiles, and fill each hole with one tile from one of the pairs. Going further, if the exact number of tiles does not need to be preserved, direct insertion of extra tiles into holes is also possible, reducing the amount of grout. Using these techniques usually slightly improves the results for most examples, but the differences are quite small.

Fig. 2 Points with equal Manhattan distance

4.2 Computation of Manhattan distance

4.2.1 Defining Manhattan distance

Using the Euclidean distance for $d(i, j)$ tends to produce hexagonally distributed tile centers, corresponding to a densest circle packing in the Euclidean metric; this is unsuitable for placing rectangular tiles which should be placed on something similar, locally, to a rectangular grid. To achieve this, we use the *Manhattan distance* instead. In the plane, the Manhattan distance between points P_1 at (x_1, y_1) and P_2 at (x_2, y_2) is defined as $|x_1 - x_2| + |y_1 - y_2|$, i.e. the distance 'across' plus the distance 'up'. On a mesh, we may also measure distances in the two preferred directions determined by the interpolated vector field. However, there is a small complication—unlike the planar case, summing distances in a curved surface 'across' then 'up' generally gives a different result to going 'up' then 'across'. We simply take the minimum of these two possibilities.

In a grid, using the Manhattan metric, each point (e.g. the red one in Fig. 2) has 8 equidistant nearest neighbors (the blue ones), and simply placing tiles at such points would lead to a grid rotated by 45° from the desired orientation. Thus, we use preferred directions at 45° to the local vector field orientation. If the user wishes to have non-square tiles of aspect ratio η, we further compensate by multiplying distances in the two directions given by the local vector field by $1/\sqrt{\eta}$ and $\sqrt{\eta}$.

4.2.2 Computing Manhattan distance

To compute Manhattan distance, we need to measure distances in the surface in preferred directions relative to the local vector field. Tracing the vector field, as done in some remeshing work, is neither efficient nor robust. Instead, we use overlapping local parameterizations as a basis for measuring distances. An approach similar to that in [16] is used to maintain a set of local parameterizations. When we need to compute the distance between two points, we consider the currently available local parameterizations to see if one exists which covers both points. If so, we simply use it, otherwise, a new local parameterization is built and stored for later reuse. For our specific problem, we use extra constraints when building charts (to help us parameterize them), and a new parameterization method; these are detailed in the next Section.

Finally, we compute the Manhattan distance using the parameterization to measure distances. For efficiency, we approximate the parameterization by a piecewise linear mapping across each triangle. Given two points, and a chart with

parameterization $X(u,v)$ that covers both of these points, the distances along a preferred direction on the surface can be calculated by integrating the norm of the appropriate directional derivative in the parameter domain. For point pairs that appear in more than one chart, we use the smallest distance in any chart, which amounts to choosing the shortest path if more than one exists. We efficiently keep track of which points are in which charts by simply using flags to indicate whether a face belongs to a specific chart; such information is recorded when the chart is constructed.

4.3 Charts and Parameterization

4.3.1 Adding a chart

When a new local chart is constructed for parameterization, we start at one of the two points between which we wish to measure the distance, and grow a region in breadth-first manner, until either the region gets too large (forces become negligible for points far apart from each other—in practice, the size can be set to say 5σ, as the energy between two such tiles is $< 4 \times 10^{-6}$) or until the accumulated change in vector field is above a certain threshold (this is to ensure that the chart can be reliably parameterized in a planar domain while following the specified vector fields). The latter criterion is defined as follows: the vector field at the center of each face can be computed by averaging the vector field at its constituent vertices. During region growth, the change in vector field direction between two adjacent triangles can be computed by isometrically mapping these two triangles (along with corresponding directions) onto a planar domain (by unfolding along the edge between the two triangles), and measuring the absolute angular differences between these two mapped vectors. The accumulated vector field change is found by accumulating such values along a path in the dual graph of the mesh. The growth of a chart is stopped if they exceed a threshold independent of mesh curvatures; $\pi/2$ is used in our experiments to make sure that such a mapping is possible.

4.3.2 Local parameterization

We need a parameterization method which specifically finds a parameterization of the surface $S = S(u,v)$ aligned with the vector field, allowing us to measure Manhattan distances by making a corresponding measurement in the planar parameter domain. It is sufficient to make sure the reconstructed vector field follows the prescribed directions. However, the vectors ∇u and ∇v cannot be expressed linearly in terms of the parameter values at the vertices. To make the computation easier and more efficient, we use a similar approach to the one in [14]. We formulate the problem in terms of minimizing the following functional over the mesh:

$$F = \int_S \left(||\nabla u - \mathbf{X}||^2 + ||\nabla v - \mathbf{Y}||^2 \right) dS, \quad (4)$$

where $\mathbf{X}$ is the vector field, and $\mathbf{Y}$ is an orthogonal vector field on the surface. However, we adapt this approach to finding a *local* parameterization by solving the following discretized functional

$$F^* = \sum_T F_T = \sum_T \left(||\nabla u - \mathbf{X}_T||^2 + ||\nabla v - \mathbf{Y}_T||^2 \right) A_T, \quad (5)$$

where T denotes summing over triangles, and values averaged over triangles, within a given chart.

Each local parameterization has low distortion and can be efficiently computed by solving a sparse linear system, having chosen some arbitrary vertex as the parameter origin.

5 Singularities and feature lines

Minor modifications to the above algorithm are needed to handle *singularities* in the vector field and *feature lines*. The latter comprise any of: open boundaries of the surface, sharp creases in the surface, and boundaries between differently colored regions of the surface; all are handled in the same manner, as they can be characterized by some curve in the surface. Creases can be extracted using automatic or semi-automatic methods [3], or they may be manually marked by the user; the same is true for segmentation to find boundaries of colored regions.

5.1 Handling singularities

Singularities are where the direction of the vector field is not well-defined. Singularities interior to a triangle, or on an edge, can be *detected* by interpolating the vector field from its values at each vertex of the mesh [1].

To *handle* a singularity, we place a *virtual* tile at the location of each singularity; it does not move in later optimization, nor do we place a tile at this location in the final tiling. Furthermore, we compute the force it exerts on nearby tiles using the *Euclidean* metric, rather than the Manhattan metric, which gives a more natural radial distribution of tiles in its neighborhood.

5.2 Handling feature lines

Firstly, we require tiles adjacent to each feature line to be aligned with it, so we additionally constrain the vector field to be parallel or perpendicular to each feature line.

Secondly, tiles adjacent to each feature line should be placed so that their centers are *half* their width from the feature line, allowing for grout. Thus, during optimization, we apply an additional force to all points within a region of influence adjacent to each feature line, of width 3 times the tile size. This force corresponds to *twice* the *Euclidean* distance between the point and the feature line, resulting in tiles being placed half the normal grout width from the feature line.

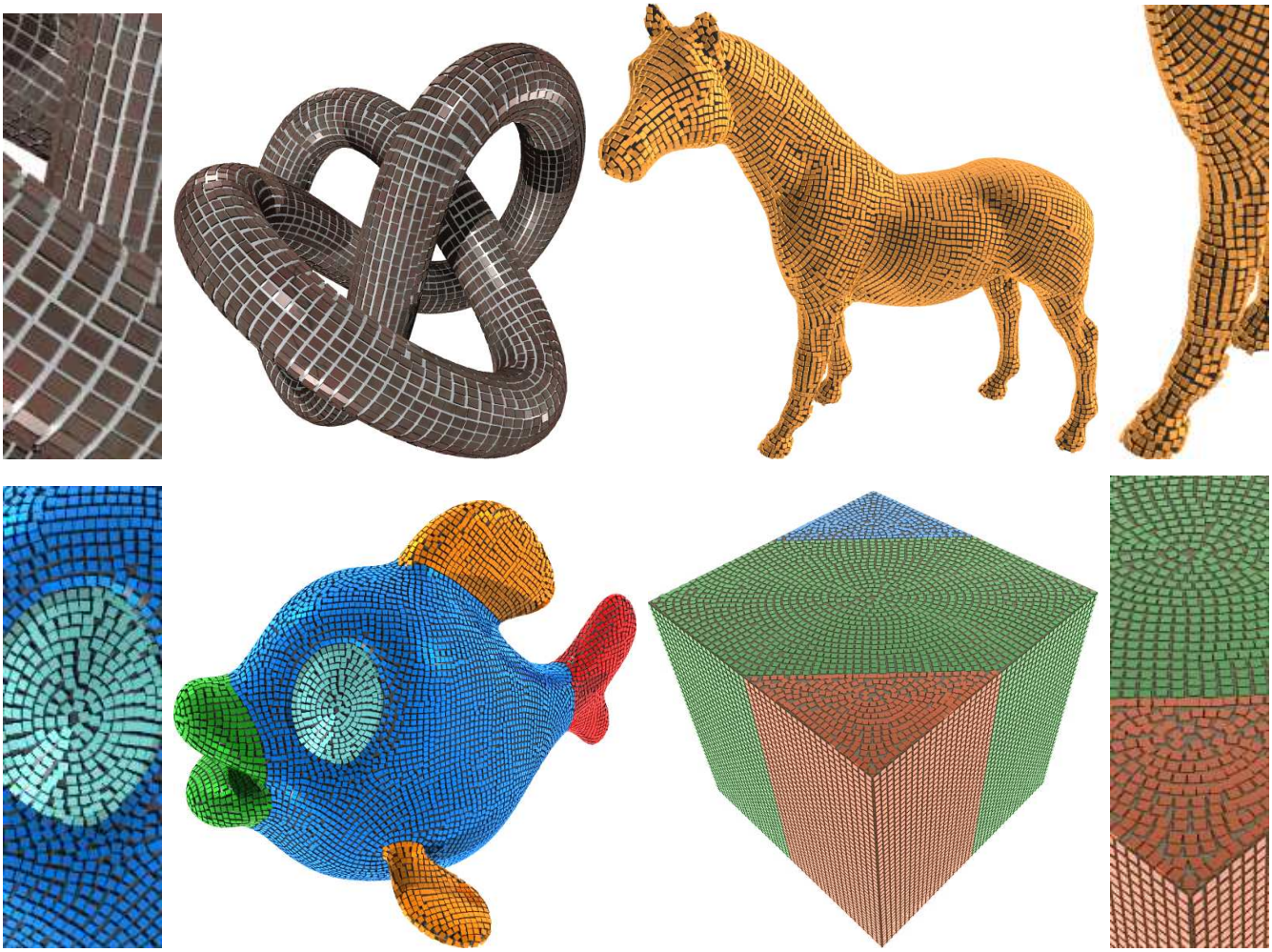

Fig. 3 Top left to bottom right: knot, horse, fish, and cube examples

6 Experimental Results

We now present various results produced using our method. As discussed earlier, rectangular tiles can be handled as well as square tiles. However, as square tiles are used in most real mosaics, we prefer to use them in most of our examples. See Fig. 3, which shows: a knot tiled with rectangular tiles, a horse tiled with square tiles, a colored fish with feature lines along the boundaries of colored regions, and a cube with feature lines along sharp edges and boundaries of colored regions. The user control in these experiments included a few user specified vectors to guide the orientation of tiles for the knot and horse models, and the specification of whether the vector field is orthogonal or parallel to each boundary or feature loop in the fish and cube examples. Such user control is necessary to give the basic desired orientation of tiles over the model.

Generally good results are obtained, as can be seen. The knot example shows that our approach works well with rectangular tiles; one potential use is to tile a surface in a way which clearly distinguishes the different principal directions. On the horse example, the tiling works well, except that it is rather ragged around the ears. However, it is fairly obvious that *no* method can produce an entirely satisfactory result for any region small compared to the tile size, especially if having complex shape or boundaries. For the fish and cube examples, the original models had several regions each of a different color; the cube also has sharp edges. Feature lines were extracted using simple color segmentation, and by detecting geometric features. User control was then used to specify whether the vector field was parallel or orthogonal to each feature edge in turn. No additional control vectors were specified for these examples—in such cases they are generally not needed, although they can be used if desired. The cube example shows how the approach copes with a combination of colored regions and geometric features. Note that it shows how our method produces an ideal result on a simple regular face.

Fig. 4 shows the result of tiling a model of genus 2. The left figure shows the initial tile positions computed by error diffusion [2]. Although an even distribution is obtained on

Fig. 4 Mosaic tiling on genus-2 'eight' model. Left: initial placement of 3,000 tiles. Right: final positions of these tiles.

a large scale, the positions of each tile within a particular triangle are chosen at random; also, no vector fields are considered. Tiles placed in this way include many overlaps and hence gaps, as shown. The result after optimization demonstrates that our method can cope well with higher genus models; note that such models usually have more singularities.

These examples took from 2 to 10 minutes to compute, using a Pentium IV 2.4GHz CPU, given meshes containing 50,000 triangles and placing about 10,000 tiles.

While we may treat each example as a *global* problem, on the other hand, it is natural to consider the tiling of regions separated by feature curves as independent tiling problems, apart from the cross-boundary constraints. Doing so leads to improved performance, since the overall problem is divided into a set of smaller subproblems which can be solved *independently*. In practice, a speed up factor of between 2 and 3 is achieved for the fish and cube examples. This factor depends on the number of separate regions and their relative sizes.

7 Conclusions

This paper has given an efficient algorithm for covering a mesh with a mosaic of rectangular tiles, while following a vector field interpolated from user inputs, and also respecting geometric and color feature lines.

As well as its aesthetic uses, this method could also be useful as a basis for carrying out quad-dominated remeshing in which each quad has almost the same size and aspect ratio, with various potential applications to analysis etc.

Tiles generated by our algorithm are generally evenly distributed with respect to the vector field. However, due to the nature of our optimization approach and the underlying model, a perfectly even distribution is not usually possible. Although our method generally produces visually acceptable results, with no visible overlaps (given a sensible choice of grout factor), an overlapping-free tiling cannot be guaranteed by our current algorithm; to provide this, it seems that a rather different approach would be required.

Our work could be extended in various ways. To improve tiling quality, a limited number of smaller or triangular tiles could be added, using a postprocessing step after the current algorithm. Other tiling elements with more complicated shapes could also be of interest.

Acknowledgment

Models in this paper are courtesy of Georgia Institute of Technology, Bruno Lévy at INRIA, France and the Shape Repository of AIM@Shape. This work was supported by the Natural Science Foundation of China (Project Number 60225016, 60321002) and the National Basic Research Project of China (Project Number 2002CB312101).

References

1. Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., Desbrun, M.: Anisotropic polygonal remeshing. ACM Transactions on Graphics **22**(3), 485–493 (2003)
2. Alliez, P., de Verdiere, E.C., Devillers, O., Isenburg, M.: Isotropic surface remeshing. In: Proc. Shape Modeling International Conference, pp. 49–58 (2003)
3. Botsch, M., Kobbelt, L.: Resampling feature and blend regions in polygonal meshes for surface anti-aliasing. Computer Graphics Forum **20**(3), 402–410 (2001)
4. Cohen-Steiner, D., Morvan, J.M.: Curve and surface reconstruction: Restricted delaunay triangulations and normal cycle. In: Proc. 19th Annual ACM Symposium on Computational Geometry, pp. 312–321 (2003)
5. Elber, G., Wolberg, G.: Rendering traditional mosaics. The Visual Computer **19**, 67–78 (2003)
6. Finkelstein, A., Range, M.: Image mosaics. In: R.D. Hersch, J. André, H. Brown (eds.) Proc. 7th International Conference on Electronic Publishing, pp. 11–22 (1998)
7. Floater, M.S., Hormann, K.: Surface parameterization: a tutorial and survey. In: Advances in Multiresolution for Geometric Modelling, pp. 157–186. Springer-Verlag, Heidelberg (2005)
8. Hausner, A.: Simulating decorative mosaics. In: Proc. ACM SIGGRAPH, pp. 573–580 (2001)
9. Kaplan, C.S., Salesin, D.H.: Escherization. In: Proc. ACM SIGGRAPH, pp. 499–510 (2000)
10. Kim, J., Pellacini, F.: Jigsaw image mosaics. ACM Transactions on Graphics **21**(3), 657–664 (2002)
11. Klein, A.W., Grant, T., Finkelstein, A., Cohen, M.F.: Video mosaics. In: Second International Symposium on Non Photorealistic Rendering, pp. 21–28 (2002)
12. Marinov, M., Kobbelt, L.: Direct anisotropic quad-dominant remeshing. In: Proc. Pacific Graphics, pp. 207–216 (2004)
13. Mount, D., Arya, S.: ANN: A library for approximate nearest neighbor searching, ver 1.1, *http://www.cs.umd.edu/mount/ANN* (2005)

14. Ray, N., Li, W.C., Lévy, B., Sheffer, A., Alliez, P.: Periodic global parameterization. ACM Transactions on Graphics p. (to appear) (2006)
15. Silvers, R., Hawley, M.: Photomosaics. New York: Henry Holt (1997)
16. Surazhsky, V., Gotsman, C.: Explicit surface remeshing. In: Proc. Eurographics Symposium on Geometry Processing, pp. 17–28. Aachen, Germany (2003)
17. Szeliski, R., Tonnesen, D.: Surface modeling with oriented particle systems. In: Proc. SIGGRAPH, pp. 185–194 (1992)
18. Turk, G.: Texture synthesis on surfaces. In: Proc. ACM SIGGRAPH, pp. 347–354 (2001)
19. Wei, L.Y., Levoy, M.: Texture synthesis over arbitrary manifolds. In: Proc. ACM SIGGRAPH, pp. 355–360 (2001)
20. Witkin, A., Heckbert, P.: Using particles to sample and control implicit surfaces. In: Proc. ACM SIGGRAPH, pp. 269–277 (1994)
21. Yang, Y.L., Lai, Y.K., Hu, S.M., Pottmann, H.: Robust principal curvatures on multiple scales. In: Proc. Eurographics Symposium on Geometry Processing, p. (to appear) (2006)

Ralph R. Martin obtained his Ph.D. in 1983 from Cambridge University and is a professor at Cardiff University. He has published over 160 papers and 10 books covering such topics as solid and surface modeling, intelligent sketch input, geometric reasoning, reverse engineering, and various aspects of computer graphics. He is on the editorial boards of *Computer Aided Design* and the *International Journal of Shape Modelling*.

Yu-Kun Lai received the bachelor's degree in computer science from Tsinghua University in 2003. He is a PhD student in the Department of Computer Science and Technology at Tsinghua University. His research interests include computer graphics, geometry processing, and CAGD.

Shi-Min Hu received the PhD degree in 1996 from Zhejiang University. He is currently a professor of computer science at Tsinghua University. His research interests include digital geometry processing, video-based rendering, rendering, computer animation, and computer aided geometric design. He is on the editorial board of *Computer Aided Design*.