Eurographics Symposium on Rendering (2005)
Kavita Bala, Philip Dutré (Editors)

Texture Tiling on Arbitrary Topological Surfaces using Wang Tiles

Chi-Wing Fu† and Man-Kang Leung†

Department of Computer Science, The Hong Kong University of Science & Technology, Hong Kong

Abstract

Synthesizing textures on arbitrary surfaces is a time consuming process. We have to analyze the surface geometry and map texture values onto the input surface adaptively. Texture tiling provides an alternative approach by decoupling the texture synthesis process into two steps: surface mapping and tile placement. This paper reformulates the texture tiling mechanism of Wang tiles for arbitrary topological surfaces. Once we created a low distortion conformal map from the input surface to a quad-based geometry, we can generate a tiling graph over the geometric dual graph of the quad-based geometry, and produce a proper tile orientation on all quad faces so that we can layout textured tiles on quads and map texture back to the input surface accordingly. Since tile placement is independent of the input surface geometry, we can perform the tiling process in no time and change texture pattern on the input surface simply by switching a tile set. No additional computation is needed. As a demonstration, we experimented texture tiling of Wang tiles on spheres, polycubes, as well as polycube-mapped models.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism: Texture

1. Introduction

High quality realistic rendering is one of the major goals in computer graphics. To achieve this goal, however, the data set size is usually very large due to the requirement of high level of detail. Regarding this, we not only have to acquire a large amount of data for the modeling; we also have to process the same amount of data during the rendering. Texture synthesis provides a practical solution for data acquisition. By acquiring just a small texture sample, texture synthesis can reproduce the statistical pattern of the texture and wrap the entire input surface with the given texture pattern.

Though conventional texture synthesis techniques help to reduce the workload in data acquisition, we still have to deal with a large amount of data generated from the texture synthesis process. Texture tiling offers an alternative approach for texturing surfaces. Instead of synthesizing textures directly on the input surface, texture tiling introduces an intermediate surface to decouple the texture synthesis process into two steps: *surface mapping* and *tile placement*. The surface mapping step analyzes the input surface geometry and constructs a surface map from the input surface to a user-specified intermediate surface. The tile placement step then arranges textured tiles on the intermediate surface so that we can map tiled texture back to the input surface based on the surface map constructed. The major advantage of this approach is that the texturing process becomes independent of the surface geometry; we only need to synthesize textures on tiles. Once a tile set is synthesized, we can employ it to texture different input surfaces; changing of texture pattern on input surfaces can be done simply by changing the reference tile set. No additional computation is needed.

According to [CSHD03], to avoid tile rotation and thus the doubling of edge colors, the use of Wang tiles for non-periodic texture tilings is restricted to surfaces that map to a plane. Thus, texture tiling of Wang tiles (or Wang cubes) is restricted to 2D Euclidean planes (or 3D Euclidean space). This paper reformulates the mechanism of Wang tiles for texture tiling on arbitrary topological surfaces. We can avoid the doubling of edge colors, and tiles are still non-rotatable. Once we created a surface map from the input surface to an intermediate surface in the form of a quad-based geometry, e.g., polycubes [THCM04], we can build a tiling graph over the geometric dual graph of the geometry so as to set up a proper tile orientation on quads. As a result, texture can be arranged on the quads and mapped back to the input surface.

1.1. Related work

Texture Synthesis: Efros et al. [EL99] extended Popat's work [PP93] and proposed an exhaustive algorithm to

† {cwfu,cskang}@cs.ust.hk

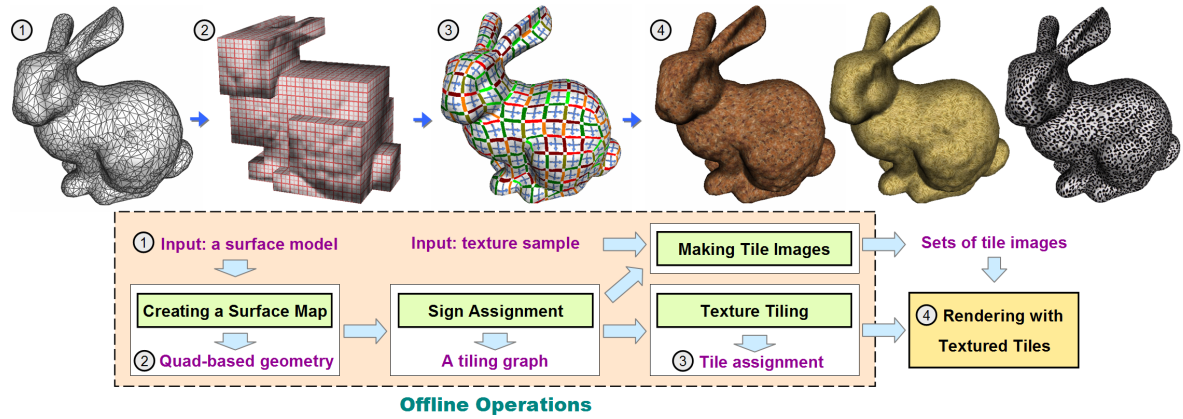


Figure 1: *The above flowchart outlines our texture tiling approach with a running example* BUNNY.

generate novel texel values by window matching. Wei et al. [WL00] accelerated the synthesis process by using tree-structured vector quantization. Praun et al. [PFH00] proposed a patch-based method that assembles textures on arbitrary surfaces [Tur01, WL01, SCA02, TZL*02, MK03].

History of Wang tiles: The theory of Wang tiles can be traced back to the early 60' when Wang [Wan61, Wan65] proposed non-periodic tiling of a plane. The tiling set consists of a set of square tiles, known as the Wang tiles, in such a way that edges of tiles are color-coded. In order to create a valid tiling of a plane by Wang tiles, all shared edges should have matched color. Grunbaum and Shepherd [GS86] examined this subject in depth and showed how to tile a plane aperiodically with a finite set of Wang tiles. Later, Culik [Cul96] proved that thirteen tiles are enough for such a non-periodic tiling [Ber66, Kar96], and Moore et al. [MRR02] applied the tiling group theory to the problem and studied the tilability of regions with boundary condition.

Tiling in Computer Graphics: Glassner [Gla98] presented various interesting tiling patterns: regular, semi-regular, and aperiodic. Neyret [NC99] proposed the symmetric edge constraint to tile regular triangular models by manually created patterns. Stam [Sta97] was the first to apply non-periodic tiling to texture creation. Wang tiles were used as texture containers for patterns such as water surface and caustic. Cohen et al. [CSHD03, SCM02] further investigated this approach, and invented an automatic method to synthesize textures on Wang tiles. Wei [Wei04] devised a Wang tile arrangement scheme in texture memory so as to correct texture filtering problem. Decaudin [DN04] applied tiling to generate and render forest scenes in real-time. Tiles were used as geometry containers [HDK01]. Stephen [Che04] further extended this concept, and employed Wang tiles to contain and generate animated flow patterns.

2. Overview of the approach

Figure 1 outlines the overall approach. The input is a surface model mappable to a quad-based geometry (intermediate surface). Different methods can be used to create such a surface map as long as the surface map is conformal and has low area distortion. Polycube-map [THCM04] is chosen for the task because of its flexibility and efficiency, however, since the proposed tiling approach is general for any quad-based geometry, models such as a quad-based icosahedron can be employed as an intermediate surface as well. After making a quad-based geometry, the next steps are sign assignment (Section 3), tile assignment (Section 4), and tile set synthesis. Note that we follow the tile synthesis method in [CSHD03] (outlined in Figure 2) to generate tile images: We first apply image quilting [EF01] to enlarge the input texture sample. Then, for each edge color in the tile set, we randomly carve a diamond shape out of the enlarged texture, and merge (dynamic programming [EF01] or graphcut technique [KSE*03]) them correspondingly to create each Wang tile in the tile set (Figure 3(a)). Providing tiles are not rotated or reflected, matching of edge colors can perfectly restore the texture pattern originally in the diamond samples.

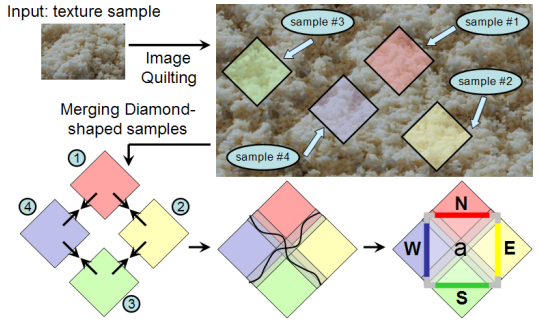


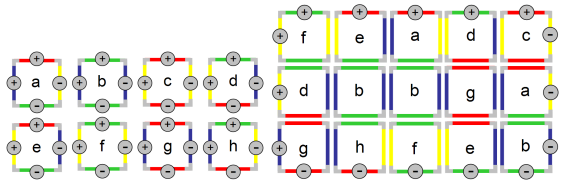
Figure 2: *Making tile images by merging diamond samples.*

3. The Sign Assignment Algorithm

3.1. Embedded Signs in Wang Tiles

Looking at the above tile synthesis method, we can see that there are actually two kinds of edges as related to the side of the diamond sample an edge contains: left/right and top/bottom. To differentiate the related side, we label N and W edges taking the bottom or right half of a diamond sample as *positive*, and S and E edges taking the top or left half

of a diamond sample as *negative*. Hence, we can explicitly match (same) colors and (opposite) signs in the tiling. Figure 3 depicts signs on Wang tiles and signs on a patch of 5×3 tiles. Note that patches of tiles always have the same sign arrangement (around the boundaries) as a single tile.



(a) (b)

Figure 3: *Signs on (a) Wang tiles and (b) a patch of tiles.*

3.2. The Sign Assignment Algorithm

Based on the signed edge concept, given a quad-based geometry as an intermediate surface, our first task is to assign signs to edges of the geometry so that we can orientate tiles on quad faces. To achieve this goal, we first represent the given quad-based geometry by a geometric dual graph, say $G = (V, E)$; Faces on the geometry are represented as vertices in the vertex set V while connectivities between faces are represented as edges in the edge set E. Thus, if we can transform G into a directed graph, called the ***tiling graph***, say $G' = (V, E')$, such that each edge in G' carries a certain direction from one quad to another, we can then determine signs on shared edges between every pair of quads.

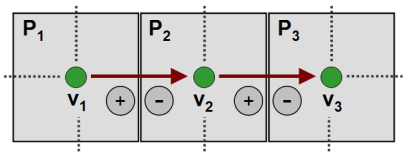


Figure 4: *Directed edges in G' produce signs on quads.*

Figure 4 illustrates the formulation. The right edge of quad P_1 (corresponding to $v_1 \in V$) is connected to the left edge of quad P_2 (corresponding to $v_2 \in V$).

> *If a directed edge in G' goes from v_1 to v_2, we assign positive sign to the edge of P_1 next to P_2 and negative sign to the edge of P_2 next to P_1.*

In this way, if we can assign directions to all edges in G so that every vertex in G' has exactly two indegrees and two outdegrees, we can put signs on edges of quad faces so that all neighboring quads can match with opposite signs automatically. Furthermore, if we can assign signs to quads in such a way that the two positive signs (negative signs) associated with the outgoing directions (ingoing directions) are adjacent to each other on the quads, the resultant sign arrangement on quads can always agree with the sign arrangement shown in Figure 3(b). The algorithm on top of the next column presents the pseudo-code for the task. It takes G as its input and generates directed edges sequentially.

Algorithm 1 SIGNASSIGNMENT (V, E)

```
E' ← ∅
while E ≠ ∅ do
  /* Randomly pick an edge in E as the starting edge */
  e0 ← (v0, v1) ← random(E)
  E ← E - {e0} and E' ← E' ∪ {v0 → v1}

  /* Pick consecutive edges at the v1 side of e0 */
  repeat
    e1 ← (v1, v2) ← search(E) such that e1 contains v1
                    and is opposite to e0 on quad v1
    if (e1 ≠ NULL) then
      E ← E - {e1} and E' ← E' ∪ {v1 → v2}
    end if
    e0 ← e1 and v1 ← v2
  until e0 = NULL
end while
return E'
```

An important strategy in the algorithm is that when we search for e_1 in E, we must select the edge opposite to the previously selected edge with respect to the quad face denoted by v_1. For example, in Figure 4, after selecting $v_1 \rightarrow v_2$, we must select $v_2 \rightarrow v_3$ so that opposite (left and right) edges of quad v_2 can receive opposite signs. As a result, when the algorithm finishes, every vertex in G' can receive exactly two indegrees and two outdegrees; thus, all quad faces can have exactly two positive signs and two negative signs accordingly. Furthermore, within each quad, the two positive (negative) signs can always locate next to each other so that the resultant sign arrangement on quads match precisely the sign arrangement shown in Figure 3(b).

Figure 5 demonstrates the algorithm on the geometric dual graph of a cube. After picking $v_2 \rightarrow v_5$, we go into the *repeat* loop of the algorithm and obtain successive edges: $v_5 \rightarrow v_4$, $v_4 \rightarrow v_6$, and $v_6 \rightarrow v_2$. Since G represents a closed surface, we can always loop back to the first edge and obtain a cycle in G. By continuing the algorithm, we obtain two more cycles: $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_1$ and $v_1 \rightarrow v_6 \rightarrow v_3 \rightarrow v_5 \rightarrow v_1$. Then, based on these directed edges, we can set up signs on quads and thus the sign layout on the cube (bottom left).

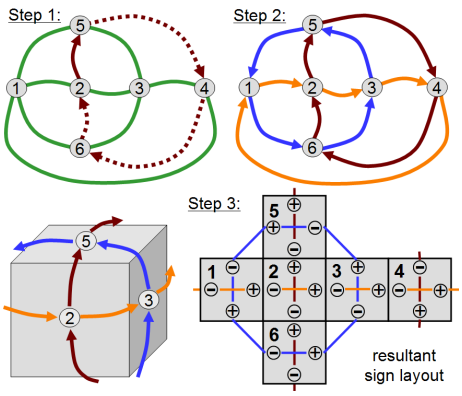


Figure 5: *Sign assignment on a cube.*

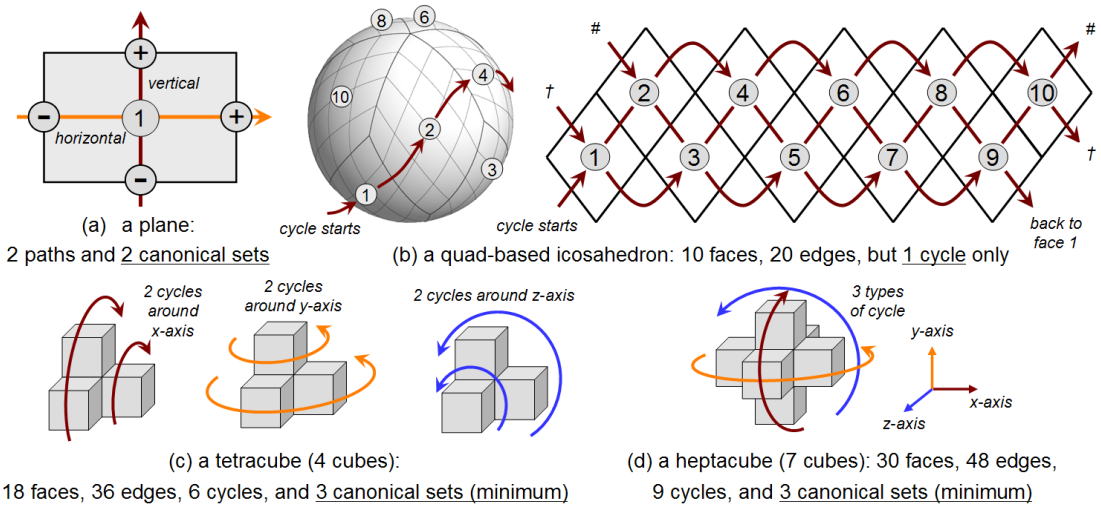


Figure 6: *Examples of Canonical Sets: (a) a plane, (b) a quad-based icosahedron, (c) a tetracube, and (d) a heptacube.*

Note that a cube always have three cycles running over its surface. Indeed, if G represents a closed surface, starting from any edge in it, we can always loop back to the first edge based on the traversal rule in the algorithm. This can be proved by Fleury's algorithm because G is always Eulerian.

4. Texture Tiling

4.1. Canonical Sets in the Tiling Graph

The sign assignment algorithm not only enables us to orientate Wang tiles on quad faces, it also helps to generalize the edge coloring scheme [CSHD03] in plane tiling†. In case of arbitrary surfaces, though we no longer have vertical and horizontal edges for color assignment, we have cycles and paths. To deal with the fact that complicated quad-based geometries could have too many cycles in G', we first devise the following definition to facilitate color assignment:

> *Two paths in G' are said to be **independent** if they do not intersect, i.e., no common vertex.*

By this means, we can group independent paths together as a set, namely the ***canonical set***, and assign them with the same group of edge colors. Note that using the word *path* in the definition above can account for cycles, and thus make the definition applicable for open surfaces as well.

4.2. Examples of Canonical Sets

Careful readers may notice that given the paths and cycles generated by the sign assignment algorithm, we can have many ways of grouping them into canonical sets. In fact, the problem of finding the minimum number of canonical sets in an arbitrary tiling graph is an NP-complete problem. This can be proved by transforming the problem into the maximum clique problem in graph theory. However, if we use polycubes as the quad-based geometry, the advantage is that because of the orthogonal structure, we only have three canonical sets regardless of the polycube shape.

Figure 6 exemplifies canonical sets in four different quad-based geometries: (a) a plane, (b) a quad-based icosahedron mapped on a sphere, (c) a tetracube, and (d) a heptacube. Note that a quad-based icosahedron is built by joining pairs of neighboring triangles in a regular icosahedron. From the figure, we can observe various interesting facts about paths and cycles. First, a planar surface has two degenerated paths and so two canonical sets, however, as a more complicated structure, the quad-based icosahedron only has one cycle in its tiling graph. Such a cycle self-intersects with itself ten times at all vertices in the graph. On the other hand, polycubes demonstrate the orthogonality property. Since cycles rotating about the same axis in the polycube space never intersect in the tiling graph, we can group them together into the same canonical set. As a result, we can always end up producing three canonical sets for any polycube model.

4.3. Creating Wang Tile Sets

Using the canonical sets extracted, we can generalize the K_h/K_v edge coloring scheme in plane tiling and define Wang tile sets for arbitrary surfaces. Since paths and cycles belonging to the same canonical set do not intersect in the tiling graph, we can assign them with same set of colors as if they are horizontal or vertical edges in the case of plane tiling.

> *Given C as the total number of canonical sets in a tiling graph, we define K_i to be the set of colors assigned to the ith canonical set where $i = 1$ to C.*

Thus, given M as the total number of colors, we can assign integers ranged $[1, M]$ to each K_i and set up a color assignment. In practice, 2 or 3 colors are sufficient for each K_i.

† For non-periodic plane tiling without boundary condition, we assign K_h and K_v colors for horizontal and vertical edges, and create $2K_hK_v$ Wang tiles, refer to [CSHD03] for the details.

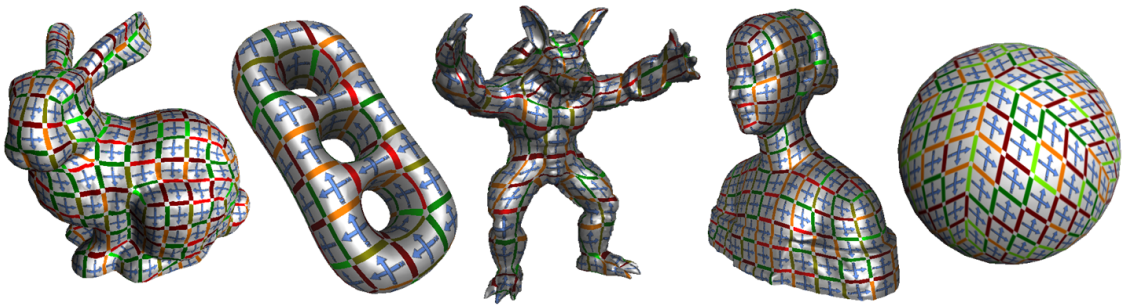


Figure 7: *Sign assignment and tile arrangement on models (left-to-right):* BUNNY, HOLES, ARMADILLO, LAURANA, *and a quad-based icosahedron (mapped on a sphere). The arrows on tiles indicate the associated signs and cycle directions.*

After setting up a color assignment using K_i's, we can start making Wang tile sets for the tiling process. For each pair of intersecting canonical sets (including self-intersection), we can produce one Wang tile set for the pair using the tile synthesis method proposed in [CSHD03]. In this way, to tile a quad on the quad-based geometry, we can look up the associated pair of intersecting canonical sets on the quad, and pick up the corresponding Wang tile set to tile the surface. By repeating this tiling process for all the quads on the geometry, we can layout tiles seamlessly over the entire surface.

Wang Tile sets for Polycube models Creating Wang tile sets for polycube models is special. Due to the fact that we always have three canonical sets for polycubes, we only have three intersections among canonical sets. Therefore, if we fix all the three K_i's, we can fix a certain Wang tile set arrangement universal for all polycube models regardless of their shapes. We thus have the following advantages.

Firstly, the synthesis of tile images becomes independent of the tiling graph, in addition to the surface geometry; thus, we can pre-synthesize sets of tile images without referring to any polycube structure. Secondly, the use of Wang tile sets on polycube models becomes flexible and dynamic. We can apply the same set of Wang tiles to different polycube models without re-synthesizing the tile images or re-tiling the polycube surface; we can switch texture on polycube models simply by changing the reference tile set.

5. Implementation and Results

5.1. Implementation of Texture Tiling

Our texture tiling system was implemented and experimented on a Pentium4 1.9GHz PC with 512MB memory. The pre-processing time for creating tile sets is surprisingly fast. Table 1 shows the computation time for generating a full set of Wang tiles for tiling polycube models. The computation time includes image quilting, extraction and merging of diamond samples, as well as tile set generation.

Texture Tiling on Polycube models When implementing texture tiling on polycube models, we can have the following optimizations based on the orthogonal structure of polycubes. First, the sign assignment algorithm can be simplified.

Table 1: *Average time for creating a full tile set.*

	Resolution of a Wang tile (pixel unit)		
	32×32	64×64	96×96
$\|K_i\| = 2$	20 seconds	65 seconds	136 seconds
$\|K_i\| = 3$	87 seconds	275 seconds	564 seconds

Due to the fact that cycles belonging to the same canonical set are all perpendicular to a particular axis in the polycube space, if we fix all cycles to be anti-clockwise (or clockwise) around their corresponding axis (x-axis, y-axis, or z-axis), polycube faces facing the same direction will always receive the same sign arrangement. Thus, we can pre-compute six sign arrangements, and assign them to polycube faces simply by checking the direction a polycube face looks at.

Secondly, we do not need to check cycle intersection or generate canonical sets. We can assign cycles to the three pre-defined canonical sets simply by checking which axis a cycle rotates about. Hence, we can fix a certain color assignment and so a fixed tile arrangement by fixing some K_i's. With these optimizations, the total pre-processing time needed for sign assignment and texture tiling is only 0.021 seconds for the BUNNY model. However, because of boundary condition in tiling, we need almost all permutations of edge colors in a Wang tile set, i.e., $\sim |K_i|^4$ tiles in a Wang tile set. Given 64×64 as the resolution of a single colored tile image (RGB) and $|K_i| = 3$, the total memory consumption for a Wang tile set is $3^4 \times 64^2 \times 3$ bytes, which is around 1 MB; hence, a full tile set is manageable and small enough to be stored in texture memory. Furthermore, to avoid texture filtering problem for texels near tile boundaries, we arrange tile images using the tile arrangement scheme in [Wei04].

5.2. Rendering and Tiling results

Figure 8 (in color pages) shows texture tiling results on different 3D models: BUNNY, HOLES, ARMADILLO, and LAURANA. The intermediate surfaces in use are all polycubes, and we employ polycube-maps provided in [THCM04] as the surface map. Twelve different texture samples are used in the experiment, and we generate Wang tile sets using a fixed tile arrangement with $|K_i|$ set to 3. The

corresponding sign assignment and tile arrangement on the models are shown in Figure 7. We can clearly see the area occupied by each tile on the objects, and the corresponding cycle directions, as indicated by the arrows on tiles.

6. Discussion and Future work

This paper presents a texture tiling approach by reformulating Wang tiling on arbitrary topological surfaces. By means of an intermediate surface, texture tiling decouples the texture synthesis process into the surface mapping step and the tile placement step. Texture synthesis becomes independent of the surface geometry. Instead of synthesizing texture on the object surface, we only need to synthesize texture on some pre-defined tiles. Once the synthesis process is done, we do not need to re-synthesize texture for the same texture sample any more. We can tile the intermediate surface by using the generated tile set, and map the tiled texture back to the input surface through the surface map created. Furthermore, since tile sets are independent of the input surface, we can change textures on the input surface simply by changing the reference tile set. The same tile set can be applied to different surfaces. We do not need to re-tile the surface or re-synthesize tile images for these operations.

In terms of practical value, the texture tiling approach works especially well with polycubes or surfaces mappable to polycubes. Due to the orthogonal structure of polycubes, we can optimize the texture tiling process and pre-synthesize Wang tile sets with fixed K_i's. Then, we can apply the same tile set to different polycube-based models regardless of their geometric structures. Both the offline texture tiling process and the run-time rendering process are highly efficient. This research work opens a new direction in surface modeling. More than just color textures, we can tile surfaces with shell textures [CTW*04], height fields, and even bidirectional texture functions (BTF) [TZL*02]. Tiling provides an efficient way in adding surface details and improving visual quality.

Acknowledgments We would like to thank Marco Tarini of the Visual Computing Lab, ISTI/CNR, for sharing the polycube-mapped models and associated fragment program, and anonymous reviewers for their comment on the paper.

References

[Ber66] BERGER R.: The undecidability of the domino problem. *Memoirs of the American Mathematical Society* (1966), 66.

[Che04] CHENNEY S.: Flow tiles. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), ACM Press, pp. 233–242.

[CSHD03] COHEN M. F., SHADE J., HILLER S., DEUSSEN O.: Wang tiles for image and texture generation. *Proc. of SIGGRAPH 2003* (2003), 287–294.

[CTW*04] CHEN Y., TONG X., WANG J., LIN S., GUO B., SHUM H.-Y.: Shell texture functions. *Proc. of SIGGRAPH 2004* (2004), 343–353.

[Cul96] CULIK K.: An aperiodic set of 13 Wang tiles. *Discrete Math. 160* (1996), 245–251.

[DN04] DECAUDIN P., NEYRET F.: Rendering forest scenes in real-time. In *Eurographics Symposium on Rendering 2004* (2004), pp. 93–102.

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. *Proc. of SIGGRAPH 2001* (2001), 341–346.

[EL99] EFROS A., LEUNG T. K.: Texture synthesis by non-parametric sampling. In *IEEE Intl. Conf. on Computer Vision* (1999), pp. 1033–1038.

[Gla98] GLASSNER A.: Aperiodic tiling. *IEEE Computer Graphics & Appplications 18*, 3 (May-June 1998), 83–90.

[GS86] GRÜNBAUM B., SHEPHARD G. C.: *Tilings and patterns*. W. H. Freeman & Co., 1986.

[HDK01] HILLER S., DEUSSEN O., KELLER A.: Tiled blue noise samples. In *VMV '01: Proceedings of Vision, Modeling, and Visualization Conference 2001* (2001), pp. 265–271.

[Kar96] KARI J.: A small aperiodic set of wang tiles. *Discrete Math. 160* (1996), 259–264.

[KSE*03] KWATRA V., SCHODL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *Proc. of SIGGRAPH 2003* (2003), 277–286.

[MK03] MAGDA S., KRIEGMAN D.: Fast texture synthesis on arbitrary meshes. In *Eurographics workshop on Rendering* (2003), pp. 82–89.

[MRR02] MOORE C., RAPAPORT I., REMILA E.: Tiling groups for wang tiles. In *SODA '02: Proc. of the 13th annual ACM-SIAM symposium on Discrete algo.* (2002), pp. 402–411.

[NC99] NEYRET F., CANI M.-P.: Pattern-based texturing revisited. *Proc. of SIGGRAPH 99* (1999), 235–242.

[PFH00] PRAUN E., FINKELSTEIN A., HOPPE H.: Lapped textures. *Proc. of SIGGRAPH 2000* (2000), 465–470.

[PP93] POPAT K., PICARD R. W.: Novel cluster-based probability model for texture synthesis, classification, and compression. In *SPIE Visual Comm. and Image Processing* (1993).

[SCA02] SOLER C., CANI M.-P., ANGELIDIS A.: Hierarchical pattern mapping. *Proc. of SIGGRAPH 2002* (2002), 673–680.

[SCM02] SHADE J., COHEN M., MITCHELL D.: Tiling layered depth images, 2002. Tech. report, U. of Washington, Seattle.

[Sta97] STAM J.: *Aperiodic Texture Mapping*. Tech.Rep. R046, 1997. European Research Consortium for Informatics and Math.

[THCM04] TARINI M., HORMANN K., CIGNONI P., MONTANI C.: Polycube-maps. *Proc. of SIGGRAPH 2004* (2004), 853–860.

[Tur01] TURK G.: Texture synthesis on surfaces. *Proc. of SIGGRAPH 2001* (2001), 347–354.

[TZL*02] TONG X., ZHANG J., LIU L., WANG X., GUO B., SHUM H.-Y.: Synthesis of bidirectional texture functions on arbitrary surfaces. *Proc. of SIGGRAPH 2002* (2002), 665–672.

[Wan61] WANG H.: Proving theorems by pattern recognition ii. *Bell Systems Technical Journal 40* (1961), 1–42.

[Wan65] WANG H.: Games, logic, and computers. *Scientific American* (1965), 98–106.

[Wei04] WEI L.-Y.: Tile-based texture mapping on graphics hardware. In *SIGGRAPH/EUROGRAPHICS Conf. on Graphics Hardware* (2004).

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. *Proc. of SIGGRAPH 2000* (2000), 479–488.

[WL01] WEI L.-Y., LEVOY M.: Texture synthesis over arbitrary manifold surfaces. *Proc. of SIGGRAPH 2001* (2001), 355–360.

Figure 8: *Texture tiling on different 3D models (from top to bottom):* BUNNY, HOLES, ARMADILLO, *and* LAURANA *(data courtesy of M. Tarini). Polycubes are used as the intermediate surface, and* $|K_i| = 3$ *in the tile set generation.*