Pattern-Based Texturing Revisited

Fabrice Neyret   Marie-Paule Cani

iMAGIS*/ GRAVIR-IMAG

Abstract

We present a texturing method that correctly maps homogeneous non-periodic textures to arbitrary surfaces without any of the difficulties usually encountered using existing tools. Our technique requires little redundant designer work, has low time and memory costs during rendering and provides high texture resolution.

The idea is simple: a few triangular texture samples, which obey specific boundary conditions, are chosen from the desired pattern and mapped in a non-periodic fashion onto the surface. Our mapping algorithm enables us to freely tune the scale of the texture with respect to the object's geometry, while minimizing distortions. Moreover, it yields singularity-free texturing whatever the topology of the object. The sets of texture samples may be created interactively from pictures or drawings. We also provide two alternative methods for automatically generating them, defined as extensions of Perlin's and Worley's procedural texture synthesis techniques.

As our results show, the method produces textured objects that look reasonable from any viewpoint and can be used in real-time applications.

Keywords: Texture Mapping, Patterns, Texture Synthesis, Non-periodic Tiling

1 Introduction

Reproducing the visual complexity of the real world is a dream for many Computer Graphics practitioners. Since every detail cannot be modeled at the geometric level, textures are very useful for adding visual complexity to a synthetic scene. They can for instance be used for representing rocks or vegetation on a distant mountain, for simulating animals' fur or skin, human clothes, or the surface aspect of a material. Most of the textures we need for modeling natural objects, either mineral, vegetable, or animal, have a common feature: they may look homogeneous at a large scale (i.e., large scale statistical properties do not depend on the location), but no visible periodicity can be found anywhere.

Texturing arbitrary shapes with such textures is a challenge for artists, since no CG tool is really adequate to fit real-world constraints: generating the texture directly on the surface is memory and time consuming (either for the CPU or the artist), while using standard image mapping results in pattern distortions, discontinuities, and obvious periodicity.

We present a practical solution to this problem, which involves no increase in computational or memory cost at rendering time over standard image mapping techniques using repetitive patterns. Our method works for arbitrary surfaces. It yields little distortion of the texture, no singularities whatever the topology of the surface, and no periodicity.

1.1 Related work

Despite years of CG research and tool development, artists still have difficult (and time consuming) work to do in order to achieve the texturing of complex surfaces. This paper focuses on homogeneous non-periodic textures, such as those we need for natural objects (textures may define any surface attribute, such as color, transparency, normal perturbation or displacement). The two mains problems to solve are *texture generation* and *texture mapping*. Let us review the solutions offered by existing tools:

Standard 2D texture mapping: The first solution consists of mapping a single image of the desired texture onto the synthetic object. To do this, a global parameterization of the object surface is required. As a consequence, there will necessarily be discontinuities of the texture somewhere on the surface if the object is closed or has a higher topologic order. Moreover, the texture may be highly distorted if the object has an arbitrary geometry. Optimization techniques such as those in [1, 11] can be used to reduce distortions, either locally, or by allowing the introduction of 'cracks', i.e., discontinuities. Entirely suppressing distortions by editing the mapping is impossible, except if the object's surface can be unfolded onto a plane (such as a cloth). This is not the case for natural shapes. A solution for the user to eliminate apparent texture distortions is to draw a pre-distorted texture that will compensate for the distortions due to the mapping. However, this requires high designer skills[2], and the work needs to be re-done from scratch for every new object.

An alternative is to use pattern-based texture mapping, which consists of repetitively mapping a small rectangular texture patch representing a sample of the desired texture pattern onto the surface. The sample image has to obey specific boundary conditions in order to join correctly with itself. More precisely, it needs to have a toroidal topology: the texture on the left edge must fit the texture on the right, and respectively the top edge has to fit with the bottom. Such texture samples can be created by editing pictures or drawings using interactive 2D painting systems. An advantage with respect to the previous approach is that, being small, the texture sample will be stored at a higher resolution, and will demand less redundant work by the artist. Moreover, it can be re-used for texturing other objects. Discontinuity and distortion problems, however, will be exactly the same as for a single texture map as long as a global parameterization is used to map the texture pattern. See Figure 1.

It should be be noted that these two techniques are the only methods available in current graphics hardware. Thus, other representations or design techniques have to be converted into this representation for rendering if real time constraints apply.

*iMAGIS is a joint project of CNRS, INRIA, Institut National Polytechnique de Grenoble and Université Joseph Fourier.
Address: BP 53, F-38041 Grenoble cedex 09, France
E-mail: [Fabrice.Neyret|Marie-Paule.Cani]@imag.fr
WWW: http://www-imagis.imag.fr/TEXTURES/

[2]This is actually done in practice in industry!

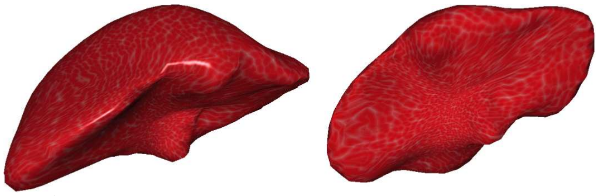

Figure 1: Standard pattern-based mapping used for applying a cellular pattern onto the geometric model of a liver. Distortions are clearly noticeable.

Interactive techniques: The problem of finding good local parameterizations for the surfaces is solved in patch-based interactive texturing systems by leaving the user to tile the surface [12, 14, 15]. In [14], the latter interactively subdivides an implicit surface into square patches. Surface geodesics are used for fitting the borders of these patches to the surface. Optimization is then used for deriving a minimally-distorted local parameterization inside each patch. This approach, which can be extended to parametric surfaces as well, can be combined with pattern-based texturing in order to cover an object with a given pattern. However, using a local instead of a global parameterization is not sufficient for avoiding texture discontinuities on closed surfaces (to be convinced, try to map a texture sample with a toroidal topology onto a cube): texture discontinuities will appear across some of the edges, since the neighboring borders of the sample image cannot be those expected everywhere.

Entirely avoiding both distortions and discontinuities can be achieved by using interactive texture painting software [8]. As in the first method, a single texture map corresponding to a global parameterization of the surface is used. However the texture content is directly designed on the object's surface before being stored as a map. The texture map may then appear distorted and discontinuous, but it will be correct when it is rendered. Depending on the user's skills, an homogeneous non-periodic texture may be designed using this method. However, this technique yields a high memory cost (as in the first approach) and consumes lots of user's time since texture details must be drawn all over the surface. Moreover, the user work is almost never re-usable on another shape.

Texture synthesis techniques: An alternative to painting the texture onto the surface is to automatically generate it, which has the advantage of saving user's time by replacing the redundant design work by a high level control of the texture features. A wide range of parametric texture synthesis techniques that are convenient for generating natural textures have been proposed [16, 21, 19, 22].

One such method is solid texturing, which involves defining a 3D material field (e.g. marble, wood) which is intersected with the object's surface to create the texture [16, 22]. No distortion nor discontinuity across the object's edges will be produced, since no surface mapping occurs. However the method is restricted to texture patterns that intrinsically come from a 3D phenomenon: it cannot capture surface properties such as the detailed appearance of the skin (e.g. regular scales). Another drawback is that synthesizing the texture during rendering will not allow real-time performance, since it is a per-pixel based computation. An alternative would be to store 3D texture tables at a high memory cost.

Other procedural techniques such as reaction diffusion [21, 19] can be used to generate a pattern-based texture directly on an object's surface. These methods are computationally costly or have a high memory cost, depending whether the texture is generated on the fly or precomputed and stored. We can note that Perlin's and Worley's techniques may also be used on surfaces (as opposed to solid material). However Perlin's noise requires a grid to be generated, so a global parameterization needs to be introduced.

Lastly, all the listed procedural techniques can easily be extended to the automatic generation of square 2D texture samples that have a toroidal topology. The latter can then be used in pattern-based mapping techniques, with the distortion and discontinuity problems discussed above. Moreover, this technique would produce patterns with obvious periodicity, which would probably spoil the natural appearance of the final object.

Towards non-periodic mapping: Artistic and mathematical work on tilings such as those of Escher and Penrose (see for instance [3, 5, 6]) can also be a source of inspiration. Escher's drawings include several tilings of the plane with complex shapes such as birds, fishes or reptiles. Penrose studies aperiodic tilings of the plane, and shows that some specific sets of tiles always form non-periodic patterns.

A first attempt to build a practical application of these ideas to texturing in Computer Graphics is Stam's work on "aperiodic textures" [17]. His aim is to render water surfaces and caustics. Stam tiles the plane with a standard grid of square patches, where he maps 16 different texture samples of an homogeneous texture. To do so in a non-periodic fashion, he uses an algorithm for aperiodically tiling the plane with convex polygons of different colors [7] (the colors of the tiles in the mathematical theory correspond to the boundaries of texture tiles). The boundary conditions between texture samples are met by using the same input noise for generating the texture in the rectangular regions that surrounds a shared edge. This method is restricted to applying textures onto a plane, otherwise the usual parameterization problems yielding distortions and discontinuities would appear. Moreover, the problem of synthesizing the texture samples is only addressed for a specific texture, and the algorithm is not explicitly described.

1.2 Overview

As shown above, none of the existing tools provides an acceptable solution to the problem of texturing arbitrary surfaces without distortions and discontinuities. With these constraints, previous methods demand too much designer intervention, are not compatible with real-time display, occupy a large memory space, or a combination of the above. This is a critical situation since most applications of Computer Graphics rely on photo-realistic texturing[3]. In this paper, we introduce a full solution to this problem, designed in the spirit of pattern-based texture mapping techniques. Our method avoids discontinuities for all surface topologies, minimizes texture distortions, and avoids the periodicity of the texture patterns.

Our solution is inspired from Escher's work since the surface will not be tiled with square patches as usual, but rather with triangles on which equilateral triangular texture samples will be mapped. It also has connections with Pedersen's geodesics [14], but they are used in our case in an automatic tiling framework, where the user just has to choose the size of the texture triangles with respect to the object's geometry. Our solution to non-periodicity is similar in spirit to the one used by Stam [17], however we have solved the more intricate problem of assigning sets of triangular texture samples onto a curved surface that may be closed, while meeting boundary conditions everywhere. Lastly, our work generalizes Perlin's and Worley's texture synthesis techniques [16, 22] by allowing the automatic generation of adequate sets of triangular texture samples. Solutions using real or hand-made images are also provided.

The remainder of this paper is developed as follows: Section 2 introduces the main features of our approach. Section 3 deals with the mapping of texture samples onto an arbitrary object geometry. Section 4 describes different methods for the generation of texture sample sets, and shows a variety of results. We conclude in Section 5.

[3] Moreover, tools have to cope with real-time constraints for video-games or flight simulators, and artist-time constraints for special effects in movies production, for which designing textures is a huge part.

2 Texturing with Triangular Patches

This paper focuses on applying textures that are homogeneous at a large scale. Moreover, as for natural textures, they should be continuous, and no periodicity should be observed. The first feature is obtained by using patterns that capture the short scale surface aspect variations, and by mapping them on the surface with low distortion (see subsection 2.1). The mapping deals with the boundary conditions at the junction between patterns, discussed in subsection 2.2. The final issue concerns the assignment method, explained in subsection 2.3.

The solution described here is designed for isotropic texture patterns, which can be found in many natural objects (for instance in most human and vegetable tissues, in rust, dust, and in numerous bumpy surfaces such as rock, ground, and roughcast wall). A possible extension enabling the introduction and control of some anisotropy is discussed in future work.

2.1 Local parameterization with triangles

As we have seen in Section 1.1, the main source of problems in usual mapping techniques is that they generally rely on global surface parameterizations. In most cases, finding a correct global mapping is simply impossible. However, Nature does not need to introduce global parameterizations to 'build' its textures; local parameterizations, together with continuity constraints, are sufficient. A tiling into continuous regions that can be locally parameterized can always be found for the continuous surfaces we wish to texture. Optimization methods will work better when applied to these local regions than to the whole surface.

Rather that defining a tiling with square patches, triangles can be used to tile a surface into regions where a local parameterization will be defined. However, to the authors knowledge, no previous work has used triangular texture patterns to design surface aspect in Computer Graphics. To address the different problem of mesh re-tiling [20], Turk produces a regular triangular surface tiling with controlled size. His algorithm fits precisely our requirements. We claim that if a polygonal tiling adequately captures an object's topology, it can be used for mapping textures. This is the case even if the tiling does not conveniently redefine the geometry, being either too coarse or too precise. Consequently, the first step of our algorithm consists of building a *triangular tiling* of the surface, computed at a user-defined scale. This tiling defines a set of local parameterizations of the surfaces, which will be used for texture mapping. More precisely, a given texture sample is going to be mapped onto each triangular patch of the tiling, whose scale thus controls the texture scale. In the remainder of the paper, we call this tiling the *texture mesh*, as opposed to the *geometric mesh* that is still used to define the shape during rendering.

2.2 Texture Samples and Boundary Conditions

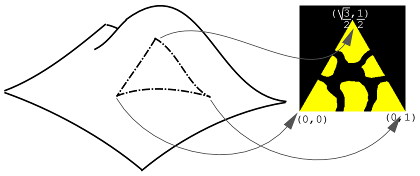

Figure 2: Each triangular patch of the *texture mesh* is mapped onto an equilateral region in a given texture sample.

The idea is to map a triangular image onto each patch of the texture mesh (see Figure 2). As these images are equilateral triangles, there will be no visible distortion if the patches of the texture mesh are approximately equilateral triangles. In order to generate a continuous texture over the entire surface, specific boundary conditions will have to be met between texture samples mapped onto adjacent patches. Basically, the two patterns in the neighborhood of the border separating two patches have to fit, which means that at least both texture values and derivatives are to be the same along a common edge. The methods presented in this paper solve for these local continuity constraints under the hypothesis that the textured patches are equilateral. If a texture sample happened to be mapped on a low quality patch (e.g. with sharp corners), the texture would be distorted and discontinuities of the texture gradient would appear on the patch edges. We have found that our solution is still sufficient in practice, since good quality meshes made of quasi-equilateral triangles can be computed for almost any surface.

Achieving texture continuity constraints using triangular tiles is more intricate than doing it with a square grid of tiles. As soon as a mesh node on a curved surface can be shared by an arbitrary (small) number of neighboring patches (either triangular or square), there is no longer anything equivalent to the 'toroidal topology' that exists when tiling the plane: no global orientation can be defined, so two patches may be connected by any edges. Moreover, this increases a priori the constraints on the texture content near the sample corners. More precisely, this enforces a zero texture gradient at those points, otherwise one would have to manage a continuity constraint between the edges of a triangle. The methods we provide for the generation of texture samples, described in Section 4, will have to cope with these boundary constraints. As our results will show, the gradient constraint at corners does not create any noticeable visual artifacts.

Since our method only relies on local parameterizations and on local continuity constraints between texture patches, it yields singularity free texturing whatever the topology of the object is (see Figure 3). However, the scale of the texture details drawn on the samples must not be too large. Otherwise, unexpected path shapes will appear at the object surface, such as those depicted in Figure 3. A practical solution to this signal processing problem consists of using patterns that are large enough to contain more than a single feature of the texture.

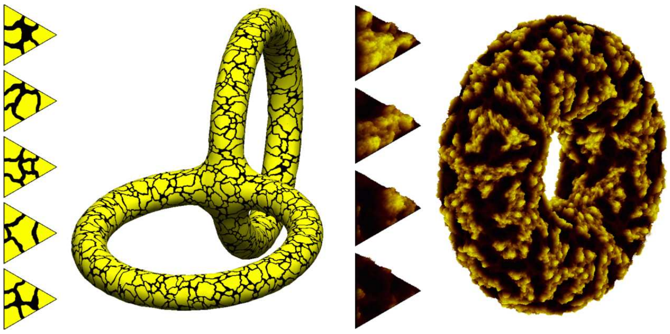

Figure 3: Two cases where unexpected path shapes appear since the lowest frequency of the texture (i.e. the grain size) is too close to the sample scale. *Left:* A naive set of texture samples designed to figure cells onto a surface. *Right:* A set of procedurally generated volumetric textures.

2.3 Assignment of Texture Samples

In order to create homogeneous textures that look like those of natural objects, several different texture samples have to be designed and non-periodically mapped onto the surface. The problem is to find how many samples are required in order to guarantee that the continuity constraints at boundaries will be respected, and to allow sufficient variations of the texture. We must also find an algorithm for assigning texture samples to the patches of the texture mesh.

The mathematical expression of this problem is not as simple as in Stam's case [17], where the mathematical theory provided a simple algorithm for achieving aperiodic mapping onto a plane, and linked the number of texture samples to the number of required boundary conditions (e.g. 16). We still have to solve a graph coloring problem (where the triangular patches represent the graph nodes and where the set of three boundaries conditions to assign them correspond to the different colors), but the graph is now a highly non-regular structure, with a varying number of neighbors per node.

A practical method for always providing a solution to continuity constraints is to use texture sample sets that include at least one texture triangle for each possible choice of three edge-constraints. More variation, or more user-control on the large-scale aspect, can be obtained by fixing a material value at each node of the texture mesh. Thus, at least one edge-constraint should be provided for each possible choice of pair of node values. A simple three step stochastic algorithm can then be used to consistently assign the triangular samples onto the surface, in a statistically non-periodic way:

1. randomly[4] choose which material value (among those used at corners of texture triangles in the sample set) is associated with each texture mesh node.

2. randomly choose which edge (among those used in the texture sample set that are compatible with the values at nodes) is associated with each geometrical edge of the texture mesh;

3. randomly assign a texture sample to each patch, among those that obey the three required boundary conditions.

The question now is: how many different texture samples do we need ? Since continuity conditions along an edge between two samples involve the gradient of the color map, the edges used in step 2 must be seen as *oriented edges*: they usually yield different boundary conditions on their two sides (to be convinced, note that a textured triangle does not smoothly weld with its mirror image, except in the special case where the gradient of the image is locally perpendicular to the common edge everywhere along it). Suppose now that we have mapped a single oriented edge e all over the texture mesh. Let us denote by E and $\bar{E}$ the different boundary conditions that the texture samples should fit on both sides of e (see Figure 4). Then, at least four texture triangles respectively obeying the conditions (E,E,E), $(E,E,\bar{E})$, $(E,\bar{E},\bar{E})$, or $(\bar{E},\bar{E},\bar{E})$ must be provided. The other possible values for the boundaries conditions (such as $(E,\bar{E},E)$ for instance) will be met by a rotated instance of one of these triangles.

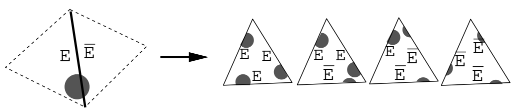

Figure 4: An oriented edge, and the set of four texture samples that need to be created to fit the different boundary conditions it produces.

In the general case of n different boundary conditions (i.e. two times the number of oriented edges), the number of texture triangles needed will be $n + n(n-1) + n(n-1)(n-2)/3$, in which the first term corresponds to the condition where the same edge is used three times, the second one to conditions with two different edge values, and the third one to solutions with three different edge values. For instance, 24 texture triangles will be needed if 2 oriented-edges are used instead of 1 (i.e. 4 boundary conditions instead of E and $\bar{E}$). Since the number of triangles required increases as a power of 3, our algorithm is not convenient for a larger number of edges. Note that if several values at nodes are used in order to constrain possible edge values, the combination of possible triangles is far smaller since edge choices have to be compatible.

In our examples, we use the minimum number of degrees of freedom, with correct results: A single kind of edge (thus two boundary conditions) is used for Figures 10,12,14. We even used the special case of symmetry mentioned above to avoid doubling the boundary condition per edge type in Figures 3(left) and 9. Noticeable repetitivity may happen with some kinds of pattern when using a small number of edge conditions. Our solution consists of providing several completely different texture samples that fit the same boundary conditions. For instance, in Figure 3 left, five texture samples fitting a single symmetric edge constraint are used. The generation of several samples fitting the same boundary conditions can be easily done using the automatic texture synthesis techniques that will be presented in Section 4. Figure 9 illustrates the use of 2 different edge conditions (using symmetry, in order to get only 2 boundary conditions, thus 4 possible triangles). An example where node values are used is presented in Figure 5, with 2 possible values (forest and ground), and symmetric boundary conditions (thus still only 4 triangles to define).

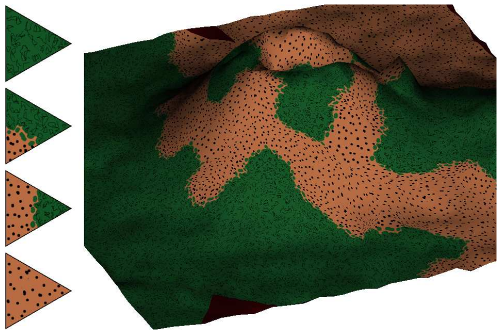

Figure 5: Mountain covered by forest. The location of forest and ground material is controlled by the values at the nodes of the texture mesh. These values are 'painted' by the user with their probability attribute (intensity of presence).

3 Mapping

In this section, we assume that we have a set of texture samples obeying adequate boundary conditions, and we describe our method for mapping them on a surface at a user controlled scale.

As suggested above, our solution for providing such control is to map texture samples onto a specifically defined *texture mesh* that tiles the surface, instead of mapping them onto the triangles that describe the object's geometry. This brings several advantages. Firstly, the texture scale becomes completely independent from the object's geometry, which is a very useful property in practice. Secondly, we can compute a high quality mesh in terms of the angular properties of the triangular patches. This will allow the mapping of equilateral texture samples without generating too large texture distortions, whatever the quality of the geometric mesh. Lastly, using a texture mesh that would be too coarse to adequately describe the object geometry is not a problem; the initial geometric mesh will still be rendered. The texture mesh just serves as a set of local parameterizations providing an image identifier and adequate texture coordinates for the geometric mesh vertices.

3.1 Overview of the texture mapping algorithm

The user first chooses the set of texture samples he wants to use, with the associated set of possible boundary conditions. He also indicates at which scale texture should be mapped by specifying

[4]This choice can be totally random to provide more variation, totally user defined, or defined with probabilities.

the desired density of texture control points (i.e., points that will be the vertices of the texture mesh) on the object's surface. Then, texture mapping is performed in the following four steps:

1. We randomly generate texture control points of the desired density on the object's surface, let them tune their relative position by simulating a repulsive force, and compute an associated high quality triangular mesh. The code we use for doing this is courtesy of Greg Turk, who uses the same process in his re-tiling algorithm [20].

2. We tile the surface with this mesh, i.e.:
   - we use surface geodesics to compute curved versions of the texture mesh edges;
   - we compute the set of geometric triangles covered by each of the resulting texture patches;
   - we compute the u, v coordinates of each geometric vertex with respect to the texture patch to which it belongs.

3. We use the algorithm described in Section 2.3 to consistently assign a specific texture sample to each patch of the texture mesh.

4. We render the object's geometry using the local u, v coordinates of the mesh vertices to map the texture samples.

A possible solution for implementing step 2 would be to adapt the set of methods introduced in [10]. In our current implementation, we rather compute geodesic curves using a standard length minimization process along a polygonal line, which is constrained to move onto the geometric mesh (the line is made of segments whose ends lie on the mesh edges). Then, we have developed a specific method, described below, for assigning u, v local coordinates to mesh vertices that lie on a texture patch without producing excessively large texture distortions. Alternative (and possibly better) solutions for implementing this part of the process can be found in [10, 4, 11].

3.2 Computing texture coordinates for mesh points

The texture mesh may have been designed at either a smaller or a larger resolution than the geometric mesh that describes the object. In the latter case, the local part of the surface that falls into a patch of the texture mesh (i.e., between three connected geodesics) may be highly curved. Computing u, v coordinates for mesh points included in this region must be done while trying to avoid texture distortions. Attention must also be paid to computing coordinates that exactly map the edges of the texture sample onto the geodesic edges of the patch, in order to avoid introducing discontinuities in the large scale texture at the junction between patches. Our solution is as follows:

To get rid of the border problem, we split the geometric triangles that are crossed by a geodesic, in order to be able to specify the exact texture coordinates along the texture patch edges[5].

The problem of computing a good u, v mapping inside each of the texture patches still remains. Since the problem is local, we have developed a simple solution that does not requires an optimization step. The basic idea is to use the three geodesic distances between a mesh vertex and the three edges of the texture patch to estimate the 'barycentric' coordinates of this vertex within the texture patch. Then, conversion into texture coordinates is immediate by analogy with the planar case. The algorithm develops as follows.

For each of the three edges of a texture patch:

1. Use a front propagation paradigm for computing the geodesic distances to the mesh vertices (see Figure 6):

   The front is implemented using a heap that stores the triangles whose three vertices are already provided with a distance value. The heap is initialized by the triangles that lie along the texture patch edge. Each front propagation step consists of taking the most reliable triangle within in the heap, i.e., the triangle which is closest[6] to this curve. The gradient of the distance within this triangle is computed. Then, for each neighboring triangle for which a vertex distance is still missing, we fold the distance gradient onto the plane of this triangle, and use the two already known values plus the gradient for evaluating the missing distance. This neighboring triangle is then inserted into the heap.

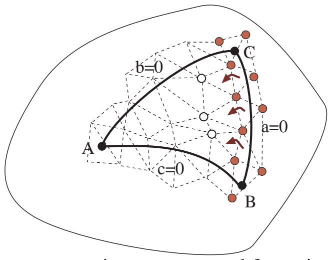

Figure 6: Front propagation process used for estimating the 'barycentric' coordinates of mesh vertices with respect to a texture patch. Distance values at pink points are already computed. The three next points for which the distance will be calculated are marked in white.

   In practice, there may be different ways of propagating distance to a given triangle, coming from several of its already computed neighbors. So we add a quality criterion to the distance value stored in the heap, and we modify a value each time we are sure quality will improve. The best quality is obtained when an vertex to be estimated falls between the two 'gradient lines' passing through the two known vertices. The estimation is less sure when it falls outside this band. The estimation is worst when the gradient is back-propagated, i.e., when the computed distance is smaller than the two known ones (then the result should only be used when no better evaluation is available).

2. Normalize all the distance values by dividing them by the value at the patch vertex that is opposite that edge.

Each vertex of the geometric mesh now stores three numbers $a, b, c \in [0, 1]$. To convert them into barycentric coordinates with respect to the three vertices of the texture patch, we divide them by their sum, so that $a + b + c = 1$. Lastly, we convert barycentric coordinates into texture coordinates with respect to the texture patch (the three corners of the image should map to $(0,0)$, $(\frac{1}{2}, \frac{\sqrt{3}}{2})$, $(1,0)$). The resulting mapping yields good results with only small texture distortions, as shown in Figure 7. However we have to keep in mind that avoiding excessive distortions is only possible 'locally': the surface's radius of curvature should not be too small relative to the patch size.

Figure 8 illustrates the control provided by texture mapping using a texture mesh: the scale of the texture can be increased while leaving the geometry of an object unchanged.

[5] The alternative solution that consists of keeping the triangles unsplit, computing a different texturing process (with half-transparent textures) for each texture patch that the triangle intersects does not work well: it does not provide enough control on the u, v coordinates near the edges of a texture patch, yielding texture discontinuities at this location.

[6] considering the maximum of the three distance values at vertices.

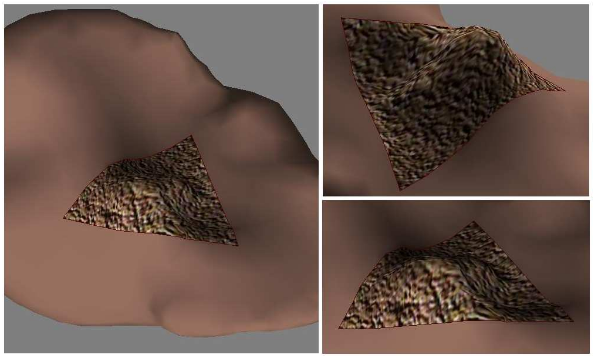

Figure 7: A texture patch mapped onto a curved region of a geometric model (views from three different viewpoints): texture distortions remain reasonable.

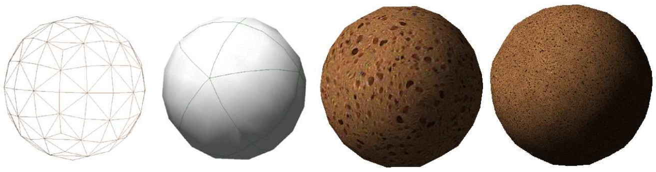

Figure 8: From left to right: the geometric mesh, which is rendered; the texture mesh, used for tuning the scale of the texture with respect to the object's geometry; the resulting textured sphere; the sphere with a finer scale texture.

4 Texture Samples Generation

4.1 Editing pictures or drawings

A first method for generating adequate sets of texture samples is direct editing, under a 2D paint system, of pictures or drawings. An example of hand-drawn texturing of a surface is depicted in Figure 9.

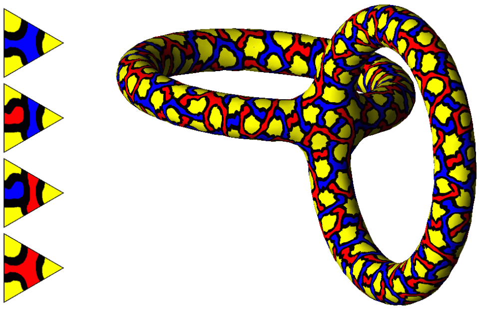

Figure 9: Texture samples drawn by hand, and the resulting image. Two different edge conditions (red, blue), both symmetric, are used.

Although it takes a certain amount of user-time, it is possible to edit real images in order to give them the required boundary conditions. The technique, that consists of copying and smartly pasting rectangular regions along edges and then eliminating texture discontinuities inside the sample, is almost the same as for square images. A single self-cyclical texture triangle, corresponding to a single symmetric edge, can be used. A more complex example of picture editing, where four different texture samples have been created for fitting the constraints associated with a single non-symmetric oriented edge, is depicted on Figure 10. The reference rectangular region figuring the oriented edge has been rotated by 180° or not when copied on the image borders, depending on which of the boundary conditions (E,E,E), $(E,E,\bar{E})$, $(E,\bar{E},\bar{E})$, or $(\bar{E},\bar{E},\bar{E})$ each of the four samples corresponds to.

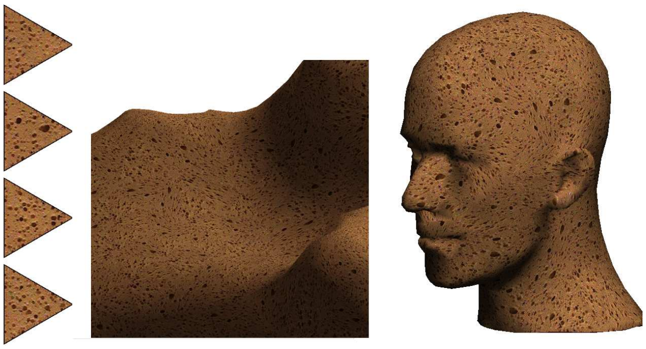

Figure 10: A set of texture samples designed by interactively editing an image of a sponge (left), and the resulting textured surface on a terrain and on a face.

We now describe two procedural synthesis techniques that can be used for automatically generating parameterized sets of texture samples thus saving user's time.

4.2 Extending Worley's algorithm

Worley's method [22] is an efficient approach for creating textures depicting small non-periodic cellular patterns such as rocks, scales, or living tissues. When applied in 2D, Worley's method basically consists of computing Voronoï diagrams of noise points randomly distributed on a plane. A square grid is used to accelerate the computation: a noise point is randomly chosen in each cell. Then, determining in which of the Voronoï region each pixel falls can be done efficiently, by only checking the noise points in the 9 closest cells. The portion of the plane covered by the noise points must be slightly larger than the square region to texture, in order to have nice Voronoï regions cross the edges. Worley's method combines the distances from a pixel point to the N nearest noise points to compute the texture value at the pixel (generally, $N \in [1..4]$). We only describe here how to deal with the nearest noise point; the other distance computations are adapted the same way.

Adapting Worley's technique for generating the texture on an equilateral triangle is easy: we just have to tile this triangle with a slightly larger triangular grid as depicted in Figure 11. A noise point is randomly chosen in each of the small triangles, and Voronoï diagrams are computed as in the standard case.

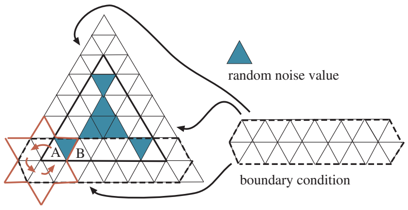

Figure 11: Triangular grid used for extending Worley's algorithm: Noise values that represent a given boundary condition along an edge are surrounded by a dashed line. Values which are the rotated copies of each other in order to maintain continuity constraints at a texture sample corner are indicated in pink.

For our pattern-based texturing application, we need to generate Worley triangles that obey specified boundary conditions along edges. More precisely, we want to be able to control the texture in the neighborhood of each triangle edge in order to ensure continuity between samples. Our solution is similar to the approach suggested in [17], and also to what we have described above for real image editing: we first generate the 'rectangular regions' representing each oriented edge. This rectangular region is implemented here by a two row grid storing the noise points that can influence the border neighborhood, on both parts of the boundary. Once again, we derive the complementary condition by rotating the rectangle that defines a boundary condition by 180°. Then, we duplicate the noise values into the adequate part of the grid, for each texture triangle that must obey this specific condition. Finally, the noise values of the inner unconstrained region of each triangle are chosen at random.

Particular attention must be paid to the achievement of texture continuity near texture sample corners. Two edges meet there, and the noise values they give to the texture triangle should thus be the same. As explained in Section 2.2, the solution is to define a texture with a given value and a zero gradient at these vertices. In terms of the algorithm above, this can be done by copying a rotated version of a given noise value into all the grid cells that surround a given vertex. In a naive implementation, this operation has to be done within two ranks of cells surrounding the corner (figured in pink), since the noise values in the second rank of cells may influence the texture gradient there. A trick for eliminating some possible visual artifacts due to symmetry is to restrict the range of possible noise point values in cells A and B (see Figure 11) so that the Voronoï region generated by the point in the blue cell between them will not intersect the edge of the texture sample. Then, this noise point value will have no influence on the texture at the vertex, enabling a (constrained) random choice for this cell. In Figure 11, all the noise points in the blue triangles can be chosen at random without spoiling boundary conditions.

Two examples of texture sample sets, and the resulting images they produce when mapped on a surface, are depicted in Figure 12.

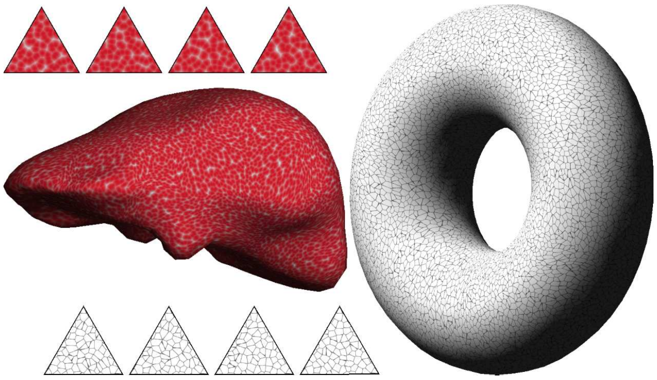

Figure 12: Sets of texture samples generated by our extension of Worley's synthesis technique, and the images produced by non-periodically mapping them onto a surface. *Left:* A human liver. *Right:* A china torus.

4.3 Extending Perlin's synthesis technique

Fractal noise based on Perlin's basis function [16] is a self-similar stochastic noise that has become a standard for generating objects that look like wood, marble, or the surface aspect of rock. One of its main features is to ensure continuity of both noise values and gradient at any point of an image (or of a volume, when the method is used in 3D, e.g. to figure smoke).

To adapt it to our texturing methodology, we first have to be able to generate the basis function on 2D equilateral triangles. Since Perlin's standard model is defined on a quadrangular grid, we modify the algorithm in the following way:

1. We first generate a pseudo-periodic noise function on a regular grid that tiles the equilateral triangle into sub-triangles (see Figure 13). This requires two steps:

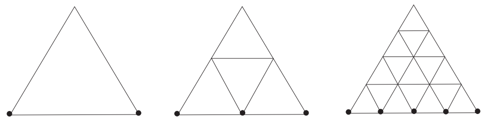

Figure 13: Triangular grids used for generating the noise function at different scales. The noise values that need to be fixed for ensuring boundary conditions are shown in bold.

   - as for textures based on Perlin's technique, we randomly associate a plane to each grid node, defined by its elevation above the node and by its normal vector;
   - we define the noise at any point inside the triangle as the barycentric interpolation of the distances to the three planes that are associated to the vertices of the small triangular cell where the point lies.

2. We define the final noise value at a pixel as the sum of instances of the pseudo-periodic noise function defined above, applied at different scales thanks to recursive subdivision of the triangular mesh, with a scaling factor that is the equal to the scale. This gives the fractal aspect to the noise.

3. The value obtained is used as usual as a seed or a perturbation to define the texture value at the pixel.

Modifying this algorithm to ensure a set of given boundary conditions around each triangle is easy: we just have to model a boundary condition as the set of noise values that control the texture values and derivatives along an edge. These values are those indicated in bold in Figure 13. We then duplicate these boundary values onto the adequate side of the grid, for all the texture samples that have to obey this specific boundary condition. Ensuring continuity at the three corners of texture sample is done as usual by giving the same mean value and zero gradient to the texture there, i.e., using specific noise values at each vertex. As in original Perlin's algorithm, all random values are precomputed in a (small) hash table, and no copy is done: instead we compute at any location which index in the random table should be accessed. To define the same control value at the vertices of two edges, one just has to ensure that the same index is produced, and rotate the built random normal vector on the fly (because adjacent triangles do not use the same frame).

Three examples of texture sample sets and the resulting images they produce are depicted in Figure 14. Note that computing procedural textures on triangular domains while ensuring continuity constraints can also be used on the fly for other kinds of applications. For instance, we have used an algorithm close from above for generating at rendering time a displacement texture modeling the crust of an evolving lava-flow without having to parameterize the flow surface [18].

5 Conclusions & Future Work

We have presented a general framework, based on triangular texture tiles, for texturing an arbitrary surface at low rendering time and memory cost. Our method has been designed for covering the surface with an homogeneous non-periodic texture such as those that can be found on many natural objects. The main features of our approach are the following: the texture can be applied at any scale with respect to the object geometry, and whatever the quality of the geometric mesh; no singularity is generated whatever the surface topology, and distortions are minimized. Moreover, using the method demands little user work, by avoiding redundancies. We describe how to use hand-drawing and real images, and we provide two automatic texture synthesis methods that adapt Perlin's and Worley's algorithms to a triangular domain. Adapting other

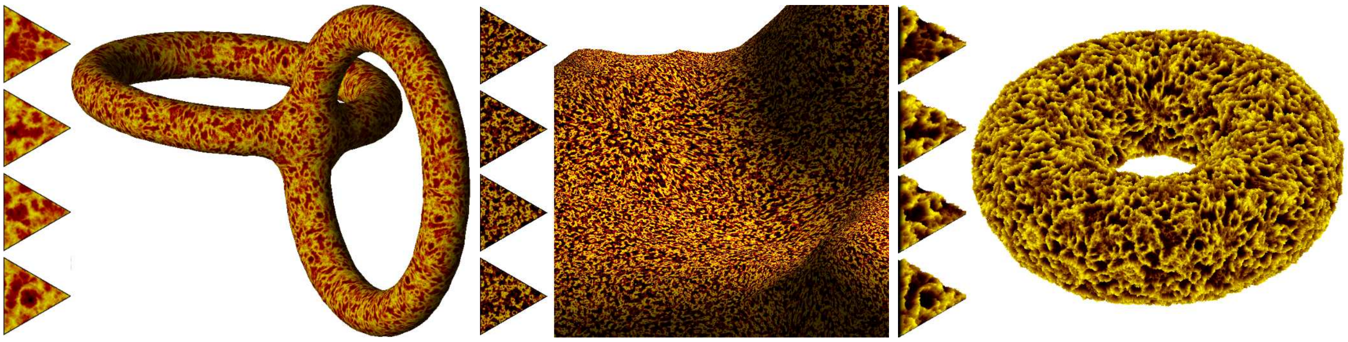

Figure 14: Three sets of texture samples generated by our extension of Perlin's synthesis technique, and the images produced by mapping them onto a surface. In the image on the right, Perlin's texture is used as displacement map, encoded with an OpenGL implementation of volumetric textures [13].

texture synthesis techniques in the same way, such as those based on a pyramidal analysis of real textures [9, 2] should be easy, as filtering kernels and pyramidal constructions can directly be 'translated' to triangular grids. Lastly, our framework is compatible with real-time applications, since the result of the texturing can be represented using classical geometric object formats.

Maintaining C^1 continuity of the texture across the surface has been achieved by building texture samples which obey specific boundary conditions, and whose border is mapped exactly onto the geodesic curves that tile the surface onto texture patches. In the current implementation, we ensure the second constraint by splitting the geometric triangles that intersect a geodesic, in order to have enough points for locally controlling the mapping. This is not a problem when large scale texture triangles are used since this splitting process will not greatly increase the number of triangles to render. However, in the case of almost flat surfaces built with a few large triangles, and that need being textured at a small scale, the tiling of the geometry may yield a high increase of rendering time. A solution would be to use a level of details approach for defining texture samples: large samples containing very small patterns would be recursively created by assembling smaller ones. Then, the former would be mapped onto a larger scale texture mesh defined on the surface, which would not split the geometry excessively. Conversely, this would provide a solution to correctly map geometric areas that are smaller than the initial sample size (such as handles or legs), without having to decrease the sample size elsewhere on the surface.

Our future work also includes the introduction of user-controlled pattern anisotropy, which is an important feature of many natural textures. Using anisotropic patterns directly in the scheme we have presented would not be a good idea, since there is no way to ensure continuity of characteristic directions between texture patches. Our idea is rather to rely on the texture mesh itself, to model the kind of anisotropy that corresponds to the stretching of an isotropic pattern: A user-defined tensor field would be used for locally specifying the amount and direction of desired anisotropy all over the surface. This field would be used to influence the generation of texture mesh control points and produce an anisotropic distribution (in the same spirit, Turk suggests in [20] to use the surface curvature). Finally, mapping the usual isotropic texture samples onto this texture mesh would result into adequately deformed patterns, with continuous directional features.

Acknowledgments

We wish to thank Greg Turk for providing his retiling code and Alexandre Meyer for adapting his volumetric textures to our framework. Thanks to David Bourguignon for creating the sponge texture samples, to Brian Wyvill and Georges Drettakis for rereading this paper, and to Jean-Dominique Gascuel for his help in video editing.

References

[1] Chakib Bennis, Jean-Marc Vézien, Gérard Iglésias, and André Gagalowicz. Piecewise surface flattening for non-distorted texture mapping. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 237–246, July 1991.

[2] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, pages 361–368. ACM SIGGRAPH, Addison Wesley, August 1997.

[3] H. S. M. Coxeter, M. Emmer, R. Penrose, and M. L. Teuber, editors. *M. C. Escher: Art and Science*. North Holland, Amsterdam, 1986.

[4] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 173–182. ACM SIGGRAPH, Addison Wesley, August 1995.

[5] M.c. Escher biography and annotated gallery. http://www.erols.com/ziring/escher_gal.htm.

[6] Andrews Glassner. Andrew Glassner's notebook: Penrose tiling. *IEEE Computer Graphics and Applications*, 18(4):78–86, July/August 1998.

[7] B. Grünbaum and G.C. Shephard, editors. *Tilings and Patterns*. Freeman, New York, 1986.

[8] Pat Hanrahan and Paul E. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH 90 Proceedings)*, volume 24, pages 215–223, August 1990.

[9] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 229–238. ACM SIGGRAPH, Addison Wesley, August 1995.

[10] Aaron Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution adaptive parametrization of surfaces. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, pages 95–104. ACM SIGGRAPH, Addison Wesley, July 1998.

[11] Bruno Lévy and Jean-Laurent Mallet. Non-distorted texture mapping for sheared triangulated meshes. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, pages 343–352. ACM SIGGRAPH, Addison Wesley, July 1998.

[12] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH 93 Proceedings)*, volume 27, pages 27–34, August 1993.

[13] Alexandre Meyer and Fabrice Neyret. Interactive volumetric textures. In G. Drettakis and N. Max, editors, *Rendering Techniques'98, Eurographics Rendering Workshop*, pages 157–168. Eurographics, Springer Wein, July 1998.

[14] Hans Køhling Pedersen. Decorating implicit surfaces. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 291–300. ACM SIGGRAPH, Addison Wesley, August 1995.

[15] Hans Køhling Pedersen. A framework for interactive texturing operations on curved surfaces. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 295–302. ACM SIGGRAPH, Addison Wesley, August 1996.

[16] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH 85 Proceedings)*, volume 19, pages 287–296, July 1985.

[17] Jos Stam. Aperiodic texture mapping. Technical Report R046, European Research Consortium for Informatics and Mathematics (ERCIM), January 1997. http://www.ercim.org/publication/technical_reports/046-abstract.html.

[18] Dan Stora, Pierrer-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret, and Jean-Dominique Gascuel. Animating lava flows. In *Graphics Interface 99*, June 1999.

[19] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 289–298, July 1991.

[20] Greg Turk. Re-tiling polygonal surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH 92 Proceedings)*, volume 26, pages 55–64, July 1992.

[21] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 299–308, July 1991.

[22] Steven P. Worley. A cellular texturing basis function. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 291–294. ACM SIGGRAPH, Addison Wesley, August 1996.