EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

Volume 21 (2002), Number 3

Texture Particles

J.-M. Dischler[†], K. Maritaud[‡], B. Lévy[*] and D. Ghazanfarpour[‡]

[†] LSIIT, University of Strasbourg, France
[‡] MSI Lab. (ENSIL), University of Limoges, France
[*] LORIA, ISA Project, INRIA Lorraine, France

Abstract

This paper presents an analytical extension of texture synthesis techniques based on the distribution of elementary texture components. Our approach is similar to the bombing, cellular, macrostructured and lapped textures techniques, but provides the user with more control on both the texture analysis and synthesis phases. Therefore, high quality results can be obtained for a large number of structured or stochastic textures (bricks, marble, lawn, etc.). The analysis consists in decomposing textures into elementary components – that we call "texture particles" – and for which we analyze their specific spatial arrangements. The synthesis then consists in recomposing similar textures directly on arbitrary surfaces by taking into account the previously computed arrangements, extended to 3D surfaces. Compared to "pixel-based" analysis and synthesis methods, which have been recently generalized to arbitrary surfaces, our approach has three major advantages: (1) it is fast, which allows the user to interactively control the synthesis process. This further allows us to propose a large number of tools, granting a high degree of artistic freedom to the user. (2) It avoids the visual deterioration of the texture components by preserving their shapes as well as their spatial arrangements. (3) The texture particles can be not only images, but also 3D geometric elements, which extends significantly the domain of application.

Keywords: texture analysis and synthesis, particles, texture mapping.

1. Introduction

Almost all real-time interactive systems (virtual reality, video games, etc.), as well as non real-time systems (lighting simulation, motion-picture effects, etc.) apply textures to surfaces in order to improve the rendering quality. Therefore, texturing plays a key role in computer graphics. However, in spite of decades of research activities in this area, decorating arbitrary surfaces in a controlled and user-friendly way, with different types of textures, remains a challenging problem. Two main approaches have been investigated in the past:

- Automatic synthesis techniques (for instance based on stochastic procedural models [7], on sample-image analysis [1, 2, 3, 8, 11, 24], on physical, chemical or biological rules [6, 22], etc.);
- Interactive texture placement / mapping [12, 14, 16, 17, 18].

With pure texture synthesis techniques, the user is freed from most painstaking manual manipulations. However, the results are generally relatively long to compute and, due to "full" automatism, hard to control very precisely. Conversely, interactive 3D texture painting systems grant users nearly unlimited freedom, while computations are much faster (interactive rates). Nevertheless, in spite of a wide collection of "paint-brushes" and interactive tools, a lot of work must still be done by hand. We believe that a "good" approach for decorating surfaces should unify all the previous advantages. That is, it should be mostly automatic and fast (i.e. perform at interactive rates), while leaving as much "artistic" freedom as possible to the user. This paper proposes a completely new, simple, fast and controllable texture analysis and synthesis technique matching well the requirements mentioned above. Unlike most recently introduced texture analysis and synthesis techniques, this one does not describe textures in the form of hierarchical sets of pixels related by a Markovian process (we may call these methods "pixel-based"). Instead, it is inspired by some well-known texture synthesis methods, based on the more or less random distribution of "large-scale" (as opposed to "pixel-scale") visual texture components, such as for the bombing textures [21], cellular textures [10], macrostructured textures [4] and lapped textures [19]. Figure 1 illustrates the basics of the method.

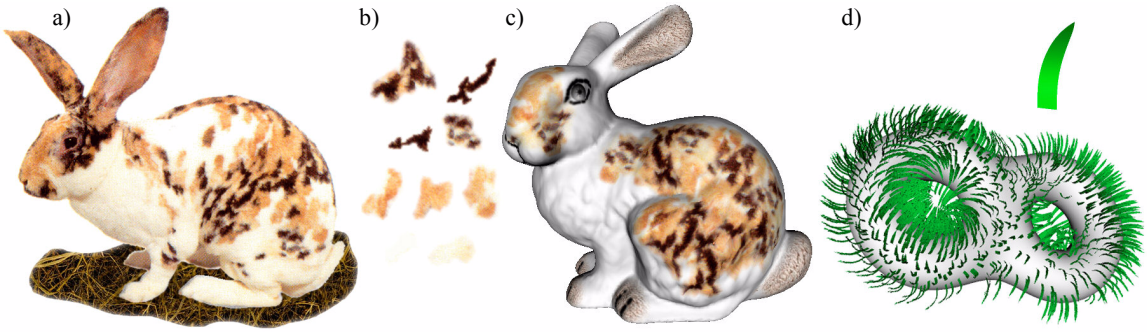

Figure 1: *Our texture analysis and synthesis technique is based on the distribution of texture particles (visual texture components) respecting specific spatial arrangements. We use a sample texture (e.g. a photo) as input (a) that we decompose into texture particles (b) in order to reproduce a similar texture directly on arbitrary surfaces (c) at nearly interactive rates (the update time is about 3 seconds for the bunny) using previously computed co-occurrences. The sample textures can also be explicit 3D models (d). Note: the eyes, ears and feet of the bunny were painted by hand (the texture of the body was synthesized automatically).*

As shown further, our main contribution is a new way of distributing these elements onto the object to be textured, so that their relative distribution mimics the initial image from which they were extracted. The synthesis process does not only work for various structured patterns, but is also fast and controllable. Figure 1 gives a small overview of what the user can expect from the method. A secondary, but yet interesting, feature of fast texture synthesis techniques (based here on a small set of texture elements) is "texture compression". Indeed, it is then possible to texture complex objects with just a few bitmap and additional data.

1.1. Related works

Bombing textures [21] consists in randomly distributing texture components called "bombs" on a plane, which permits to define various types of random patterns. The cellular texturing approach [10] is another highly interesting technique for distributing texture components (called "cells", in this case) on arbitrary surfaces using a relaxation algorithm. The relaxation makes the distribution converge on a stable solution matching user-specified properties, such as orientation, size or distance to neighboring cells. Macrostructured textures [4] likewise consist in distributing geometric texture components (in this case extracted from photos), on surfaces or inside volumes. These components are called "macrostructures". From a certain point of view, lapped textures [19] may also be considered as a kind of cellular approach since it also consists in distributing texture components (large texture pieces called patches in this case) directly on surfaces, in a way similar to cells, i.e. by considering distances to neighboring patches.

All of these techniques perform well, and allow users to control some texture properties. However, all have in common the fact that the distribution properties must be provided manually by the user, which means integrating it directly and procedurally into the synthesis procedure. For instance, the sample images are not used to automatically extract some information about spatial arrangements (e.g. Dischler and Ghazanfarpour [4] only use the sample images to extract the shapes and colors of the macrostructures, not their spatial arrangements). In fact, all of these texturing methods, based on particles, only consider low order statistical arrangements (that is the proximity of neighboring particles), but fail to consider more structured / specific spatial arrangements.

1.2. Method principles

This paper extends the previously mentioned texture synthesis techniques by introducing an analytical process; i.e. a process based on the analysis of sample images. More specifically, we extend the method of the lapped textures [19] (and in some way, the chaos mosaic [26] and patch-based textures [15]) by "splitting" (segmenting) the textures into more elementary features that we call "texture particles", instead of large texture pieces regrouping several elements of the texture. Texture particles are elementary visual texture components such as for example individual bricks in a brick wall.

Note that, because of the "large pieces" used especially for lapped textures, some visible discontinuities may appear, even in spite of alpha blending and particularly in the presence of low frequency features due to overlapping. By using a finer decomposition, we are in many cases able to avoid such visible discontinuities (See Figure 9 for a comparison with lapped textures in the "Results" section). We analyze the 2D spatial arrangements of the texture particles using co-occurrences. Then, the obtained 2D information is extended to the case of arbitrary surfaces, by applying a geodesic-like metrics and by defining a frame on the surface. In this

paper, we use the terminology of "texture particle", but one could also use the terminology of "bomb", "cell" or "macrostructure". We observed that our simple approach performs well with various types of textures, providing in many cases, faster and better results than with Markovian "pixel-based" methods, especially when these are generalized to arbitrary surfaces [23, 25, 27]. We usually obtain results close to "image quilting" [9], but with a higher degree of user control (beyond straight "reproduction"). In fact, in our case, as for our macrostructured technique [4] the user can precisely control, at nearly interactive rates, many visual properties (by moving / adding / suppressing features, or by changing their density, shapes or colors). Unlike "image quilting" [9] that does not seem to be very easily generalizable to arbitrary surfaces because it is based on rectangular tiling (thus requiring a usual (u,v) parameterization), our technique can also be easily generalized to arbitrary surfaces, which avoids the usual problems of texture mapping (discontinuities and/or distortions).

Moreover, our "texture particles" are not limited to texture images but can also be "geometric" textures (e.g. 3D textures based on "explicit" 3D geometric models, such as thorns or grass). Globally, our method defines a new scale between the pixel scale [23, 25, 27] and the large scale (entire texture pieces / tiles) [9, 19, 15, 26]. Each texture particle represents a texture "feature" with a specific semantics (it is a set of pixels or a geometric feature, with a given signification, such as the individual bricks in a brick wall). As in chaos mosaic methods [15, 26], the fact of using large features instead of pixels explains the high speed of the synthesis, even on arbitrary surfaces in our case, compared to other methods [23, 25, 27]. This also guarantees the preservation of the shapes of the features, which improves the visual quality and fidelity with the provided sample texture.

The rest of the paper is organized as follows. The next section describes the details of the method. The first part of that section deals with the segmentation of sample texture images (models) into particles. The second part deals with the spatial arrangement analysis and the fast synthesis in 2D. Finally, the last part concerns its extension to arbitrary surfaces, further using appropriate interactive tools allowing the user to control the synthesis process. Section 3 shows some graphical issues. A comparative study with some other existing synthesis methods is proposed. Some limitations of our method are also discussed. Finally, we conclude this paper and propose some future directions.

2. Texture analysis and synthesis

The analysis step consists in expressing the example texture $T(i,j)$, $(i,j) \in [1,N]^2$, in the form of a set of spatially organized visual components called "texture particles". For images, the particles correspond to sets of pixels, with a given signification. More formally:

$$T(i,j) = P_1(i,j) \otimes P_2(i,j) \otimes \ldots \otimes P_{n_P}(i,j)$$

where all P_k represent the particles, n_P their number and $\otimes$ an image recomposition operator. For 3D textures, the particles simply correspond to 3D models. After segmentation and classification, the particles are analyzed – especially their spatial organization – by computing co-occurrences.

2.1. Segmentation into particles and classification

The goal of segmentation is to identify zones in images that have similar visual characteristics. That is what we need as we want to identify a set of particles that are representative of each of the main elements of the texture. However, as deeply studied in the field of computer vision, most of the methods propose automatic solutions, but only for very specific cases. For example, Lefebvre and Poulain [13], focused exclusively on brick walls and wood textures. Hence, they could adapt their segmentation technique to these specific cases, in order to provide a completely automatic technique, which extracts the individual bricks. Nevertheless, in the general case, the user must perform the classification since this is an intelligent process, strongly depending on the type of application. For example, Premoze et al. [20], segment satellite images of mountain landscapes and classify features by using a *training set*. The approach requires the user to click on individual pixels that are assigned to a given class (e.g. the user indicates that the selected pixel represents snow, rock or forest). For these user-selected pixels, statistical feature vectors are computed. The global segmentation and classification is then performed by assigning all the remaining pixels to the classes with the "closest" statistics (according to a certain weight function, which is, in their case, based on a so-called *normal distribution maximum likelihood Bayes classifier* [20]). Another more "brute-force" method for segmenting textures simply consists in using edge-detecting scissors or a "lasso". For instance, this has been used for scissoring texture pieces in the case of lapped textures [19].

In our case, we want to leave as much freedom as possible to the user, but without losing the ability to deal with a large spectrum of textures. Therefore, entirely automatic segmentation methods cannot be used. Yet, we want to minimize the user's work. Hence, we implemented two approaches. The first one is a "brute-force" scissoring technique similar to the one used by the lapped textures. This technique works for all types of textures, but is inappropriate for textures characterized by a lot of small features, since it might require too much work from the user, in these cases. The second method is based on the observation that texture features are often discernable because of their color, which contrasts with the other parts. Some sophisticated color

quantizations in special color spaces might be used, but a simple RGB quantization often produces good enough results to detect these features. Additionally, we simply applied a Gaussian filter before quantization, in order to remove noise. This method for isolating texture particles has also been used in another recent paper [5].

Figure 2 illustrates the segmentation of a biscuit texture, a texture representing knots and a red and green marble texture. The images on the left are the original images, while the segmented images are on the right and some of the user-selected texture particles in the middle (we do not show *all* particles in Figure 2, to avoid overloading it). For the biscuit, we have identified four classes: a sort of light brownish "background" pattern, light brownish spots, dark brown spots and the small black holes. For the knots, we have identified two classes: the black holes (these are also a sort of "background" pattern) and the knots. For the marble, we have identified four classes: a dark red "background" pattern, light red spots, black spots and green spots. In all cases, we used both approaches for extracting the particles (sometimes scissors, sometimes the segmentation simply by clicking onto the color zones). Also note that, in all cases, we additionally use a morphological dilatation of the particles (see next section), in order to apply alpha blending. Therefore, in the middle of Figure 2, the borders of the particle are smooth.

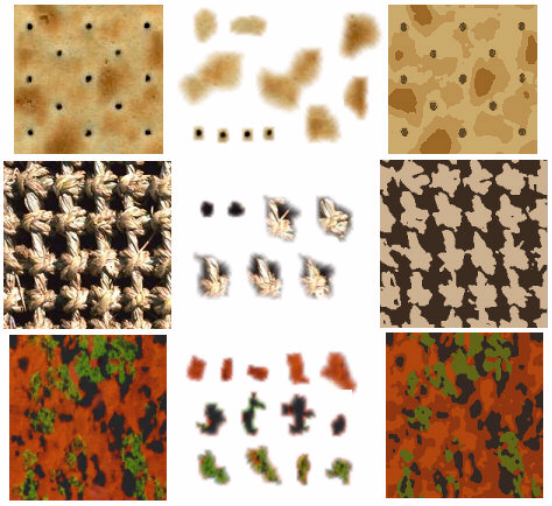

Figure 2: *Decomposing texture images into particles.*

At this point, the user's work seems more important than with completely automatic analysis and synthesis methods (it is in fact comparable to the work required by lapped textures), for which the user only supplies the input image. But, this actually minimal work has a major advantage: by identifying features, we are able to provide the user with much more tools for controlling the synthesis process.

2.2. Analyzing spatial arrangements

Once the particles have been identified, their spatial organization can be analyzed. In what follows, we will distinguish particles that are "complete" and "incomplete". Particles having too much of their pixels on the border of the image T are supposed to be incomplete (i.e. clipped by the limited texture image size). It would be more accurate to consider as incomplete a particle having at least one pixel on the border, but we experienced that if the number of pixels on the border is sufficiently low with respect to the global particle size, this has no visual consequence for the final synthesis. Inversely, by using particles that have too many pixels on the borders of the original image, visual artifacts appear. Indeed, features look like clipped by a rectangle, thus introducing discontinuities.

Let us consider a complete particle P_k. We will now analyze the positions of its neighboring particles (for the same class *only*). Therefore, we apply repetitively a morphological dilatation operation (note that a similar dilatation is used to enlarge the particles in order to apply alpha blending, as shown in the middle part of Figure 2) defined as:

$$\bigcup_{b \in B} P_{k,b} = \{p + b \mid p \in P_k, b \in B\}$$

where B denotes the structuring kernel (in our case a square of size 3x3). At each dilatation step, the size of the particle P_k "grows" by a one-pixel-thick contour.

After a certain number of iterations, P_k begins to reach and overlap the neighboring particles $P_{k'}$ (of its class). These can now be used to compute co-occurrences. We note that the number of dilatation iterations has been made proportional to the size of the particle P_k, in order to only keep direct neighbors and to avoid too important enlargements. But, inversely, dilatations are performed until at least one neighbor is reached in each quadrant. Figure 3 illustrates the obtained neighbors of two classes of particles: the small dark holes in the biscuit texture and the dark brown spots.

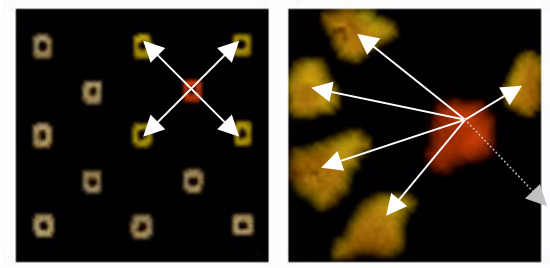

Figure 3: *Neighboring particles are detected for each class by morphological dilatation.*

Once we have detected all the closest neighbors, we can compute a list L_k of co-occurrences related to P_k, where, in our case, one co-occurrence is defined as a

triplet (q, dx, dy). q is an index value giving the quadrant in which the neighboring particle lies. (dx, dy) represent the distances between the bounding boxes of the concerned particles (see Figure 4). It is important for the synthesis process to have co-occurrences for all four quadrants; otherwise, one would not fill in "isotropically" a complete plane (or a complete 3D surface). Indeed, the particles must be propagated in all possible directions to avoid empty zones. In cases for which there is a missing particle in one quadrant, we simply use the symmetric co-occurrence (i.e. the one of the opposite quadrant). The gray arrow on the right of Figure 3 represents such a missing particle.

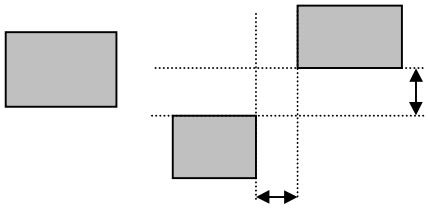

Figure 4: *Computing co-occurrences for neighboring particles by considering their bounding boxes.*

We use the distance between couples of particle bounding boxes – and not the distance between their centers, for example – to be able, during the synthesis, to take into account the possibly varying sizes of the particles, without introducing an overlapping effect. Yet, for some co-occurrences the values of dx and dy may be negative, which means that the boxes are effectively overlapping. For anisotropic particles not aligned with the axis, we rotate their boxes to best fit the particles. For each particle, we compute a list of co-occurrences, thus obtaining, for each class, a complete set of co-occurrence lists (this set must contain at least one list of co-occurrences). This set characterizes the spatial distribution of the particles of a given class. Similar sets can be computed for all other classes of particles.

There are, however, two exceptions for which we must consider alternative spatial arrangements for the classes:

- In some cases, the model image T may be too small to compute any co-occurrences for a given class, for example because it contains too few particles (only one, two or three). In such cases, we assume that the distribution is "random", and we only take into account the respective distances between the particles (how far they are from each other). In the case there is only a single particle, we use as distance the dimensions of the input image. This represents an acceptable compromise, since, except in some rare cases, the texture sample is also too small for observers to notice particular spatial arrangements.
- There exists for most textures (not necessarily all) one specific class, which we called the "background" pattern in Figure 2. The particles of this specific class must be handled differently, since they are used to cover densely the whole plane. To cover an entire surface, we do not need to take into account any spatial arrangement. Therefore, we simply use a technique similar to lapped textures that ensures the whole surface is covered without holes.

Once all sets of co-occurrence lists have been computed for every class of structures, it is possible to synthesize resembling textures directly on a plane, by using a "seeding" procedure, which propagates particles by starting from one germinal particle.

In practice, the user starts by selecting a position where he puts the first particle of the first class. Then, the system chooses randomly one list of co-occurrences out of the concerned set. New randomly chosen particles are put on the respective locations given by the co-occurrences, if there is not already a particle close to this location. The proximity of particles (to avoid "too much" overlapping) is checked by using the bounding boxes of the particles. If these are too much overlapping, then no new particle is created, since we assume that there is already one on the concerned location. The fact that we check whether there are already particles is important to make the system stop once the entire surface has been covered. Once one class has been processed, we start again with the next class. Figure 5 illustrates the steps of this procedure in the case of the textures of Figure 2. Each texture is constructed progressively, by adding more and more texture particles, according to the spatial arrangements, and by processing each class sequentially. We note that in all cases the first class corresponds to the background (the first image on the left), which has been processed using a technique similar to lapped textures. Each image shows the result of the processing/"seeding" of the particles of one class.

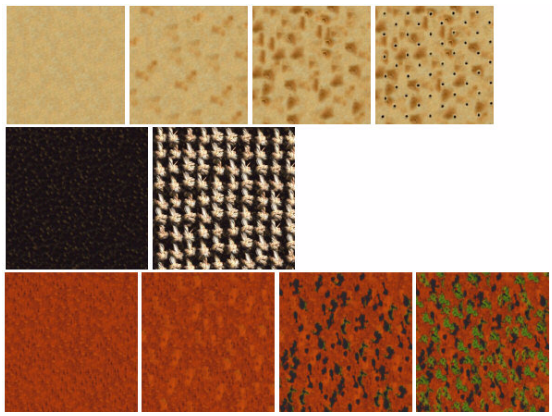

Figure 5: *Propagating the particles according to the co-occurrences in order to reconstruct step-by-step (images from left to right) a similar texture.*

We note that we do not consider correlations among the positions of particles of different classes. Neither do we consider correlations between particle shapes and positions, which is the case for example with mosaic-like patterns. This represents currently a limitation of the method, which will be further discussed in the "Results" section. Due to blending, the order of the classes in which they are processed is naturally crucial. For example, for the biscuit, it is important to process the class corresponding to the dark holes at last; otherwise, some of them might be hidden by particles of the next classes.

2.3. Generalization to arbitrary surfaces

All co-occurrences have been computed on a plane using Euclidean distances and a 2D frame. On a 3D surface, we also need a local frame to be able to transpose the computed co-occurrences. In practice, we use a tangential vector field defined interactively by the user. It determines, as for the lapped textures, the orientation of the texture. Next, we need a distance measurement to estimate the distance separating two points on the 3D surface. We assume that the surface is locally flat enough to compute a distance directly using an intersection with a plane. The distance $dist(P_1,P_2)$ between two points P_1 and P_2 on a surface is estimated by computing the intersection of the plane defined by the vectors (P_1P_2, N_1) with the surface. N_1 is the normal at P_1 and the plane passes through P_1. As one can see, if the surface is not very smooth, the local normals N_1 and N_2 (which is the normal at P_2) may be quite different from each other and from larger-range "gradients" computed around P_1 and P_2. This might result in a computed distance bigger than the real shortest distance. Therefore, in the case of rough surfaces, we recommend to keep the minimum between $dist(P_1,P_2)$ and $dist(P_2,P_1)$.

If the surface is made of polygons and, if some topological information has been pre-computed (for each edge, we store the indices of the corresponding two faces), then this intersection can be computed very quickly by sequentially following the polygon edges until the face containing P_2 is reached. The final distance $dist(P_1,P_2)$ corresponds to the sum of all computed segment lengths using an Euclidean distance. The method works very well, no matter how irregular the mesh is.

With this "pseudo-distance" measurement and with a local frame on the surface, we are now able to distribute the particles with respect to the previously computed sets of co-occurrences. As for the 2D case, the seed step is initialized by selecting a random (or user-defined) point on the surface, on which one seed particle is "planted". New surrounding particles are then added by taking into account the previously computed co-occurrences. However, as for the 2D case, before adding any new particle, a proximity test is made in order to check whether a particle is already in the neighborhood or not. To do so, we store, for each face, a list of the particles located on this face. With the pre-computed topological information, we get rapidly all the particles close to a given face (by visiting the adjacent faces, and their adjacent faces, and so on).

The algorithm stops when no more new particles can be added. It is easy to understand that this algorithm necessarily stops, at the latest when the surface is densely covered by particles (no more can be added because of the close proximity of others), or when the border of the surface is reached.

3. Results

Figure 6 (see color section) illustrates a comparison of our method with some other texture analysis and synthesis techniques. From left to right we show: the model, Wei and Levoy synthesis, Ashikhmin synthesis, lapped textures (applied to a plane), image quilting and our method. In most cases, our method provides results roughly comparable, from a spatial point of view, to image quilting, but without its perfect "regularity". Indeed, compared to image quilting, our method adds some randomness because of the fact that the particles do not all have the same size. Otherwise, the results would be quite similar. Our method also introduces a slight blurring effect due to alpha blending. However, the results are better than with Wei and Levoy, Ashikhmin and lapped textures, since we take into account the specific alignments. Indeed, while these methods can provide good results with many non-structured textures, they were not designed for highly-structured textured and therefore fail for such cases. For the examples of Figure 6, the 2D synthesis is extremely fast with our method (about 0.1 seconds), thanks to the hardware-accelerated alpha blending of textured polygons for rendering the particles. The seeding process itself (for the particle classes) is fast and requires only a fraction of second on a plane (the given timing does not include the synthesis of the background, which is done only once for a plane, and which is never modified nor controlled by the user).

Figure 7 (see color section) illustrates some extra textures (not highly structured, this time) that we compared to Wei and Levoy, and Ashikhmin. The top row shows, from left to right, the model, Wei and Levoy and Ashikhmin synthesis. The bottom row shows the reproduction using our method on the left. The right-hand image further illustrates the fact that the user can control some visual aspects, and even make them vary spatially: such as altering the probability of one specific particle according to the location, density of particles, size, orientation, etc. This makes the textures look somewhat different, but yet in a controlled way.

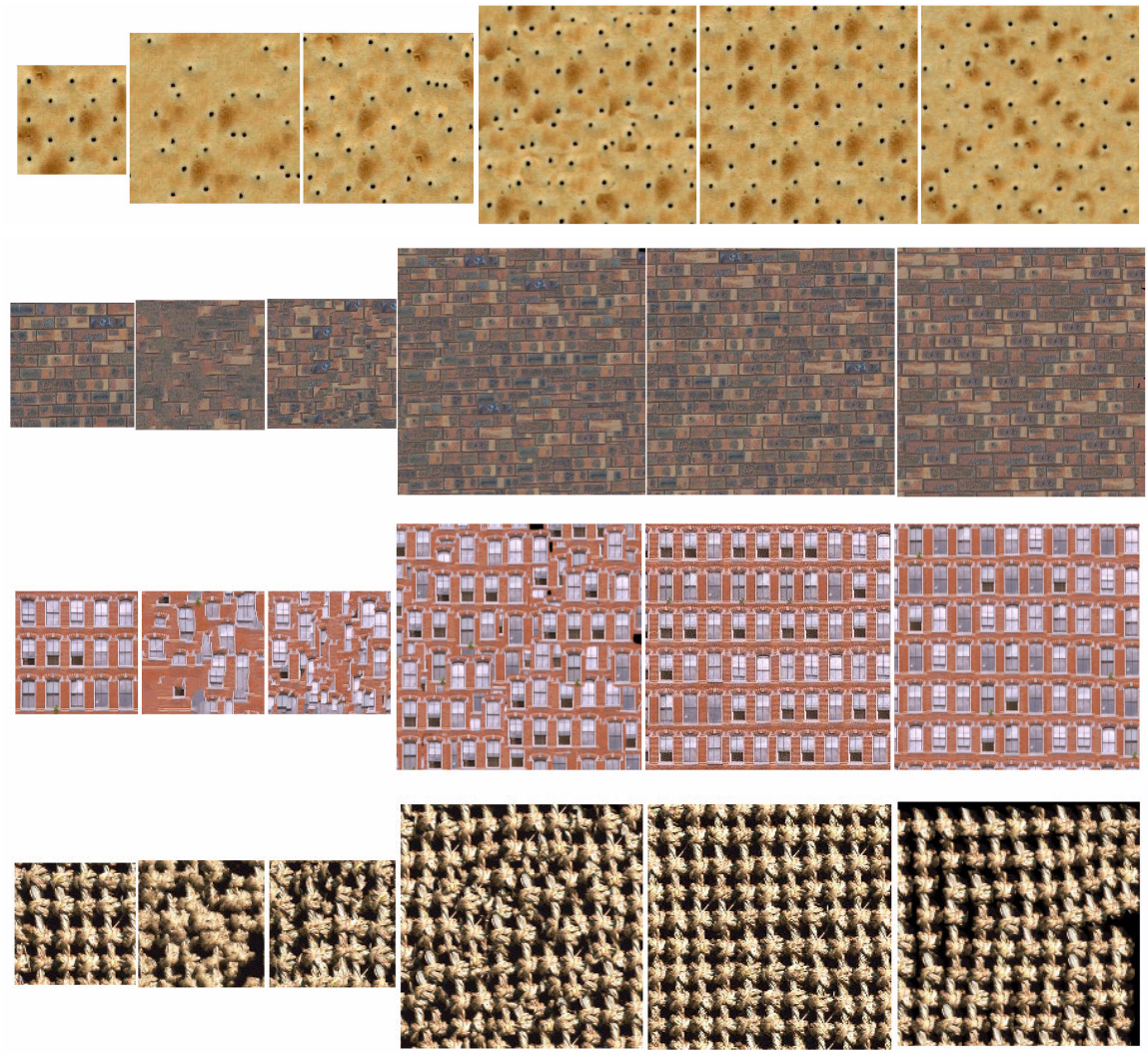

Figure 6: *Some examples of 2D synthesis of structured textures. From left to right: the model, Wei and Levoy synthesis, Ashikhmin synthesis, lapped textures (applied to a plane), image quilting and our method.*

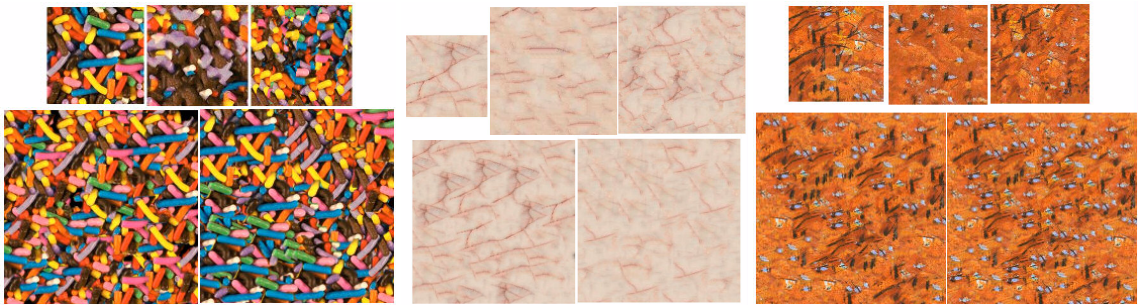

Figure 7: *Three more textures. For each example, the top row shows the model, followed by Wei and Levoy synthesis and then Ashikhmin synthesis. The two bottom images show our results: the left-hand image shows a basic larger reproduction; the right-hand image shows some variations obtained by modifying the density of specific particles.*

Figure 8 (see color section) shows some textures applied to arbitrary objects. The images demonstrate again that the user can control the distributions and probabilities of particles on different surface locations. The user can control individual particle properties, which is well demonstrated by the "double torus" made of colorful sweets. Also the marble on the statue illustrates user control: there are veins everywhere except on the nose, on the mouth and on the left eye, which has been explicitly specified by the user (vein particles having their center close to these regions have simply been removed). For this, the user simply selects the parts (respectively the facets of a mesh or the pixels of an image, for synthesis on a plane or a 3D surface) of each region of the surface to be textured. Then, the user can modify the parameters of each texture element in each region.

Figure 8: *Texturing arbitrary surfaces.*

We also used the same horse texture as Wei and Levoy [25] for qualitative comparison in Figure 8, which shows that our method preserves the square shapes better. For most of these textures, the actual synthesis process required less than 10 seconds (10 seconds was the worst case) on a PC with AMD Athlon (1.2 GHz) and GeForce2 graphics card. This timing, which does not include the analysis and decomposition into particles, depends on the number of particle classes. Here, there are only two classes: the background and one particle class. The timing also does not include the "background" synthesis, which needs to be done only once in a pre-process for a given object (independently of the type of the texture) using a lapped-textures-similar method and by using always the same "background" patch shape. In fact, apart from the "background" which is processed separately, the actual synthesis time mainly depends on the size and density of the features, not "that much" on the complexity of the surface (the complexity of the surface intervenes in the computation of the pseudo-distances). The main complexity of the algorithm lies in the test whether there are already particles on a location or not.

Figure 9 (see color section) shows a comparison between our technique and lapped textures, which are based on large texture pieces regrouping multiple particles. The top part shows, from left to right, the model, the texture patch used for the lapped textures, and some (not all) of the particles used for our technique. The bottom part shows the result of synthesis on a 3D model of skull, first (left) in the case of the lapped textures, then (right) with our technique.

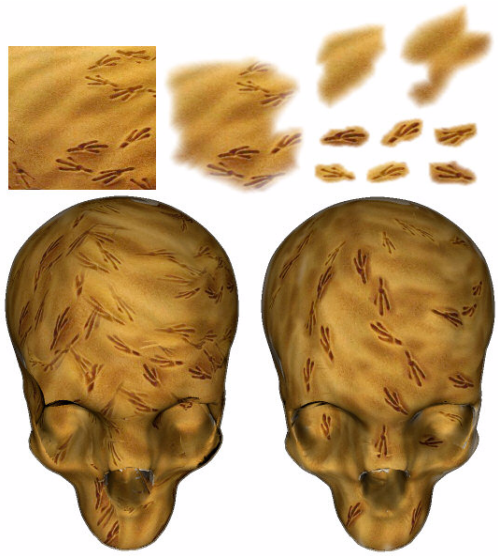

Figure 9: *Comparison of lapped textures (left) and our method (right).*

Further possibilities, such as controlling the size of the particles or mixing particles from different texture models, are offered to the user (see Figure 10, Figure 11 and color section). On the left of Figure 10, the size of the sweets decreases progressively in the center of the image, while on the right, the size of the black holes in the biscuit is changed randomly for each particle (each hole). For mixing textures, we actually mix the particles

instead of the texture characteristics [2], We first generated both textures separately; then we simply overlapped both synthesis results by removing some leaf particles. In fact, we only kept the leaf particles that were centered on the white mortar between the bricks.

Figure 10: *Examples of particle size variation.*

Figure 11: *A simple example of particle mixing.*

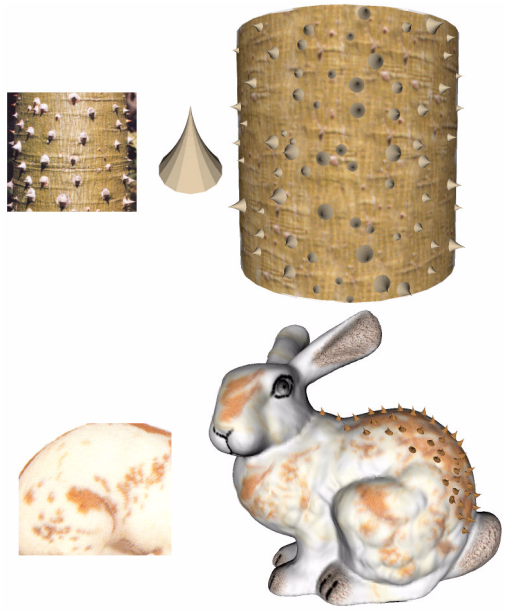

Figure 12: *The method also works for 3D "geometric" features, such as thorns.*

In this paper, we have essentially presented and described a method for synthesizing texture images, but the method can also be used for 3D textures. Figure 12 (see color section) illustrates this. The 3D shapes of the particles were modeled "by hand", but the positions were extracted from the models, so the synthesis could perform automatically (in the case of the bark, we neglected the curvature of the tree limb, by simply computing the distances in terms of pixels between the picks). We also applied some hand-modeled thorns to the back of the bunny. The distribution matches the one of the previous bark texture.

Finally, Figure 13 (see color section) illustrates the limitations of the method. Textures characterized by complex correlated spatial arrangements (for which shapes are correlated to the positions), such as for mosaic patterns or checkerboards, cannot be processed correctly. For example, in the case of a checkerboard, the corners of the black, respectively white, tiles are touching each other, which results in a more complex spatial structure than with the horse texture of Figure 8. In fact, to ensure that mosaic-like particles cover the entire plane (with no holes), the particles must overlap (like lapped textures) by increasing their global frequency. Consequently, the frequency of the texture is also augmented. Nevertheless, our results remain in this case more acceptable than with Wei and Levoy and Ashikhmin (both are also shown for comparison on the first row of Figure 13, respectively the middle and right-hand textures).

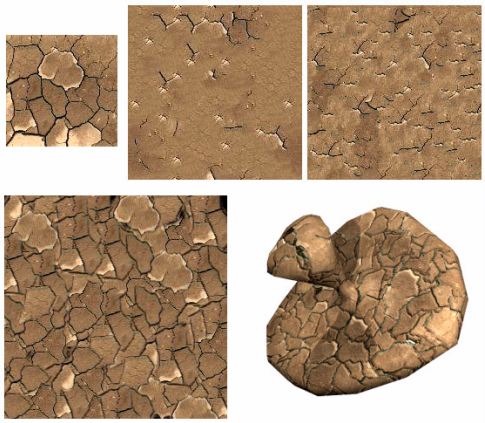

Figure 13: *An example of limitation of the method.*

4. Conclusions and future directions

We have presented a novel and simple approach for analyzing and synthesizing a large panel of different natural textures, including 3D textures, by using sample images (or 3D models) and by extending the principles of the bombing / cellular / macrostructured and lapped textures. The synthesis is innovative in that it introduces

a new level of texture analysis between the usual two extremities of pixel and large-scale levels (large texture pieces). It is fast, effective (since it reproduces correctly a large number of patterns) and grants users much more control than other synthesis methods.

As we have mentioned, some textures, however, cannot be correctly processed with this approach. In particular, textures with high-order correlations among feature shapes and distributions. We intend to address specifically these textures in our future works. For such textures, it is necessary to change the shape of a particle with respect to its position and to neighboring particles. Naturally, this will necessarily introduce some distortions, but the goal would be to keep these distortions as low as possible: it is indeed not possible to apply a checkerboard for example on any arbitrary surface, without distortions and/or discontinuities. Unlike a "brute-force" texture-mapping technique, which does not consider the "semantic" of the underlying texture, an analysis and decomposition into precise visual features can provide a solution, which better minimizes visual artifacts.

References

1. M. Ashikhmin. Synthesizing natural textures. *Proc. of 2001 ACM Symposium on Interactive 3D Graphics*, 2001.
2. Z. Bar-Joseph, R. El-Yaniv, D. Lischinski and M. Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Trans. on V&CG*, **7**(2):120-135, 2001.
3. J. S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. *Computer Graphics (Proc. of Siggraph'97)*, pp. 361-368, 1997.
4. J.-M. Dischler and D. Ghazanfarpour. Interactive image-based modeling of macrostructured textures. *IEEE CG&A*, **19**(1):66-74, 1999.
5. J.-M. Dischler, K. Maritaud and D. Ghazanfarpour. Coherent bump map recovery from a single texture image. *Graphics Interface*, 2002.
6. J. Dorsey and P. Hanrahan, Modeling and rendering of metallic patinas. *Computer Graphics (Proc. of Siggraph'96)*, pp. 387-396, 1996.
7. D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin and S. Worley. Texturing and modeling: a procedural approach. *Academic Press*, 1994.
8. A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. *IEEE Int. Conference on Computer Vision*, pp. 1033-1038, 1999.
9. A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. *Computer Graphics (Proc. of Siggraph'01)*, pp. 341-346, 2001.
10. K. W. Fleischer, D. H. Laidlaw, B. L. Currin and A. H. Barr. Cellular texture generation. *Computer Graphics (Proc. of Siggraph'95)*, pp. 239-248, 1995.
11. D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. *Computer Graphics (Proc. of Siggraph'95)*, pp. 229-238, 1995.
12. A. Herzmann and D. Zorin. Illustrating smooth surfaces. *Computer Graphics (Proc. of Siggraph'00)*, 2000.
13. L. Lefebvre and P. Poulin. Analysis and synthesis of structural textures. *Graphics Interface*, 2000.
14. B. Levy and J. L. Mallet. Non-distorted texture mapping for sheared triangulated meshes. *Computer Graphics (Proc. of Siggraph'98)*, pp. 343-352, 1998.
15. L. Liang, C. Liu, Y. Xu, B. Guo and H.Y. Shum. Real-time texture synthesis by patch-based sampling. *Tech. Report MSR-TR-2001-40, Microsoft Research*, 2001.
16. P. Litwinowicz and G. Miller. Efficient techniques for interactive texture placement. Computer Graphics (Proc. of Siggraph'94), pp. 119-122, 1994.
17. J. Maillot, H. Yahia and A. Verroust. Interactive texture mapping. *Computer Graphics (Proc. of Siggraph'93)*, pp. 27-34, 1993.
18. H. K. Pedersen. A framework for interactive texturing on curved surfaces. *Computer Graphics (Proc. of Siggraph'96)*, pp. 295-302, 1996.
19. E. Praun, A. Finkelstein and H. Hoppe. Lapped textures. *Computer Graphics (Proc. of Siggraph'00)*, 2000.
20. S. Premoze, W. Thompson and P. Shirley. Geospecific rendering of alpine terrain. *Proc. of EG Workshop on Rendering*, 1999.
21. B. J. Schachter and N. Ahuja. Random pattern generation processes. *Computer Graphics and Image Processing*, **10**:95-114, 1979.
22. G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proc. of Siggraph'91)*, **25**(4):289-298, 1991.
23. G. Turk. Texture synthesis on surfaces. *Computer Graphics (Proc. of Siggraph'01)*, 2001.
24. L.-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. *Computer Graphics (Proc. of Siggraph'00)*, pp. 479-488, 2000.
25. L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. *Computer Graphics (Proc. of Siggraph'01)*, 2001.
26. Y. Xu, B. Guo and H.Y. Shum. Chaos Mosaic: fast and memory efficient texture synthesis. *Tech. Report MSR-TR-2000-32, Microsoft Research*, 2000.
27. L. Ying, A. Hertzmann, H. Biermann and D. Zorin. Texture and shape synthesis on Surfaces. *Proc. of EG Workshop on Rendering*, pp. 298-308, 2001.