Fast Multi-Scaled Texture Generation and Rendering

Anton V. Pereberin

Keldysh Institute of Applied Mathematics RAS

Moscow, Russia

Abstract

The multi-scaled model for stochastic texture representation and the method providing real-time rendering of textures formalized by this model are introduced. Both “abstract” and “natural-like” textures can be generated. Being much simpler then existing stochastic texture models it satisfies the requirements of real-time texture mapping: compact data representation, scalability, random pixel access. The rendering algorithm is simple enough to be implemented in hardware.

Keywords: *stochastic textures, texture-mapping, multi-scaled representation, wavelet transform.*

1. INTRODUCTION

At least two objects are usually referred to as textures. First is an ordinary image processed and stored in a way convenient for mapping purpose [6], e.g. image of a palace facade to be mapped to the corresponding geometry to create realistic 3D model of the building. Storing such a texture in explicit form is expensive (as hundreds of textures are to be stored in graphical device memory simultaneously) so the image is to be compressed. Texture compression methods have to satisfy some special requirements. In particular, the decompression algorithm is to be as much simple and fast as possible and suitable for implementation in hardware. Then it must provide random access or local reconstruction, i.e. the ability to evaluate an arbitrary pixel of the image without reconstruction of the whole object. Moreover, a texture is to be represented in a way convenient for mapping on different resolution levels. This is usually achieved by *mip-mapping*, i.e. storing the sequence of 1:2, 1:4, 1:8, etc., scaled copies in addition to the initial image.

Another object is texture in its initial meaning, i.e. texture of material (wood, paper, marble, textile, etc.), texture of sandy, or water, or ground surfaces, texture of leather and so on. Also different “abstract” patterns can be treated as textures.

Such textures are usually processed in the following way. Given a relatively small sample of a texture, it is to be spreader over any desired size. The easiest way is simple tiling of the initial sample, but this produces poor result as tiling leads to periodic effect that looks unnatural.

There exist several *stochastic models* [1][3][4] to represent such textures. All of them are based on the hypothesis that textures can be formalized as probabilistic distributions. A texture sample is a sample from such a distribution. It should be analyzed in attempt to capture the distribution. If distribution is found properly, then initial sample and image, generated according to the distribution, must be perceived as two samples of the same texture, though not the same images.

In [1][2] the iteration method is used for texture synthesis and analysis: the inputs are texture and random noise samples, they are sequentially converted to the texture image of desired size. In [3] the Laplasian pyramid is build to analyze texture sample, on the synthesis phase the pyramid is transformed in a way preserving high-resolution features (deterministic component) and affecting low-resolution features (probabilistic component). In [4] textures are modeled as Markov Random Fields.

The idea to represent a texture with a small object containing all the information necessary for generation looks attractive, as this representation is sufficiently compact. Moreover, the size of such a representation doesn’t depend on size of the output. Unfortunately, generation textures from samples is not suitable for real-time applications. All the techniques mentioned above require sufficiently complicated and time-consuming calculations. Thus if real-time texture-mapping is required, the image of the desired size is to be generated *before* the rendering phase and then stored using texture compression techniques which do not take into account the special structure of the image.

Our task was to find a model for texture representation, which is probably not so powerful as existing models are, but satisfying the requirements of real-time texture mapping, mentioned above.

First a method for fast creation of new artificial textures was developed. The idea was to take some trivial image (base element) composed by a user in a minute by means of simple graphic editor and to generate new image from randomly scattered scaled and rotated copies of base element.

The next step was to modify a model in a way providing realistic approximation of some natural textures.

On the third step the compact texture data representation and fast rendering algorithm was developed.

The remainder of the paper is organized as follows. Section 2 contains the detailed description of the texture representation model. Some examples and results are introduced in Section 3. In Section 4 the proposed model is compared with wavelet transform of images. Section 5 introduces Layer Control Masks, the effective rendering technique. In Section 6 some implementation details and estimation of calculation complexity and data size are also discussed. The concluding Section 7 contains some ideas on model enhancement and the proposal for further research.

2. THE MODEL DESCRIPTION

As it was mentioned above, the idea behind the model was to compose an object from randomly scattered scaled and rotated copies of some small and simple trivial image (*base element*). In practice, however, not all the possible scales and rotations of base element are used; the place the copy of base element can be dropped to is not absolutely random also.

2.1 Replications

The *replication* is one copy (maybe scaled and transformed as described below) of base element to be placed to output image. The point of output image is called *replication point* if it has non-zero probability to be the origin of one of the replications.

Assume then that the base element is a square bitmap with side size $N = 2^K$ pixels. Then the scaled versions of the element are also squared bitmaps with side size 2^k, $k = \overline{1, K}$. Index k is called *resolution level* or *resolution.*

Elements can be replicated with shift equals to one half of their side size. That means that on the resolution level k, which corresponds to image side size 2^k, the replication points are $\left(2^{k-1}i, 2^{k-1}j\right)$, $i, j \in \mathbf{Z}$.

In each replication point the following *events* can take place:

- Base element can be replicated (*positive replication*), or negative of the base element can be replicated (*negative replication*), or base element can be not replicated at all (*no replication*).
- Base element can be *rotated* to 90°, 180° and 270°, or not rotated, (i.e. rotated to 0°).
- Base element can be *mirrored* or not mirrored.

For the particular model one can specify the probability of each of these events.

2.2 Composing Image from Replications

We assume that base elements can have pixels with both positive and negative intensity. Base element background has zero intensity and is considered to be “transparent”.

At the initial step the output image is the rectangular of desired size with zero intensity.

Then replications of base element are placed to the output image. The element can be simply added, but other operations are also available.

Assume that a is the current intensity value of some pixel of output image, b is the pixel value of a replication which is to update a and $\tilde{a}$ is the updated intensity of the pixel. Then the following operations are available:

- simple addition

$$\tilde{a} = a + b.$$

- non-zero application

$$\tilde{a} = \begin{cases} a, & b = 0 \\ b, & b \neq 0 \end{cases}.$$

- “maximum” application

$$\tilde{a} = \begin{cases} a, & |a| \geq |b| \\ b, & |a| < |b| \end{cases}.$$

The two latter operations are not linear, and not commutative, i.e. their result is depended on the order of replication. This feature can be used to control “transparency” of replications.

The probability of choice of one of these operations for each replication can be also specified.

Replications of equal resolution form *layers* of output image. One *weight coefficient* can be assigned to each layer. In this case all the replications of the layer are to be multiplied by the corresponding coefficient. This controls the contrast of the layer and consequently the layer significance in output image. Note that the terms *resolution level* and *layer* are closely connected with each other, but the are not equivalent (see Section 2.4 below).

The order of replication can be different. One of the easiest ways is *layer-to-layer* order. It is possible to specify whether to move from top level to bottom or vice versa.

The final step of the generation is adding some “background” intensity value to each pixel of an image.

2.3 Model Parameters Specification

To create a particular model one has to specify number of layers and tables of probabilistic distribution of events taking place in replication points of each level (*event tables*).

In the simplest case event table has only one cell. It means that the only distribution is used for all the replication points of a layer. The polar situation is when the particular distribution is specified for each replication point, but this seems to have no sense. Usually event table determines the event probabilities for a group of several neighbour replication points.

As to the base element it can be either included or not included in model specification. The latter case means that the model was intended to be used with different base elements.

Such parameters as background intensity of the whole image, weight coefficients for each layer and order of generation (top-bottom or bottom-top) are to be specified also.

2.4 Scaling

The proposed model provides the easy way to generate 1:2, 1:4, 1:8, etc., scaled copies of textures. Indeed, by default the top layer (layer 0) corresponds to resolution K of base element, layer 1 corresponds to resolution $(K-1)$, etc. If to shift this correspondence (e.g. layer 0 to resolution $(K-1)$, layer 1 to resolution $(K-2)$, etc.) than scaled (reduced) version of the texture will be generated. It is also possible to magnify the texture (e.g. layer 0 corresponds to resolution $(K+1)$), but in this case the base element is to be stretched to resolutions higher than K.

3. EXAMPLES

Random Rotation Model

The simplest model is called “random rotation”. The 2×2 event table consists of two staggered zero entries (which corresponds to “no replication” with probability 1), the two remained entries are the same and specify the uniform distribution of rotation angle (Tab. 1). The table is the same for any layer. The number of layers can be different but usually is 3 or 4. Weight coefficients can be different also but in the simplest case they all are equal to 1.

The base element is not specified. Originally this very model was used to generate new textures. So the model except the base element was pre-defined for a user could compose new base elements for it.

Instead of “negative application” the following pre-processing of base elements was used for this model: the base element was summarized with its mirrored and inverted copy. The mean of such a pre-processed element is always 0.

Examples of textures, generated by “random rotation” model are shown on Fig. 1 (corresponding base elements are placed to top-left corner of each image, the leftmost texture is presented in two scales).

Cheese

The “cheese” is a simple example of natural texture approximation. The more or less realistic model of cheese is just a number of holes scattered along a plane. Holes can be of different size, their shape and orientation are arbitrary. There must not be too many holes, so the probability of a hole replication must not be very high. We simplified this model by using the only shape for all the holes (Fig. 2, left) and only two scales. Nevertheless, the result looks like a piece of cheese (Fig. 2, right).

Brick Wall

The brick wall appeared to be an interesting example. The “ideal” brick wall is a regular structure, and can be represented easily. The task was to add some sort of irregularity to make the result look more natural. The “half a brick” image was used as the base element (Fig. 3, left). In the ideal case each two elements in any row are to be 180° rotated copies of each other, and two neighbour rows are to be shifted copies of each other. We affected this order by adding a small probability for the element to be oriented not in the proper way. One of the possible distributions is introduced in Tab. 3. The result is shown on Fig. 3, center. The “realism” of the image can be increased by adding some sort of a noise (Fig. 3, right). This noise is just the very base element replicated with a low weight coefficient (0.2) on layers 2 and 3 according to some uniform distribution (Tab. 2).

4. CONNECTION WITH WAVELET THEORY

One can notice that the introduced model looks similar to image wavelet synthesis [7] (i.e. inverse wavelet transform) and so-called *random wavelet expansion,* introduced in [5].

Here is the well-known formula of 2D dyadic wavelet reconstruction:

$$I(x,y) = v\varphi(x,y) + \sum_{k=0}^{K} \sum_{i,j=-\infty}^{+\infty} w_{ij}^{(k)} \psi(2^k x - i, 2^k y - j)$$

Suppose now that base element is a mother wavelet $\psi(\bullet,\bullet)$ (as it was mentioned above, in some experiments base elements were pre-processed to satisfy at least one attribute of the real wavelet, i.e. to have zero mean). Instead of wavelet coefficient $w_{ij}^{(k)}$, weight coefficient $w^{(k)}$, which is one per resolution k, is used. Each scaled and shifted copy of $\psi(\bullet,\bullet)$ is transformed (rotated, inverted, etc., or simply vanished) by the functional $\mathbf{W}[\,\bullet, \xi]$. The functional is controlled by random variable ξ which distribution depends, in general, on layer k and space disposition. The “low-resolution” part is expressed by the mean intensity v. Thus we get the following formula:

$$I(x,y) = v + \bigoplus_{k=0}^{K} \bigoplus_{i,j=-\infty}^{+\infty} w^{(k)} \mathbf{W}\left[\psi(2^k x - i, 2^k y - j), \xi_{ij}^{(k)}\right].$$

Symbols $\bigoplus$ are used instead of $\sum$ to show that operations similar but not identical to addition can be used. (Note that in general the choice of the operation is also controlled by random variable).

The main common feature of both formulas is that they represent an object $I(\bullet,\bullet)$ as a collection of scaled and shifted copies of some element $\psi(\bullet,\bullet)$. And, consequently, both expressions provides good scalability.

5. LAYER CONTROL MASK

For texture mapping purpose it is desirable not to generate a texture as the whole image, but to calculate small patches or even single pixels of the texture (*local generation*). The generation scheme described above is badly suitable for local generation as it uses random-number generator. Indeed, all the instances of a texture generated according to some model are samples from the same probabilistic distribution, but they are not the same image. In case of local generation different patches or pixels belong to different samples, i.e. they do *not* belong to the same images. Even two attempts to calculate one particular pixel can give two different results.

So the problem is to find a technique that guarantees that all the locally generated patches or pixels belong to the same image.

One of the possible solutions is to calculate the events in each replication point *before* the generation phase started and to store the results in special data structures.

Since in the proposed model the layers are generated independently from each other, the structure storing pre-calculated events is actually a set of 2-dimentional arrays. Each array corresponds to one layer. Each entry of an array corresponds to one replication point and contains a code describing replication event in this point. Such an array is called *layer control mask* (LCM).

Note that only one byte is enough to code all the possible events. Indeed, 2 bits are necessary to code positive/negative/none replication, 2 bits for the four possible rotation angles, 1 bit for mirroring and 2 bits to code the replication operation. Seven bits total.

LCMs provides both global and local generation. Global generation is performed in nearly the same way as described above, the only difference is that the generator uses prepared event codes instead of computing them. For local generation not all the entries of LCMs are used but only those corresponded to specified spatial area.

One can see that the use of LCMs not only solves the problem of local generation but also splits the generation process into two phases: (a) LCM generation and (b) texture rendering using prepared LCMs. Note that only the second phase is essential to be real-time. The LCM generation contains the main part of necessary calculations. Actually these calculations are not very complicated in existing model, but if the model is enhanced (see Section 7.1) the calculations will become more sophisticated and time-consuming. Moreover, the description of the model (and hence the input data representation) can be more complicated also. Nevertheless the increase of the complexity of the LCD generation it is not critical as this phase is performed independently from rendering and cannot affect the speed of the latter. The rendering phase uses simple non-intelligence algorithm and primitive input data format (2D arrays of bytes). So it can be implemented in hardware and performed very fast.

The disadvantage of LCMs is that they are of finite size. If not to use LCMs than it is possible to generate the texture of any desired size avoiding periodic effect. Besides, the size of input data (the distribution description) doesn't depend on output image size. If LCMs are used, the output image can be of any size also. But if size of LCMs is not enough to cover the size of the output image they are to be tiled and this sooner or later will lead to periodic effect. The larger the LCMs are the less appreciable the periodic effect is. But the large LCMs affect the compactness of texture representation. Some compromise between output image quality and data size is to be found.

First note that even not very large LCMs can guarantee sufficiently large output image without periodic effect. Indeed, if the model consists of the only layer and the base element size is 32×32 pixels than the LCM of 64×64 entries provides the generation of an image of size 1024×1024 pixels without tiling (remind that the distance between two replication points is one half of base element side size, hence $64(32/2)=1024$).

It seems that the smaller the size of base element is the larger the size of LCM must be. E.g., if the base element size is only 16×16 pixels then to generate 1024×1024 output image without tiling the size of LCM must be 128×128. However, if the model consists of at least two layers there is usually no necessity to make all the LCMs cover the output image size. E.g., the model consists of 2 layers (0 and 1), the base element size corresponded to layer 0 is 32×32. Then the base element size on layer 1 is 16×16 pixels. Now assume that the size of both LCMs is 64×64 and the desired size of the output is 1024×1024 pixels. Though tiling presents in layer 1 but when united with layer 0, which is free of tiling, then periodic effect is hardly perceptible. Sometimes even better results can be achieved if LCMs sizes are not multiplies of each other, are not multiplies of power of two and maybe not square at all. E.g., size of layer 0 LCM is 50×60, size of layer 1 LCM is 75×45. It is obvious that the period of the output is much larger than the period of any of the separate layers.

Other ways of periodic effect decay without considerable increase of representation data size are also available.

6. DATA REPRESENTATION AND IMPLEMENTATION NOTES

As it was mentioned above, the use of LCMs permits to divide the generation process into two phases.

The implementation details of the first phase, the LCM generation, are not discussed here. The only thing can be mentioned is that since the probabilistic distribution is specified independently for each layer, the calculation of LCMs can be parallelized easily.

Now consider the texture representation after LCM generation. Obviously it must be both compact and easily interpreted.

A base element is represented as 8-bit bitmap of specified size (in our experiments it was usually 128×128, 64×64 or 32×32 pixels). Not only the base element itself but also its copies of lower resolutions are to be represented. We used two representations in our experiments. The first one is Haar transformed image [7], which has exactly the same size as initial image but allows to reconstruct it with any dyadic resolution relatively fast. The second way is storing all the scaled copies in explicit form (this approach is similar to mip-mapping). This requires more space for the representation but provides more effective rendering (see below).

The representation of LCMs was discussed in Section 5.

Let us consider the "brick wall" representation for example. The model consists of 3 layers: 0, 2 and 3. The size of layer 0 LCM is 40×40, size of the other (it is the same for both remained layers) is 20×20. The size of the base element is 32×32 pixels, its 8×8 and 4×4 pixels copies are to be stored also (the explicit representation is considered). Hence the result is $40^2+20^2+32^2+8^2+4^2=3104$ bytes plus at most 20 bytes for additional information (including weight coefficients, generation order flag, etc.). This data is enough to generate texture with period 640×640 pixels. The 8-bit bitmap of the same size occupies 409600 bytes which is approximately 130 times larger than the proposed representation. Even if to add missed resolution levels of the base element (16×16 and 2×2 pixels) and to use different LCMs for layers 2 and 3 than the size of the representation will not exceed 3800 bytes which is approximately 107 times smaller than the whole image size.

The representation can be even more compact if some compression methods are applied to it. One of the possible approach is to code the regular structure of zeros ("no replication" events) in LCMs. E.g., LCMs generated according to event tables Tab. 1 or Tab. 3. have many regularly structured zero entries, thus they can be coded in a way which can reduce the size of LCM representation approximately twice or even more. Moreover, such kind a compression can hardly affect the rendering speed. Some techniques of fast compression and decompression can be applied to base elements also.

Now let us pay attention to rendering phase and discuss the evaluation of one separate pixel of a texture.

Since almost all the data is stored in 2D arrays of bytes, access to any entry of any LCM and any pixel of any resolution level of base element is trivial. (However, if the base element is represented by its Haar transform, then additional calculations are required to get its pixels). Given point coordinates, the rendering module can easily find entries of LCMs corresponded to specified point. Then, according to event codes, for each replication covering the point it has to evaluate corresponding pixel value. For almost all the events this evaluation consists just in finding necessary pixel in one of the scaled copies of base element, and only for "negative replication" the sign of the value is to be changed then.

For the sake of simplicity assume now that simple addition only is used for generation. According to the model definition, in any point at most four replication of the same layer can meet. So, at most 3 additions per layer are to be performed. Then obtained result is to be multiplied by weight coefficient (1 multiplication). Then values of all the N layers are to be summarized ($(N-1)$ additions) and mean intensity is to be added also (1 addition). Thus the number of operations (excluding the search of replications pixels) to render one pixel of N-layered texture doesn't exceed $4N$ additions (or similar operations) and N multiplication.

As well as for the LCM generation sufficient portion of calculations are performed independently for each layer, so it can be parallelized. Note, however, that for parallel computations scaled copies of base element are to be stored in explicit form rather than evaluated from transformed representation.

7. CONCLUSION. FURTHER WORK

The model for representation of both "abstract" and some "natural-like" scalable textures has been introduced. It was supplied with effective generation and rendering technique. By means of this technique the most complicated calculations were encapsulated into the pre-processing phase. This permits to perform rendering phase very fast and even to implement it in hardware. The algorithm provides pixel-wise rendering, which is very important for texture-mapping purposes. The representation of texture is sufficiently compact (3-10 Kb), thus a large number of textures can be stored in graphical device memory.

Obviously, the model is not free of limitations. Sufficiently large class of objects can hardly be represented by existing variant. So the one of the tasks for further research is enhancement of the model which will enlarge the class of textures can be represented.

Another direction of further work is development of *texture analysis module* and is actually a new research project.

7.1 Model Enhancement

Some trivial enhancements of the model can make it more flexible. The "power of two" restrictions on base element size, resolution level construction and replication shifts can be weakened. On the other hand this may demand additional data for model representation and also may affect the scalability of the output.

In existing model the probabilistic distribution of events is specified independently for each layer. The opportunity to specify the distribution for neighbour replications on different layers can be added. But this can require synchronization between layers, which can make the LCM structures more complicated.

Some other modifications of this kind can be made also.

The more serious enhancement is implementation of more sophisticated stochastic models, models using conditional distribution, e.g., Random Markov Fields [4].

Note that the more sophisticated model is used, the more complex calculations are required. But this will concern only the phase of LCM calculation, and this phase is not hardware-implemented and must not be real-time. As to the rendering phase, it will work with the same (or may be slight different) LCM structures, and thus will be as simple and fast as it is now.

7.2 Analysis Module

We assume that the structure of wide range of textures can be approximately represented in a way similar to one proposed in this paper, i.e. as on or maybe more "base elements" and simple data structures controlling their replication.

One of the possible approaches is to use different modifications of wavelet transform to capture such a structure. Wavelet transform is a powerful tool for space-frequency analysis and the use of hierarchical, multiresolution or wavelet-based methods for texture analysis is not new [1][3][5]. Moreover, the existing model has many features common with dyadic 2D wavelet transform, and the use of similar methods for both synthesis and analysis seems to be promising.

8. ACKNOWLEDGEMENTS

This work was made as a part of research program performed in accordance with the Research Agreement between Moscow State University and Intel Technologies, Inc.

9. REFERENCES

[1] D. J. Heeger and J. R. Bergen. Pyramid-Based Texture Analysis/Synthesis. *Proceedings of SIGGRAPH'95,* pp. 229-238, 1995.

[2] T. F. El-Maraghi. An implementation of Heeger and Bergen's Texture Analysis/Synthesis Algorithm, 1997. http://www.cs.toronto.edu/~tem/2522/texture.html

[3] J. S. De Bonet. Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images. *Proceedings of SIGGRAPH'97,* pp. 362-368, 1997. http://www.ai.mit.edu/~jsd/Research/TextureSynthesis/

[4] A. A. Efros and T. K. Leung. Texture Synthesis by Non-Parametric Sampling. *IEEE International Conference on Computer Vision,* Corfu, Greece, Sept. 1999.

[5] D. Mumford and B. Gidas. Stochastic Models for Generic Images, 2000. http://www.dam.brown.edu/people/mumford/Papers/Generic5.pdf

[6] A.V. Pereberin. Hierarchical Approach for Texture Compression. *Proceedings of GraphiCon'99,* pp. 195-199, 1999.

[7] E. J. Stollnitz, T. D. Derose and D. H. Salesin. *Wavelets for Computer Graphics. Theory and Applications.* Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

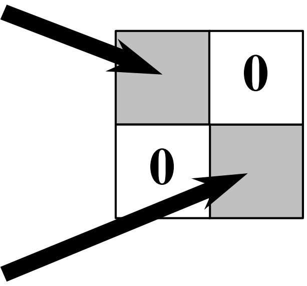
About the author:

Anton V. Pereberin is the postgraduate student of Keldysh Institute of Applied Mathematics. Interested in multiresolution methods of graphical information representation.

E-mail: avpereb@newmail.ru

Tab. 1. The “random rotation” model

Pos. repl.	1.0
Neg. repl.	0.0
No repl.	0.0
Rot. 0°	0.25
Rot. 90°	0.25
Rot. 180°	0.25
Rot. 270°	0.25
Mirroring	0.0
Addition	1.0
Non-zero	0.0
Maxinium	0.0



Tab. 2. The “simple noise” model.

Pos. repl.	0.33
Neg. repl.	0.33
No repl.	0.34
Rot. 0°	0.25
Rot. 90°	0.25
Rot. 180°	0.25
Rot. 270°	0.25
Mirroring	0.5
Addition	1.0

Tab. 3. The “brick wall” model.

Pos. 1.0 0° 0.8 90° 0.05 180° 0.15 Non-z. 1.0	0	Pos. 1.0 0° 0.2 180° 0.8 Non-z. 1.0	0
0	0	0	0
0	Pos. 1.0 0° 0.75 90° 0.05 180° 0.2 Non-z. 1.0	0	Pos. 1.0 0° 0.1 90° 0.02 180° 0.88 Non-z. 1.0
0	0	0	0

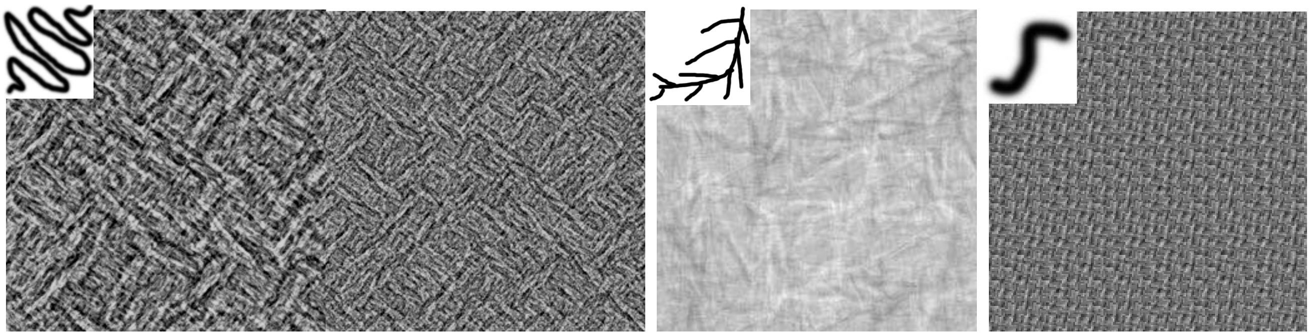


Fig. 1. Examples of “random rotation” model.

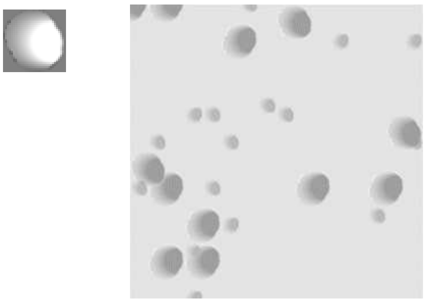


Fig. 2. “Cheese”: base element and output image.

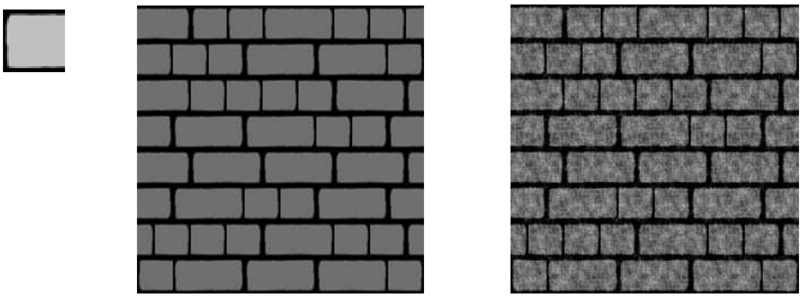


Fig. 3. “Brick wall”: base element, simple output image and advanced output image.