Proc. VIIth Digital Image Computing: Techniques and Applications, Sun C., Talbot H., Ourselin S. and Adriaansen T. (Eds.), 10-12 Dec. 2003, Sydney

Synthesising Textures Using Variable Neighbourhood Searching

Minh Tran and Amitava Datta

School of Computer Science & Software Engineering
The University of Western Australia
35 Stirling Highway, Crawley, WA 6009, Australia
{minh,datta}@csse.uwa.edu.au

Abstract. Texture synthesis aims to define and reproduce discriminating image features. These features are used to associate with and differentiate between two textures. Often texture is composed of a pattern with an element of randomness in each feature's appearance, position and orientation. The goal is to imitate the sample texture in such a way that sample and synthesised texture are perceived to be generated by the same source. One of the recent texture synthesis methods selects an output pixel by searching with its already generated neighbourhood for a corresponding match in the sample image. The neighbourhood size is fixed and blurring of texture features often results. Our method attempts to avoid this problem by enabling a dynamic, accelerated neighbourhood search. The window size varies with each output pixel and is determined by the current neighbourhood intensity configuration.

1 Introduction

A texture exists as a surface description of an object. It is usually composed of an oriented regular structure or a relationship relevant between constituent texture elements. In computer graphics, the purpose of a texture is to depict surface detail which does not necessarily occur in the geometry of the object surface [4]. Mapping texture onto object models is a less intensive method of graphical object design compared to explicit surface detailing. Texture synthesis was initially introduced to eliminate repetition and visible seams evident in texture mapping inappropriately sized samples [10]. Tiling texture copies together produces texture discontinuities once mapped onto the surface. On the other hand, stretching the texture to cover a larger surface area can cause image distortion. Synthesising textures to any desired size will alleviate the shortcomings of limited sized texture mapping.

Our approach is based on the work accomplished by Wei & Levoy [15] and Ashikhmin [1]. They both use neighbourhood searching to find pixels with similar neighbourhoods in the sample image to assign as the new pixel in the output image. While Wei & Levoy's approach tends to blend texture elements together, the approach by Ashikhmin produces output that contains patches of the original sample with visible seams. These are the dominant image artefacts found in

the output images, and our approach attempts to avoid these shortcomings by finding a compromise between these two approaches. We have implemented a method that generates an output on a per pixel basis using an accelerated, variable neighbourhood search.

1.1 Motivation and Applications

Synthesising textures to an arbitrary size is advantageous in computer graphics and image processing. Generating novel views of the physical world from texture samples is less scrupulous than recreating it from scratch [5]. Constrained synthesis can be used to remove image artefacts; the surrounding texture is used as the input sample and synthesised over the selected area. Boundary information and a texture sample can be stored to allow image compression; each region is reconstructed separately with its texture [6]. Texture synthesis can also be extended to cover arbitrary 3D models.

Texture synthesis can be used to improve the artistic capabilities of graphical designers. Recently, texture synthesis methods have been used to combine two different textures together and transfer or swap textures between their corresponding objects [5]. Texture transfer can, for example, give a polar bear the skin of a orange. Texture mixing creates textured combinations such as moss on a wall. This can be extended into 3D to create temporal textures, for example, moss can creep or grow along a wall. However, this method of design cannot (yet) emulate the interaction of the material it is representing with its surroundings, for example, multiple ripples of water interference or shadows. Often, the size and shape of the target object does not suit the texture sample available. The sample can be used as a training example to generate an output, perceived to be the same texture.

1.2 Overview

In the next section we discuss past texture synthesis methods with particular interest in Wei & Levoy [15] and Ashikhmin's [1] work. We describe our methodology in Section 3 and discuss our approach in Section 4. Section 5 covers our results and in Section 6, we conclude with suggestions of further work.

2 Previous Work

The key to texture synthesis is to find what features and statistics must be shared between two images to be perceived as the 'same' texture. Emulating these features can give the impression that the same stochastic process produced these samples. Heeger & Bergen match histograms of multi-resolution filter responses [8]. Bar-Joseph et al. match wavelet transforms decomposed into a tree [2]. Portilla & Simoncelli use complex wavelet coefficients to constrain their texture synthesis algorithm [14]. Efros & Freeman and Liang et al. match patch edges [5, 10]. Kwatra et al. initially match rectangular patches and use a

graph cut method to calculate the optimal irregular patch shape to be pasted on the output image [9]. De Bonet matches and shuffles regions at different resolutions, sampling from the original decomposed texture corresponding to the same resolution [3]. Efros & Leung, Harrison, Wei & Levoy, and Ashikhmin match pixel neighbourhoods [6, 7, 15, 1].

The Markov Random Field (MRF) is a popular model used to abstract the complex properties of textures. This theory, in relation to images, states that each pixel in an image has an independent local spatial property characterised by its surrounding neighbours. Therefore, the generative process of textures in this model is assumed to be stochastic, local, and stationary [10]. Several texture synthesis methods [6, 10, 11, 13, 15, 17] adopt the MRF model in their synthesis process. Most of these approaches settle for an approximation to the more time-consuming optimal solution. Due to the spatial locality of the sampling procedure adopted by these researchers, the methods can be subdivided into pixel-based and patch-based synthesis. Thus one can infer a pixel by analysing its neighbourhood (pixel-based synthesis) or conversely, assign a neighbourhood patch given a pixel (patch-based synthesis).

2.1 Patch-based Synthesis

Patch-based approaches apply the idea that once a pixel has been generated, the next or neighbouring pixels have already been predetermined since each pixel is characterised by its neighbours. Patches of the sample texture are pasted onto the output image. The size of the patch must be large enough to encapsulate the largest regular repeating pattern in the image. The shape of patch can be square [5, 10], rhombic [11], arbitrary [16], or optimally determined by a graph cut method [9].

The arrangement of patches must preserve image fidelity by obscuring patch seams and maintaining an element of randomness. Tiling patches with similar overlapping edges together prevents image discontinuity. Patch edges are concealed by blurring [10, 16] or redefined by a minimum-cost edge function [5, 9]. Randomness can be preserved by arbitrarily selecting a patch from a set of possible patch candidates. This approach is less computationally intensive than pixel-based methods since synthesis is handled as a collection of pixels at a time instead of one at a time. However, this approach can produce artefacts such as image discontinuity caused by blending two features together located at shared patch edges.

2.2 Pixel-based Synthesis

Pixel-based synthesis methods incrementally generate output pixels from coercing either a random noise image or an empty image to display similar intensity statistics as the sample image. Typically, the new pixel is determined by evaluating its neighbours and determining which pixel in the sample image has the closest matching neighbourhood intensities. This sample pixel becomes the new

output pixel thus the final texture only contains pixels originating from the sample image.

Sample texture patterns are preserved in the output image by the size and shape of the neighbourhood comparisons. Wei & Levoy [15] and Ashikhmin [1] use an L-shaped causal neighbourhood where only already generated pixels are used in the comparison. Efros & Leung [6] grow pixels from the centre, spiralling outwards, using a square-shaped neighbourhood, but only including already generated pixels.

An image output artefact displayed by most pixel-based approaches is the blending of texture elements. While patch-based methods preserve edge information between texture objects, pixel-based methods tend to blur these features. This produces the artefact of 'blobs' where texture features are fused or smudged.

Explicit pixel generation is often computationally expensive, requiring some form of acceleration. Wei & Levoy [15] treat the search as a nearest neighbour search using a tree-structured vector quantisation method (TSVQ). Each pixel neighbourhood comprises a vector; input sample vectors are stored in a tree determined by a generalized Lloyd algorithm. Output pixel vectors are compared with that of the tree using a best-first search instead of an exhaustive search (such as Efros & Leung's method [6]).

Another approach to improve texture synthesis speed is to simplify the pixel selection process. Ashikhmin's greedy algorithm [1] restricts the search for a match to a selection from valid candidates. He adopts the principles similar to the patch-based approach where a pixel's previously determined neighbours affect its selection. A new pixel is generated by locating input pixels corresponding to each neighbourhood pixel, and appropriately 'forward-shifting' to the candidate pixel (controlled by the neighbourhood pixel's position relative to the pixel being generated). Therefore, the search for a match is reduced to a comparison of (at most) the total number of neighbourhood pixels, instead of the entire image. If the neighbourhood of the candidate does not lie entirely within the input image, a candidate is selected at random. However, this method grows regions of textures with visible seams between regions so, like patch-based synthesis, it is only suited for natural textures composed of mostly high frequency information.

3 Methodology

One of the problems facing texture synthesis research is finding a balance between generating an output that looks too repetitive and synthesising an output that is so random, it does not represent the original sample at all. To avoid repeated copying of discriminating features, an element of randomness is often introduced. Previously, this was achieved by randomly selecting a pixel or patch from a set of candidates [1, 2, 5]. However, this randomness parameter must be controlled to prevent the output texture patterns deviating too far from its sample.

Wei & Levoy [15] identify texture features by using a neighbourhood search accelerated by TSVQ. Perhaps the blurring of object edges in the texture can

be explained by their uniform window size used in each search. If the target neighbourhood window lies between two elements containing edges, the match found may contain intensity values that are an average of the target neighbourhood. This can be avoided by using a dynamic shaped neighbourhood window, the size of which is dependent upon the neighbourhood intensities belonging to the target pixel. If the size of the window is restricted to enclose a single or part of a texture feature, then the match found would be one that closely reflects the neighbourhood intensity values of the target pixel. However if the selection of the window size is too small, not enough information is captured within that window to distinguish it from the rest of the sample, and the resultant output looks flat with little variance of colour. If the window size selected is too large, the structure of texture features are better encapsulated, however computational time is increased and blurring can result.

4 Our Approach

The selection of window size varies with each pixel being generated. We specified the bounds of the variable window size to range from 2 to 10. Figure 1 illustrates how these numbers relate to the width of the neighbourhood window. The shape of window is causal, adapted from Wei & Levoy's L-shaped neighbourhood. It examines only already generated pixels to find a match. The output pixels are initialised to noise. Each pixel is traversed in raster scan order starting from the top left hand corner. The neighbourhood window is initially the minimum value and the RGB intensities are compared to the average values. If the difference between these two values exceeds a certain threshold (λ) the window size is set as its current size and a match is searched for. If it lies within the threshold limit, the next pixel is encountered and the width of the window expands as it reaches the final pixel corresponding to the current width. The order in which the neighbourhood pixels are traversed is shown in Figure 1. When the window size (w) is equal to 1, the pixels examined start from position 1 and increments to last pixel corresponding to that size, in this case, position 4. The window size increases to 2 and the pixels at position 5 onwards are processed.

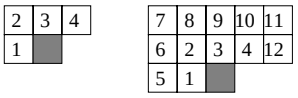

Fig. 1. An example of causal neighbourhoods with window sizes $w = 1$ and 2 respectively. Even though the actual maximum width of these neighbourhoods reaches $2w+1$, the size w depicts the number of pixels spanning to the left, right and above the candidate pixel. The grey pixel represents the pixel being generated. The surrounding numbered pixels depicts the order in which the pixels are traversed to determine the window size.

To improve the efficiency of our algorithm, we used ANN to accelerate our neighbourhood matching. ANN decomposes the data points into either a kd-tree or a box-decomposition tree. In high dimensions, searching such data structures for nearest neighbours perform significantly better with respect to time, than a brute force method [12]. Given a degree of error (ϵ) specified by the user, a faster running time can be achieved. We used the kd-tree data structure since it produced comparable results at a significantly less processing time than box-decomposition trees (see Figure 3 for kd-tree synthesis results and Table 1 for time comparisons).

Variables needed in the ANN search include the array of data points, total number of points, dimension, bucket size, splitting rule, and maximum points visited. The last three variables affect the time and accuracy of the search and the default settings were used. The data array arrangement is illustrated in Figure 2. The indices in the neighbourhood window correspond to the order in which the pixels are placed in a vector. The processing is done using RGB values, thus each pixel has three constituent values. In the example given in Figure 2, the size of a vector of four pixels extends to twelve to cater for the RGB value of pixels. Each pixel in the sample image has a corresponding neighbourhood vector. The vectors are collected for each neighbourhood size and placed in the corresponding kd-tree.

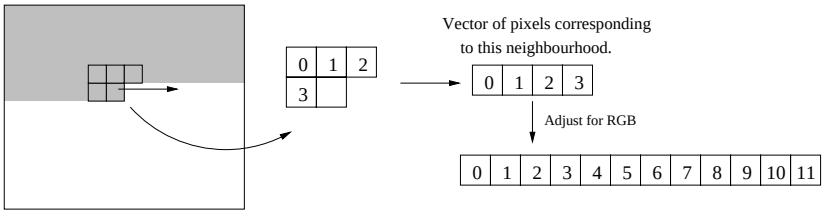

Fig. 2. This figure illustrates how a neighbourhood arrangement corresponding to a pixel is stored as a vector. The indices indicate the order in which the pixels are stored in the vector. The extension of each pixel by 3 caters for its RGB value. Neighbourhood vectors corresponding to each pixel is placed in a kd-tree.

Before any neighbourhood searching is conducted, kd-trees corresponding to each neighbourhood size are determined for the texture sample using ANN. During the pixel generation phase, the appropriate kd-tree can be searched according to the corresponding window size selected. The ANN search is fast and efficient. It requires a specified error bound (ϵ). This value determines how accurately ANN searches for a match. A value of $\epsilon = 0$ returns the exact nearest neighbour. A value greater than 0 returns an approximate nearest neighbour and thus introduces an element of randomness into the match found.

5 Results

A few results gained from this approach can be viewed in Figure 4. The algorithm was implemented in C++ and run on a Pentium IV 2.4GHz 512MB RAM PC. We used an error bound of $\epsilon = 10$ and all results were generated at 200×200. We implemented a λ detection method to determine the threshold value. This value is responsible for determining the size of the window. It is defined by the average difference between a non-edge pixel and an edge pixel. A Canny edge detector determined the edges of the sample at a threshold of 0.5. The pixels were then traversed in scan line order and an average of non-edge pixel intensities was accumulated. Once an edge pixel was reached, the difference between that average and the edge pixel was also accumulated and averaged over the whole image. Specifying the wrong threshold value can send some parts of the pixel generation to get stuck in the wrong search space and produce garbage; this is similar to the problem raised by Efros & Leung [6] however the shape of these clustered pixels have a characteristic jagged appearance. This may be due to the direction of the pixel generation. Harrison [7] states that the direction in which pixels are generated affects the quality of the output. The optimal direction is dependent upon how and where a pixel is constrained by its neighbours.

Highly structured textures require window sizes that are large enough to encapsulate the largest repeating feature. Figure 4(f) illustrates the effect of specifying a window size that is too small. The feature elements of this texture are captured, even the parts cut off at the edge of the image, but the structure between these elements is not. By increasing the maximum window size to cater for the pattern, this problem can be alleviated. However, this will require more processing overhead, particularly memory usage and search time.

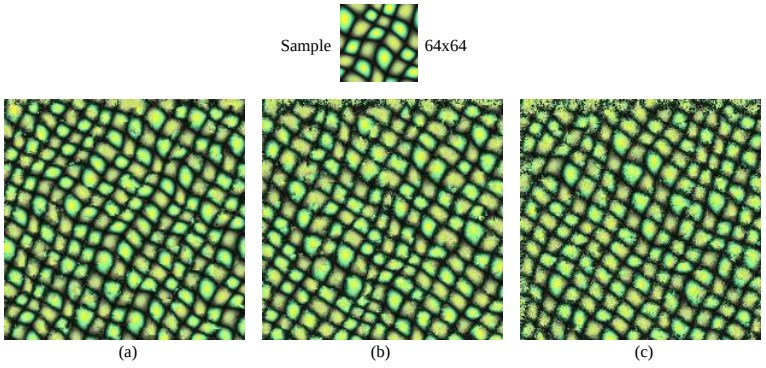

Fig. 3. A texture sample and its size is provided at the top of this figure. The results below this sample display the effects of quality of output with varying error bound values for the kd-tree data structure. Synthesis results are 200×200. **(a)** $\epsilon = 0$ **(b)** $\epsilon = 10$ **(c)** $\epsilon = 20$.

ϵ	Kd-tree Preprocessing	Kd-tree Synthesis	Bd-tree Preprocessing	Bd-tree Synthesis
0.0	4 secs	168 secs	11 secs	616 secs
10.0	4 secs	17 secs	11 secs	26 secs
20.0	4 secs	8 secs	11 secs	11 secs

Table 1. This table shows the preprocessing and synthesis times obtained for the results belonging to kd-tree (Figure 3) and box decomposition tree (bd-tree).

In our approach, there is a sacrifice of quality for speed and efficiency. This trade-off is captured in the error bound value, ϵ, specified in the ANN search. The effects of different error bounds with the output image is summarised in Figure 3 and the times given in Table 1. As expected, the better output was obtained from the smaller ϵ value at the cost of time. The output generated at ϵ = 20.0 contained more noise producing a 'fuzzy' effect. On the other hand, the results gained at $\epsilon = 10.0$ were comparable to that of $\epsilon = 0.0$ at the fraction of the time cost, hence we set $\epsilon = 10.0$ throughout our experiments.

Since the output image is initialised to white noise, early neighbourhood measurements contain mostly noise pixels, which affects the quality of the output image. Typically, a strip of pixels located on the top and left hand side of the image are inconsistent with the rest of the output image. An example of this is evident in Figure 3. However, this can be easily fixed by passing the algorithm again over these sections.

6 Conclusion and Future Work

Our method is a compromise between the current work of Wei & Levoy [15] and Ashikhmin [1]. This approach attempts to avoid image blurring of texture edges of Wei & Levoy's method and prevent seams evident in Ashikhmin's algorithm by allowing the size of the neighbourhood window to vary for each ANN search. We have used a variable neighbourhood search to maintain local spatial consistency within each texture feature. The size of the neighbourhood window is determined by change in their intensity values from the average. Our results are comparable to recent work but there is still room for improvement.

The window bounds unique to a texture can be measured during the preprocessing phase when the texture edges have been located. Statistics gathered from the edge thresholded image can assist in a educated guess of the optimal minimum and maximum window sizes. Appropriate window sizes can prevent the blurring of texture features and also encapsulate texture structure. The majority of an output pixel's neighbourhood may correspond to a general texture feature or area of the sample. Therefore the difference between non-edge pixels and its next edge can be collected for each pixel since the difference between non-edge pixels and pixel edges vary within each texture image. During the synthesis phase, the majority location of a pixel's neighbourhood in the output image can be measured and the corresponding λ (see section 4) value used accordingly.

References

1. Michael Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, pages 217–226, 2001.
2. Ziv Bar-Joseph, Ran El-Yaniv, Dani Lischinski, and Michael Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Transactions on Visualization and Computer Graphics*, 7(2):120–135, April–June 2001.
3. Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. *Computer Graphics*, 31(Annual Conference Series):361–368, 1997.
4. David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling*. AP Professional, 2nd edition, 1998.
5. Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In Eugene Fiume, editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 341–346. ACM Press / ACM SIGGRAPH, 2001.
6. Alexei A. Efros and Thomas K. Leung. Texture synthesis by non-parametric sampling. In *IEEE International Conference on Computer Vision*, pages 1033–1038, September 1999.
7. Paul Harrison. A non-hierarchical procedure for re-synthesis of complex textures. In *WSCG 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 190–197. University of West Bohemia, 2001.
8. David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *International Conference on Image Processing*, volume 3, pages 648–651, October 1995.
9. Vivek Kwatra, Arno Schodl, Irfan Essa, Greg Turk, and Aaron Bobick. Graphcut textures: Image and video synthesis using graph cuts. In *Proceedings of SIGGRAPH 2003*, 2003.
10. Lin Liang, Ce Liu, Yingqing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. Technical Report MSR-TR-2001-40, Microsoft Research, March 2001.
11. Yanxi Liu and Yanghai Tsin. The promise and perils of near-regular texture. In *Texture 2002, the 2nd International Workshop on Texture Analysis and Synthesis, associated with the European Conference on Computer Vision 2002 (ECCV'02)*, 2002.
12. David M. Mount. *ANN Programming Manual*. University of Maryland, 1998.
13. Rupert Paget and I. D. Longstaff. Texture synthesis via a noncausal nonparametric multiscale markov random field. *IEEE Transactions on Image Processing*, 7(6):925–931, June 1998.
14. Javier Portilla and Eero P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 4(1):49–71, 2000.
15. Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 479–488. ACM Press / Addison Wesley Longman, 2000.
16. Y. Xu, S. Zhu, B. Guo, and H. Shum. Asymptotically admissible texture synthesis. In *Second International Workshop of Statistical and Computational Theories of Vision*, July 2001.
17. Alexey Zalesny and Luc Van Gool. A compact model for viewpoint dependent texture synthesis. *Lecture Notes in Computer Science*, 2018:124–143, 2001.

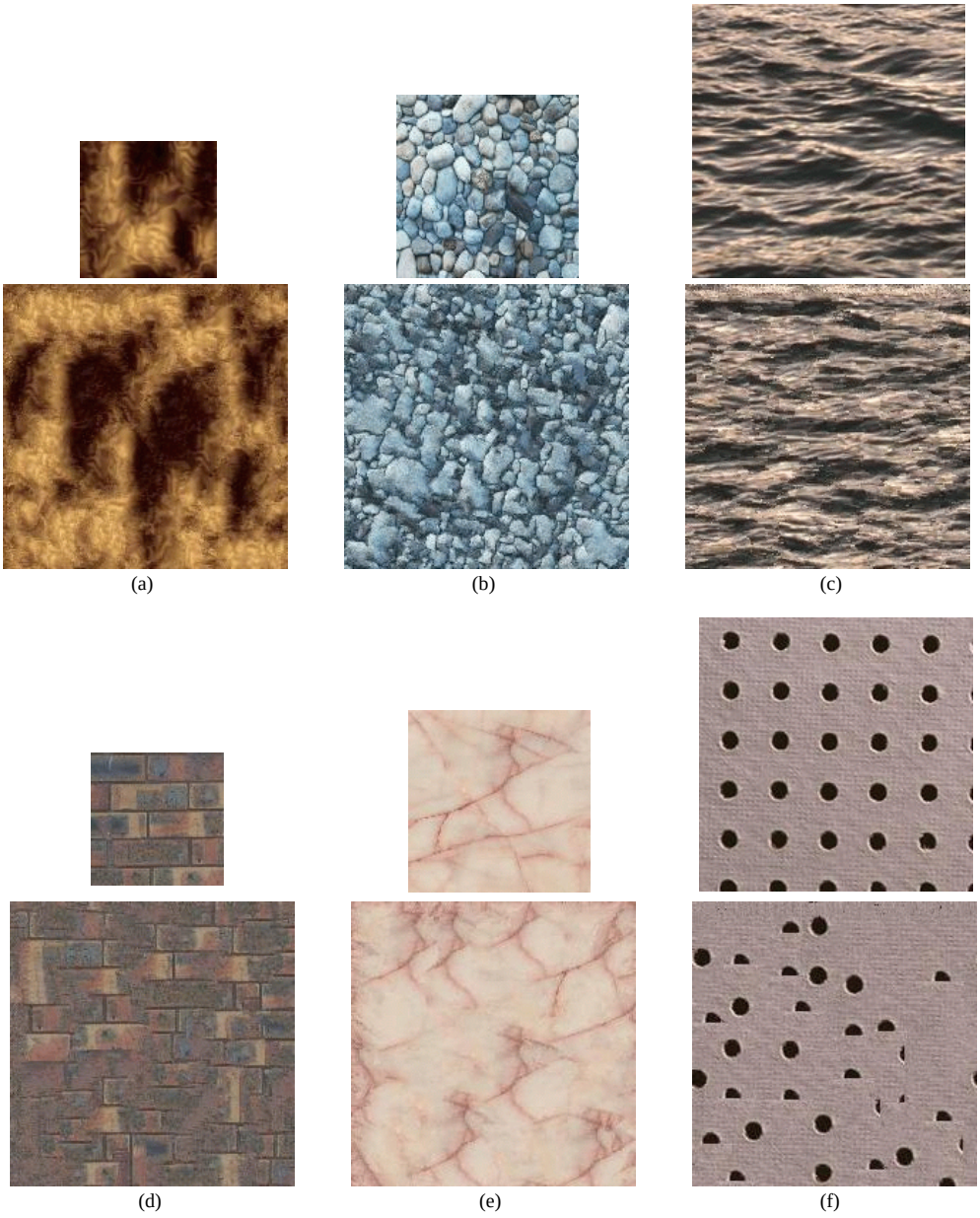

Fig. 4. Some results. Each pair contains the sample (top) and output (bottom). All output images are 200×200. Samples sizes are as follows: **(a)** and **(d)** 96×96, **(b)** and **(e)** 128×128, and **(c)** and **(f)** 192×192.