Developing Texture Images with Texture Generation and Syntheses

Daniela Ilieva
Department of Computer Science & Engineering at Technical University of Varna
1 Studentska street
Varna 9009, Bulgaria
ilievadaniela@mail.bg

ABSTRACT: *In this work, we addressed how to develop texture images which have both texture generation and text synthesis. We have utilized natural features like terrain, fire, clouds, water, fog and vegetation to develop the images with graphics. The small digital images sometimes do not have the visible boundaries and markings for which we have used texture synthesis. We have experimented with our procedures and found the results interesting and helped to generate the textures.*

Keywords: Texture Image, Texture Generation, Texture Synthesis

Received: 19 January 2023, Revised 3 March 2023, Accepted 12 March 2023

DOI: 10.6025/jmpt/2023/14/2/29-38

1. Introduction

An image texture is a set of metrics calculated in image processing designed to quantify the perceived texture of an image. Image texture gives us information about the spatial arrangement of color or intensities in an image or selected region of an image.

In 3D computer graphics, a texture is a digital image applied to the surface of a three-dimensional model by texture mapping to give the model a more realistic appearance.

In image processing, every digital image composed of repeated elements is called a "texture".

Texture can be arranged along a spectrum going from stochastic to regular.

Image textures can be artificially created or found in natural scenes captured in an image. Image textures are one way that can be used to help in segmentation or classification of images. To analyze an image texture in computer graphics, there are two ways to approach the issue: Structured Approach and Statistical Approach.

The image texture can be a photograph of a “real” texture or can be created by a procedural order.

2. Procedural Texture Generation

Procedural techniques [1], [2] are code segments or algorithms that specify some characteristic of a computer generated model or effect. Procedural texture does not use a scanned-in image to define the color values. Instead, it uses algorithms and mathematical functions to determine the color.

One of the most important features of procedural techniques is abstraction. In a procedural approach, rather than explicitly specifying and storing all the complex details of a scene or sequence, we abstract them into a procedure and evaluate that procedure when and where needed. This allows us to create inherent multiresolution models and textures that we can evaluate to the resolution needed.

We also gain the power of parametric control, allowing us to assign to a parameter a meaningful concept (e.g., a number hat makes mountains rougher or smoother). This parametric control unburdens the user from the lowlevel control and specification of detail.

Procedural models also offer flexibility. Procedural techniques allow the inclusion in the model of any desired amount of physical accuracy. The designer may produce a wide range of effects, from accurately simulating natural laws to purely artistic effects.

One major defining characteristic of a procedural texture is that it is synthetic - generated from a program or model rather than just a digitized or painted image.

The advantages of a procedural texture over an image texture are as follows:

• A procedural representation is extremely compact. The size of a procedural texture is usually measured in kilobytes, while the size of a texture image is usually measured in megabytes. This is especially true for solid textures, since 3D texture images are extremely large. Nonetheless, some people have used tomographic X-ray scanners to obtain digitized volume images for use as solid textures.

• A procedural representation has no fixed resolution. In most cases it can provide a fully detailed texture no matter how closely you look at it (no matter how high the resolution).

• A procedural representation covers no fixed area. In other words, it is unlimited in extent and can cover an arbitrarily large area without seams and without unwanted repetition of the texture pattern.

• A procedural texture can be parameterized, so it can generate a class of related textures rather than being limited to one fixed texture image.

The disadvantages of a procedural texture as compared to an image texure are as follows:

• A procedural texture can be difficult to build and debug. Programming is often hard, and programming an implicit pattern description is especially hard in nontrivial cases.

• Evaluating a procedural texture can be slower than accessing a stored texture image. This is the classic time versus space trade-off.

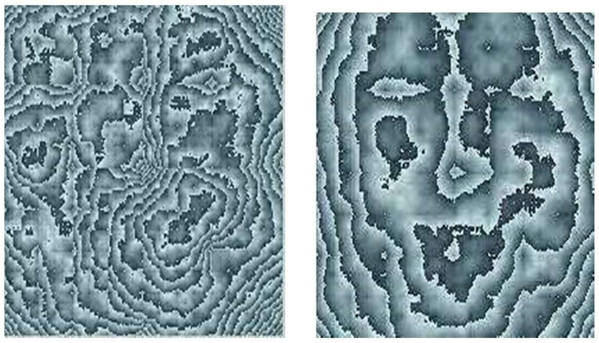
• Aliasing can be a problem in procedural textures. Antialiasing can be tricky and is less likely to be taken care of automatically than it is in image-based texturing.

To generate irregular procedural textures, we need an irregular primitive function, usually called noise. This is a function that is apparently stochastic and will break up the monotony of patterns that would otherwise be too regular.

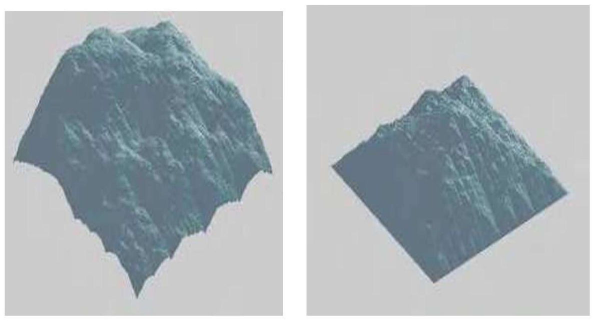
The obvious stochastic texture primitive is white noise*,* a source of random numbers, uniformly distributed with no correlation whatsoever between successive numbers.

Procedural texturing uses fractal noise extensively. The noise function simply computes a single value for every location in space. We can then use that value in literally thousands of interesting ways, such as perturbing an existing pattern spatially, mapping directly to a density or color, taking its derivative for bump mapping, or adding multiple scales of noise to make a fractal version. While there are infinitely many functions that can be computed from an input location, noise's random (but controlled) behavior gives a much more interesting appearance than simple gradients or mixtures of sines.

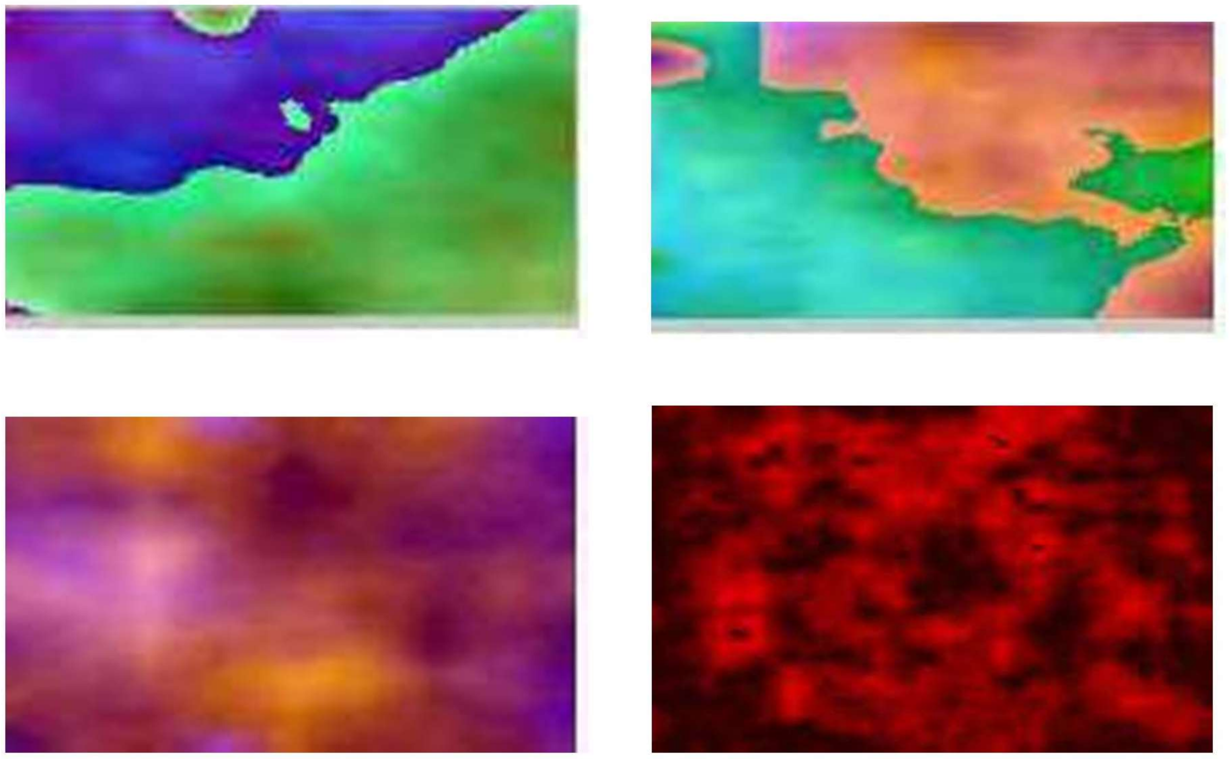
Below are shown the images (figure 1) procedurally generated with midpoint displacement procedure that can be used in terrain scenes, scenes with landscapes, clouds and meadows. The images (a) are created by a procedure and then the intensity of each point is used as a height for receiving 3D terrain (b). The images (c) are created with different color scheme and control parameter.

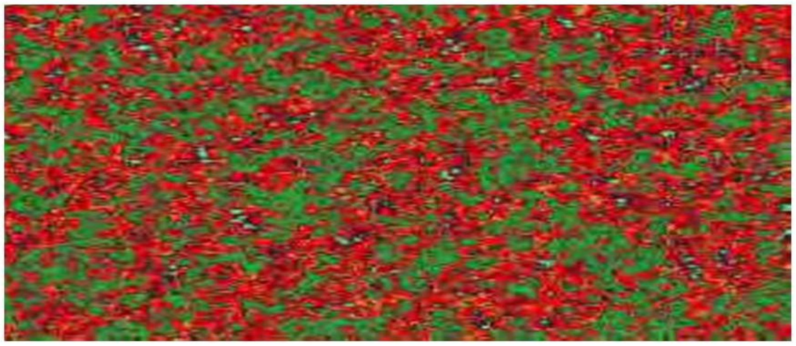


(a)



(b)





(c)

Figure 1. Generated texture images by generalized stochastic subdivision (midpoint displacement procedure)

3. Texture Synthesis

The texture synthesis problem may be stated as follows:

Given an input sample texture, synthesize a texture that is sufficiently different from the given sample texture, yet appears perceptually to be generated by the same underlying stochastic process.

Texture synthesis algorithms are intended to create an output image that meets the following requirements:

• The output should have the size given by the user.

• The output should be as similar as possible to the sample.

• The output should not have visible artifacts such as seams, blocks and misfitting edges.

• The output should not repeat, i. e. the same structures in the output image should not appear multiple places.

Like ithe most algorithms, texture synthesis should be efficient in computation time and in memory use.

3.1. Pyramid-Based Texture Synthesis

Pyramid-based texture synthesis [Heeger and Bergen] [3] is a very first 'fast' algorithm that generates a new texture by matching certain statistics with the training sample. In the method, a texture is decomposed into pyramid-based representations, i.e. an analysis and a synthesis pyramids are constructed from the training and the output textures respectively.

3.2. Multi-Resolution Sampling

Multi-resolution sampling [4], proposed by DeBonet, also constructs an analysis and a synthesis Guassian/Laplacian pyramids in texture synthesis. But it has two major improvements over the previous method. First, multi-resolution sampling extracts a set of more detailed and sophisticated image features by applying a filter bank onto each pyramid level. Second, multi-resolution sampling explicitly takes the joint occurrence of texture features across multiple pyramid levels into account, while the previous method processes each pyramid level separately.

3.3. Pixel-Based Non-Parametric Sampling

Pixel-based non-parametric sampling [5], [6], [7], constrains pixel sampling using a similarity metric defined with respect to a local neighbourhood system in an MRF (figure 2).

The method assumes an MGRF model of textures so that a pixel $i \in g$ only depends on the pixels in a local neighbourhood $N_i \in g$. The distance between neighbourhoods N_i and N_j, e.g., the sum-of-squaredifference (SSD), provides a metric of pixel similarity between *i* and *j* . To synthesise a pixel *i*, the algorithm searches for a pixel *j* from the training texture that minimises the distance between N_i and N_j, and then uses the value of pixel *j* for pixel *i* .

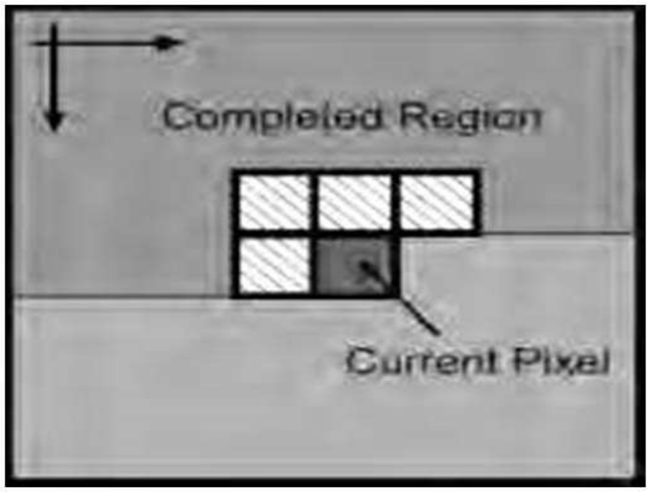


Figure 2. The '*L*'-shaped neighbourhood in a pixel-based nonparametric sampling (a 3X3 neighbourhood window in this case). Each square represents a pixel and the arrows indicate the synthesis is performed in a raster scanline order

3.4. Block Sampling
Block sampling [8] is a natural extension to the previous pixel-based methods, which improves time efficiency of the synthesis by using image blocks as basic synthesis units. So, instead of pixel by pixel, block sampling synthesises a texture on a block by block basis (figure 3).

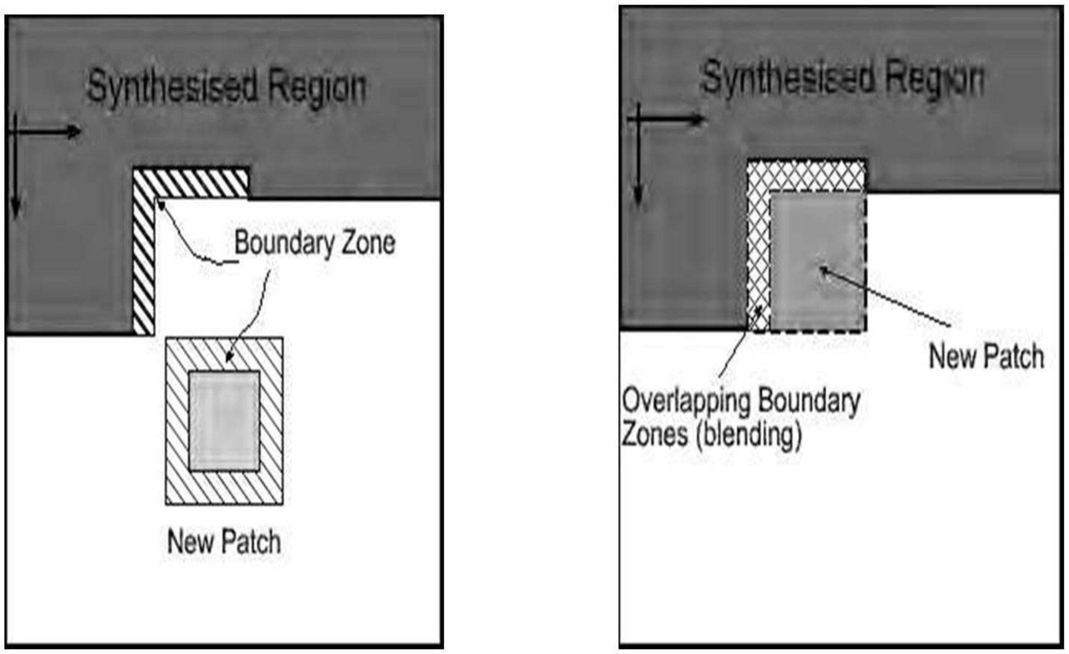


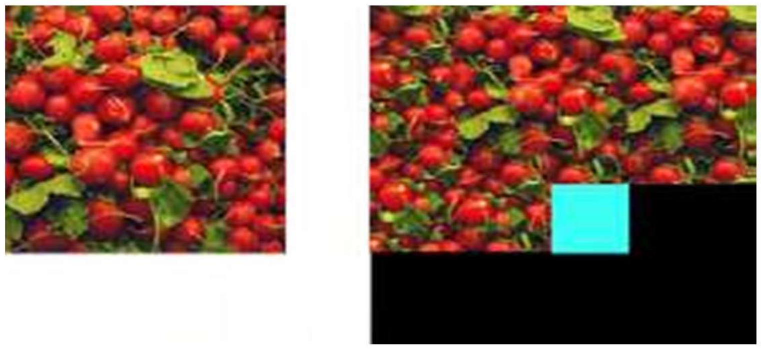
Figure 3. Patch-based non-parametric sampling: (a) The boundary zones in a new and a already synthesised patches. (b) The overlapping boundary zones after a new patch is placed into the output texture. The arrows indicate the synthesis is performed in a scanline order

3.5. Image Quilting

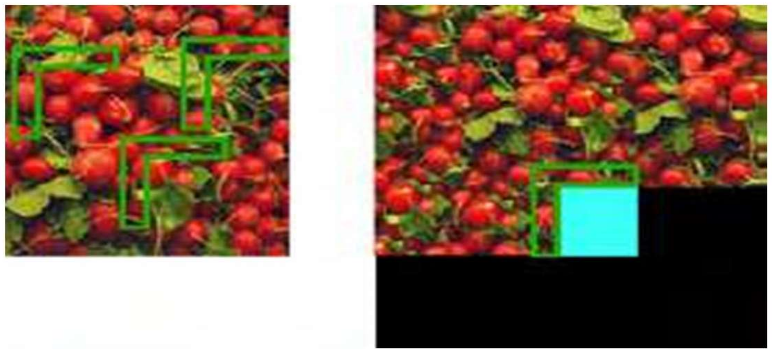
Image quilting [9,10] improves the patch-based nonparametric sampling by developing a more sophisticated technique to handle the boundary conditions between overlapped image patches. Instead of using the oversimplified blending technique, image quilting exploits a *minimum error boundary cut* to find an optimal boundary between two patches. An optimal cut defines an irregular path separating overlapping patches, so that each patch provides the synthetic texture only image signals on its side of the path.

Results of our algorithm based on block sampling procedure with image quilting (Figure 4):

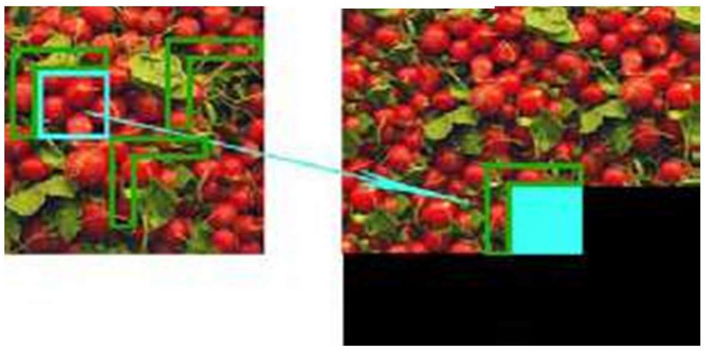
Step 1: Define patch that will be filled from the input image. It will be synthesized in step *M*



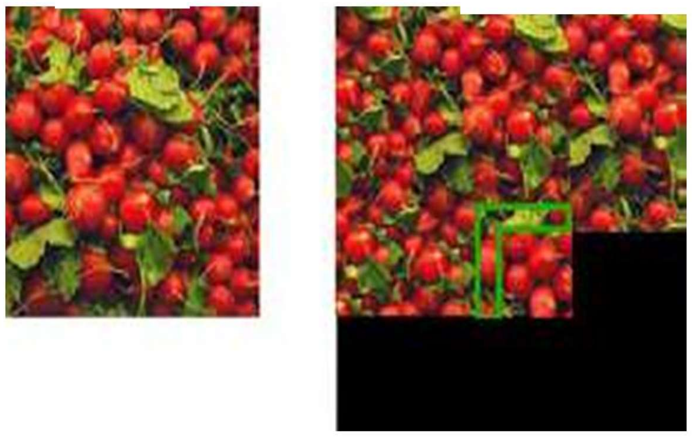
Step 2: Create a "neighbor" - an image formed around a patch.



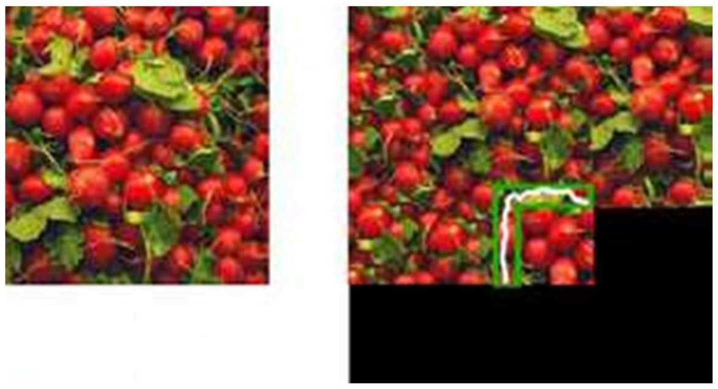
Step 3: Detecting matching "neighbors"



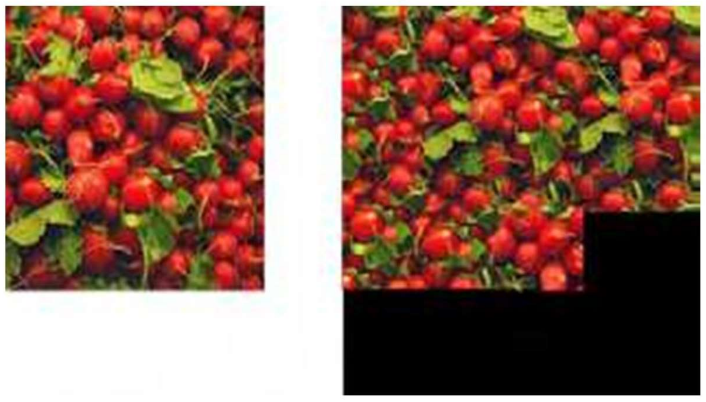
Step 4: Selecting the best of all "neighbors"



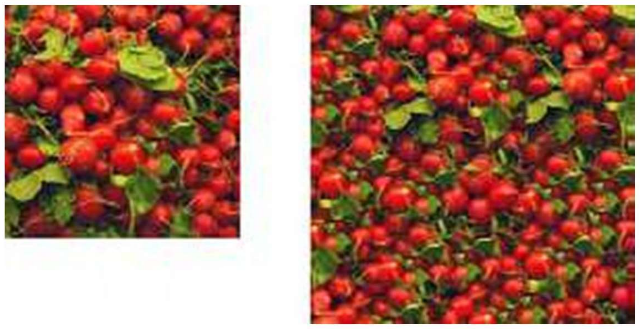
Step 5: Copy selected "neighbor" and patch of the original image.



Step 6: Implement a strategy to remove the gap



Step 7: Forming of the patch from step *N*



Step M: Synthesis the whole texture image

Figure 4. Algorithm for texture image synthesis finding an optimal neighbor

Algorithm to search for optimal "patch" is based on the following steps:

- Forming of piece with all-black color (Figure 5).
- Forming of mask image J_i

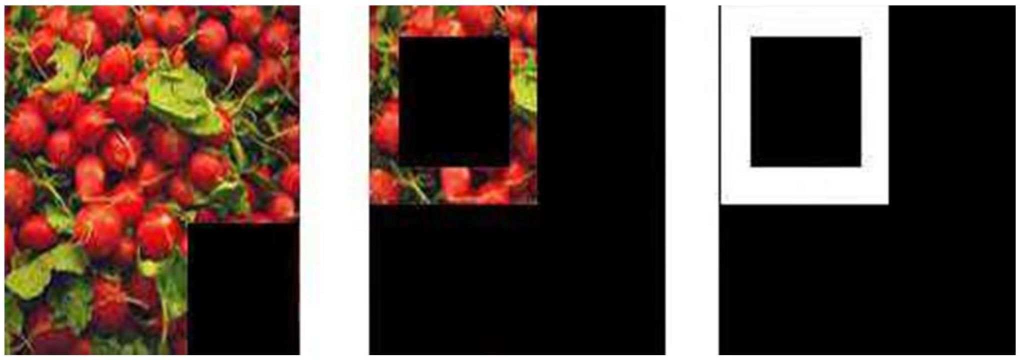


Figure 5. Algorithm to search optimal "patch"

Calculate (Equation 1) the error Ei ($x0$) between Ii and T (boot image) for each displacement $x0 = (\Delta x, \Delta y)$ of the input texture T (Figure 6).

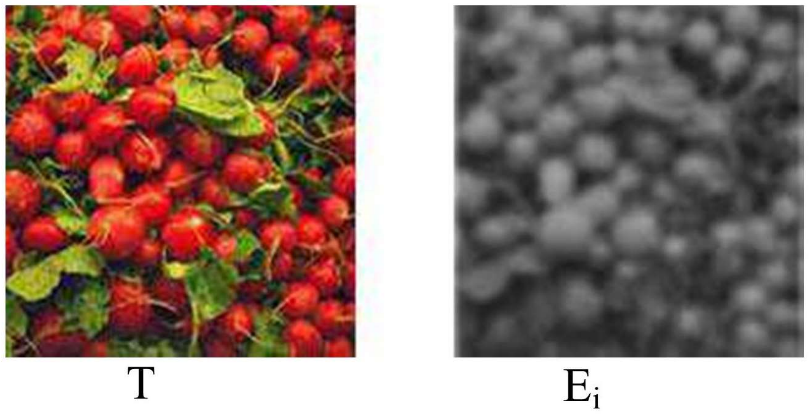


Figure 6. Determination of the error image

$$E_i(X_0) = \frac{1}{k_i} \sum_X \sum_c J_i(x)\, w_c\, (I_{i,c}(x) - T_{i,c}(x+x0))^2 \qquad (1)$$

Determination of error overlap of Si mask image *Ii* (Equation 2) and *Pi* selected piece (Figure 7).

$$S_i(X) = \sum_c J_i(x)\, w_c\, (I_{i,c}(x) - P_{i,c}(x))^2 \qquad (2)$$

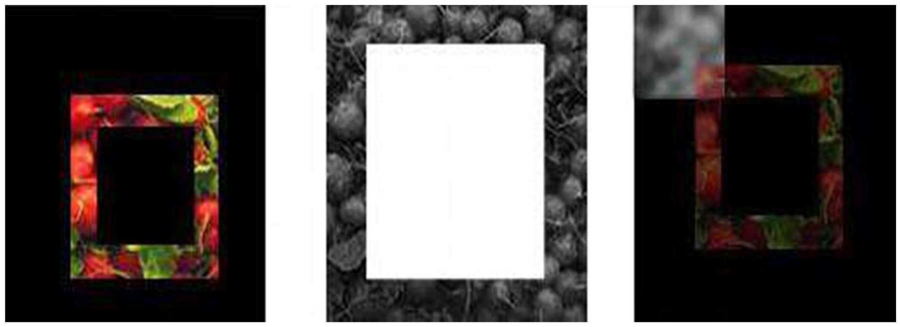
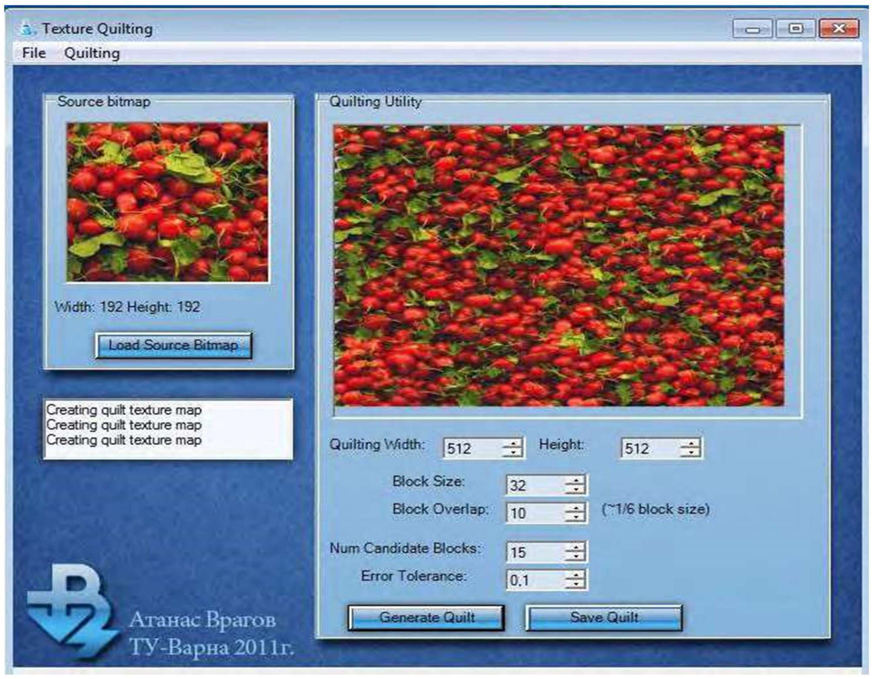


Figure 7. Determination of error overlap Si

When found the patch with a S_i = min to copy selected patch in the i-th image area.

• The algorithm is repeated until all texture image is created.

• Developed application is shown in Figure 8.



Width: 192 Height: 192

```
Creating quilt texture map
Creating quilt texture map
Creating quilt texture map
```

Quilting Width: 512 Height: 512

Block Size: 32

Block Overlap: 10 (~1/6 block size)

Num Candidate Blocks: 15

Error Tolerance: 0,1

Figure 8. Texture synthesis application

4. Conclusion

The results obtained are with satisfactory appearance and speed of preparation and can be used for computer graphics and visualization.

References

[1] Ebert, D., Musgrave, K., Peachey, D., Perlin, K. & Worley, S. (2003). *Texturing and Modeling. A Procedural Approach, 3rd edn. Morgan Kaufman Publishers*. ISBN: 1-55860-848-6.

[2] Schneider, P. and Eberly, D. (2002). *Geometric Tools for Computer Graphics*, 3rd edn. Morgan Kaufman Publishers.

[3] Heeger, D.J. and Bergen, J.R. Pyramid-based texture analysis/synthesis. *In: SIGGRAPH*, pages 229–238 (1995).

[4] DeBonet, J.S. (1997) Multiresolution sampling procedure for analysis and synthesis of texture images. *Computers and Graphics, 31(Annual Conference Series)*, 361–368.

[5] Efros, A.A. and Leung, T.K. (1999) *Texture synthesis by nonparametric sampling*. In: ICCV, 2, 1033–1038.

[6] Wei, L. and Levoy, M. (2000) Fast texture synthesis using treestructured vector quantization. In: Siggraph 2000, Computer Graphics. Proceedings (edited by K. Akeley). *Addison Wesley Longman: Reading*, USA, pp. 479–488. ACM Press / ACM SIGGRAPH.

[7] Ashikhmin, M. (2001) Synthesizing natural textures. *In: The Proceedings of 2001 ACM Symposium on Interactive 3D Graphics*, pp. 217–226.

[8] Liang, L., Liu, C. and Shum, H.Y. (2001). *Real-Time Texture Synthesis by Patch-Based Sampling*. Technical Report MSR-TR-2001-40. Microsoft Research.

[9] Efros, A.A. and Freeman, W.T. (2001) Image quilting for texture synthesis and transfer. *In: Proceedings (edited by E. Fiume), SIGGRAPH 2001, Computer Graphics*, 341–346. ACM Press / ACM SIGGRAPH.

[10] Zhou, D. and Gimel'farb, G.L. (2004) Texel-based texture analysis and synthesis. *In Proc. Image and Vision Computing,* (New Zealand) (IVCNZ 2004). Akaroa: New Zealand, 215–220.