Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 35 (2012) 210 – 216

Procedia Engineering

www.elsevier.com/locate/procedia

International Meeting of Electrical Engineering Research
ENIINVIE 2012

Multi-seed texture synthesis to fast image patching

Jose M. Celaya-Padilla[a], Carlos E. Galvan T.[b], J. Ruben Delgado C.[c], Issac Galvan-Tejada[d], Ernesto Ivan Sandoval[e]

[a] *Tecnolgico de Monterrey, Campus Monterrey*
Bionformatic Department
[b] *Universidad Autonoma de Zacatecas*
Unidad Academica de Ingenieria Electrica
[c] *Tecnolgico de Monterrey, Campus Monterrey*
Autonomous Agents in Ambient Intelligence
[d] *Universidad Autonoma de Zacatecas*
Unidad Academica de Ingenieria Electrica
[e] *Universidad Autonoma de Zacatecas*
Unidad Academica de Ingenieria Electrica

Abstract

Actually we have a large number of devices which can take pictures, from a digital camera to a cell phone, one of the problems is that usually at moment to take the picture we don't see some errors that can be present in the image until we observe it in a computer or printed image, this is a problem, because is almost impossible to recapture the moment in a new photo, to find a solution for this problem come the need implement a method to correct some of the defects that appear. In this paper we compare two texture synthesis methods and propose an algorithm to patch an original image, using a multi seed generated texture image.

Keywords: Synthesis, Texture, Image, Patch

1. Introduction

One topic in Image processing, is Texture synthesis which have multiple applications in digital media, like games, photos, computer aided design (CAD), and are extensively used to increase realism on surfaces[1]. The texture synthesis problem can be formulated as follow: Given a finite sample from a texture seed (an image), the goal is to synthesize other samples from the same texture[2]. There are many approaches to tackle this problem, Efros et al [3] propose one of the most famous methods due the simplicity of the algorithm and the excellent results. Therefore, an efficient texture synthesis method is demanded in real-life applications[4]. In texture synthesis there are two principal methods, pixel-based methods and

Email address: jose.cpadilla@gmail.com (Jose M. Celaya-Padilla)

1877-7058 © 2012 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
doi:10.1016/j.proeng.2012.04.182

patch-based methods which are generally more efficient and the features inside patches can be well kept, but inconsistencies may exist between different patches[5]. Our work is motivated by the approach presented in [3],[2] but principally by [6] because they use texture synthesis to fix images. We propose an easy but functionally method using Multi-Seed Texture Synthesis algorithm to patch a given image.

This paper is organized as follow. A theoretical background for this work and a review of some existing methods for texture synthesis is given in Section 2. In Section 3 we describe, analyze and compare two fundamental methods for texture synthesis. Section 4 presents our contribution and algorithm to do a multi seed texture synthesis and patching. Experimental results are presented in Section 5. Finally, Section 6 presents conclusions and future work.

2. Background and Current State of the Art

Given the importance in these days of digital images, there is a lot of work on texture synthesis, because is required in many fields like games industry, film industry and even by the common user to fix images taken with digital cameras or smartphones.

To achieve this task there are many algorithms that model textures such as Markov Random Fields (or in a different mathematical form, Gibbs Sampling), and generate textures by probability sampling [7], these complex algorithms generates the texture synthesis pixel by pixel, which carry a lot of computational cost. Another approach to texture synthesis is using patches from the original image to synthesis textures like in [8], [9] and [5] which is faster than pixel-based algorithm due the transfer of a whole patch each time and not just a pixel. One of the most important problems to tackle in the patch approach is the disjointedness in the edges of patches. Efros et al. in [3] propose an overlap region between patches to smooth the edges, this simple idea works amazing, and nowadays is one of the most used algorithm in texture synthesis, other works just propose a different way of calculate the overlap region. [10] propose the use of tile sets, this approach operates by extracting some sample patterns randomly from input texture and introduce a fast search algorithm to choose the optimal cutting curve. [5] propose a improved graph cuts which get better results than other classical methods finding the minimum cost path to cut the patch but increase the computational cost. Since the patching approach works faster and work for many kind of textures (stochastic and structured) texture synthesis to fix images like in [6] use a patching approach.

3. Random square blocks and Efros-freeman method

In this section is described and analyzed the performance of two fundamental methods in texture synthesis.

3.1. Random square blocks

Developing a robust and general texture synthesis algorithm has been proved that is a difficult task by a huge number of approaches already done without a successfully solution for all the problems. Xu et.al. [5], inspired by the Clone Tool in PHOTOSHOP, propose a much simpler approach yielding similar or better results. The idea is to take random square blocks from the input texture and place them randomly onto the synthesized texture. The statistics being preserved here are simply the arrangement of pixels within each block. While this technique will fail for highly structured patterns (e.g. a chess board) due to boundary inconsistencies, for many stochastic textures it works remarkably well[5].

We summarize the algorithm in the following pseudo code

```
1. for i to image_width
2.   for j to image_height
3.     output_image = input_image[ random_block(x,y) ];
4.     j = j + block_height;
5.   i = i + block_with;
```

First we calculate the dimensions of the destination image that will contain the synthesis texture, and then we generated a nested cycle in order to fill the entire destination image. We have to increase the counter variables in a reason of the block size, and then we take a random rectangle from the original image and put this rectangle in the next location available in the destination image.

3.2. Efros-freeman method

With the growth of computing power, spread and use of digital cameras, the need of better results correcting images has been exploited so nowadays have emerged a lot of algorithms for topics related to image processing, and, texture synthesis is one of them. The algorithm proposed by Efros-freeman is a more complex algorithm that get better results than Random square blocks in images with a highly structured patterns (like a chess board).

We summarize the algorithm in the following pseudo code

```
1. for i to image_width
2.   for j to image_height
3.     ov = overlap(previus_mask,current_mask);
4.     texture_block = find_nearest_overlap(input_image,ov);
5.     output_image = copy_block(texture_block);
6.     j = j + texture_block_height;
7.   i = i + texture_block_with;
```

As we can see is the same structure like the Random square blocks algorithm, but in the nested cycles we have two principal functions that make this algorithm better. The first, function overlap, which is a function that find the region of overlapping between immediately past window and the current window, once this region is funded the algorithm call the next important function in this proposed method, find_nearest_overlap, this function in fact, try to find the most similar overlap region in the original image. To find this region in the original image Efros-freeman just mention that we need a method to do this task, do not propose a specific method. In the literature we can find a lot of approaches to solve this task like pixel-based methods, probabilistic methods, brute force methods and specific cross correlation methods. We chose to use cross correlation method because in the literature is one of the most used methods due the simplicity to program and the efficiency that it has.

Once we have this tool to find the most similar overlap region, the next steps are almost the same that in the Random square blocks algorithm. We cut that region and put it in the next available location in the destination image.

3.3. Efros-freeman and Random square blocks comparation

In order to evaluate the effectively of both algorithm we first test in a independent way the two synthesis algorithms using MatLab running on a MacBook with a core i5 processor at 2.4 GHz, 4 Gb RAM, MAC OS Lion 10.7 and in a HP Pavilion dv2000 with a T1700 core duo processor at 1.7 GHz, 3 Gb RAM running Linux Ubuntu 11.04.

Figure 1 shows the texture synthesis result from both algorithms, we can see that Efros-freeman gives us a better result to the view in contrast with Random square blocks but Efros-freeman take much time to be generated because the algorithm complexity, hence Random square blocks is better if we want a fast synthesis with less details like in real time games or in stochastic images when there is not well defined patterns.

Figure 2 shows the texture synthesis result from both algorithms in a more complex image, due the pattern, Efros-freeman give us an excellent result and in almost the same time than Random square blocks because as this picture has a definite pattern is easy to find a overlap region.

4. Fast texture synthesis and patch method

In this section is described the texture synthesis and patching method that we propose.

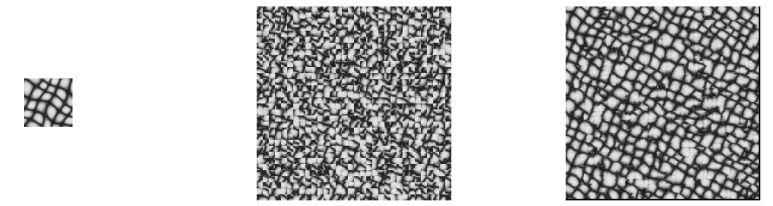


Fig. 1. Imagen generated by the texture synthesis algorithms using 5 pixels block (a) Sample texture; (b) Random square blocks; (c) Efros-freeman

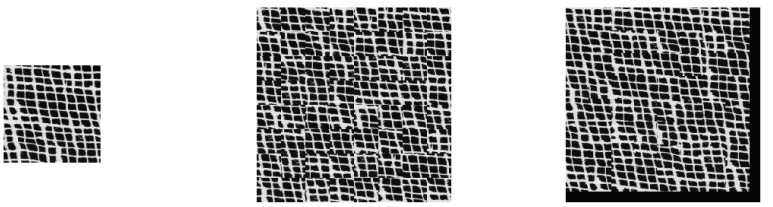


Fig. 2. Imagen generated by the texture synthesis algorithms using a 10 pixels block (a) Sample texture; (b) Random square blocks; (c) Efros-freeman

4.1. Patching and Proposal method

The texturing patching process includes a variety of issues such as access, sampling, and filtering, texture patching is the process to translate one texture to an image, this is commonly used to extract objects from an scene, and fill full the empty area with some texture that blend with the rest of the image, this can be found widely on animated movies, videogames and digital content. The mapped image, usually rectangular, is called a texture map or texture, and is used to generate a new texture, of N by M size, which fill full the image, in order to do that, we developed and strategy to over come this problem, first we find the area that we want to fill, in order to do that, we search for the specific area of interest, in this case, we arbitrary edited an image to draw a square figure, with a non common color, such as Cyan, or Magenta, this area is showed in (figure 3), once the area is detected, the dimensions of the local area are obtained and used to calculate the number of different mask's to be extracted, this is done by diving the area's high by the mask high, once we know the number of mask's we extract a maskj of the adjacent area, this maskj, will work as seed, to generate a texture area, after the new texture is generated we transfer the mask to the destination image, after, we extract a new maskj+1, an repeat the process until the whole destination image is completed, the representation of this algorithm is presented as follow.

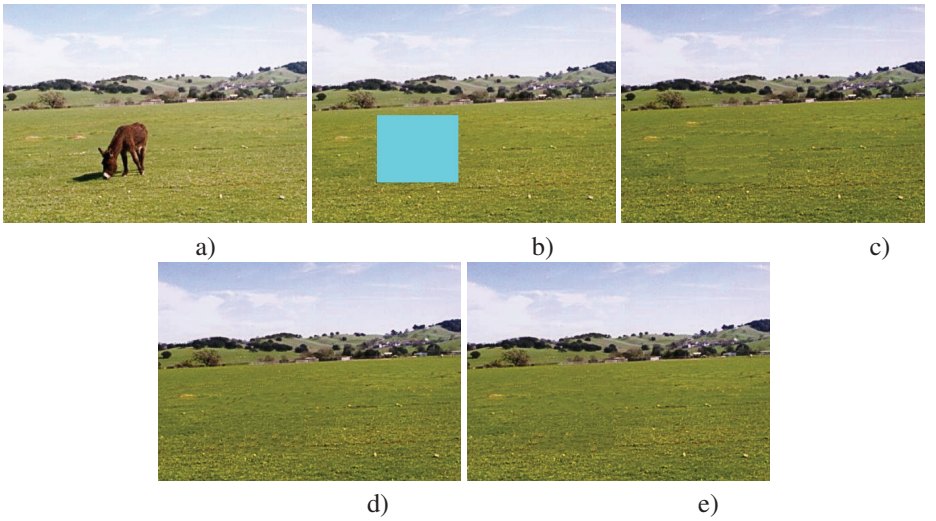
```
1. for i to image_with
2.  for j to image_height
3.   If current_position = area_of_interest then
4.    seed = find_adjacent_seed(i,j);
5.    texture_block = Generate_texture(seed);
6.    output_image=Apply_Texture_mask(input_image,texture_block);
7.   end
```

a) b)

Fig. 3. Test images (a) Original Image; (b) Edited input image

5. Results

In order to evaluate the efectivity of the proposed algorithm to patch images we used MatLab running on a MacBook with a core i5 processor at 2.4 GHz, 4 Gb RAM and Mac OS Lion 10.7 and in a HP Pavilion dv2000 with a T1700 core duo processor at 1.7 GHz, 3 Gb RAM running Linux Ubuntu 11.04.



a) b) c)

d) e)

Fig. 4. Results of using our proposal (a) Original Image;(b) Edited image; (c) Efros Patch; (d) Random square blocks; (e) Our Poposal

The algorithm was able to find the previously designated area and generate the best texture nearest to the location, after the texture is generated and applied to the area, the overall result was very good especially in stochastic backgrounds in some cases as is showed in the figure 4 the result image showed a smooth transaction between the background and the patched area better than using just a single seed Efros texture synthesis.

Figure 5 shows the algorithm performance in a more complex image due the gradient generated by the shadows in the sand, we can see a darker square in the right bottom of the mask square, but despite the quality of the patch is good enough when is not needed a lot of details and better than single seed Efros texture synthesis in which we can see clearly a square where the synthesized texture is used.

6. Conclusions and future work

The synthesis of texture is an important aspect of computer graphics, in addition to a wide range of applications in image processing. However, it is difficult to develop an overall process for texture synthesis.

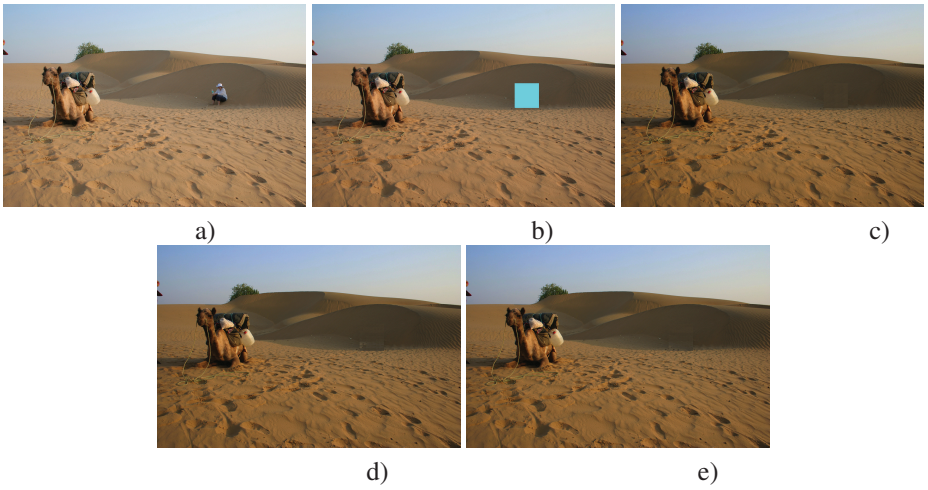


Fig. 5. Results of using our proposal in a complex image (a) Original Image;(b) Edited image; (c) Efros Patch; (d) Random square blocks; (e) Our Poposal

This work analyzes how the different algorithms affect the synthesized speed and quality, and considers texture to be a nature stochastic image, the first algorithm, despite of being simple and basic, delivery good results when applied to images containing stochastic textures. While the second algorithm is more complex, but delivers an excellent texture results with a wide range of seeds, we applied our multi seed patch matching method in several practical images and all the experimental results are very convincing and promising. As future work we propose use parallels texture generation for patching, and using GPU acceleration, the proposal of an algorithm that automatically select a window-size in non-square windows for elongated textures, also improve the smoothness of the result texture in patched images using gradient direction based algorithms.

7. Acknowledgement

The authors thank Movie.com for partially financing the research. Jorge I. Galvan-Tejada, Carlos E. T. Galvan, thank PROMEP for the support of the project, to Michael Day for the images used in the project.

References

[1] L. V. B. G. H.-Y. S. Jingdan Zhang, Kun Zhou, Synthesis of progressively-variant textures on arbitrary surfaces, ACM Transactions on Graphic 22(3) (2003) 295–302.
[2] A. Efros, T. Leung, Texture synthesis by non-parametric sampling, in: Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, Vol. 2, 1999, pp. 1033 –1038 vol.2. doi:10.1109/ICCV.1999.790383.
[3] W. T. F. Alexei Efros, Image quilting for texture synthesis and transfer, 2001.
[4] X. Changzhen, G. Fenhong, Fast texture synthesis via patch-based circular linked lists, in: Communications, Computers and Signal Processing, 2009. PacRim 2009. IEEE Pacific Rim Conference on, 2009, pp. 545 –550. doi:10.1109/PACRIM.2009.5291313.
[5] K. Zou, Y. Li, Z. Li, R. Li, X. Xu, Improved graph cuts for patch-based texture synthesis, in: Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on, 2009, pp. 122 –125. doi:10.1109/ICCSIT.2009.5234981.
[6] C. Xiao, M. Liu, N. Yongwei, Z. Dong, Fast exact nearest patch matching for patch-based image editing and processing, Visualization and Computer Graphics, IEEE Transactions on 17 (8) (2011) 1122 –1134. doi:10.1109/TVCG.2010.226.
[7] M. L. Li-Yi Wei, Fast texture synthesis using tree-structured vector quantization, Gates Computer Science Building, Stanford.
[8] J.-F. Wang, H.-J. Hsu, H.-M. Wang, Constrained texture synthesis by scalable sub-patch algorithm, in: Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on, Vol. 1, 2004, pp. 635 –638 Vol.1. doi:10.1109/ICME.2004.1394272.

[9] X. Changzhen, G. Fenhong, Z. Jiancheng, Q. Dongxu, Patch map for fast texture synthesis, in: Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on, 2007, pp. 501 –504. doi:10.1109/PACRIM.2007.4313283.

[10] G. Xu, S. Ma, Robust tile-based texture synthesis using texture element, in: Electronics and Information Engineering (ICEIE), 2010 International Conference On, Vol. 2, 2010, pp. V2–179 –V2–183. doi:10.1109/ICEIE.2010.5559738.