Bio-Medical Materials and Engineering 24 (2014) 2219–2227
DOI 10.3233/BME-141034
IOS Press

GPU-based skin texture synthesis for digital human model

Zhe Shen[a,*], Lili Wang[a] , Yaqian Zhao[a], Qinping Zhao[a] and Meng Zhao[b]
[a]*State Key Laboratory of Virtual Reality Technology, Beihang University, Beijing, China*
[b]*Northeastern University at Qinhuangdao, Qinhuangdao, China*

Abstract. Skin synthesis is important for the actual appearance of digital human models. However, it is difficult to design a general algorithm to efficiently produce high quality results. This paper proposes a parallel texture synthesis method for large scale skin of digital human models. Two major procedures are included in this method, a parallel matching procedure and a multi-pass optimizing procedure. Compared with other methods, this algorithm is easy to use, requires only a small size of skin image as input, and generates an arbitrary size of skin texture with high quality. As demonstrated by experiments, the effectiveness of this skin texture synthesis method is confirmed.

Keywords: Texture synthesis, virtual human, parallel computing, L2 normal form

1. Introduction

Digital human models are widely used in many medical applications, including virtual human, anatomical teaching, pharmacological experiment, athletic training, and virtual surgery. However, digital human models are generally based on human body slices [1], such as the Visual Human Project in American and Virtual Human in China. As digital human models focus on body structures, body details, including skin, hair, brows and eyelashes, which affect the actual appearance are generally ignored. When human body slices are used, the integrity of skin cannot be maintained, thus it needs to be generated by photographs or by hand drawings. As either way is tedious, digital human's skin would always show a uniform flat color.

In computer graphics, texture is used to produce realistic model appearance with little cost [2]. Sample based texture synthesis reconstructs and reorganizes the sample texture, which could synthesize a new texture image with high resolution while maintaining the visual effect.

In this paper, skin is treated as a texture sample, and we propose a GPU (Graphics Process Unit) based multithreads parallel algorithm for skin texture synthesis. Specifically, skin textures are synthesized using a two-pass algorithm. In the first pass, a patch based synthesis approach is used to synthesize the coarse texture. In the second one, the textures are optimized by a self-adaptive non-parametric global texture optimization algorithm. For both passes, a universal multithread parallel algorithm is employed to compute the best match neighborhood in sample texture using GPU. Besides, a novel re-

*Corresponding author: Zhe Shen, State Key Laboratory of Virtual Reality Technology, Beihang University, Beijing, China. Tel.: +86 135 8179 4654; Fax: +86 10 8233 9903; E-mail: shenzhe@vrlab.buaa.edu.cn.

weighting scheme ensures fast convergence of the energy function. As confirmed by the result, this method is effective for not only skin texture, but also some other physiological elements.

In Section 2, a brief introduction will be given on the development of texture synthesis. Section 3 then presents the GPU-based skin texture synthesis algorithm in detail. Results of the algorithm are given in Section 4 with conclusions summarized Section 5.

2. Related works

2.1. Per-pixel texture synthesis

A non-parameter sample method proposed by Efros [3] describes the texture by MRF (Markov Random Field). In MRF models, the color of a certain pixel is only affected by its neighboring pixels, which acts as the theoretical basis of neighborhood texture synthesis. Later, Wei improved Efros's method by using an *L*-shape neighborhood to find the best match pixel [4,5]. Wei's method searches a match pixel using the known neighboring pixels. Meanwhile, tree-structured vector quantization is also employed to reduce the synthesis time.

Ashikhmin proposed the 1-Coherence algorithm to accelerate the texture synthesis per pixel [6]. For an undetermined pixel, there is a high probability that the best match pixel is the pixel close to the last best match pixel and thus removing pixels with low match probability could speed up the match process. Zelinka proposed a jump map method based on K-Coherence algorithm [7], which divides the synthesis process into two sub-procedures, texture analysis and texture synthesis.

Although per pixel texture synthesis is easy for implementation, yet it takes a long time when applied in large textures. As a result, for textures with structures, it is hard to maintain the structural information using per pixel texture synthesis.

2.2. Per-patch texture synthesis

As an improvement from per pixel texture synthesis, per patch texture synthesis was later proposed. Instead of finding a best match pixel, per patch method now finds a block of pixels.

The image quilting texture synthesis method [8] proposed by Efros synthesizes the texture by searching the best match patch in a line scanning order, and computes a minimal cost path by dynamic planning between the new texture patch and the old one. Although a satisfactory effect can be achieved, this method is low in synthesis processing. By expanding the image quilting method, Schlömer proposed the semi-random texture synthesis later [9]. Besides, Cohen proposed the Wang tiles method [10], which combines the Wang tile algorithm in theory of computation and image quilting. Wang tiles method defines a series of square texture blocks, with each edge of a block marked in a certain color and edges with the same color could combine seamlessly. During the synthesis, a block which meets the constraint of color is chosen to form a big image.

However, although per patch texture synthesis is much faster than the per pixel method, it is still time consuming, and the seam between two adjacent patches needs to be handled.

2.3. Global texture optimization

Kwatra proposed the global texture optimization method [11], which uses two energy functions, Exception and Maximization, and thus converts texture synthesis into energy function minimization. To

minimize, the E and M functions are executed iteratively until the energy function is below a threshold. Therefore, a low energy value could make the texture look like the sampler.

Inspired by Kwatra's method, coarse textures can be refined by global optimization algorithm without iterative execution. Instead, with the statistical data, weight is defined to ensure quick convergence of the energy function.

2.4. GPU-based texture synthesis

Texture synthesis is a computation-intensive work. With the development of hardware and software, parallel computing is becoming more and more powerful. It is an inevitable choice to implement texture synthesis in a parallel style.

Lefebvre proposed a parallel controllable texture synthesis method [12] based on Wei's method. Image pyramids are constructed for sampler and target texture with the latter being divided into four groups by the sampled interlaced isolation column and row. This algorithm can be implemented in parallel within groups and in series between groups. Afterwards, Lefebvre proposed appearance vector space texture synthesis [13], in which the vector contains more information due to its large size of pixel neighborhood, and thus is beneficial for pixel match.

Risser proposed structured image hybrids synthesis method [14], which improves the coordinates stabilization, and made a global structure image blending. Huang proposed a two-stage large scale texture synthesis method based on random search [15]. The first (initial) stage will fill the target texture by random texture patch, while the second (iterate) stage implements the stochastic searching and texture patch broadcasting in parallel to adjust the texture. This method is also implemented using GPU, but it still needs several seconds to complete the work.

Chen proposed a method to synthesize large textures using GPU in real-time [16]. This method first analyzes the periodicity and the optimized patches size of exemplar before distributing patches with and filling in the vacant regions. Chen later improved the method by generating bigger patches using patches from the synthesized part [17]. Interactive operations were also added into texture synthesis to integrate user control information and the structure of exemplar. The method was then implemented on GPU to guarantee an interactive computation time [18]. Wang presented a just-in-time texturing method. The method implemented on GPU and modeling a texture patch as an umbrella to address the texture diversification and mipmapping problem in a short time [19].

In this paper, we combine the patch based synthesis method and the global optimization method. Firstly, a coarse texture using patch based method is synthesized, which is then refined by global optimization method. With both steps implemented on GPU, a common size texture could be worked out within a second.

3. Texture synthesis on GPU

Texture synthesis begins with initializing the target texture with a white Gaussian noise. A patch from target texture is then selected in line scanning order, traversing the sampler to find a best match patch. Once the best match path is found, it is copied to the target texture. These steps are repeated iteratively until each target texture is matched with a patch. Finally, global optimization method is employed to refine the entire target textures. Throughout the synthesis process, parallel computing is used to search for the best match patch and refine the global texture.

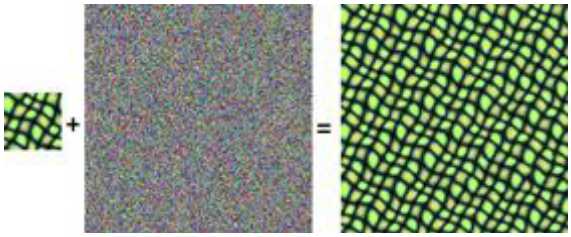


Fig. 1. Texture synthesis using white Gaussian noise.

3.1. Initial texture

Textures are similar locally but show randomness globally. To make the texture features random, we initialize the target texture with white Gaussian noise (Figure 1). The purpose of this step is to convert the random texture into an image like the exemplar.

The first best match patch is completely random, as the initial texture is random. The succeeding match patchs will also be random due to the accumulation of random data. However, in local areas, we use patches selected in the exemplar to compose the texture, so it is similar in small scales, which ensures the target texture is similar locally while random globally.

3.2. Texture patch matching

Texture patch matching is implemented in a line scanning order. A texture patch t_i with size $N*N$ in target texture is selected, followed by traversing the exemplar S to compute the L_2 distance of each patch s_i with size $N*N$ corresponding to t_i as in Eq. (1).

$$BestMatchBlock = \min(\forall_{s\in S} \sum_{i=1}^{N*N} (s_i - t_i)^2) \tag{1}$$

Short distances indicate small differences, which implies that two patches are more identical. After obtaining all distances in the exemplar, the patch with the shortest distance is chosen as the best match, and later copied into t_i as the synthesis result.

After finishing one patch in target texture, it will move on to the next undetermined patch in a line scanning order. Please be noted that the displacement distance D should be smaller than N, which could guarantee the continuity between patches as well as the data dependency of the subsequent patches. As a result, an appropriate relation between N and D could ensure the stability of the synthe-

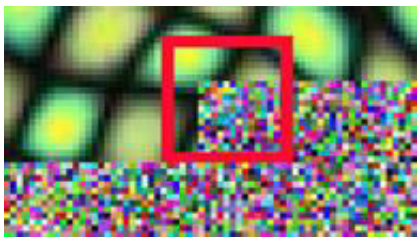


Fig. 2. Texture synthesis by blocks.

sis result. And in this paper, we choose N=2D.

As there are only horizontal movements in a row, we only have vertically overlapped areas in the first row. However, in other rows, overlapped areas exist both horizontally and vertically (Figure 2).

3.3. Self-adaptive global optimization

When searching for the matching patches, only part of the decided information is needed, and the remaining information is random data. There is some high frequency information between two decided patches, which makes the texture appears as if it consists of blocks. As a result, global optimization algorithm will be used to adjust the result with the energy function defined as Eq. (2).

$$E_t(x;\{z_p\}) = \sum_{p\in X} \left\| x_p - z_p \right\|^2 \tag{2}$$

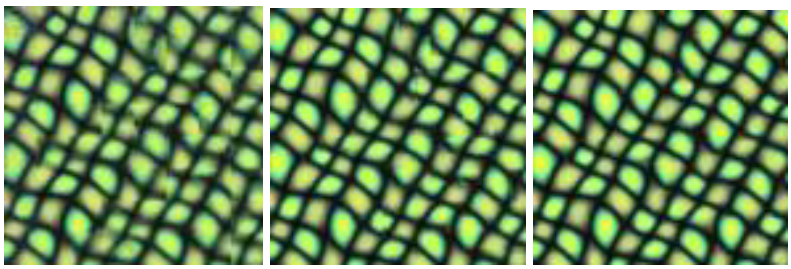
where X stands for the target texture and Z stands for the sample texture. x stands for the vectorized X, i.e., x is formed by concatenating the intensity values of all pixels in X. Therefore, the global energy is the sum of the L_2 distance in the neighborhood.

In this paper, a self-adaptive optimization is proposed to accelerate synthesis. The probability distribution function of the best match patch is firstly computed. If there is a region with high probability, it could be a repeatable texture with similar structures, which only needs one time optimization.

To ensure fast convergence of the energy function, high frequency information can be eliminated by adding a weight based on Eq. (1), as show in Eq. (3).

$$\mathrm{E_t(x;\{Z_p\}} = \sum_{\mathrm{p\in X}} \omega_{\mathrm{p}} \left\| x_p - \mathrm{z_p} \right\|^2 \tag{3}$$

where ω_p stands for the weight, which is given by $\omega_p = P(x_p)\left\| x_p - z_p \right\|^{r-2}$ with $r = 0.8$. $P(x_p)$ is the probability of finding the best match in a high frequency success match region for the current patch.



(a) Without optimization (b) 1 time optimization (c) 2 time optimization

Fig. 3. Results of different optimization times.

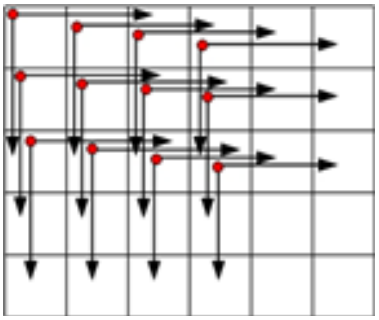


Fig. 4. Parallel structure sketch map.

After each optimization process, the size of the neighborhood will be reduced to a quarter, which ensures that the optimization can be implemented in a smaller region with better results. As shown in Figure 3, Figure 3(a) gives the coarse texture synthesized by patch based method, in which the borders of the patches can be seen. Figure 3(b) shows the texture with one time optimization, which is better than (a) but still contains some discontinuous pixels. Figure 3(c) gives the texture after a two time optimization, whose quality could satisfy the requirement of application. As a result, the minimum number of optimization is chosen to be three. As shown in Kwatra's method, r is set to 0.8, which guarantees stable results.

3.4. GPU-based algorithm analysis and design

Although parallel algorithm requires low data dependency, yet texture synthesis based on MRF model is highly data dependent in which the color of subsequent pixels are determined by the previous pixels. As a result, the analysis and design of parallel algorithm should maintain data consistency.

Task partitioning of texture synthesis indicates that searching the best match patch is computation intensive and is the core work of the algorithm. More importantly, the searching procedure could be completed separately, which allows the computing the best match patch in parallel using the proposed method as shown in Figure 4.

At the beginning of the parallel algorithm, we will launch equal number of threads as the number of pixels in the sample texture. Each thread is corresponding to a certain pixel, and it is used to compute the L_2 norm of the neighborhood starting from the corresponding pixel. Once the results are obtained, we will compute the best value by data reduction on GPU, in which each thread compares the two values and records the smaller one. This process is repeated until only one value is left. The time complexity of multi-thread data reduction is $O(\log_2 n)$, which is more efficient than that on CPU.

For *N*N* sample texture, the traditional algorithm needs to compute *N*N* times to find the best match. In comparison, this method allows the results to be computed in one computation cycle. Meanwhile, the method is insensitive to the size of the exemplar.

4. Results and analysis

Experiments are implemented using parallel tool CUDA 5.0 (Compute Unified Device Architecture) with the hardware configurations detailed as follows. The hardware includes an Intel Core i3-3220 CPU and a 4 G RAM. The display card is NVIDIA GeForce GTX 650 with 1 G video memory, 384

stream processors, 128 bit bus width, 5000 MHz video memory frequency, 1100 MHz core frequency and 3.0 compute capability.

Figure 5 shows the skin state in three conditions with the small images showing the skin texture sample and the large images showing the synthesized texture. The synthesized texture has the same feature as the skin sample, but in a larger size. And the synthesized result could be directly applied on a model as texture.

Figure 6 shows a digital hand model textured with two different textures. The left figure shows the hand model textured with a simple uniform color, and right one shows the model with the texture synthesized using the method proposed in this paper. It can be seen that the right model could recognize the skin characteristics of human, including sweat pores and wrinkles, which indicates that the synthesized texture is more realistic.

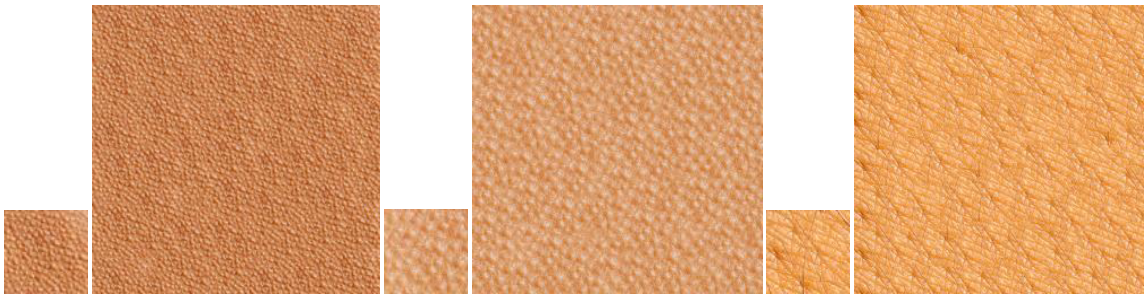


Fig. 5. Skin textures in different conditions.

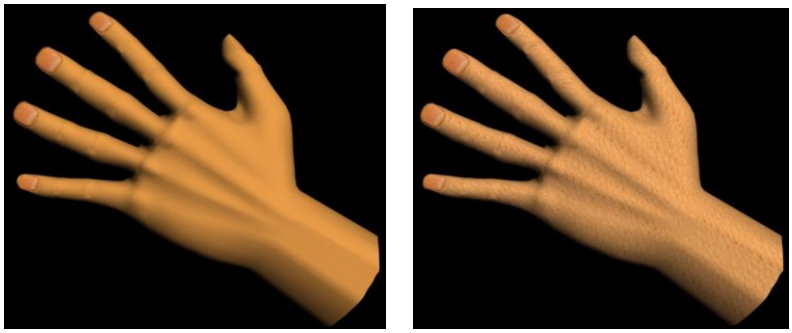


Fig. 6. Hand model with different textures.

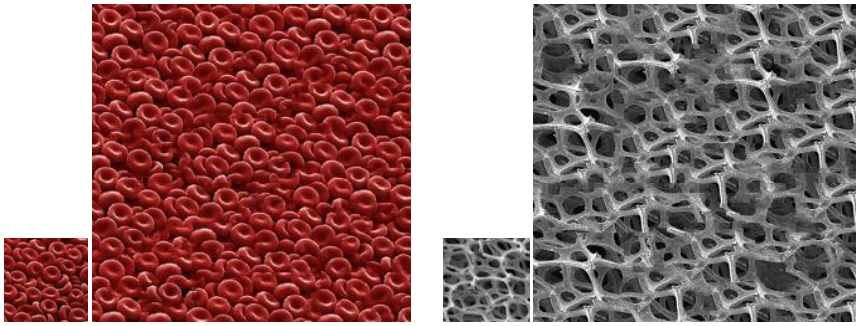


Fig. 7. Textures of blood platelets and bone tissue under microscope.

Besides the skin textures, some other physiological elements can also be synthesized using this method. Figure 7 shows the samples and satisfactory synthesized results of blood platelets and bone tissues under microscope.

The computation efficiency of this method is compared with that of Huang's parallel algorithm. Using Huang's method, it takes about 6 seconds to synthesize a texture with a size of 200*200. While with this method, it only takes 0.903 second to synthesize a 256*256 texture. However, as the proposed method is insensitive to the size of exemplar, synthesizing exemplars with a size of 64*64 or 128*128 size would still require the same time.

5. Conclusion

In this paper, we present a new method to synthesize textures from exemplar on GPU for digital human model. This algorithm could be implemented on regular desktop GPU systems. Compared with the other texture synthesis algorithms, the method takes shorter time to synthesize same quality results, thus could meet the requirement of interactive applications. As confirmed by experiments, this algorithm is stable and insensitive to the size of exemplar. Future work will aim at expanding the algorithm to the organ surface texture synthesis and solid texture synthesis.

Acknowledgement

This work was supported by grant from the National Natural Science Foundation of China (61190121) and the Hebei Institute of Humanities and Social Science Research Projects (SD135005).

References

[1] The online resource for the visible human project, http://www.nlm.nih.gov/research/visible/visible_human.html, accessed at May 1, 2014.
[2] L.Y. Wei, S. Lefebvre, V. Kwatra et al., State of the art in example-based texture synthesis, Proc. of Eurographics State of the Art Report, Eurographics Association Press, Aire-la-Ville, 2009, pp. 93–117.
[3] A.A. Efros and T.K. Leung, Texture synthesis by non-parametric sampling, Proc. of the 7th IEEE International Conference on Computer Vision **2** (1999), 1033–1038.
[4] L.Y. Wei and M. Levoy, Fast texture synthesis using tree-structured vector quantization, in: Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, New York, 2000, pp. 479–488.
[5] L.Y. Wei, Texture synthesis by fixed neighborhood searching, Ph.D. Dissertation, Stanford University, 2002.
[6] M. Ashikhmin, Synthesizing natural textures, in: Proc. of the 2001 Symposium on Interactive 3D Graphics, ACM Press, New York, 2001, pp. 217–226.
[7] S. Zelinka and M. Garland, Towards real-time texture synthesis with the jump map, in: Proc. of the 13th Eurographics Workshop on Rendering, Eurographics Association Press, Aire-la-Ville, 2002, pp. 99–104.
[8] A.A. Efros and W.T. Freeman, Image quilting for texture synthesis and transfer, in: Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, New York, 2001, pp. 341–346.
[9] T. Schlömer and O. Deussen, Semi-stochastic tilings for example-based texture synthesis, Computer Graphics Forum **29** (2010), 1431–1439.
[10] M. F. Cohen, J. Shade, S. Hiller et al., Wang tiles for image and texture generation, ACM Transactions on Graphics **22** (2003), 287–294.
[11] V. Kwatra, I. Essa, A. Bobick et al., Texture optimization for example-based synthesis, ACM Transactions on Graphics **24** (2005), 795–802.
[12] S. Lefebvre and H. Hoppe, Parallel controllable texture synthesis, ACM Transactions on Graphics **24** (2005), 777–786.
[13] S. Lefebvre and H. Hoppe, Appearance-space texture synthesis, ACM Transactions on Graphics **25** (2006), 541–548.

[14] E. Risser, C. Han, R. Dahyot et al., Synthesizing structured image hybrids, ACM Transactions on Graphics **29** (2010), 85-1–85-6.
[15] Z.Y. Huang, F.Z. He, S.L. Zhang et al., Random search based large scale texture synthesis, Journal of Computer-Aided Design & Computer Graphics **23** (2011), 1091–1098.
[16] X. Chen and W.C. Wang, Real-time synthesis of large textures, Journal of Software **20** (2009), 193–201.
[17] X. Chen and W.C. Wang, Reusing partially synthesized textures for real–time synthesis of large textures, Chinese Journal of Computers **33** (2010), 769–775.
[18] Y. Chen and L.L. Wang, Interactive texture synthesis based on geometry mesh, Journal of Computer-Aided Design & Computer Graphics **25** (2013), 1480–1488.
[19] L.L. Wang, Y.L. Shi, Y. Chen et al., Just-in-time texture synthesis, Computer Graphics Forum **32** (2013), 126–138.