Technology and Health Care 25 (2017) S357–S365
DOI 10.3233/THC-171339
IOS Press

GPU-based interactive near-regular texture synthesis for digital human models

Zhe Shen[a,b,*], Chao Wu[c], Bin Ding[a,b], Zhipeng Yuan[b], Naisheng Zhang[b], Meng Zhao[a,b] and Yuan Zhao[a,b]
[a]*Northeastern University, Shenyang, Liaoning, China*
[b]*Northeastern University at Qinhuangdao, Qinhuangdao, Hebei, China*
[c]*Habei Agricultural University, Baoding, Habei, China*

Abstract. Near-regular texture is a common feature for both nature scene and 3D human models. However, traditional texture synthesis has only been able to produce a single result. This study proposes a parallel, and interactive, method for the texture synthesis of irregular appearance in 3D human models. The new method includes two major procedures: (1) iterative edge extraction and processing, and (2) parallel texture synthesis, which generates a texture with higher quality. In this paper, the effectiveness of near-regular texture synthesis algorithm is demonstrated experimentally.

Keywords: Interactive method, texture synthesis, 3D human model, near-regular texture, parallel computing

1. Introduction

A wide range of medical applications – including anatomical teaching, the construction of virtual humans, pharmacological experiments, athletic training, and virtual surgery – require the use of human digital models. 3D human models are usually modeled by human body slices, like Visual Human Project in the United States and the Virtual Human Project in China [1]. With the development of digital technology, 3D human models now mainly focus on the structures of human body, so that certain bodily details that affect actual human appearance (including skin, hair, eyebrows and eyelashes) are frequently ignored. Because the bodies are transformed to slices by CT or other technology, the appearance's integrity would be broken. Even though most human structures and organs are similar, each digital model is unique, and therefore different from all others.

In computer graphics, however, digital human models are designed to be universal. Even so, these models always have their own individual features, i.e. a bruise or birthmark. These kinds of textures do not have obvious structures, but they are not totally random. We therefore call this kind of texture "near-regular texture."

In order to improve this problem, we propose an interactive method to add some random effect on the appearance of these models. We use a Graphics Process Unit (GPU) based multithreads parallel algorithm to achieve interactive texture synthesis. We synthesized near-regular textures in a two-pass way. We extracted the edge of the near-regular texture in the first pass; in the second pass, we optimized the

*Corresponding author: Zhe Shen, School of Computer and Communication Engineering, Northeastern University at Qinhuangdao, Qinhuangdao, Hebei, China. E-mail: shenzhe@neuq.edu.cn.

0928-7329/17/$35.00 © 2017 – IOS Press and the authors. All rights reserved
This article is published online with Open Access and distributed under the terms of the Creative Commons Attribution Non-Commercial License (CC-BY-NC 4.0).

texture via a global optimization algorithm, which implemented in a self-adaptive, non-parametric way. Furthermore, we employed a multithread, parallel algorithm which executed in GPU. Our experiment results confirmed that this method is effective not only for near-regular textures like skin, but also for additional physiological textures.

The paper's structure is as follows: in the next section (Section 2), we offer a brief introduction on the development of interactive texture synthesis and GPU-based near-regular texture synthesis. In Section 3, we explain the GPU-based interactive near-regular texture synthesis algorithm in detail. The next Section 4 shows the results of our algorithm. At last in Section 5, we summarize the study's conclusions.

2. Related works

2.1. Per-patch texture synthesis

Compared with other synthesis method, per-pixel texture synthesis method has a low efficiency. So researchers proposed per-patch texture synthesis. Rather than finding the best-matching pixel, the per-patch texture synthesis method finds a block which contains more pixels.

Efros, for instance, suggested an image-quilting texture synthesis method to synthesize texture in a linear order by finding a best-matching patch [2]. Although this method could produce a relative ideal result, the synthesis processing time is longer. Based on Efros's method, Schlömer proposed a semi-random texture synthesis [3]. Additionally, Cohen designed the Wang-tiles method [4], which combines the image quilting with the Wang tile algorithm. The Wang tiles method using a series of square texture blocks with different colored-edges to synthesis texture. Same color edges could be combined seamlessly. During synthesis, the blocks that meet the color parameters are chosen to synthesis a large texture.

Per-patch texture synthesis method is much faster than per-pixel method, but it is still hard to meet the real-time interaction request.

2.2. Interactive texture synthesis

In 2013, in order to control different level elements of texture (structures in high-level and details in low-level), Ma proposed an easy and intuitive method. These controls can be manually specified, while the corresponding geometric and dynamic repetitions can be computed automatically [5].

Yang proposed a method capable of adjusting the sample in a local area. Part of the texture image could be synthesized under the control of the reference sample texture, resulting in an interactive process [6].

Further research was done following Ma and Yang. Sivaks used neighborhood match to optimize the automatic generation of the process control diagram, and to reduce the store and time costs [7]. Sibbing improved the edge-growing algorithm, and applied it to maintain the synthesis feature [8]. Finally, Rosenberger employed control maps to synthesize the non-stationary textures [9].

2.3. GPU-based texture synthesis

Texture synthesis is a computation-intensive process. With the development of GPU, desktop parallel computing hardware and software architecture has become more powerful. Implementing texture synthesis in a parallel mode is, by now, a natural and logical next step.

Generally, image pyramids are constructed for sample and target textures. Meanwhile, the image pyra-

mids method can be executed in a parallel way. Lefebvre introduced appearance vector space to handle texture synthesis. The appearance vector could indicate more structure information due to its larger neighborhood. This method is thus useful for pixel matching [10].

A structured image hybrids synthesis method was proposed by Risser. In this kind of synthesis, the stabilization is coordinated and the global structure image is blended [11]. Huang suggested a random searching based large-scale texture synthesis method, which include two-stage process [12]. The first stage is initialization, which fills the target white texture image using random color; the second stage is iteration, which implements texture-patch matching to optimize the whole texture. This method is also implemented in a parallel way on GPU, but the synthesis process still needs seconds to complete.

Chen proposed a GPU based real time algorithm to synthesize large-scale textures [13]. The algorithm first analyzes the periodicity of the sample texture, and then distributes patches and fills in any vacant areas. Later, an improved method was proposed which could generate a larger size patches from the synthesized texture area [14]. Another improvement was the addition of interactive operations to texture synthesis in order to integrate user control information and exemplar structure. To guarantee an interactive computation time, this method was later implemented on GPU [15]. Wang then proposed a just-in-time method to synthesis texture on GPU. This method considers the texture patch as an umbrella shape, which could eliminate the texture distortion and mapping problems in a short time [16].

In order to get a satisfied result, we combine the global optimization algorithm and the patch-based texture synthesis method. A coarse texture is synthesized using the patch-based method, and then the global optimization algorithm was implemented to refine the coarse texture. Since we implemented both steps on GPU, we were able to achieve a common-size texture within one second.

3. Texture synthesis on GPU

3.1. Algorithm

The first step of near-regular texture synthesis is initializing the target texture image with a random color, such as white Gaussian noise. Then select a patch from target texture in linear order, and looking for a best-matching patch in sample texture. When the best-matching patch is found, copy it to the target texture at the right positon. Repeat these steps until all target texture pixels are confirmed. The global optimization algorithm is then employed to erase the border between two patches. All these steps are implemented in a parallel way on GPU.

Since the initialization of target texture is random, the result of first best-matching patch would also be random. The succeeding results of best-matching patches are equally random, due to the accumulation of initial offset. But in local areas, we used patches selected from exemplar to compose the texture, so the appearance of target texture is consistent with the sample texture. This ensue the local similarity and global randomness of the target texture.

Figure 1 shows the pseudo code of our algorithm. In our algorithm, *Is* denotes sample texture, *Io* denotes output texture, P_s denotes a pixel in I_s, P_o denotes a pixel in *Io*, $N(p_o)$ denotes neighborhood of pixel p_o.

Furthermore, we summarize the algorithm in the pseudo code can be seen in Fig. 1.

The algorithm's architecture is flexible and made up of several components. Some of these components are as follows:

```
Function   I_o ← TextureSynthesis (I_s, outputSize)
1          I_o ← Initialize (outputSize);
2          foreach Neighborhood size S_i from low image pyramid to high
3              loop through all texture patches start from P_s(x,y) of I_o
4                  N(p_o) ← BuildNeighborhood (x,y)
5                  C ← FindBestMatch (x,y);
6              S_i = S_i/2;
7          I_o ← ReconImage (C);
8          return I_o;
```

```
Function   C ← FindBestMatch (x,y) // Implement on GPU for each thread
1          N_s ← BuildNeighborhood (x,y);
2          N_a^best ← NULL; C ← NULL;
3          loop through all pixel (x,y) of I_s
4              N_a ← BuildNeighborhood (x,y);
5              if Match (N_a, N_s) > Match (N_a^best, N_s)
6                  N_a^best ← N_a;
7          C ← N_a^best ;
8          return C;
```

Fig. 1. Pseudo code of the algorithm.

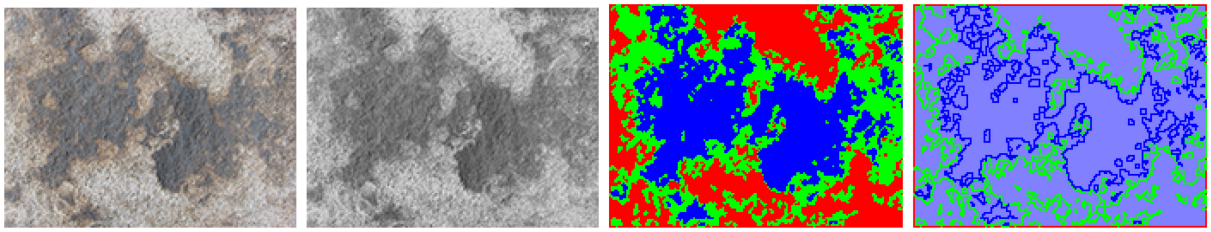


Fig. 2. Near-regular texture edge extraction.

Synthesis Ordering: The function *TextureSynthesis* has a linear order. In line 3 of the function *FindBestMatch*, however, the process order is carried out in a parallel mode (Section 3.5).
Neighborhood: We used the fixed-size neighborhood to simplify computation and to facilitate GPU implementation. After each process of implementation, we reduced the neighborhood by a half size. We found that a smaller size was able to optimize the border of adjacent texture patches (Section 3.6).

3.2. Edge extraction

The sample plot's boundary refers to the half-rule texture image grayscale, according to the grey value after the discretization of the boundary's hierarchical structure (shown in Fig. 2).

The image is divided into five layers based on the pre-defined threshold value of n. In our method, we always assigned n to stand for 3 or 4.

Layer i masks the other layers' information to show only its own boundary (shown in Fig. 3).

In our study, we used an iteration in order to extract the edge of a large area. We started with point p, found p's adjacent boundary point q, and performed the process iteratively until we found all the boundary points. We were able to finish this process due to the boundary's circularity.

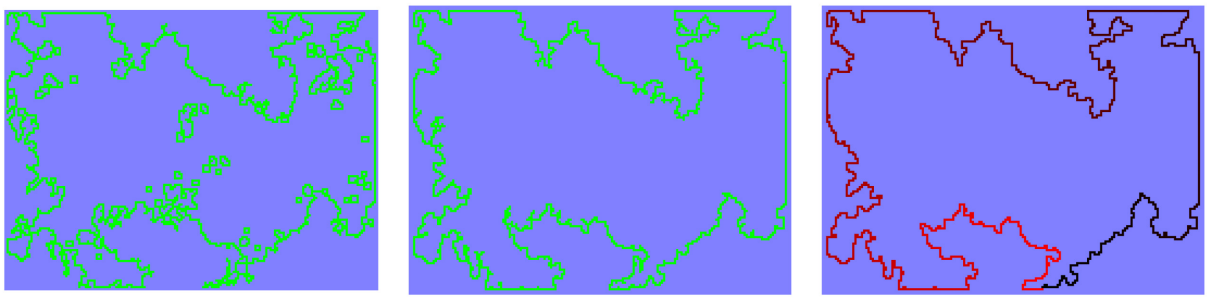


Fig. 3. Near-regular texture edge extraction result.

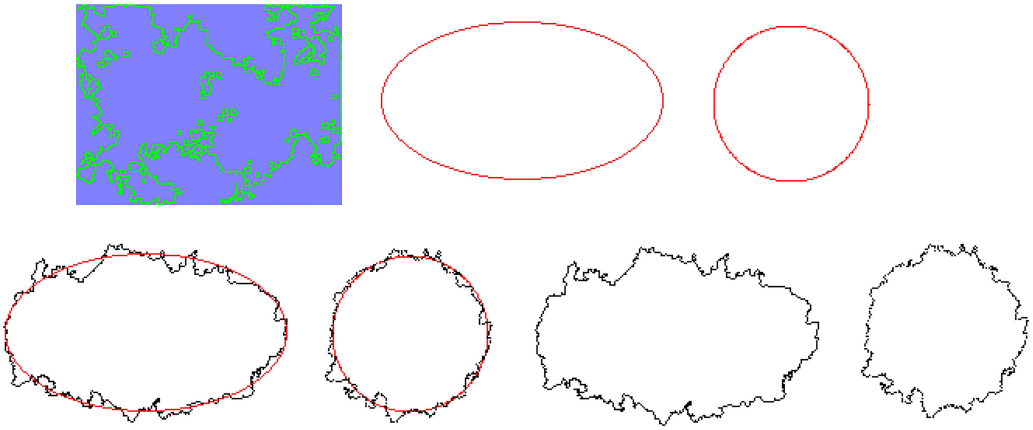


Fig. 4. Edge disturbance.

We then repeated this process for each layer, in order to extract all the boundaries of the sample texture with different grey values.

3.3. User input and edge disturbance

After successful edge extraction, the user inputs the control edge. The result is a disturbance in the input edge information, according to the extracted information. Our process ensures that the user input has an appearance that consistently matches the sample texture (shown in Fig. 4).

Following edge disturbance, and in order to fill the closed area, our method launched the patch-based, parallel texture synthesis process.

3.4. Texture patch matching

The texture patch matching process implemented in a linear order. Select an $N * N$ size patch t from target texture, and then traversed each patches s_i with same size in the sample texture S, and compute the L_2 distance between s_i and t as in Eq. (1), and R, G, B are RGB vectors of each pixel.

$$BestMatchBlock = \min\left(\forall_{s\in S}\sum_{i=1}^{N*N}((R_{s_i}-R_t)^2+(G_{s_i}-G_t)^2+(B_{s_i}-B_t)^2)\right) \quad (1)$$

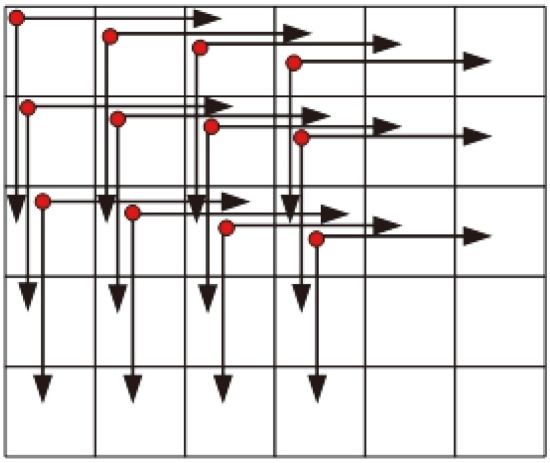


Fig. 5. Parallel structure sketch map.

A small result of L_2 distance indicate small texture patch differences, implying that the adjacent patches usually more similar than patches that far away from each other. When all L_2 distance are calculated, the patch with the minimum value is chosen as the best match.

After one best-matching patch confirmed in the target texture, the next undetermined patch is selected in a linear scanning order. The displacement distance D along the scanning direction should be smaller than N, which means the size of patch. The overlap area could guarantee the continuity and the data dependency of subsequent patches. Consequently, the appropriate relation between N and D has the capacity to ensure the stability of the synthesis result. In our particular study, we chose N = 2D.

Figure 5 is a sketch of parallel structure on GPU. Each pixel corresponds to a thread. When computing the best-matching neighborhood, each thread works with the neighborhood whose initial pixel corresponds to that particular thread.

3.5. Self-adaptive global optimization

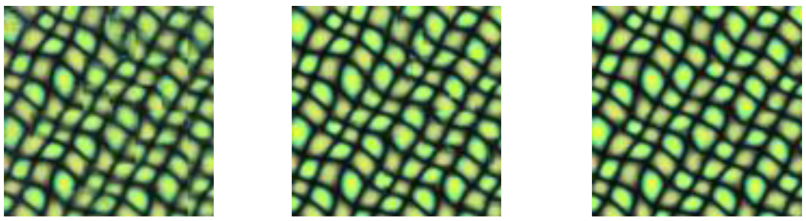
After the first-pass of synthesis, we could get a coarse texture result. The coarse texture has a similar appearance with the sample texture, but there are obvious borders between any two confirmed patches; which divided the texture into several blocks. A self-adaptive global optimization algorithm was used to erase the border between two patch of the synthesized result. The selected energy function as show in Eq. (2).

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

In Eq. (2), X indicates the pixels in target texture image; x indicates the result of vectorized X, i.e., x is formed by concatenating the intensity values of all pixels in X; Z indicates the pixels in sample image. Therefore, the global energy of the target texture is the sum of the L_2 distance of each pixel.

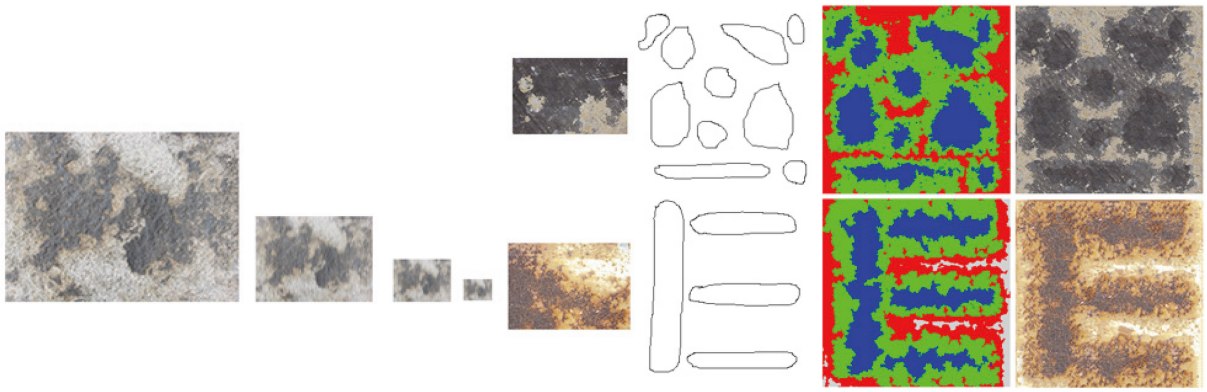
We add a self-adaptive weight in Eq. (3) to make the energy function convergence fast, so that the high frequency information can be eliminated effectively.

$$E_t(x;\{z_p\} = \sum_{p\in X} \omega_p \|x_p - z_p\|^2 \tag{3}$$



(a) No optimization (b) One-time-pass optimization (c) Two-time-pass optimization

Fig. 6. Comparison of different optimization passes.



(a) (b)

Fig. 7. Near-regular texture synthesis result in LOD.

Here, ω_p stands for the weight, given by $\omega_p = P(x_p)\|x_p - z_p\|^{r-2}$ with $r = 0.8$. $P(x_p)$ stands for the probability of finding the best-matching patch in sample texture for the current patch of target texture, in a high-frequency, successful match region.

After each pass of optimization, the neighborhood's size is reduced by a quarter. The smaller neighborhood could refine the target texture by erase the slight borders. Figure 6a shows the coarse texture which has sharp borders. Figure 6b shows the one-time-pass optimization which has mild borders. Figure 6c shows the two-time-pass optimization which has no obvious borders. Two-time-pass optimization result is the texture whose quality is able to satisfy the application requirements. Consequently, we selected three as the minimum number of optimization. As shown in Kwatra's method, in order to get a stable result, r should be set to 0.8.

4. Results and analysis

We tested our experiments with such software architecture: Operating system is Microsoft Windows 7 professional edition, develop tool is Visual studio 2008 with CUDA 5.0.

The hardware list as follow: CPU model is Intel Core i3-3220, the random access memory size is 8 G, the video card is NVIDIA GeForce GTX 650 with 1 G DDR3 RAM, 1100 MHz core frequency, 5000 MHz video memory frequency, 128 bit bus width, 384 stream processors.

Figure 7a shows the result of a near-regular texture synthesis in a LOD (Level of Detail) way. The texture resolution on the right has a half size of the texture resolution on the left.

Figure 7b shows the result of user input synthesis, which could benefit from adding further interactive information to control the synthesis.

In our experiments, we implemented Rosenberger's algorithm with a series of pyramid images to speed up the process. Both our method and Rosenberger's algorithm are implemented on the same hardware.

5. Conclusions

In this paper we proposed an interactive method to synthesize near-regular textures on GPU for 3D digital human models. Compared with other interactive parallel algorithms, the method presented here requires less time to synthesize results of the same or greater quality. Therefore, our proposed method meets the requirements of interactive applications. Our experiments show that the algorithm is stable and effective, and the time consumption is insensitive to exemplar's size. Further research still needs to be conducted in order to make our algorithm applicable to organ surface texture synthesis as well as solid texture synthesis.

Acknowledgments

This research was supported by grants from the Fundamental Research Fund for the Central Universities (N152303006), the Scientific Research Foundation of Northeastern University at Qinhuangdao (XNB201603), the Natural Science Foundation of Hebei Province (F2017501072), the Scientific Research Fund of Hebei Education Department (QN2016307), the Natural Science Foundation of Hebei Province (F2016501073), Beijing Natural Science Foundation (No. 4164092), and the Doctoral Scientific Research Foundation of Liaoning Province (No.201601016).

Conflict of interest

None to report.

References

[1] The online resource for the visible human project is: http://www.nlm.nih.gov/research/visible/visible_human.html, accessed at May 1, 2014.
[2] Efros A.A. and Freeman W.T., Image quilting for texture synthesis and transfer, Proc. of the 28th Annual Conference on Computer Graphics and Interactive Techniques, New York: ACM Press (2001), 341–346.
[3] Schlömer T. and Deussen O., Semi-stochastic tilings for example-based texture synthesis, Computer Graphics Forum **29** (2010), 1431–1439.
[4] Cohen M.F., Shade J., Hiller S. et al., Wang tiles for image and texture generation, ACM Transactions on Graphics **22** (2003), 287–294.
[5] Ma C., Wei L.Y. and Tong X., Discrete element textures[J], ACM Transactions on Graphics **32**(4) (2013), 90: 1–90: 10.
[6] Yang F., Wang J., Shechtman E. et al., Expression flow for 3D-aware face component transfer[J], ACM Transactions on Graphics **30**(4) (2011), 60.
[7] Sivaks E. and Lischinski D., On neighbourhood matching for texture-by-numbers[C], Computer Graphics Forum **30**(1) (2011), 127–138.
[8] Sibbing D., Pavić D. and Kobbelt L., Image synthesis for branching structures[C], Computer Graphics Forum **29**(7) (2010), 2135–2144.

[9] Rosenberger A., Cohen-Or D. and Lischinski D., Layered shape synthesis: automatic generation of control maps for non-stationary textures, Acm Transactions on Graphics **28**(5) (2009), 89–97.
[10] Lefebvre S. and Hoppe H., Appearance-space texture synthesis, ACM Transactions on Graphics **25** (2006), 541–548.
[11] Risser E., Han C., Dahyot R. et al., Synthesizing structured image hybrids, ACM Transactions on Graphics **29** (2010).
[12] Huang Z.Y., He F.Z., Zhang S.L. et al., Random search based large scale texture synthesis, Journal of Computer-Aided Design & Computer Graphics **23** (2011), 1091–1098.
[13] Chen X. and Wang W.C., Real-time synthesis of large textures, Journal of Software **20** (2009), 193–201.
[14] Chen X. and Wang W.C., Reusing partially synthesized textures for real-time synthesis of large textures, Chinese Journal of Computers **33** (2010), 769–775.
[15] Chen Y. and Wang L.L., Interactive texture synthesis based on geometry mesh, Journal of Computer-Aided Design & Computer Graphics **25** (2013), 1480–1488.
[16] Wang L.L., Shi Y.L., Chen Y. et al., Just-in-time texture synthesis, Computer Graphics Forum **32** (2013), 126–138.