

# Data Embeddable Texture Synthesis with Fast Data Extraction

✉ <sup>1</sup>Charushila Vijay Rane, <sup>2</sup>Sandip Raosaheb Patil

<sup>1</sup> JSPM's Rajarshi Shahu College of Engineering, Tathawade, Pune-411033

[ranecharushilav@gmail.com](mailto:ranecharushilav@gmail.com)

<sup>2</sup> Bharati Vidyapeeth College of Engineering for Women, Dhankawadi, Pune-411043

[srpatil44@gmail.com](mailto:srpatil44@gmail.com)

Received: 18th July 2020, Accepted: 27th July 2020, Published: 31st August 2020

## Abstract

Texture synthesis which grows source texture image into a bigger texture image is applied for data embedding, instead of using a readily available digital image as a carrier for data. The patch based texture synthesis is used to embed the data in patches. The selection of a suitable patch for texture synthesis is dependent on data to be hidden. At receiver, embedded data is recovered along with source texture. The efforts are taken to improve the speed of the data extraction. In the process of data extraction, the matched candidate patch is identified at every patch location of stego synthetic texture image. If it is done by referring to the energy value of the kernel region than by comparing the whole patch with all candidate patches for similarity, data extraction is almost 40 times faster.

## Keywords

*Texture Synthesis, Data Embedding, Data Extraction, Energy.*

## Introduction

For secure communication of secrets or private information such as in military or banking applications, cryptographic methods are used. By cryptography, plain text information is converted into cipher text. The cipher text can attract the attackers because of meaningless and random nature. It is essential to hide the existence of secret information in data transit. Generally, in steganography, the information is hidden in the cover image by altering the pixel values which may lead to distortion in the stego image. In contrast to this, data is embedded through the process of texture synthesis. This approach of steganography is more secure since pixel values are not altered. The volume of embedded data is directly proportional to the dimensions of output stego synthetic texture image. Depending upon the length of data to be hidden, size of stego synthetic texture image can be varied. Source texture is recovered at receiving end and is used for the next iteration of data exchange. In our proposed work, for the data extraction process, if energy values of candidate patches are used for distinguishing them from one another instead of finding patch structural similarity, data extraction time is reduced by the factor  $\approx 40$ . For example, in data embeddable texture synthesis of sample from D20 texture image of Brodatz Album, data extraction time is reduced from 4668.68 sec to 117.35 sec i.e. by a factor of 39.78.

## Related Works

H. Otari *et al.* [1] uses LBP patterns for photograph readable data hiding and S.C. Liu *et al.* [2] combined Art image generation with data hiding provides less embedding capacity. Kuo-Chen Wu *et al.* [3] uses patch based texture synthesis for data hiding and by using other approaches of texture synthesis and data hiding, image quality and embedding capacity can be increased. The execution time is 6.8% to 8.7% more if compared with a pure texture synthesis process. As per experimentation of Efros Alexei A. and Thomas K. Leung, pixel by pixel texture synthesis is slow and facing problems like growing garbage, verbatim copying [4]. Wei Li-Yi, and Marc Levoy uses multiresolution synthesis for textures containing large scale structures [5]. In patch based sampling of Liang Lin *et al.*, texture synthesis can be faster and boundary treatment is needed [6]. A. A. Efros and W. T. Freeman synthesized a texture by stitching patches of existing texture images. Minimum error boundary cut is taken at overlap of two blocks. Performance is better for random textures and semi-structured textures. It is a fast and simple algorithm, extended for texture transfer but excessive repetition and distortion at boundaries are observed in this algorithm [7]. Sylvain Lefebvre and Hugues Hoppe use neighborhood matching method for efficient parallel synthesis with addition of user controls [8]. Instead of using single exemplar image as in traditional texture synthesis, Charles Han *et al.* use some input exemplars at multiple scales for texture synthesis [9]. Jian Muwei *et al.* proposed efficient wavelet transform based texture synthesis algorithms [10]. Joshi Mangala S. *et al.* used simultaneous AR model to estimate parameters for texture synthesis [11]. Data hiding can be done by histogram shifting [12] and prediction error [13]. Qin Chuan *et al.* proposed simultaneous data hiding and compression [14]. Schottle Pascal and Rainer Bohme

analyzed adaptive embedding in presence of attackers [15]. Weiyi Wei *et al.* [16] propose data hiding through texture synthesis based on LBP. This approach gives increased embedding capacity, robustness and good visual effect. Texture synthesis can be done by pixel based, patch based and parameter estimation based methods. Data hiding can be done in spatial domain and Transform domain.

### Proposed Method

Ahead of understanding the actual algorithm, one must know the basic terminology used such as source texture, patch, kernel region, boundary region, source patches and candidate patches.

Texture image sample of size  $S_w \times S_h$  which is the seed for data embeddable texture synthesis is called as source texture. A part of the source texture with size  $P_w \times P_h$  is called a patch. The central part of patch with size  $K_w \times K_h$  is a kernel region which is surrounded by boundary region of width  $P_d$  such that

$$P_w = K_w + 2 P_d \quad (1)$$

$$P_h = K_h + 2 P_d \quad (2)$$

The source texture of size  $128 \times 128$  is further divided into 16 equal and non overlapping blocks called kernel blocks. Each kernel block has the size of  $32 \times 32$ . All borders of every kernel block are extended with the depth of 8 pixels to generate source patches.

In texture synthesis, the most suitable patch is selected from a number of candidates to paste at the current working location. So we need to generate a set of candidate patches. To generate candidate patches, a window of  $48 \times 48$  size is considered. This window is moved on a source texture of size  $128 \times 128$  pixels with a jump of one pixel each time.

### Message embedding procedure

Message embedding procedure is explained in this section. The corresponding block diagram is shown in figure 1. The text message is embedded in the stego texture image through the texture synthesis process. It consists of index table generation, candidate patches and source patches generation from source texture and message oriented texture synthesis.

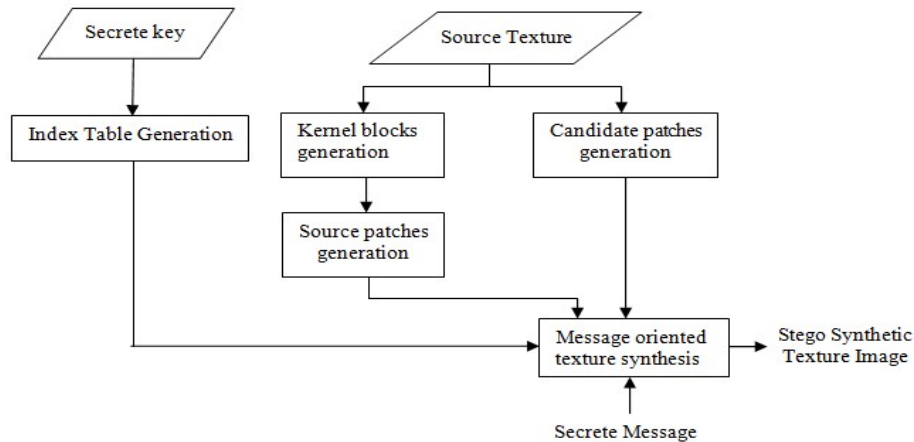


Figure 1: Message Embedding Procedure

**Index table generation** - Index table is generated to keep track of type of patch pasted in texture synthesis process. This will help to retrieve the source texture exactly at receiver. Index table contains 12 columns and 12 rows. The size of the index table depends on the quantity of secret message to be hidden. Total numbers of entries in the index table are 144. Out of 144 locations of stego synthetic texture image, 16 are used for source patches pasting and remaining 128 locations are filled up by data oriented candidate patches.

**Candidate patches generation** - Candidate patches are generated from source texture of size  $S_w \times S_h$  by taking a jump of one pixel every time and cropping a patch of size  $P_w \times P_h$ . Number of candidate patches generated is given by  $N_{CP} = (S_w - P_w + 1) \times (S_h - P_h + 1)$  (3)

One parameter is required which differentiates all the candidate patches from one another. Energy value at kernel of each candidate patch is calculated by using the equation 4.

$$E_i = \frac{1}{MN} \sum_{j=1}^M \sum_{k=1}^N |P_i(j, k)|^2 \quad (4)$$

Where,  $E_i$  is overall energy in the kernel region of  $i^{\text{th}}$  candidate patch.

$P_i(j, k)$  is pixel at location  $(j, k)$  in the kernel region of  $i^{\text{th}}$  candidate patch.

$M$  and  $N$  denote the size of the kernel region of  $i^{\text{th}}$  candidate patch.

The Energy value of candidate patch's kernel region is able to distinguish the candidate patches from each other. Only unique candidate patches are retained.

**Source patches generation** - The source texture of size  $S_w \times S_h$  is divided in non overlapping kernel blocks of size  $K_w \times K_h$ . These kernel blocks are extended with the width  $P_d$  of 8 pixels. If the kernel block is situated across the boundary of source texture, then symmetrical contents of that kernel block at boundary are used for expansion purposes otherwise the kernel block will be extended in neighboring Kernel blocks to generate boundary. Number of source patches generated is given by  $N_{SP}$  as in equation 5.

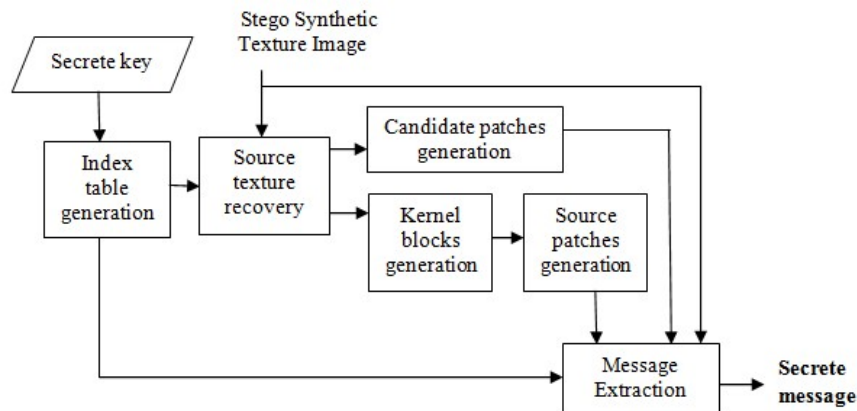
$$N_{SP} = \frac{S_w}{K_w} \times \frac{S_h}{K_h} \quad (5)$$

**Source patches pasting** - Key is used to decide upon the locations to be used for source patches pasting. While selecting the locations, the boundary area of the work bench is skipped. Out of remaining locations, alternate rows and columns are the candidate positions for source patches pasting to reduce the variety of overlapping of source patches with candidate patches. 16 locations are randomly selected for source patches pasting. These locations of scan line order are converted into corresponding row number and column number. For e.g. Location no.40 means (row no=4, column no=4), since the workbench consists of 12 rows and 12 columns. Source patches are pasted at these randomly selected locations in the texture synthesis process.

**Data Embeddable texture synthesis** - All locations of the work bench, other than source patches positions are synthesized by using candidate patches. Mean square error is calculated at the overlapping area between each candidate patch and already synthesized region. Then candidate patches are arranged as per MSE values. The candidate patch having minimum MSE value indicates the best match with the synthesized area. In pure texture synthesis, this patch with rank 1 is pasted at the current working location. To make texture synthesis data embeddable, the candidate patch having rank equal to the ASCII value of the character from secret message to be hidden is selected. One character is concealed in the selected candidate patch. Stego synthetic texture image is formed with message oriented patch selection and compared with source texture image to calculate SSIM value for quality checking purpose. To enhance the quality of synthesized texture, a minimum error path through the overlapped region is traced while stitching the selected candidate patch in the workbench. This is the optimal boundary between selected candidate patch and the previously synthesized surrounding patches [7].

### Message extraction procedure

Message extraction is done from received stego synthetic texture image at the receiving end. The block diagram given in figure 2 explains the message extraction process.



**Figure 2: Message Extraction Procedure**

**Recovery of source texture** - At the receiving end, with the help of the same secret key, index table is generated as in data embedding procedure. With the help of the index table, the locations of source patches pasting are traced. The kernel part of source patches is extracted and organized in order to build original source texture back. Source texture is recovered which gives SSIM value equal to 1 with source texture of data embedding procedure. It assures the exact recovery of source texture.

**Source patches and candidate patches generation** - From the recovered source texture, source patches and candidate patches are generated by following the procedure as in data embedding. Every candidate patch must be unique.

**Matched patch retrieval** - The locations of stego synthetic texture, at which data is embedded in terms of selected candidate patches, are referred one by one. The stego kernel region of size  $32 \times 32$  is extracted from the current working

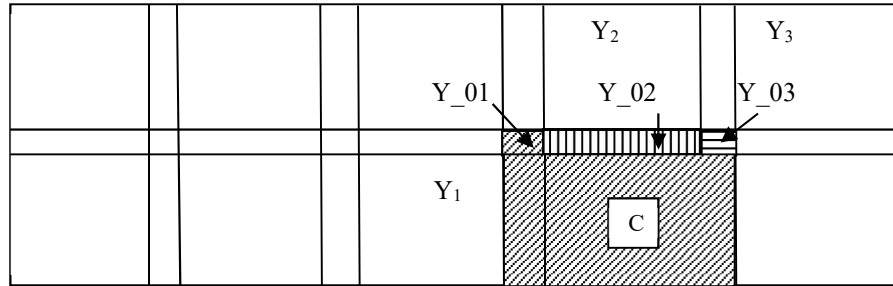
location. Then it is compared with kernel region of every candidate patch to find exact matching patch with stego kernel region of current working location. It is referred as matched patch of that location. The energy value of the stego kernel region is compared with the energy of every candidate patch's kernel region.

MSE value calculation - For extraction of character from pasted candidate patch, MSE value calculation is needed. At the current working location, the entire candidate patches set is tested one by one. MSE value of overlapping area between each candidate patch and already synthesized surrounding patches is calculated.

Data extraction - All the candidate patches are ranked as per MSE values calculated for current working location. The rank of the matched patch of that working location can be located from the sorted MSE index. This rank is nothing but the ASCII value of the concealed character.

Extraction of data from the first row of stego synthetic texture - In the data extraction from the first row of stego synthetic texture, the MSE value calculation involves only vertical overlap with adjacent patch.

Extraction of data from row number 2 onwards of stego synthetic texture - In the data extraction from row  $\geq 2$  of stego synthetic texture, the MSE value calculation involves vertical as well as horizontal overlap with adjacent patches.



**Figure 3: Horizontal Overlap and Vertical Overlap**

If the extraction of data is to be done from the given shaded patch C at current working location of figure 3 at  $(i=2, j=4)$  i.e 2<sup>nd</sup> row and 4<sup>th</sup> column, then

$$\text{Overlap}_{\text{horizontal}} = Y\_H = Y\_01 + Y\_02 + Y\_03 \quad (6)$$

$$Y\_01 = Y_1 (1:8, 41:48) \quad (6a)$$

where  $Y_1 = \text{matched\_patch} \{i, j-1\}$ , i.e.  $Y_1$  is a matched patch at the same row but previous column of current working location.

$$Y\_02 = Y_2 (41:48, 9:40) \quad (6b)$$

where  $Y_2 = \text{matched\_patch} \{i-1, j\}$ , i.e.  $Y_2$  is a matched patch at the previous row but same column of current working location.

$$Y\_03 = Y_3 (41:48, 1:8) \quad (6c)$$

where  $Y_3 = \text{matched\_patch} \{i-1, j+1\}$ , i.e.  $Y_3$  is a matched patch at the previous row and next column of current working location.

$$\text{Overlap}_{\text{vertical}} = Y\_V \quad (7)$$

$$Y\_V = Y_1 (1:48, 41:48) \quad (7a)$$

where  $Y_1 = \text{matched\_patch} \{i, j-1\}$ .

Subtract the region of  $8 \times 8$  pixels which is common in horizontal and vertical overlap for rows and columns greater than 1 i.e for the case  $\{i > 1 \ \&\& \ j > 1\}$

## Results and Discussion



The experimentations are performed on a system having i5-3230M CPU @2.60GHz and 4 GB RAM. 64 bits operating system. The software used for algorithm development is Matlab R2014a. The source textures used for texture synthesis are Brodatz gray textures as well as some colored textures. Texture synthesis is done in scan line order. Source patches are distributed at different locations in the workbench, those are selected based on key and remaining positions are filled with candidate patches that are selected based on data to be hidden. For exact data retrieval, uniqueness of candidate patches must be checked.

At receiver, the source patches are extracted with the help of index table to rebuild the source texture. In the data extraction process, the matched patch with every location of stego synthetic texture is identified among all candidate patches by comparing the kernel regions. Matched patches are identified based on the energy value of the kernel region. The data extraction is performed at a faster rate as per the results mentioned in Table 1. In our method, data

extraction time is almost the same irrespective of the source texture used and is very less if compared with the Kuo-Chen Wu et al method.

Data embeddable texture synthesis is performed for colored textures also. Matched patch of current working location in synthesized texture is recognized based on energy value of the kernel region in the data extraction process. Sample textures and resultant stego synthetic textures are shown in Figure 4.

**Table 1: Data Extraction Time by Kuo-Chen Wu et al.[3] Method and Proposed Method**

Texture	Parameters	Kuo-Chen Wu et al.[3]		Proposed Method	
		Without mincut	With mincut	Without mincut	With mincut
 D20.gif	Data Extraction time (sec)	4668.68	4166.51	117.35	120.46
	SSIM value	0.0237	0.0242	0.0237	0.0242
Data Extraction time Reduction Factor		Without mincut	$\frac{4668.68}{117.35} = 39.78$		
		With mincut	$\frac{4166.51}{120.46} = 34.59$		
 D101.gif	Data Extraction time (sec)	5545.14	5075.66	123.57	120.7
	SSIM value	0.0456	0.0482	0.0456	0.0482
Data Extraction time Reduction Factor		Without mincut	$\frac{5545.14}{123.57} = 44.87$		
		With mincut	$\frac{5075.66}{120.7} = 42.05$		

(a) Sample Textures



ropenet.jpeg

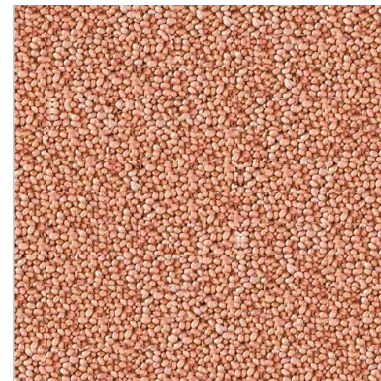
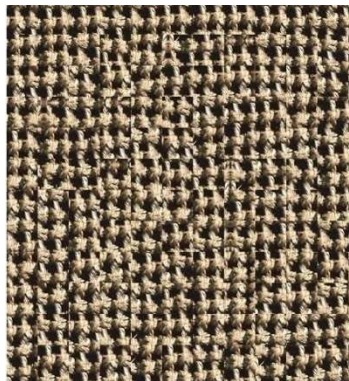


Almond.jpeg



Peanut.jpeg

(b) Stego Synthetic Texture Images







**Figure 4: (a) Sample Textures (b) Stego Synthetic Texture Images for Color Textures**

In data embeddable texture synthesis of coloured textures the parameters like data extraction time and SSIM value of stego synthetic texture with original texture image are noted in Table 2.



**Table 2: Data Embeddable Texture Synthesis for Color Texture**

Texture	ropenet.jpeg 	Ganache.png 	Almond.jpeg 	Peanut.jpeg 
Data Extraction time (sec)	113.69	132.329	116.212	113.577
SSIM value	0.1154	0.6529	0.6166	0.4086

The data extraction by our proposed method is very fast as compared to the counterpart method by Kuo-Chen Wu et al.[3] . For ropenet stego synthetic texture of size  $192 \times 192$  with 8 Bits per patch hiding capacity, computing time is 1680 seconds even though the system with higher configurations (i7-2600 3.4GHz CPU and 4GB memory) is used whereas in our proposed method, for ropenet stego synthetic texture of size  $488 \times 488$ , with one character per patch hiding capacity, data is accurately extracted in 113.69 seconds.

### Conclusion and Future work

This work proposes the fast data embeddable texture synthesis. From a sample of texture image, large sized texture image is synthesized by patch based texture synthesis. The patch selected for synthesis is dependent on data to be hidden. The pieces of source texture are key based randomly distributed in stego synthetic texture so that it can be retrieved accurately at receiver terminal for further round of secure data transmission.

Different Brodatz textures are considered for experimentation. In the data extraction process, if patch identification is done with the help of energy values of patch instead of observing similarity, the process is faster by approximately 40 times. Same algorithm is experimented on colored textures also. Fast retrieval of embedded data is observed.

In texture synthesis, if fixed patch size is used, the quality of stego synthetic texture image is not good enough. To improve the output stego synthetic texture quality, patch size is decided from pattern size of sample texture instead of using fix sized patch.

In data embeddable texture synthesis, as the patches are placed as per data to be hidden, the structure of stego synthetic image is disturbed as this work is in spatial domain.

Further improvement in quality of output stego synthetic texture is possible if worked in frequency domain.

### References

1. Otori Hirofumi, and Shigeru Kuriyama. "Data-embeddable texture synthesis." In International Symposium on Smart Graphics, pp. 146-157. Springer, Berlin, Heidelberg, 2007
2. Liu Shan-Chun, and Wen-Hsiang Tsai. "Line-based cubism-like image—A new type of art image and its application to lossless data hiding." IEEE Transactions on Information Forensics and Security 7, no. 5 (2012): 1448-1458.
3. Wu Kuo-Chen, and Chung-Ming Wang. "Steganography using reversible texture synthesis." IEEE Transactions on Image Processing 24, no. 1 (2014): 130-139.
4. Efros Alexei A., and Thomas K. Leung. "Texture synthesis by non-parametric sampling." In Proceedings of the seventh IEEE international conference on computer vision, vol. 2, pp. 1033-1038. IEEE, 1999.
5. Wei Li-Yi, and Marc Levoy. "Fast texture synthesis using tree-structured vector quantization." In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp. 479-488. ACM Press/Addison-Wesley Publishing Co., 2000.
6. Liang Lin, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. "Real-time texture synthesis by patch-based sampling." ACM Transactions on Graphics (ToG) 20, no. 3 (2001): 127-150.
7. Efros Alexei A., and William T. Freeman. "Image quilting for texture synthesis and transfer." In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 341-346. ACM, 2001.
8. Lefebvre Sylvain, and Hugues Hoppe. "Parallel controllable texture synthesis." In ACM Transactions on Graphics (ToG), vol. 24, no. 3, pp. 777-786. ACM, 2005.
9. Han Charles, Eric Risser, Ravi Ramamoorthi, and Eitan Grinspun. "Multiscale texture synthesis." In ACM Transactions on Graphics (TOG), vol. 27, no. 3, p. 51. ACM, 2008
10. Jian Muwei, Shuan Liu, and Junyu Dong. "Fast Texture Synthesis Using Wavelet Coefficient Fitting." In 2008 International Symposium on Intelligent Information Technology Application Workshops, pp. 491-495. IEEE, 2008.

11. Joshi Mangala S., Prashant P. Bartakke, and M. S. Sutaone. "Texture representation using autoregressive models." In 2009 International Conference on Advances in Computational Tools for Engineering Applications, pp. 386-390. IEEE, 2009.
12. Li Xiaolong, Bin Li, Bin Yang, and Tieyong Zeng. "General framework to histogram-shifting-based reversible data hiding." IEEE Transactions on image processing 22, no. 6 (2013): 2181-2191.
13. Rad Reza Moradi, KokSheik Wong, and Jing-Ming Guo. "A unified data embedding and scrambling method." IEEE Transactions on Image Processing 23, no. 4 (2014): 1463-1475.
14. Qin Chuan, Chin-Chen Chang, and Yi-Ping Chiu. "A Novel Joint Data Hiding and Compression Scheme Based on SMVQ and Image Inpainting." IEEE transactions on image processing 23, no. 3. March 2014
15. Schottle Pascal, and Rainer Bohme. "Game theory and adaptive steganography." IEEE Transactions on Information Forensics and Security 11, no. 4 (2015): 760-773.
16. Wei, Weiyi, and A. Chengfeng. "Robust Steganography Using Texture Synthesis Based on LBP" International Conference on Security with Intelligent Computing and Big-data Services. Springer, Cham, 2018.