An Improved Image Super Resolution and Its Parallel Implementation Based on CUDA

Chenyan Feng[1], Xiaoyun Zhang[2], Zhiyong Gao[3]
[1]Institute of Image Communication and Information Processing
[2]Shanghai Jiao Tong University. Shanghai, China
katharine@sjtu.edu.cn, xiaoyun.zhang@sjtu.edu.cn, zhiyong.gao@sjtu.edu.cn

ABSTRACT: *Image super resolution (SR) based on self-similarity has achieved impressive results by exploiting the local similarity across various scaled images. However, it is time consuming and far from meeting practical applications' requirements and it is prone to unnatural visual artifacts such as facets and enhanced noise. In this paper, with the powerful general purpose GPU, an accelerated parallel implementation based on NVIDIA CUDA architecture has been proposed. Then denoising is combined and implemented together during the SR processing using the patch matching information of similarity, and it makes the images clearer and cleaner. Also, subpixel accuracy of interpolation and searching is applied to improve the similarity matching process, which make the upscaled images more natural-looking. Experimental results demonstrate that the proposed method achieves an impressive speedup rate and produces improved visual quality images in comparison with other state-of-the-art SR.*

Keywords: Image Super-resolution (SR), Parallel CUDA Implementation, Self-similarity, Patch Matching, Image Denoising, Subpixel Interpolation

Received: 18 July 2015, Revised 11 August 2015, Accepted 18 August 2015

1. Introduction

The objective of image super-resolution (SR) is to estimate a high-resolution (HR) image with several low-resolution (LR) images. And singe-image SR has been widely used because of its robustness and implementation. Previous works on single-image SR can be roughly divided into four categories: interpolation-based, learning-based, and reconstruction-based [5].

Interpolation-based SR methods like bilinear or bi-cubic have been popularly applied due to the simple implementation and low complexity. But they tend to wipe off high frequency details of images which result in blurred HR images. The learning-based methods [13, 15, and 20] can recover HR images with much more details by training sets of HR/LR image pairs. The reconstruction-based SR methods enforce the similarity constraint between the original LR image and the down sampling counterpart of the HR image. Alternatively, example-based nonparametric methods were used to predict the missing high frequency band of the HR image by relating the image pixels at two spatial scales using a universal set of training example patch pairs. Taking one step further, self-example based methods were proposed, taking advantage of the local self-similarity to reduce the dependence on the selection of training sets.

Since self-similarity of images has been proven by recent studies [2, 3, 9], image SR can be regularized based on these self-similar examples instead of some external databases [3]. Although the self-similarity-based methods achieved desirable results, they are prone to unnatural visual artifacts such as facets and enhanced noise. Also the computational cost of most of the SR schemes is getting higher, especially as image size increases.

And since most image processing schemes are inherently parallel, GPU with multiprocessors are very suitable to accelerate the implementation to make the algorithms more practical. Based on the NVIDIA's graphic card and CUDA platform developed for general parallel computation, it's easy to get higher performance than the implementation on CPU

In this paper, we redesign the implementation of the self-similarity-based SR [2] to accelerate this scheme with CUDA architecture. And we improve this algorithm by adding denoising process into the patch matching loop. While searching the optimal patch for the HR image around the corresponding position of original image, we utilize the information of these candidate patches to estimate the characteristic of this small area—whether the original patch contains some noise—so that different approaches to obtain optimal patches are applied according to the characteristic. Another improvement is refining the data precision by subpixel, which turns out to be an effective way to obtain high visual quality HR images.

The rest of the paper is organized as follows. Section II introduces the related work of local self- similarity. Section III shows the details of implementation of GPU acceleration. Section IV describes the denoising process and shows how the subpixel is used to get a better natural looking image. Section V gives some experimental results to support.

2. Related Work of Self-similarity

Basically, the image formation process can be roughly represented by equation (1),

$$I_0 = DQ_kL + n_k \quad 1 \le k \le K \qquad (1)$$

In equation (1), L is a vector form of the original highresolution images while I_0 is the observed low-resolution images. Q_k denotes the degradation process such as warping or blurring for image, is the matrix representing the down-sampling operator, and denotes additive noises in the image formation process.

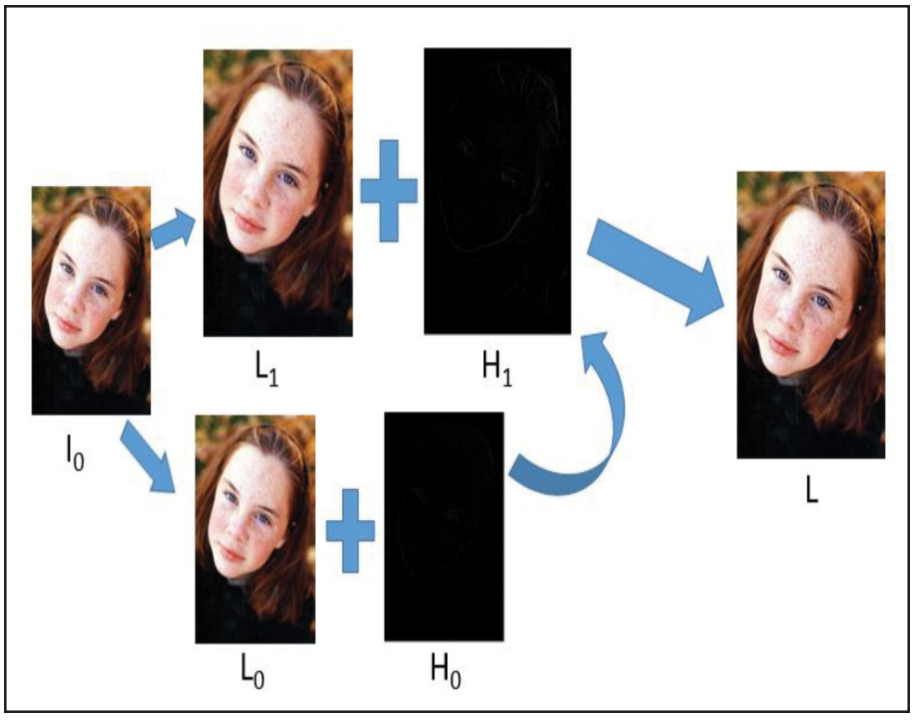

Figure 1. The specific implementation of the LSE process

According to [2], in natural images, the edges and other singularities in natural images are locally scale invariant upon small scaling factors (normally less than 1.25) and the high-resolution patches can be reconstructed through localized search in the original image. Known as the Local self-example (LSE) algorithm, it addressed the SR problem by predicting and filling the high frequency band for the coarsed high-resolution image. This process can be represented by equations (2), (3), (4),

$$L_0 = U(D(I_0)) \quad (2)$$

$$H_0 = I_0 - L_0 \quad (3)$$

$$L = U(I_0) + H_1,\ H_1 = T(H_0) \quad (4)$$

where L is the reconstructed image, U is the interpolation operator used to achieve the coarse version of L, L_o, is the low frequency part of I_o with same size, and H_o, H_1 are the corresponding high frequency band of l_o and L represents the mapping between H_o and H_1: foreach patch in ($L_1 = U(I_o)$), we will search the best-match one around the corresponding position of L at localized region, and, add the very patch in H0 with the same coordinate of the best-match patch into .

Every patch in a natural image can find a most similar patch in its slightly upscaled version at localized regions around the relative point. That means the upper frequency band can also be reconstructed from by patch matching, Fig. 1 briefly shows the specific implementation of LSE algorithm.

As mentioned about, the original MATLAB and C++ implementation of LSE is pixel by pixel, and the matching process has to search in the local window. And the experiments have shown that when the image size become 1920x1080 or larger (which is often the case), the C++ implementation on our DELL workstation will cost about more than 80s dealing with single image. But this character also shows the algorithm is inherently parallel. And LSE produces impressing results for natural images. In Fig2, a),b),c), the images with scalar factors 1.25, 1.252, and 1.253(applying the implementation shown in Fig 1 several times to get the bigger scalar factor),shows that LSE is capable of producing plausible fine details across the image. However, two problems in this algorithm still remain to be solved. Firstly, whenever there are some noises in the smooth area, which is very common in natural images, LSE algorithm tends to magnify the noise by adding the high-frequency band. As shown in patches (B), (B1), and (B2) in Fig. 2, the noise intensity increases as the scalar factor getting bigger. Secondly, as mentioned by Freeman, LSE algorithm will produce false line-like edges in fine-detailed cluttered regions, which makes the results appear somewhat faceted, as like patches (A), (A1), and (A2).

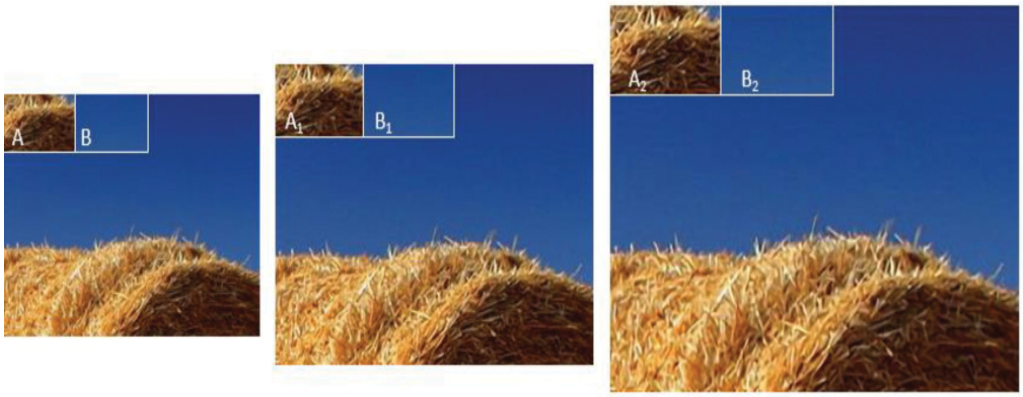

Figure 2. Three upscaled images using LSE with the scalar factor 1.25 for each step

3. GPU Implementation Design

CUDA is the parallel computing platform and programming model developed by NVIDIA, and provides many friendly, full-featured APIs to help harness the power of the GPU. The LSE algorithm is highly parallel for implementation as discussed before. Hence we design our parallel implementation for LSE based on CUDA environment.

3.1 Basic Concepts of CUDA

There are some basic definitions needed to know about CUDA programming.

1. A CUDA "multiprocessor creates, manages, schedules, and executes threads in groups of 32 parallel threads called warps" [22];

2. A CUDA multiprocessor contains a number of blocks of threads, and multi-blocks are grouped into 2-D or 3-D grids;

3. Threads in the same block can communicate and share data through shared memory.

3.2 Design for CUDA Implementation

The algorithm is implemented as the following steps:

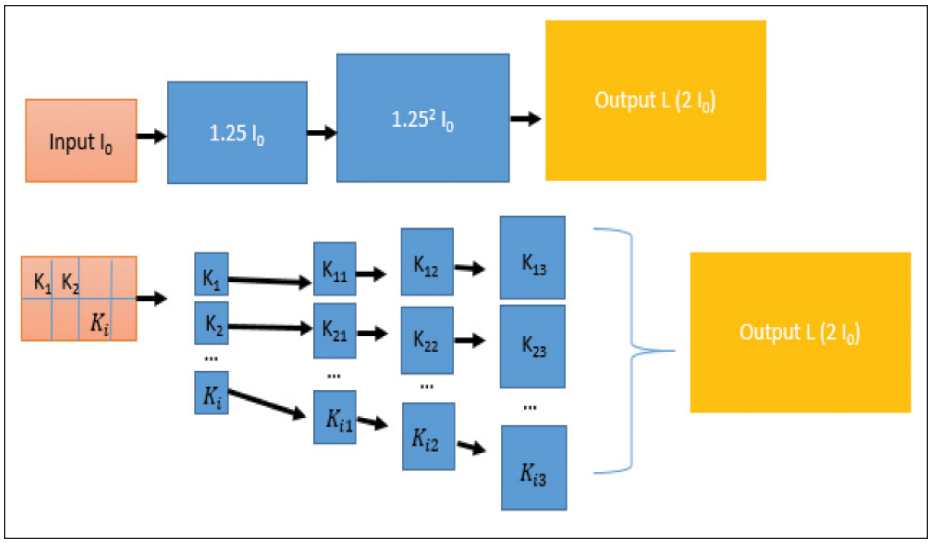

Figure 3. The CUDA implementation design

1. Read the input YUV image (with the size $H_i \times W_i$) from the binary file;

2. Load the image file from CPU (the host) memory to GPU (the device) global memory since the global memory is large enough to store the file and this can avoid transfer data frequently between CPU and GPU;

3. Processing the image on GPU.

Suppose that the target scalar is sa , so according to the introduction of LSE in last sector, we need repeat the procedure (shown in Fig.1) $n = sa / 1.25$ times to get the target HR image, which results to keeping transfer data from GPU's global memory to the compute unit to support kernel function. Actually the time spend on data transferring is the part we can save for more important computing.

So we redesign the implementation structure for parallel efficiency: we separate the image into many patches and upscale every patch of the input image for n times independently as shown in Figure 3.

1. Generate a kernel with block size (block_x, block_y), and global size (global_x, global_y).The number of threads $N = global_x \times global_y \times block_x \times block_y$ is equal to the output number of pixels $N = sa^2 \times H_i \times W_i$. Besides the number of threads in each block should be a multiple of 32 (a warp).

2. Since LSE has to search the best fit high-frequency patch for each pixel within the local window, the data used for two adjacent pixels has lots of overlap. To avoid fetching data from global memory for every pixel, the shared memory is used to store the data.

When designing the implementation, we allocate one thread to work on the whole process for every output pixel. And due to a), in the first step of LSE, there will be some threads cannot be used, and further study of the optimization are still working on.

4. Denoising in SR Process

4.1 Denoising with Patch Matching Information

It's of great importance to remove noise in natural image for perceptual experience. However, from the flowchart of LSE algorithm in Figure 1, it can be seen that the noise is treated as normal image texture and due to the misalignment and the randomness the noise shows, it turns out that the noise is enhanced through LSE algorithm. Lots of schemes have been proposed to address the problem. But most of them are time-consuming[11] or dependent on complicated math theory and difficult to apply to hardware design [12].

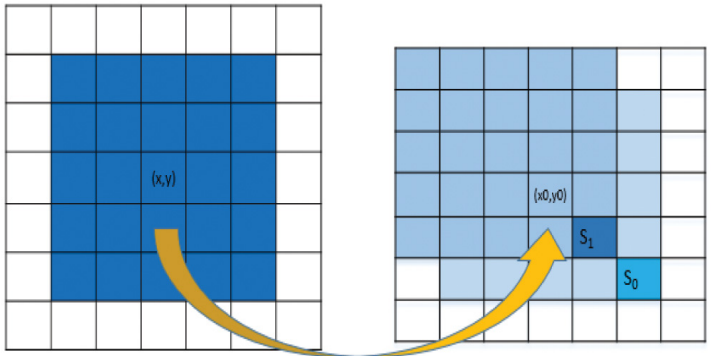

Figure 4. For every patch, the candidate patches in input image

We explore an effective method to measure the image content by making full use of patch information through reconstructing H_1. In LSE, the detail of high-frequency matching is as follows:

For a patch $P(f \times f)$. with the center point (x, y) in L_1, as shown in Figure 4, we first get the corresponding patch P_0 centered at (x_0, y_0) in L_0, $x_0 = [x/c]$, $y_0 = [y/c]$ (c denotes the scalar factor and [ ] means get the data's integer part). In order to find out the best-fit patch for P, we set a window ($w \times w$ in size) in L_0 to search. Then we use SAD to measure the similarity of the two patches. For the searching window, the most similar patch P_s which has the smallest SAD with P, is recorded from the w^2 candidate patches around P_0. Hence the high frequency patch with the same coordinate in H_0 will be added to H_1.

In order to find out whether patch P is corresponding to the noise part or not, we need to analyze the w^2 candidate patches $\{P_i, i = 1, 2, .., w^2\}$. And the SADs $\{S_i, i = 1, 2, .., w^2\}$, between each candidate patch and P, will be used to estimate the character of P, Then we normalize the SADs as follows,

$$S_i = {S_i}\Big/{\sum_{i=1}^{w^2} S_i} \tag{5}$$

Since noise is usually random distributed and texture areas generally have apparent edges, we can see that the SADs of patch $\{P_i\}$ containing texture details have larger variance than those containing some noise. Furthermore, there will be at least one S_i significantly smaller than others according to local similarity theory. Hence, the variance of $\{S_i\}$ texture can be used to determine whether the matched patch in high frequency band H_1 should be added into the upscaled low frequency image H_1.

An adaptive thresholdfor each patch based on the SAD distribution is set to decide which kind of content the patch contains.

$$S_v = \frac{1}{w^2}((S_1 - S_m)^2 + (S_2 - S_m)^2 + \cdots + (S_{w^2} - S_m)^2) \tag{6}$$

$$S_v < \delta \tag{7}$$

In the above equations (6) and (7), S_m denotes the mean of the $\{S_i\}$, S_v denotes the variance of $\{S_i\}$ δ and is the threshold. If P is considered to be a noised one, we will drop the corresponding high-frequency band to suppress the natural noises.

To take a further step to smooth the noised patch in low frequency L_1, patches $\{P_i, i = 1, 2, .., w^2\}$ are combined by weighted

average.

4.2 Subpixel Interpolation and SR Matching

Through LSE's matching loop, we are supposed to find an exact match for each patch. But it's difficult to access a perfect match due to the discrete resampling process in down-sampling and up-sampling for small scaling factors shown in Fig. 1. Freeman et.cl have pointed out this limitation of LSE algorithm.

As a result, there are many misaligned patches in H_1 generated by LSE, which generally lead to the visual facet effect. After analyzing lots of experimental data, it turns out that the exact match may be not found at integrate pixel position, but between two integrate pixels.

Index	-3	-2	-1	0	1	2	3	4
Half-filter	-1	4	-11	40	40	-11	4	1
Quarter-filter	-1	4	-10	58	17	-5	1	

Table 1. The coefficients of the filter for interpolation

Index	-3	-2	-1	0	1	2	3	4
Half-filter	0	0	-4	36	36	-4	0	0
Quarter-filter	0	0	-4	53	18	-3	0	0

Table 2. The altered coefficients of the filter

We extend the local self-similarity on natural images to subpixel level to improve the prediction accuracy to address the unnatural-looking problem.

$A_{-1,-1}$				$A_{0,-1}$	$a_{0,-1}$	$b_{0,-1}$	$C_{0,-1}$	$A_{1,-1}$				$A_{2,-1}$
$A_{1,0}$				$A_{0,0}$	$a_{0,0}$	$b_{0.0}$	$c_{0.0}$	$A_{1,0}$				$A_{2,0}$
$d_{-1,0}$				$d_{0,0}$	$e_{0,0}$	$f_{0,0}$	$g_{0,0}$	$d_{1,0}$				$d_{2,0}$
$h_{-1,0}$				$h_{0,0}$	$i_{0,0}$	$j_{0,0}$	$k_{0,0}$	$h_{1,0}$				$h_{2,0}$
$n_{-1,0}$				$n_{0,0}$	$p_{0,0}$	$q_{0,0}$	$r_{0,0}$	$n_{1,0}$				$n_{2,0}$
$A_{-1,1}$				$A_{0,1}$	$a_{0,1}$	$b_{0,1}$	$c_{0,1}$	$A_{1,1}$				$A_{2,1}$
$A_{-1,2}$				$A_{0,2}$	$a_{0,2}$	$b_{0,2}$	$c_{0,2}$	$A_{1,2}$				$A_{2,2}$

Figure 5. A map of integer pixel and subpixel position, the capital letters represent integer pixels, and the rest are subpixel

In our research, subpixels are interpolated using a filter derived from discrete Fourier transform (DCT) and inverse Fourier transform. An 8-tap separable DCT-based interpolation filter is used for 1/2 precision, and a 7-tap separable DCT-based interpolation filter is used for 1/4 precision (if necessary) as shown in Table 1.

In Figure 4, the capital letters represent integer pixels, and the rest are subpixels to be solved. With the help of Table1, we solve $a_{0,0}$ and $b_{0,0}$ by

$$a_{0,0} = \sum_{i=-3}^{3} A_{I,j}\, Qfilter(i) \quad (8)$$

$$b_{0,0} = \sum_{i=-3}^{3} A_{I,j}\, Hfilter(i) \quad (9)$$

$c_{0,0}$ and $d_{0,0}$ are calculated in a similar way as shown in (8), (9). $e_{0,0}$,$f_{0,0}$, $g_{0,0}$, $i_{0,0}$,$j_{0,0}$, $k_{0,0}$, $p_{0,0}$, $q_{0,0}$ and $r_{0,0}$ are then calculated with their vertical neighbors $a_{0,0}$, $b_{0,0}$ and $c_{0,0}$:

$$e_{0,0} = \sum_{i=-3}^{3} a_{0,i}\, Qfilter(i) \quad (10)$$

$$k_{0,0} = \sum_{i=-3}^{4} c_{0,j}\, Hfilter(i) \quad (11)$$

...,

It has been proved that the subpixels interpolated by the filter has improved the precision accuracy of the patch-matching process so that the reconstructed images turn out to be much more natural with accurate texture.

Actually, as data surged, subpixel interpolating and matching process brings us a severe time-consuming challenge.

So we reduce the number of reference pixels from eight to four and slightly adjust the filter's coefficients to reduce computational complexity. The altered coefficients are shown in Table. 2

5. Experiments and Analysis

In this part we will demonstrate some experiments results.

5.1 GPU Implementation

Our platform of CPU is Intel (R) Xeon (R), E5620 @2.40GHz, and GPU is GT750Ti (750Ti), NVIDIA. Here are some basic parameters of 750Ti listed in Table 3.

CUDA cores	640
Number of multiprocessors	5
GFLOPs	1305.6
Memory bandwidth	86.4GB/sec
Shared memory	48KB
L2 Cache	2048KB

Table 3. The main parameters of GT750Ti

	CUDA	**C++**	**MATLAB**
Image1(1920x1080)	179ms	53s	347.45s
Image2(1920x1080)	185ms	58s	378.79s
Image3(1920x1080)	164ms	49s	320.62s

Table 4. Time cost of implementation of CUDA, C++ and MATLAB

We took the 1920x1080 pixels images as our mainly study object. 32x32threads are grouped as a block since the maximum number of threads in each block is 1024. And each thread deal with one pixel. So the grid size is (120, 68), and the block size is (32, 32), and the corresponding patch of input image is 16x16.

Considering the continuity of the data and edges, we load 32x32 pixels to each shared memory and 1024Bytes* 4=4KB for floating point images. Taken the intermediate variable into account, the total usage of shared memory is about 24/48=50%. In table. 4 we list some results of our experiment with the above parameters.

The following table 4 shows that processing the same 2k image, the time cost of CUDA/C implementation is much less than C++ implementation and the MATLAB implementation.

5.2 Improvement of Algorithm

We test our algorithm on the Berkeley Segmentation Database. And the parameter for our experiment is: patch size for 5×5, the search window 3×3, and the small upscale factor 1.25 for every SR step.

In order to demonstrate the effectiveness, we compare our algorithm with the current state-of-the-art upscale algorithms, NEDI [12], ICBI [19], and LSE [2]. In Fig 6, obviously, we can see that the HR images produced by our algorithm looks more natural than LSE, like the words on the chip, while have shaper edges and reserve more details than ICBI and NEDI. And compared to LSE, the proposed algorithm has smoother and clearer results with shaper edges. And about the other Fig 6, there is some noise in the original images. We can see that LSE has the highest intensity of noise, especially in the smooth area. And the proposed algorithm successfully remove the noises while preserve the fine texture well.

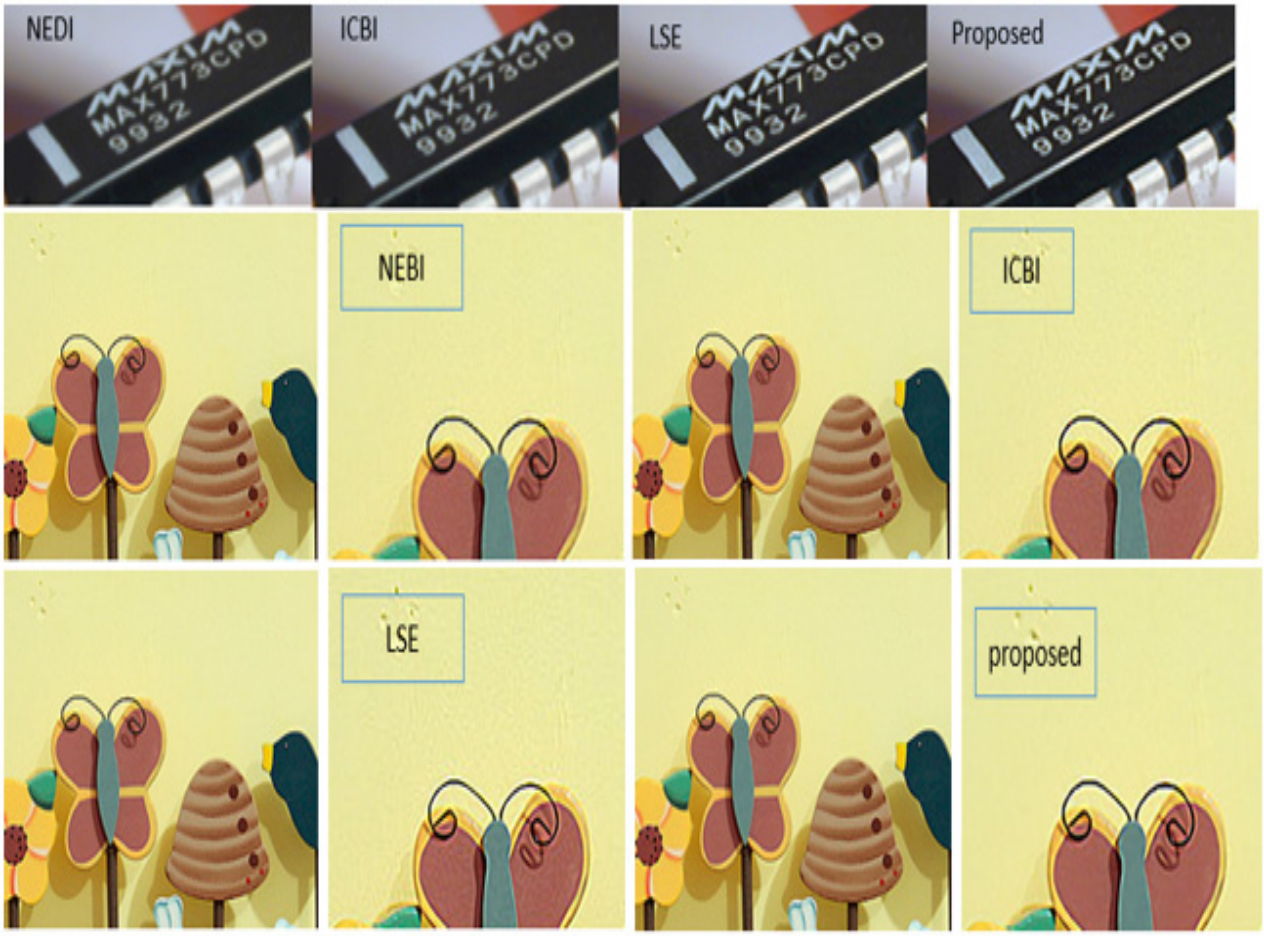

Figure 6. The results of improved LSE

References

[1] Zhang, K., Gao, X., Tao, K, T., Li, X. (2013). Single Image Super-Resolution With Multiscale Similarity Learning. *IEEE Trans Neural Networks and learning systems*, 24 (10) October 2013.

[2] Freedman, G., Fattal, R. (2011). Image and video upscaling from local self-examples. *ACM Transactions on Graphics* (TOG), 28 (3) 1–10.

[3] Glasner, D., Bagon, S., Irani, M. (2009). Super-resolution from a single image. *IEEE International Conference on Computer Vision.*

[4] Chen, S., Mulgrew, B., Grant, P. M. (1993). A clustering technique for digital communications channel equalization using radial basis function networks. *IEEE Trans. Neural Networks*, 4, 570–578, July.

[5] Zhiwei, X., Xiaoyan, S. (2010). Robust Web Image/Video Super-resolution. *IEEE Trans. Image Processing*, 19 (8) 547–588, August.

[6] Freeman, W. T., Jones, T. R., Pasztor, E. C. (2002). Example based super-resolution. *IEEE Computer Graphics and Applications*, 22. 56–65.

[7] Zontak, M., Irani, M. (2011). Internal statistics of a single natural image. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

[8] Shan, Qi, et al. (2008). Fast image/video upsampling. *ACM Transactions on Graphics* (TOG). 27 (5) ACM.

[9] Ebrahimi, M., Vrscay, E. (2007). Solving the inverse problem of image zooming using self-examples. *International Conference on Image Analysis and Recognition,* 117–130.

[10] Fattal, R., et al. (2007). Image upsampling via imposed edge statistics. *ACM Transactions on Graphics* (TOG) 26 (3) 95.

[11] Vivek, K. (2005). et al. Texture optimization for example-based synthesis. *ACM Transactions on Graphics* (TOG). 24 (3). ACM.

[12] Li., Xin., Michael, T. Orchard. (2001). New edge-directed interpolation. *Image Processing, IEEE Transactions on* 10 (10) 1521-1527.

[13] C. Hong, D. Yeung, and Y. Xiong. Super-resolution through neighbor embedding. Computer Vision and Pattern Recognition, 2004. Proceedings of the 2004 IEEE Computer Society Conference on Vol. 1.

[14] Lin, Z., Shum, H.Y. (2014). Fundamental limits of reconstruction-based super-resolution algorithms under local translation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26 (1) 83-97.

[15] Yang, J. et al. (2008). Image super-resolution as sparse representation of raw image patches. Computer Vision and Pattern Recognition, CVPR.

[16] JCTVC. (2013). High Efficiency Video Coding (HEVC) Test Model 10 (HM10) Encoder Description. JCTVC-L1002_v3, Jan.

[17] Fan, W., Yeung, D. (2007). Image hallucination using neighbor embedding over visual primitive manifolds. *In*: Proceedings *IEEE Conf. Computer Vision and Pattern Recognition*, 1–7.

[18] Yang, J., et al. (2013). Fast image super-resolution based on in-place example regression. Computer Vision and Pattern Recognition. CVPR 2.

[19] Andrea, G., Asuni, N. (2011). Real-time artifact-free image upscaling. *Image Processing*, *IEEE Transactions on* 20 (10) 2760-2768.

[20] Yang, M., Wang, Y. F. (2013). A self-learning approach to single image super-resolution. Multimedia, *IEEE Transactions on,* 15, (3), 2013.

[21] http://www.nvidia.com/object/cuda_home_new.html. Technologies of NVIDIA home.

[22] NVIDIA (2013). NVIDIA CUDA Programming Guide, July.