A Computationally Efficient Approach for Exemplar-based Color Image Inpainting using GPU

Dibyam Pradhan*, Naveen M.*, Sai Hareesh A.*, P.K. Baruah, V. Chandrasekaran
Sri Satya Sai Institute of Higher Learning, Prasanthi Nilayam, India
{dibyam4,naveenjoinsu}@gmail.com, {saihareesha,pkbaruah,vchandrasekaran}@sssihl.edu.in

***Abstract*—Image inpainting refers to the process of reconstructing the original image from a damaged one in a visually plausible way. We propose a new gradient-based algorithm for exemplar-based inpainting by making use of L_∞ norm. We implement the most time consuming step of the algorithm on the GPU and compare the serial execution timings against the parallel execution timings. The parallel implementation has an average speedup of 14 over the serial implementation. The results obtained from our approach are perceptually on par and in many cases better than the state-of-the-art approaches to date.**

***Index Terms*—Inpainting, Exemplar-based inpainting, L_∞ norm, L^2 norm**

I. Introduction

The baton of delivering high performance has been carried to a great extent in the recent times by the graphic processing units (GPUs). The evolution of the CUDA programming model has made it possible for the modern GPUs to use massive multithreading for gaining huge application performance. Depending on how well the algorithm lends itself to parallelization, the GPU implementations provide correspondingly greater performance as compared to the CPU implementations.Image inpainting is one such area which could benefit significantly by the use of GPUs.

Image inpainting refers to the process of reconstructing the original image which has been damaged due to factors such as ageing, wear and tear and occlusion. The challenge lies in the fact that the observer seeing the inpainted image should not be able to guess that the image had been tampered with. There are a lot of inpainting techniques available in literature. Some of them are based on PDEs[1], some are statistical-based techniques[2] and some exemplar-based techniques[3] [4]. Due to the greater accuracy of inpainting, the recent times has seen an increasing focus on exemplar-based methods for image inpainting by researchers. The crux of the exemplar-based methods lies in searching the best exemplar or the best patch in the undamaged portion of the image that will be used for filling the damaged portions of the image. Criminisi et al.[3] proposed an algorithm that assigns a priority to each patch on the source region(undamaged region) and finds the best exemplar based on a best-first greedy strategy. Shen et al.[5] have followed a gradient-based inpainting approach with the help of a Poisson equation. The order in which the patches are filled is decided by the values of the average gradients of the patches on the boundary of the to-be filled region.

The approach that we follow is quite similar to that of Hareesh et. al.[6]. Hareesh et.al. have followed a gradient-based approach for filling the damaged portions by choosing a simple function that is a linear combination of the gradient and logarithm of gradient in order to decide the filling priority. The best exemplar is chosen such that it minimizes the L^2 norm between the pixels in the best exemplar and the pixels in the current patch on the boundary of the damaged region(fill region). We varied the algorithm as in [6] by changing the norm from L^2 to L_∞ norm. We observed that this approach leads to much better results than in [6]with regards to both the quality of inpainting and the execution timings of the application as a whole. Apart from adopting a new approach, we also parallelized the most time consuming step of the algorithm using CUDA and achieved remarkable results. We perform a comparison of the CPU and the GPU based implementations and notice a speedup factor of 11 to 18 over the CPU implementation.

In this document, we first discuss the related work on implementing image inpainting on GPUs(next section). We then discuss our sequential exemplar-based inpainting algorithm in Section 4. Section 5 gives the details of the CUDA implementation of the algorithm. Section 6 discusses the results. Section 7 deals with conclusion and future work.

*Student Author

II. Related Work

The literature abounds in algorithms for image inpainting but not many of them have been implemented on the GPU. This may be due to the inherent sequential nature of the algorithm or may be due to the high complexity of the algorithm. However, there are some inpainting algorithms that have been implemented on the GPUs. Kwok et al. [7] have proposed an efficient algorithm for exemplar-based inpainting, in which they separate the exemplars into the frequency coefficients and select only the relevant coefficients. The search for best exemplar is done by the use of a search-array data structure, which can easily be ported to the GPU.

Rosner et. al. [8] have presented efficient algorithms for image warping and image inpainting for frame interpolation and their implementation on the GPU. For each pixel on the boundary of the fill region, they propagate its intensity to the fill region and calculate its distance to the boundary of the fill region. Depending on this distance and the intensity values, the pixel is inpainted. All the above steps are implemented in GPUs. Their GPU implementation had a speedup factor of about 6-10 over the CPU implementation. Chong [9] has followed a texture-synthesis approach to image inpainting. He assigns weights to all the pixels in the undamaged portion of the image and based on these weights, he determines the pixel to be replaced as the damaged pixel that is most constrained by its neighbours. He then replaces the chosen damaged pixel by the pixel with the best neighbourhood match. The determination of the to-be replaced pixel and its replacement is carried out on GPU. Yousef et.al.[10] have tried to optimize the exemplar-based image inpainting method by reducing the number of queries and the arithmetic intensity of each query and by using a different color space, YCbCr. They also achieved good performance improvements.

The approach that we follow and its parallel implementation is not only efficient but also the inpainting results are visually much better than most of the methods that exist in literature[3][6].

III. Exemplar-Based Inpainting Algorithm

We shall now discuss our exemplar-based algorithm. Let us consider an image I which has been tampered with and a region R which comprises of the tampered portion of the image that has to be inpainted(see figure 1). Let B be the boundary of R. The inpainting of the image is carried out by filling the pixels along the boundary B of R by using sequences of patches from the source region, $S = I \backslash R$. This means that the boundary B keeps getting reduced as the damaged or the fill region gets filled and inpainting ends when the boundary B no longer exists.

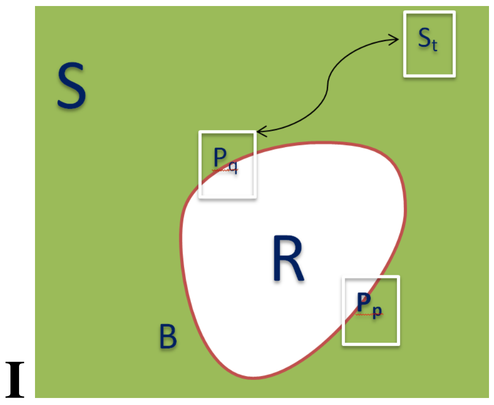


Fig. 1. Depiction of the notations used

The steps of the algorithm could be outlined as under:

1) Locate and mark the boundary B of the fill region.
2) For all $p \in I$, associate a confidence term $C(p)$ such that initially, $C(p) = 0, \forall p \in R$, $C(p) = 1$ otherwise.
3) For each pixel, say $p \in R$, construct a rectangular patch P_p with p as its center. As illustrated in [3], there needs to be a patch-filling order for inpainting of the patches along the boundary. Hence, we compute the priorities of every pixel along the boundary B, by $K(p) = C(p).D(p)$, where $D(p)$ is the data term that signifies the strength of the gradient function at that point as described in [6].
4) Find the patch P_q which has the maximum priority among all the patches centered along the boundary B.
5) Find a patch S_t, in the source region S that is most similar to the patch P_q. The choice of S_t is done such that it minimizes the norm $d(S_t, P_q)$, where d denotes L_∞ norm in the CIE color Lab space (L^*, a^*, b^*). This patch S_t is the known as the best exemplar.
6) Copy image data from S_t to P_q, $\forall p \in P_q \cap R$.
7) Update $C(p), \forall p \in P_q \cap R$.
8) Repeat steps 3 to 7 till B is empty.

IV. Our Implementation

We shall now discuss the serial implementation of the above algorithm and then its parallel implementation. The serial implementation of the algorithm was done in Matlab in conjunction with C. Matlab provides a way of integrating C code into the Matlab code with the help of mex functions. Step 5 of the algorithm, which consists of the search for the best exemplar was implemented in C with the help of mex-files. The rest of the algorithm was implemented in Matlab.

We analyzed the serial implementation of the algorithm and found that the most expensive step in the above algorithm is the search for the best exemplar,i.e. step 5. We found that 70% of the time taken by the entire application is spent on step 5. This suggested us to analyze step 5 of the algorithm and consider possibilities for parallelization. On careful analysis. we found that this step has a lot of scope for parallelization. The serial implementation of step 5 consists of the following steps:

1) For each patch in the source region S(see figure 1), calculate the sum-squared error (SSE)of each pixel in the current patch of S over the corresponding to-be filled pixel in the target patch P_q.
2) Find the patch with the least error. This patch becomes the best exemplar.

The above steps have a lot of data-parallelism and could be easily parallelized with the help of the CUDA programming model. Also, NVIDIA provides good support for using Matlab with CUDA and hence, we ported step 5, the best exemplar search to GPUs using CUDA.

We shall now first discuss the parallel implementation of step 5 as follows:

1) Copy the source image from the CPU host memory to the GPU texture memory and copy the image which has been marked with the region to be filled(fill image) and the current patch from the CPU host memory to the global memory in GPU.
2) We then launch a kernel with the total number of threads equaling the total number of patches in the source region S.
3) Make each thread responsible for a patch in the source region S. Each thread now calculates the SSE(sum-squared error) for all the pixels in its patch and stores the error values in global memory. Each patch is associated with an error value.
4) After all the threads have completed their work, we just need to find the patch with the minimum error which becomes the best exemplar. This step is done sequentially.

V. Results

The runtime measurements for both the CPU and the GPU implementations were made primarily on an Intel Quadcore machine equipped with NVIDIA Tesla C2050 graphics card. We perform the tests on three images of different sizes. The first image is the well-known bungee image (206 x 308) as can be seen in figure 2. The second image is windows xp's beach image (238 x 180) and a surfing image(750 x 500) (see figures 3 and 4). All these images are three channel images. In the case of the bungee image, we wish to remove the bungee jumper from the original image in such a way that the observer would not notice that there was a jumper in the original image. Hence, we mark this region with green color.

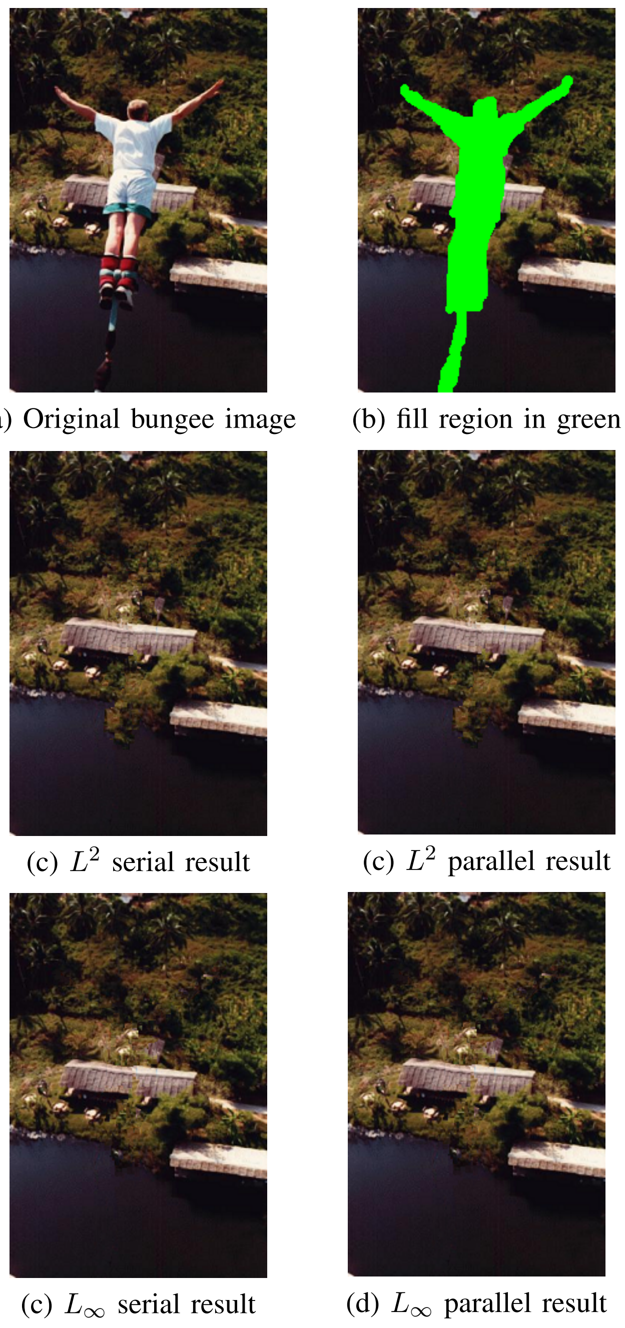

(a) Original bungee image
(b) fill region in green
(c) L^2 serial result
(c) L^2 parallel result
(c) L_∞ serial result
(d) L_∞ parallel result

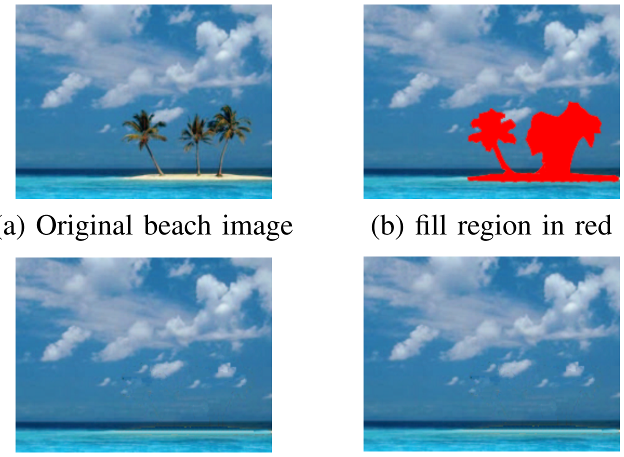
Fig. 2. Results of inpainting on the bungee image

Similarly, we mark the fill region with red color in the beach image. In the surfing image, we wish to remove the region marked by green color.

We first perform quality test for the inpainted image by visual inspection and compare the resultant images of our CPU and the GPU implementation which uses L_∞ norm against the approach followed in [6] which uses the L^2 norm. We then compare the runtime measurements of our implementation against the approach in [6]. Not only did we parallelize our method, we also parallelized the method as in [6] using a similar approach as above to have a better comparison of the results. We shall now discuss the results in detail.

A. Quality Test

Figures 2 to 4 display the results of the CUDA and C implementations of the algorithm for the three images. Figure 2 also depicts the results of the L_∞ approach. The quality of the results in the case of CUDA implementation is the same as that of the serial implementation without any plausible change as could be seen in figure 2. This was the case for all the images. Also, it was observed that the quality of the image for our proposed L_∞ norm yielded much better results in terms of quality of inpainting and execution time than the L^2 norm as can be seen in figure 2.



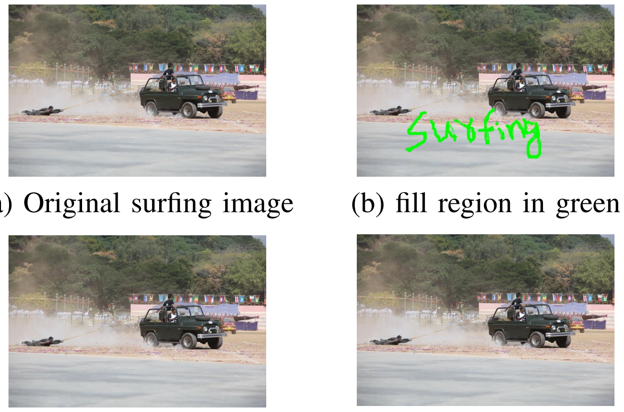
(a) Original beach image (b) fill region in red

(c) L_∞ serial result (d) L_∞ parallel result

Fig. 3. Results of inpainting on the beach image

B. Runtime Test

The runtime calculation for all the images is done first on the Quadcore machine with Tesla C2050 for both the serial and parallel implementations of the best exemplar search part of the algorithm. For the purpose



(a) Original surfing image (b) fill region in green

(c) L_∞ serial result (d) L_∞ parallel result

Fig. 4. Results of inpainting on the surfing image

Image name	Serial timing	CUDA timings	Speedup
bungee	13.2444	1.0455	12.66
beach	6.0831	0.5043	12.06
surfing	151.1735	8.0349	18.81

TABLE I
EXECUTION TIMINGS IN SECONDS FOR BEST EXEMPLAR SEARCH USING THE L^2 NORM IN TESLA C2050

image name	Serial timing	CUDA timing	Speedup
bungee	12.8163	1.1048	11.60
beach	5.598	0.4954	11.30
surfing	126.8789	7.0616	17.97

TABLE II
EXECUTION TIMINGS IN SECONDS FOR BEST EXEMPLAR SEARCH USING THE L_∞ NORM IN TESLA C2050

of correctness of the results, we only note down the total time spent by the entire application in searching the best exemplar. Table 1 displays the runtime measurements for the serial versus CUDA implementation (of best exemplar search) using the L^2 norm and the effective speedup obtained. A speedup factor of 12, 12 and 18 were obtained in the case of bungee image, beach image and the surfing image respectively. Table 2 displays the runtime measurements for the L_∞ norm and the speed-up obtained. In this case, a speedup of 11 to 18 were obtained for the three images. This proves that the CUDA implementation outperforms the serial implementation with respect to the execution timings. Note that the execution timings for the proposed method (L_∞) is much lesser than that of the L^2 approach. Hence, the proposed method is better than the L^2 method both in terms of quality of inpainting as well as execution

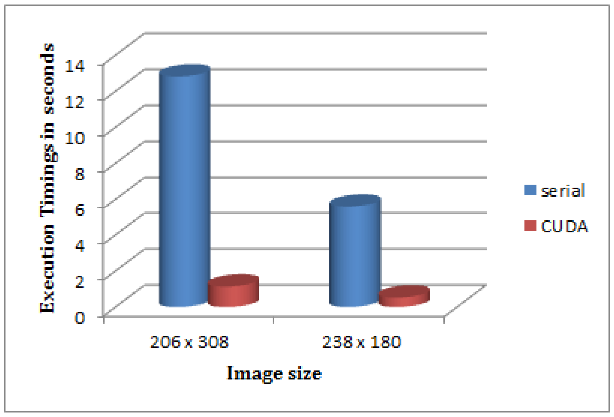


Fig. 5. Execution timings against image sizes for the bungee image(206x308) and the beach image(238x180) using our proposed L_∞ norm

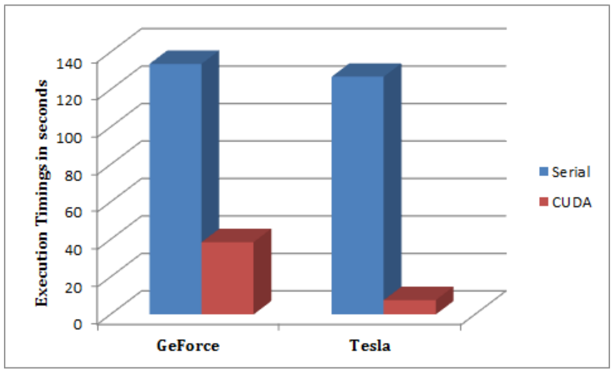


Fig. 6. Execution timings for the surfing image(750x500) for the Tesla C2050 and GeForce 310M using our proposed L_∞ norm

timings.

Figure 5 compares the execution timings for the bungee image and the beach image against the image size for the proposed method. In both cases, there is a seemingly greater reduction in the execution time in parallel implementation as against the serial implementation with an increase in the size of the image. Figure 6 compares the execution timings for the surfing image when implemented in the core i3 machine with NVIDIA GeForce 310M graphics card as against the implementation on Tesla C2050 machine. As is expected, the reduction in execution timings after parallelization is far better in Tesla than in GeForce 310M.

VI. Conclusion and Future Work

A new gradient-based algorithm for exemplar-based color image inpainting using the L_∞ norm was proposed and the results were found to be computationally more efficient as well as visually more plausible than the existing methods. The most time consuming step of the algorithm was implemented parallely on the GPU and an average speedup of 14 over the sequential implementation was observed.

For future work, we propose to consider the usage of the L_∞ norm in other inpainting approaches. The proposed exemplar-based approach along with the parallel implementation could be extended to image segmentation, image blurring and also super-resolution as these methods are computationally more intensive than the inpainting methods. Also, the proposed algorithm could be implemented totally in CUDA by using the OpenCV library for image processing.

Acknowledgment

We dedicate this work to our founder Chancellor, Bhagawan Sri Sathya Sai Baba. We also acknowledge NVIDIA Pune, for providing all the computational facilities.

References

[1] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester, “Image inpainting,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 2000, SIGGRAPH ’00, pp. 417–424, ACM Press/Addison-Wesley Publishing Co.

[2] Anat Levin, Assaf Zomet, and Yair Weiss, “Learning how to inpaint from global image statistics,” *Computer Vision, IEEE International Conference on*, vol. 1, pp. 305, 2003.

[3] A. Criminisi, P. Prez, and K. Toyama, “Region filling and object removal by exemplar-based image inpainting,” *IEEE Transactions on Image Processing*, vol. 13, pp. 1200–1212, 2004.

[4] Jiying Wu and Qiuqi Ruan, “Object removal by cross isophotes exemplar-based inpainting,” in *Proceedings of the 18th International Conference on Pattern Recognition - Volume 03*, Washington, DC, USA, 2006, ICPR ’06, pp. 810–813, IEEE Computer Society.

[5] Jianbing Shen, Xiaogang Jin, Chuan Zhou, and Charlie C. L. Wang, “Technical section: Gradient based image completion by solving the poisson equation,” *Comput. Graph.*, vol. 31, pp. 119–126, January 2007.

[6] Sai Hareesh Anamandra and Venkatachalam Chandrasekaran, “Exemplar-based color image inpainting using a simple and effective gradient function,” in *IPCV*, 2010, pp. 140–145.

[7] Tsz-Ho Kwok, Hoi Sheung, and Charlie C. L. Wang, “Fast query for exemplar-based image completion,” *Trans. Img. Proc.*, vol. 19, pp. 3106–3115, December 2010.

[8] Jakub Rosner, Hannes Fassold, Peter Schallauer, and Werner Bailer, “Fast gpu-based image warping and inpainting for frame interpolation,” *International Conferences on Computer Graphics, Vision and Mathematics, GraVisMa 2010.*

[9] Hamilton Chong, “Gpu image inpainting via texture synthesis,” *http://www.eecs.harvard.edu/ hchong/goodies/inpaint.pdf.*

[10] Mohamed Yousef and Khaled F. Husien, “Par xii: Optimized, data-parallel exemplar-based image inpainting,” *SIGGRAPH 2011 poster.*