Parallel-Friendly Frame Rate Up-Conversion Algorithm Based on Patch Match

Xu Jiang(✉), Li Chen, Zhiyong Gao, and Xiaoyun Zhang

Shanghai Jiao Tong University, Shanghai, 200240, China
{xu.jiang,hilichen,zhiyong.gao,xiaoyun.zhang}@sjtu.edu.cn

Abstract. In this paper, we propose a parallel-friendly frame rate up-conversion (FRUC) algorithm based on the patch match. The key points of the algorithm are the generation of self-similarity patch, parallel true motion estimation and motion vectors post-processing dealing with occlusion problem. Our algorithm is fully parallel at each step and can be implemented on GPU. The GPU implementation achieves up to 48 times speedup over its CPU implementation. Compared with the traditional FRUC algorithm based on 3-D Recursive Search, our algorithm has a good performance in handling the fine motion structure and occlusion problem.

Keywords: FRUC · Parallel true motion estimation · GPU · Patch Match

1 Introduction

Frame rate up-conversion (FRUC) is the technique that increases the frame rate by generating the intermediate frame between the original frame in the video. FRUC is commonly applied to increase the time resolution and reduce the effect of motion blur. The main approaches can be classified into three steps, motion estimation (ME), motion vectors post-processing (MVPP), and motion compensated interpolation (MCI). Among the three steps, ME is the most basic and important step. 3-D Recursive Search (3DRS) [1] is a good performing true-motion estimation algorithm with low computational complexity. And many improved methods [2, 3] based on 3DRS have been proposed. Because 3DRS is based on the assumption that most objects are much larger than a block, it recursively uses Motion Vectors (MVs) of spatial and temporal neighboring blocks to estimate the MV of a block [2]. The assumption indicates that it is not a good motion estimation method for a small object. And the spatial recursion, which depends on the precomputation of the spatial neighbor block, is obviously sequential execution. This prevents the algorithm to be applied in parallel computation. Besides 3DRS, optical flow estimation is a more accurate algorithm. But the high computational complexity limits the optical flow estimation in practical application.

Recently, numerous research proposed parallel-friendly algorithms which are suitable to be implemented on GPU to gain significant speedup [4]. In [7], this parallel true motion estimation algorithm adjust the 3DRS to use the temporal recursion only. But this result in more convergence time. In [8], the algorithm applies jump flooding scheme

X. Yang and G. Zhai (Eds.): IFTC 2016, CCIS 685, pp. 146–154, 2017.
DOI: 10.1007/978-981-10-4211-9_15

proposed by Rong and Tan [5] for Patch Match. In [6], the Patch Match is applied in EPPM algorithm, and it proves that carefully crafted local methods can reach quality on par with global ones.

In this paper, we propose a parallel-friendly algorithm for FRUC, and implement it on GPU. We focus on improving the ME performance of the small object and handling the occlusion problem. Compared with 3DRS, proposed algorithm is based on dense Motion Vector Field (MVF), which can handle the fine motion structure. Compared with optical flow estimation, the adapted Patch Match in parallel reduces the high computational complexity. Our algorithm can also be divided into three steps. The detail of proposed algorithm is presented in Sect. 2. Section 3 describes the implementation of the proposed algorithm in CUDA. The result of the proposed algorithm is presented in Sect. 4. Section 5 concludes the paper.

2 Parallel-Friendly Frame Rate Up-Conversion Algorithm

Our algorithm can be classified into three steps: ME, MVPP and MCI. ME is based on Patch Match to realize parallelization, and through this step, we obtain the forward and backward MVF. Then MVPP is designed to eliminate and correct the outliers in forward and backward MVF. Finally, the MCI step generates the intermediate frame. The outline of proposed algorithm is depicted in Fig. 1.

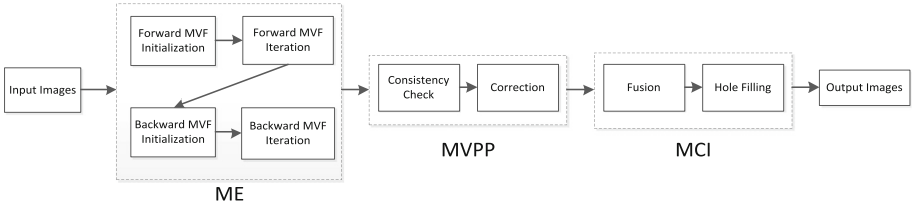


Fig. 1. Outline of the proposed algorithm.

2.1 Motion Estimation

The main idea of original Patch Match is to initialize a random correspondence field and then iteratively propagate good guesses among neighboring pixels [6]. The method has been adapted for the application of FRUC. Two significant changes are made to enhance the performance. Firstly, we change the block to self-similarity patch, which can improve the accuracy of ME. Secondly, we restrict the search range from the whole image to a certain window. Because it is impossible for the object in a video to move from one side to another side within two frames. The adapted Patch Match can be divided into 4 parts: Patch Generation, Initialization, Propagation and Random Search. The process of acquiring the forward MVF is presented as follow.

Patch Generation. To compute the matching cost the of a MV, we allocate a block $\mathbf{B}(x,y)$ for each pixel (x,y) to compute the Sum of Absolute Differences (SAD). Notice

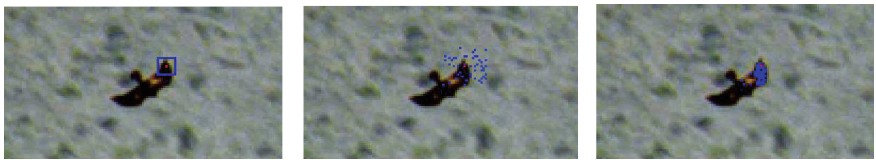
that, with the increase of the radius, the computational load shows quadratic increase. This suggests a natural way to reduce the computational complexity which is that we only use the n most similar pixel to compute SAD. In other words, we ignore dissimilar pixel to the center pixel. For each pixel (x,y), we select the n most similar pixel from its neighborhood, store them in the array **Patch**(x,y), and only use the pixel belonging to the **Patch**(x,y) to compute the matching cost. Specifically, the matching cost is defined as follows,

$$\mathrm{D}(\mathbf{a}, \mathrm{MV}(\mathbf{a})) = \sum_{\mathbf{b} \in \mathrm{Patch}(\mathbf{a})} |\mathrm{F_t}(\mathbf{b}) - \mathrm{F_{t+1}}(\mathbf{b} + \mathrm{MV}(\mathbf{a}))| \tag{1}$$

where $\mathrm{F_t}(.)$ represents the frame at time t, $\mathbf{a}(x_a,y_a)$ is the pixel in $\mathrm{F_t}$, $\mathbf{b}(x_b,y_b)$ represents the pixel belonging to **Patch**$(\mathbf{a})$, **MV**$(\mathbf{a})$ is the motion vector at position $\mathbf{a}$.

Then a problem raised is that the brute-force preselection of n similar pixels for each pixel actually can be too slow, especially when patch size is large, which may cancel out a large portion of the speed gain of the Patch Match [6]. Note that the straight implementation of the selection process takes $O(Mn)$ complexity for each pixel.

For the sake of speeding up, our algorithm utilize the similarity between the adjacent pixel to get the approximate n most similar pixel. The step of **Patch**(x,y) generation is as follows: For each pixel (x,y), we randomly select n pixels from its neighbor pixel and store them into a vector **Patch**(x,y) in order of their similarity to the center pixel (x,y); Then we merge the adjacent pixel similarity vector **Patch**$(x-1,y)$, **Patch**$(x,y-1)$, **Patch**$(x+1,y)$ and **Patch**$(x,y+1)$ into **Patch**(x,y). Since we do not intend to search exactly the top n similar pixel for each pixel, the algorithm does not iterate more. All the steps is depicted in Fig. 2, with $n = 50$ and patch size is 21×21. Note that all the pixel in one patch tend to own the same true MV intuitionally, which improves the accuracy of SAD criterion, and we will use this property in the MVPP step.



(a) Block (b) Patch after Initialization (c)Patch after merging

Fig. 2. Illustration of patch generation. (a) shows the 7×7 block around the center pixel. (b) shows the random initialization within 21×21 range around the center. (c) shows the final patch after merging. All the members belong to one patch can be seen as part of one object, so they usually own the same true motion vector.

Initialization. The MVF can be initialized by assigning random MV, or by using prior information. When initialize the forward MVF, we use the random values clamped to a certain value w (usually one eighth of the width of the image), because it is impossible for the object in a video to move from one side to another side within two frames. Then we use formula (1) to compute the matching cost for each MV. Note that after the

acquirement of the forward MVF, we can use the prior information to initialize the backward MVF, which can reduce the number of iterations for backward MVF.

Iteration - Propagation. After initialization, we perform the iterative process of propagation to improve the accuracy of MVF. The propagation mode is based on jump flooding algorithm [5] to realize parallel computing, but we adapt the method for the FRUC. This process contains many rounds. For example, there are $\log l + 1$ rounds with halved step length of l, $l/2$, $l/4$, …, 1. Figure 3 clarifies how the MVs is propagated. In each round with step l, we renew the $\mathbf{MV}(\mathbf{a})$ by $\mathbf{MV}(\mathbf{a} + \mathbf{b})$, as b satisfies the follow equation:

$$\mathbf{b} = arg\ min\ \mathrm{D}(\mathbf{a}, MV(\mathbf{a} + \mathbf{b})),\ \mathbf{b} \in \{(\pm l, 0), (0, \pm l), (0, 0)\} \tag{2}$$

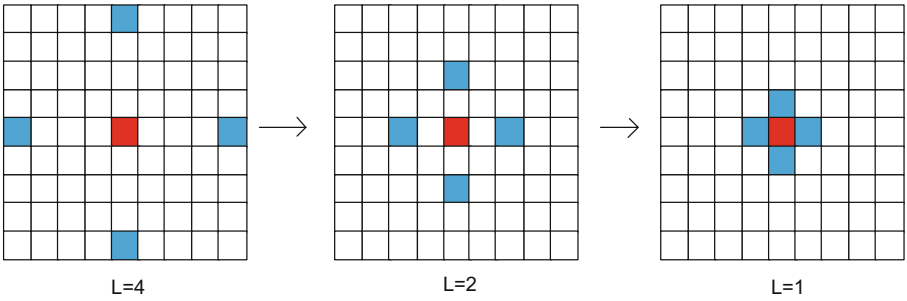


Fig. 3. Illustration of propagation. In each round, the red pixel inquiries the MVs of four blue neighbors as candidate MVs. (Color figure online)

Iteration - Random Search. Then we perform the iterative process of random search to improve the MVF. Let $\mathbf{mv}_0 = \mathbf{MV}(\mathbf{a})$. We attempt to improve $\mathbf{MV}(\mathbf{a})$ by testing a sequence of candidate MVs at an exponentially decreasing distance from $\mathbf{mv}_0$:

$$\mathbf{u}_i = \mathbf{mv}_0 + w\alpha^i \mathbf{R}_i \tag{3}$$

where $\mathbf{R}_i$ is a uniform random in $[-1,1] \times [-1,1]$, w is the clamped value mentioned above, and α is a fixed exponentially decreasing ratio. We examine $\mathbf{u}_i$ for $i = 0, 1, 2.$ until $w\alpha^i$ is below 1 pixel. In our application, $\alpha = 1/2$.

After computing with 2–3 iterations, the forward MVF has almost always converged in practice. Then follow almost the same operations except the initialization step, computing 1–2 iterations, we can also get backward MVF. Figure 4 illustrates the intermediate result of all the steps in motion estimation.

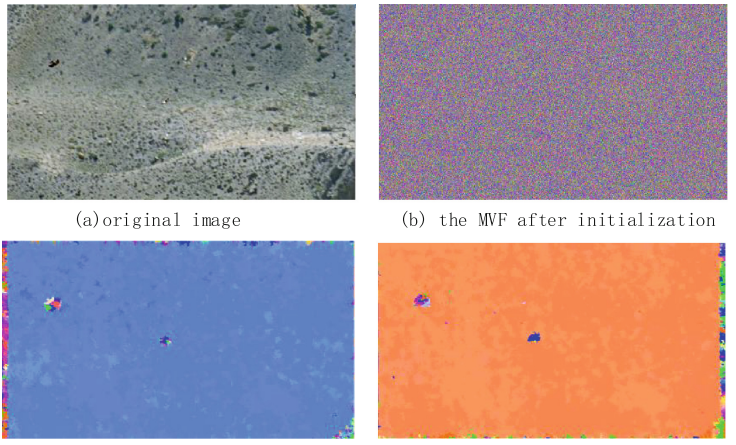


Fig. 4. Illustration of Motion Estimation. (a) is the original image. (b, c, d) show the MVF at different stages, with magnitude visualized as saturation and angle visualized as hue. For (c) and (d), there is some noise introduced by the SAD metric. But most noise gathers in the occlusion region (just like the up and left side of the forward MVF, and down and right side of the backward MVF).

2.2 Motion Vector Post-processing

After Motion Estimation, we can get the forward and backward MVF. As shown in Fig. 5, there are a lot of noise in both MVFs before MVPP. Because of the aperture problem, the SAD metric alone is not reliable enough to determine the true motion [3]. In addition to the noise introduced by SAD metric, there are many wrong MVs in occlusion region. So it is necessary to take MVPP step to eliminate outliers and to handle occlusion problem.

Consistency Check. After computing forward and backward MVFs between two frames, we explicitly perform forward-backward consistency check [9] to detect wrong MVs. All the inconsistent MVs is marked and wait to be corrected by the next step.

Correction. After the Consistency Check, we treat the consistent MVs as high-credibility MVs, and we will use these MVs to fix the wrong MVs. As we assumed in the Patch step, most pixels in the same Patch own the same true MV. So for each wrong **MV**(x,y), we will select 5 alternative **MV**(i,j), (i,j) belong to **Patch**(x,y) and **MV**(i,j) is high-credibility MVs. Vector Median Filter is applied to choose the fixing MV from 5 alternative MVs. The fixing result is depicted in Fig. 5.

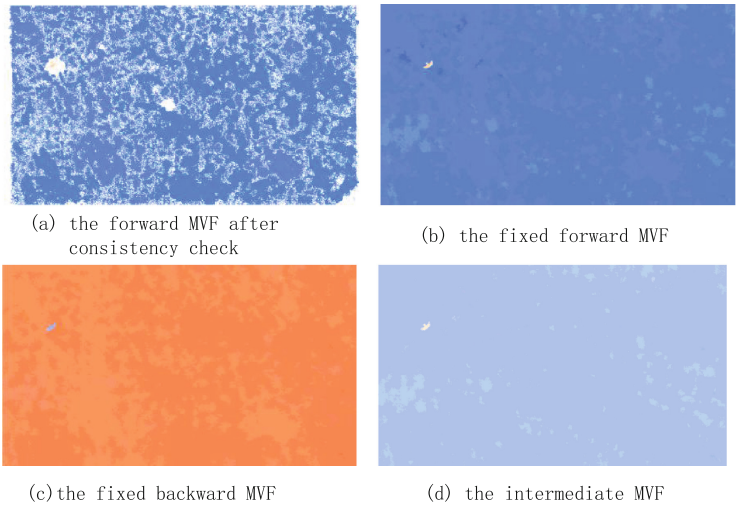


Fig. 5. Illustration of MVPP and MCI. (a) shows the result of consistency check. All the blank region is the inconsistent region. (b) and (c) shows the MVF after correction. (d) shows the fusion result of the forward MVF and backward MVF.

2.3 Motion Compensated Interpolation

After the MVPP, we get two more reliable MVFs. Then we fuse the forward and backward MVFs into intermediate MVF for interpolation. But there may be still some holes remained after fusion. So we use the vector median filter again on the intermediate MVF to fill all the holes. The intermediate frame is constructed by the full intermediate MVF. Before outputting the intermediate frame, a 3×3 median filter is applied on the intermediate frame to filter the pepper noise along the motion edge of the object. The full intermediate MVF is shown in Fig. 5.

Fusion. For the pixel $\mathbf{a}(x_a,y_a)$ in F_t, and pixel $\mathbf{b}(x_b,y_b)$ in F_{t+1}, we define the intermediate MVF as follows:

$$\begin{aligned} \mathrm{MV}_{inter}(\mathbf{a} + \frac{1}{2}MV_{forward}(\mathbf{a})) &= \frac{1}{2}MV_{forward}(\mathbf{a}) \; or \\ \mathrm{MV}_{inter}(\mathbf{b} + \frac{1}{2}MV_{backward}(\mathbf{b})) &= -\frac{1}{2}MV_{backward}(\mathbf{b}) \end{aligned} \tag{4}$$

There may be conflicts in some positions. For example, pixel **a** and **b** may map to the same position. In order to avoid conflicts, we give first priority to the consistent MVs for selection and second priority to the fixed MVs. If two conflicted MVs are of the same priority, we will choose the one with smaller matching cost.

Hole Filling. There may be still some holes remained after fusion. So we use the Vector Median Filter again on the intermediate MVF to fill all the holes. For each hole, we will

choose 5 nearest MVs in Manhattan distance on intermediate MVF as alternative MVs. Then Vector Median Filter is applied to choose the filling MV from alternative MVs.

3 CUDA Implementation

On account of the inherently parallel-friendly nature, our algorithm is implemented efficiently on GPU with CUDA. In detail, the block size 16×16 and the whole thread size 960×540 (equal to the resolution of the video) is unified in our CUDA implementation. The outline of our program is described in three part:

For ME part, only two kernel function is needed. The first kernel function is paralleled at patch-level, i.e. each thread in the kernel takes charge of the generation of self-similarity patch for each pixel. The patches of different threads are mutually exclusive. The main task of second kernel function is to acquire MVF. Since the threads of different blocks can not be synchronized, we have to store two MVF, one for the former round, and one for the current round. In initialization procedure, forward MVF use random MVs, and backward MVF shares the information from forward MVF. In the propagation procedure, each thread inquiries the MVs stored in the former round, compares it to the one in current round, and choose the best MVs for each pixel. After propagation, random search is applied to improve the MVF.

For MVPP part, two kernel function is designed for Consistency Check and Correction respectively. The Consistency Check function is paralleled at pixel-level, and each thread in the kernel check the consistency of the MV for the corresponding pixel. Then the Correction function is launched twice to correct the wrong MVs for forward MVF and backward MVF.

For MCI part, the Fusion and Hole Filling procedure correspond to two kernel function. The Fusion function is paralleled at MV-level, and each thread map the corresponding MV into the intermediate MVF. Then the Hole Filling function is launched to fill the holes remained in intermediate MVF.

4 Experiment Result

In our implementations, the hardware is Core^{TM} i7-3770 k 3.4 GHz and NVIDIA^{TM} GeForce GTX 780Ti with 1 GB dedicated graphic memory. Our algorithm is inherently parallel and can be implemented on GPU with CUDA as mentioned above. We choose the video with the resolution 960×540 for experiment. The GPU implementation takes 521 ms on average to generate an intermediate frame. Compared with 25 s for the CPU implementation, the GPU implementation achieves up to 48 times speedup over its CPU implementation. So far, the only parallel FRUC algorithm [7] has some defects, like long convergence time and less effective than the 3DRS. Therefore, the FRUC algorithm [10] based on 3DRS with 8×8 block is chosen as benchmark algorithm.

4.1 Subjective Evaluation

We choose three classic videos with different features to judge the performance of our FRUC algorithm. Figure 6 presents the results of our FRUC algorithm and benchmark algorithm mentioned above. For the first video, the most difficult part is to estimate the motion for the little black hawk on the top left corner, because it is very easy to be disturbed by the large background. Our algorithm shows a good performance for the fine motion structure. For the second video, the fast move of caterpillar and large occlusion region is a big challenge. The intermediate frame presents that our algorithm can tract the fast motion and handle occlusion well. For the third video, it is hard to keep the gird of window. As depicted in the result, we keep main grid, but still failed in some positions. It suggest that there is still a potential for improvement.

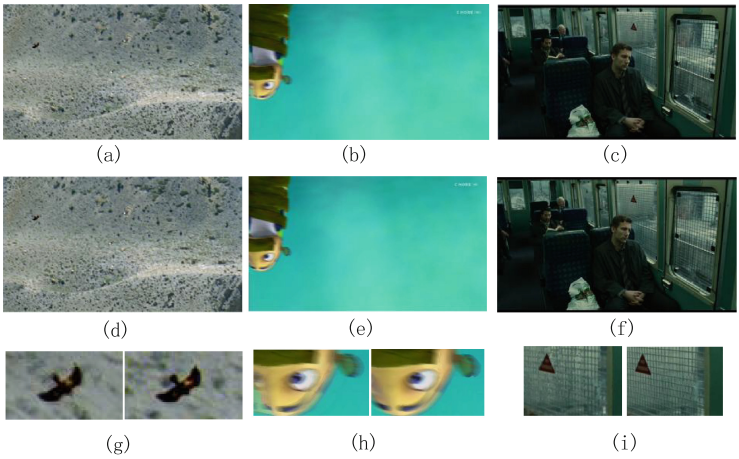


Fig. 6. Visual comparison of interpolated frames by different FRUC algorithms. (a, b, c) benchmark algorithm. (d, e, f) proposed algorithm. (g, h, i) are the partial enlarged details of the result. Left side is from benchmark algorithm, and right side is from proposed algorithm.

4.2 Objective Evaluation

PSNR is applied to evaluate the objective quality of the proposed algorithm and the benchmark algorithm. Table 1 presents the PSNR of the first 30 interpolated frames (to avoid the influence of the scene change) of the attackOfTheHawks, HMoveFast, and COMTrain sequences. As shown in the table, the proposed algorithm provides better PSNR performance than the benchmark algorithm.

Table 1. Average PSNR (dB) obtained by proposed algorithm and bench mark algorithm.

Sequences	attackOfTheHawks	HMoveFast	COMTrain
Proposed	28.23	36.57	24.28
Benchmark	26.34	34.23	23.36
Gain	1.89	2.34	0.92

5 Conclusion

In this paper, we propose a parallel-friendly FRUC algorithm based on patch match. This algorithm is inherently parallel and can be implemented on GPU with CUDA. The speedup and the quality of intermediate frame are analyzed. The GPU implementation achieves up to 48 times speedup over its CPU implementation. Compared with the FRUC based on 3DRS, proposed algorithm has advantage in the fine motion structure and occlusion problem.

Acknowledgment. This work was supported in part by Chinese National Key S&T Special Program (2013ZX01033001-002-002), National Natural Science Foundation of China (61133009, 61221001, 61301116), the Shanghai Key Laboratory of Digital Media Processing and Transmissions (STCSM 12DZ2272600).

References

1. De Haan, G., Biezen, P.W.A.C., Huijgen, H., et al.: True-motion estimation with 3-D recursive search block matching. IEEE Trans. Circ. Syst. Video Technol. **3**(5), 368–379, 388 (1993)
2. Kim, D.Y., Park, H.W.: An efficient motion-compensated frame interpolation method using temporal, information for high-resolution videos. J. Disp. Technol. **11**(7), 1 (2015)
3. Braspenning, R.A.C., Haan, G.D.: True-motion estimation using feature correspondences. In: Proceedings of SPIE - The International Society for Optical Engineering, vol. 5308, pp. 396–407 (2004)
4. Yu, P., Yang, X., Chen, L.: Parallel-friendly patch match based on jump flooding. In: Zhang, W., Yang, X., Xu, Z., An, P., Liu, Q., Lu, Y. (eds.) IFTC 2012. CCIS, vol. 331, pp. 15–21. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34595-1_3
5. Rong, G., Tan, T.S.: Jump flooding in GPU with applications to Voronoi diagram and distance transform. In: Symposium on Interactive 3d Graphics, Si3d 2006, Redwood City, California, USA, 14–17 March 2006, pp. 109–116 (2006)
6. Bao, L., Yang, Q., Jin, H.: Fast edge-preserving patchmatch for large displacement optical flow. IEEE Trans. Image Process. **23**(12), 4996–5006 (2014). A Publication of the IEEE Signal Processing Society
7. Michielin, F., Calvagno, G., Sartor, P., et al.: A parallel true motion estimation method based on binarized cross correlation. In: IEEE Third International Conference on Consumer Electronics, Berlin, pp. 1198–1202 (2013)
8. Barnes, C., Shechtman, E., Finkelstein, A., et al.: PatchMatch: a randomized correspondence algorithm for structural image editing. Acm Trans. Graph. **28**(3), 341–352 (2009). Article 24
9. Hosni, A., Rhemann, C., Bleyer, M., et al.: Fast cost-volume filtering for visual correspondence and beyond. IEEE Trans. Pattern Anal. Mach. Intell. **35**(2), 504–511 (2013)
10. Guo, Y., Chen, L., Gao, Z., et al.: Frame rate up-conversion method for video processing applications. IEEE Trans. Broadcast. **60**(4), 659–669 (2014)