I-015

GPU を用いたぼけ・ぶれ画像のブラインド復元
Blind Image Restoration for Blurred Images implemented on GPU

大竹 翔太 Shota Otake　永田 敦史 Atsushi Nagata　後藤 富朗 Tomio Goto　平野 智 Satoshi Hirano　櫻井 優 Masaru Sakurai

***Abstract*— 一般に、ぼけやぶれを含んだ画像は理想的とは言えない。そこで、ぼけやぶれを含んだ画像から光の拡散係数（ぼけ・ぶれ）を推定し、逆畳みこみを行うことでぼけ・ぶれを除去した画像を得るブラインドデコンボリューションと呼ばれる手法が存在する。しかし、ぼけ・ぶれの推定には大きな計算コストが必要となり、リアルタイム処理が実用上の課題である。そこで、本研究ではぼけぶれ推定の処理に GPU を用いることで処理の高速化を図った。GPU を用いて処理を行った場合と CPU のみで処理を行った場合の実行時間を測定し、ぼけ・ぶれ補正に GPU を用いることの有用性を示す。**

Keywords*—*ぼけ・ぶれ補正; GPU 実装;; ブラインドデコンボリューション

I. はじめに

スマートフォンやデジタルカメラの普及に伴い、写真を撮影することは一般的となっている。しかし、多くの写真には手振れやピントのずれによって望ましくない画質劣化が生じている。この問題を解決するために、光学的な手振れ補正機能がいくつかのデジタルカメラに組み込まれているが、実際には不十分であり画像処理による補正が必要である。

ローカルパッチに対して光の拡散係数(PSF)の推定と理想画像の推定を交互に繰り返して画像復元を行う手法が提案された [1]。しかし、この手法は処理コストが大きく処理に時間がかかってしまう。本論文では、ブラインド画像復元アルゴリズムを GPU に実装し、処理時間の削減を行った。GPU に実装した場合の処理時間と CPU に実装した場合の処理時間を比較し、GPU に実装することの優位性を示す。

II. ぼけ・ぶれモデル

ぼけ・ぶれ画像復元は未知の一様なぼけ・ぶれ関数である PSF を求めることで行われる。PSF は式(1)で表される。

$$g = f * h + n \tag{1}$$

ここで、$*$ は畳み込み演算子であり、g はぼけ・ぶれ画像、f は理想画像、h は PSF、n は未知のノイズである。一般に、ノイズ n はガウシアンノイズとして考えられる。本論文では、このモデルに従って画像復元を考える。

一枚のぼけ・ぶれ画像からの画像復元はブラインドデコンボリューションと呼ばれる。ノンブラインドデコンボリューテョンが既知の PSF を用いるのに対し、ブラインドデコンボリューションでは PSF が未知である。一般に、ブラインドデコンボリューションは不良設定問題であり、最適な PSF を一意に求めることは困難である。

III. ブラインドデコンボリューション

本論文では、理想画像の推定と PSF の推定に評価関数の最小化の繰り返しを用いている。特に、理想画像推定では TV (Total Variation) 正則化によってテクスチャを除去しショックフィルタを掛けることによってエッジを強調している。これは評価関数の収束性能を向上させる役割を持つ。さらに、デコンボリューションでは処理時間を削減し復元性能を向上させるために D. Krishnan らが提案した手法 [2] を用いている。また、PSF 推定フェーズでは画像の一部であるパッチに対して処理を行うことにより高速化を図っている。

Fig. 1 に、ブラインドデコンボリューションのブロック図を示す。パッチ抽出、理想画像推定、PSF 推定、最終デコンボリューションで構成される。まず、PSF 推定フェーズのために画像の一部（ぼけ・ぶれを多く含むエッジ成分）をパッチとして抽出し、輝度成分に対して処理を行う。PSF 推定フェーズでは効果的な推定のために繰り返し毎に PSF サイズを段階的に大きくする。初期状態は 3×3 ピクセルに設定され、ステップ毎に $\sqrt{2}$ 倍となり、最終的に元の PSF サイズとする。最後に推定された PSF とぼけ・ぶれ画像を用いて最終デコンボリューションが行われ、復元画像を得る。

A. パッチ選択

計算時間を削減するため、PSF 推定フェーズの処理を考慮して最適なパッチを選択する。ラプラシアンフィルタとソーベルフィルタ(垂直、水平、両方)からエッジマップを作成し、最適なパッチを選択する。

B. 理想画像推定(x-step)

理想画像推定はデコンボリューション、TV 正則化、ショックフィルタによって構成される。一時的なデコンボリューション画像は式(2)によって得られ、D. Krishnan らの手法 [2] を用いて、式(2)を最小化する。ルックアップテーブルを用いることによって処理時間は大幅に削減できる。また、R. Liu らの境界処理 [3] を用いることでリンギングを削減している。

$$\min_x \sum_{i=1}^{N} \left(\frac{\lambda}{2} (x \otimes h - g)_i^2 + \sum_{j=1}^{J} |(x \otimes f_i)_i|^{\alpha} \right) \tag{2}$$

デコンボリューションを行った後、式(3)に示される Rudin-Osher-Fatemi (ROF) モデルの TV 正則化 [4] を行い、骨格成分を抽出する。式(3)は Chambolle の射影法を用いて最小化される。

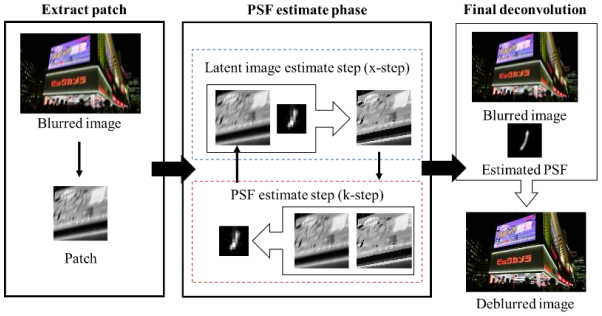

Fig. 1. Processing Flow of Blind Deconvolution

$$f_0 = \mathrm{argmin}_f\{\|f-g\|_2^2 + \lambda_r \mathrm{TV}(f)\} \quad (3)$$

TV 正則化を行った後、式(4)に示すショックフィルタ[5]によってエッジを強調する。

$$f_{t+1} = f_t - \mathrm{sign}(\Delta f_t)\|\nabla f_t\| \quad (4)$$

C. PSF 推定 (k-step)

PSF は勾配分布∇fから式(5)によって推定される。ノイズを減らすために、閾値処理された勾配分布が用いられる。式(5)は共役勾配法によって最小化される。

$$h = \mathrm{argmin}_h\{\|\nabla f * h - g\|_2^2 + \lambda_h\|h\|_1\} \quad (5)$$

D. 最終デコンボリューション

最終デコンボリューションでは、PSF 推定フェーズで最終的に推定された PSF を用いて画像復元を行う。理想画像推定処理と同じく D. Krishnan らの手法を用いて式(2)を最小化する。最終デコンボリューションは RGB 成分それぞれに対して施される。

IV. GPU 実装

処理の高速化について検討するため、最終デコンボリューションを GPU に実装した。開発環境として、CUDA を用いた。CUDA は NVIDIA 社が開発した並列計算プラットフォームであり、GPU を汎用計算に用いることを容易にする。CUDA は拡張された C 言語から Visual Studio 等の既存のコンパイラを用いて実行ファイルを生成する。ソースコードは識別子によって CPU 計算と GPU 計算部に分割されそれぞれコンパイルされる。一般に、大規模な並列計算は多くのプロセッサーを用いて行われるが、1 つの GPU は多くのプロセッサーを内蔵しているため、CPU で実装するより高速化が可能となる。

V. 実験結果

計算時間を比較するため、CPU と GPU の両方にデコンボリューション処理を実装し、実行時間を計測した。実験環境を TABLE I,II,III に示す。GPU には NVIDIA のハイエンド GPU である GeForce GTX Titan を用いた。Fig. 2 にブラインドデコンボリューションの結果を示す。また、TABLE IV に処理時間を示す。GPU での最終デコンボリューションの処理時間は、メインメモリと GPU のメモリ間の画像転送時間も含んでいる。TABLE IV より、Full HD 画像に対しては、GPU 実装が CPU 実装より 10 倍高速化できることを確認した。

VI. おわりに

本稿では、デコンボリューション処理を GPU に実装することによって処理の高速化に成功した。しかし、動画像アプリケーションに対しては十分ではない。本稿では最終デコンボリューションのみ GPU 実装したが、他の処理も GPU に実装することによって更なる高速化を行うことが期待できる。今後の課題として、PSF 推定等のブラインドデコンボリューションの GPU 実装および性能改善が挙げられる。

TABLE I. EQUIPMENT ENVIRONMENT

Software	Windows 7 64bit + CUDA 6.5 + Visual Studio 2013
GPU	NVIDIA GeForce GTX Titan
CPU	Intel Xeon E5-2360(2.6GHz)
Memory	DDR3-1600 32GB

TABLE II. GPU SPECIFICATIONS

Name	Ge Force GTX Titan
Core	GK110
Core Clock	876 MHz
Multi Processors	14
CUDA Core	2688
Memory	6144 MB
Memory Clock	3004 MHz
Memory Interface Width	384 bit
Memory Bandwidth	288.4 GB/Sec

TABLE III. EXPERIMENTAL PARAMETERS

Patch size			256×256
Kernel size			31×31
Kernel estimate iteration for each step			5
Latent image estimation	*Deconvolution*	*Iteration*	6
		λ_d	1500
	TV	*Iteration*	10
		Restraint λ_r	0.03
		Gradient step size τ	0.125
	Shock Filter	*Iteration*	1
		Strength dt	1.0
		Attenuation rate	0.9
PSF estimation	*Iteration*		30
	Threshold		0.075
Final Deconvolution	*Iteration*		6
	λ_f		1500

(a) Blurred image (b) CPU (c) GPU

Fig. 2. A part of Restored Image

TABLE IV. PROCESSING TIME

Process	**Processing Time [ms]**			
	482×482		**1920×1080**	
	CPU	**GPU**	**CPU**	**GPU**
Kernel estimation (CPU)	2760		3182	
Final deconvolution (CPU or GPU)	913	382	6657	627
Total	3673	3142	9839	3809

参考文献

[1] H. Senshiki, et al., “Blind Restoration of Blurred Images Using Local Patches,” *Proc. GCCE*, pp. 320-321, Oct. 2015.

[2] D. Krishnan and R. Fergus, “Fast Image Deconvolution using Hyper-laplacian Prior,” *Proc. ANIPS*, pp. 1033-1041, Dec. 2009.

[3] R. Liu and J. Jia, “Reducing Boundary Artifacts in Image Deconvolution,” *Proc. ICIP*, pp. 505-508, Oct. 2008.

[4] Osher and E. Fatemi, “Nonlinear Total Variation based Noise Removal Algorithms,” *Physica D*, Vol. 60, pp. 259-268, 1992.

[5] S.J. Osher and L.I. Rudin, “Feature-oriented Image Enhancement using Shock Filters,” *SIAM Journal on Numerical Analysis*, Vol. 27, pp. 910-940, 1990.