GPUを用いたビデオ映像のリアルタイム安定化

高橋賢治[†] 藤澤誠[††] 三浦憲二郎[†††]

この論文では，ビデオ映像に含まれる振動成分を取り除くための処理の計算をGPUを用いて行う手法を提案する．映像の安定化処理には，グローバルモーションの推定，振動補正，モザイキングの３つの処理を行うが，CPUでこれらの処理を行うと処理時間が長く，グローバルモーションの推定とモザイキングが処理の大半を占めている．そこで，並列処理が可能なGPUで計算処理を行うことで処理時間の短縮を図った．提案した手法は，ビデオ映像のフレーム画像をテクスチャデータとしてGPUに転送し計算を行い，計算結果をオフスクリーンバッファに描画し，ピクセルの値を読み込むことによって結果を得る．GPUからCPUにピクセルを読み込む速度，GPUとCPUの計算速度を考慮して，それらに対する計算負荷を最適にバランスさせることでリアルタイムでの処理を達成した．

Real-time Video Stabilization with GPU

Kenji Takahashi,[†] Makoto Fujisawa[††] and Kenjiro T. Miura[†††]

This paper proposes a fast computational method of video stabilization using the Graphics Processing Unit (GPU) that removes the unwanted vibrations from videos. The video stabilization is composed by estimation of the global motion, removal of the undesired motion and mosaicking. When these are processed with CPU, the computational cost for the global motion estimation is very high. We improve the speed of this computation with GPU that the parallel processing is possible. Our method can obtain the result by forwarding the frame image of the video to GPU as texture data, and drawing the calculation result to the offscreen buffer. We have achieved a real-time processing by optimizing the load balance bewteen CPU and GPU on the speeds of reading pixels and calculations of GPU and CPU.

1. 緒　　言

近年集積化技術の進歩により，ビデオカメラの小型化，低価格化が進み，一般に普及するようになり，様々な場面でビデオカメラが使用されるようになった．小型カメラは近年，災害時に素早く情報収集を行うために，人間が立ち入れない場所で被災者の探索を行うロボットや，上空から災害状況を確認する無人ヘリコプターなど，遠隔操作用のレスキューロボットにも搭載されている．しかし，ロボット自身の振動や，荒れた路面上や地震で障害物が散在する状況下で走行することから，ロボットに搭載されたカメラから送られてくる映像にはゆれが生じてしまう．そのため，即座の状況判断が困難になることや，オペレータが画面酔いして操作に影響が出る可能性が考えられる．したがって，映像のゆれによる影響を抑えるためには，リアルタイムでの動画像処理を行いゆれを軽減させる必要がある．

現在，デジタルカメラのために開発，研究されているゆれを軽減する手法は，電子式，光学式，イメージセンサーシフト式，レンズユニットスイング式手ぶれ補正などがあげられる．しかし，これらはカメラに搭載される補正機能であり，そのカメラで撮影した映像だけしか補正できず，それらは必然的にカメラの大型化，高価格化を招いてしまう．近年では，デジタルカメラの普及やPCの発達により，一般の家庭用PCでも動画像の処理などが簡単に行えるようになっており，汎用性を高めるためにPCを利用した安定化処理が望まれる．しかしながら，動画像はデータ量が多く，それらを処理するにはCPUでは負荷が大きいためリアルタイムでの処理は難しい．そこで，本研究では並列処理による高速演算が可能なGPU(Graphics Processing Unit)を用いたリアルタイムでの映像の安定化手法を提案する．

† 静岡大学大学院工学研究科
Graduate School of Engineering, Shizuoka University
†† 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology University
††† 静岡大学創造科学技術大学院
Graduate School of Science and Technology, Shizuoka University

コンピュータビジョン分野においても，GPU を使い画像処理を高速化するための研究が盛んに行われている．GPU でグラフィック計算以外のより一般的な処理を行う際には，CPU とのデータの受け渡し方法が問題となる．一般的には CPU から GPU へはテクスチャを，その逆にはオフスクリーンバッファを用いる．より詳しいことは文献 1) を参考にしてほしい．文献 1) は画像処理を GPU で行うためのフレームワークを示した．テクスチャやオフスクリーンバッファは平面画像データであり，画像処理で扱う静画像と親和性が高い．また，動画像も各フレームごとでは静画像として扱えるため，動画像処理を GPU で高速化する研究も数多くなされている．Strzodka と Garbe[2] は動画像に対するローカルモーション推定とその視覚化を GPU によってリアルタイムに実行した．Sinha ら[3] は動画像の特徴追跡を GPU で処理することで KLT 特徴追跡アルゴリズム[4] を CPU 処理に比べて 15 倍以上高速化し，リアルタイムでの実行を可能とした．この特徴追跡を用いてグローバルモーション推定を行うことも可能である[5] が，映像中の動物体の動きに全体の動きが作用され，安定した推定が難しいと考え，我々は Litvin らの安定化手法[6] を GPU で高速化する．動画像の隣接したフレーム間の動きをアフィン変換と仮定し，その差分値の合計をエラー値として最小化手法で最適解を求める．このとき，CPU 処理における計算時間のボトルネックとなっているエラー値計算部分を GPU で並列に計算するために提案されたベクトル要素の合計値計算手法[7] などを用いて高速に計算する．

2. 関 連 研 究

映像の安定化には，カメラの動き (グローバルモーション) を推定し，それを基に振動補正を行う．Litvin らの手法[6] は，グローバルモーションにカルマンフィルタを適用することによってゆれを抑えた動きを求め，フレーム画像を変形させる．さらに，変形による画像の劣化をモザイキングを行うことで補間する．しかし，モザイキングでは映像の中の物体の動き (ローカルモーション) を補間しきれない．Matsushita らは，ローカルモーションを用いた Motion inpainting によってこの問題を解決し，より良画質な映像を生成する手法を提案した[8]．また，彼らは，階層的な運動推定[9] を行うことで，グローバルモーションの推定時間の減少に成功しており，ガウスカーネルを用いることによって振動補正を行っている．

3. 映像の安定化

3.1 グローバルモーションの推定

映像の安定化を行うにはグローバルモーション (カメラの動き) を知る必要がある．グローバルモーションは隣接するフレーム間の動きを求めることによって推定される．フレーム I^n から I^{n+1} までのピクセル座標 $\mathbf{x} = (x, y)$ の変化は，

$$\mathbf{x}_{n+1} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \mathbf{A}_n^{n+1}\mathbf{x}_n + \mathbf{b}_n^{n+1} \tag{1}$$

で表すことができる．このアフィン変換 $(\mathbf{A}_n^{n+1}, \mathbf{b}_n^{n+1})$ は

$$E(n, n+1) = \sum_{\mathbf{x}\in\chi} (I^n(\mathbf{x}_n) - I^{n+1}(\mathbf{A}_n^{n+1}\mathbf{x}_n + \mathbf{b}_n^{n+1}))^2 \tag{2}$$

の最小値を求めることによって得ることができる．ここで，χ は画面平面上全ての座標値を表す．ここでは，高速化のために平行移動と回転の 3 変数に限定し，最小値探索には勾配値 (導関数) を用いる BGFS 法[10] を用いる．

3.2 振 動 補 正

推定したグローバルモーションをもとに，Matsushita らの方法[8] を用いて振動を補正する．補正するフレームの前後 k フレームを利用して補正変換 S_n を

$$S_n = \sum_{m=n-k}^{n+k} T_n^m \star G(k) \tag{3}$$

によって求めることができる．ここで，T_n^m はフレーム n から m までのアフィン変換，G はガウスカーネル，そして $\star$ は畳み込み演算子である．得られたアフィン行列を用いて振動補正を行う．

$$\bar{\mathbf{x}}_n = S_n\mathbf{x}_n = \bar{\mathbf{A}}_n\mathbf{x}_n + \bar{\mathbf{b}}_n \tag{4}$$

3.3 モザイキング

振動補正したフレームには未定義領域が発生するため，Litvin らのモザイキングを用いた手法[6] で補間する．周囲のフレーム $\bar{I}^{n+m}$ を補間の対象となるフレーム $\bar{I}^n$ の位置に変形 $(\bar{I}^{n+m} \rightarrow \bar{\bar{I}}^{n+m})$ させ，式 (5) を用いて補正をかけることによって未定義領域のピクセルを補間する．モザイキングの結果を図 1 に示す．

$$\bar{I}^n = \frac{1}{\sum(n,m)} \sum_{-M\le m\le M, m\neq 0} E(n,m)\bar{\bar{I}}^{n+m} \tag{5}$$

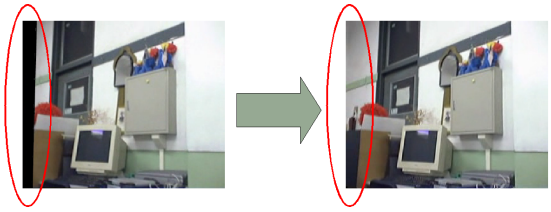


図 1 未定義領域を補間

4. GPU での実装

前節で提示した安定化処理を CPU 上で行うと非常に時間がかかる．グローバルモーションの推定とモザイキングの処理時間が大半を占め，二つを合わせると 320×240[pixel] の映像において約 3[sec/frame] である (論文 11)．これは，グローバルモーション推定では式 (2) の計算において輝度値の差を 1 ピクセルずつ計算する処理を，最小値探索のため何度も計算を繰り返していること，およびモザイキングにおいて，多くの画像データを扱う必要があるためである．そこで，本研究では論文 11) のグローバルモーションに改良を加え，モザイキングについても GPU のピクセルの並列処理能力を用いて計算速度を向上させる．

4.1 グローバルモーション推定

画像データは CPU から GPU にテクスチャデータとして送る．GPU は本来描画専用であるため，ここではディスプレイに表示させないオフスクリーンバッファである FBO(Frame Buffer Object) を用いる．オフスクリーンバッファには，他に天谷ら[11] の用いた PBuffer(Pixel Buffer) があるが，FBO は GPU を 2 つ用いて高速化する技術である NVIDIA SLI が利用可能，OS に依存しないなどの利点がある．

画像の差分はテクスチャ画像をアフィン変換して行う．ただし，場面が急激に変わるフレーム間や画面内を動的な物体が多く占める場合 (ローカルモーションが大きい場合) は，対応点がとれずにグローバルモーション推定に失敗する可能性があることに注意しなければならない．また，ピクセル座標値のアフィン変換 $\boldsymbol{A}_n^{n+1}\boldsymbol{x}_n + \boldsymbol{b}_n^{n+1}$ によって，輝度値が定義されていない領域 (未定義域) を参照する場合，そのピクセルはエラー値の計算から除外する．エラー値 E は最終的に有効であったピクセル数 χ_e で補正する $\hat{E} = (\chi/\chi_e)E$. ただし，$\alpha = \chi_e/\chi$ が小さい場合 (例えば 1/5) に正しい結果が出ない可能性がある (実際のカメラの動きが小さくても最小化手法の反復の初期において $\boldsymbol{A}$ や $\boldsymbol{b}$ の値が大きくなることがある)．本研究では $\alpha < 1/5$ 以下では未定義域のピクセルの輝度値を 0 としてエラー値を計算し，エラー値が意図的に大きくなるように実装した．また，最終的なエラー値がある値より大きい場合は，グローバルモーション推定の失敗，または，シーンチェンジとみなして振動補正を行わないようにしている．もちろん，実際のカメラの動きがあまりに高速であった場合，これは不正確な結果を生むことに注意する必要がある．

画像幅を W_f, 画像高さを H_f として，求めた差分画像に対し，

$$W_f \leq 2^n, H_f \leq 2^n \quad (6)$$

(n は最小の整数)

となる $2^n \times 2^n$ を描画範囲とした FBO を確保し，GPU の並列計算の特性を生かし，図 2 のようにして描画範囲を画像の幅と高さの半分にした FBO に貼り付けて画像サイズを 1/4 にする．このとき描画する画像のピクセル座標 (x, y) の値は，元の画像のピクセル座標

$$\begin{matrix}(2x, 2y) & (2x+1, 2y) \\ (2x, 2y+1) & (2x+1, 2y+1)\end{matrix} \quad (7)$$

の値を合計したものを格納する．

これを繰り返し，画像サイズを小さくしていく．計算がある程度進んだら FBO のピクセルのカラー値に格納して CPU にリードバックし，1 ピクセルになるまで CPU で合計することで計算結果が得られる．図 3 に，$2^n \times 2^n$(n は 0 から 7 までの整数) におけるリードバックのサイズによる計算時間の変化を示した．ここで使用した映像は最小値探索に BGFS 法を使っており，フレームレートが 30[fps], 10[sec] の 320×240[pixel] である．なお，実行環境は CPU が Intel® Core™2 Duo E6850，GPU が NVIDIA® GeForce® 8800 GTX，メモリが 4GB である．

これをみると，合計値計算において，GPU 上で最後まで計算を行い 1×1 にするよりも，GPU 上で $2^4 \times 2^4 = 16 \times 16$ まで計算を行い，リードバックの後，残りを CPU 上で処理すると最速になることがわかる．

これは，本研究で用いた GPU(NVIDIA GeForce8800 GTX) では最大 128 個の並列計算が可能であるが，n が 4 のとき $2^4 \times 2^4 \times 4 = 1024$，n が 3 のとき $2^3 \times 2^3 \times 4 = 256$，n が 2 のとき $2^2 \times 2^2 \times 4 = 64$ (GPU 上では RGBA の 4 色の計算を行っている) と，並列計算できる数に比べて計算点数が少なると，効率よく並列計算ができなくなるため，計算時間が増加すると考えられる．

4.2 モザイキング

モザイキングは図 4 に示すように，計算処理と画像生成の 2 つの処理から成り，このうち画像を生成する

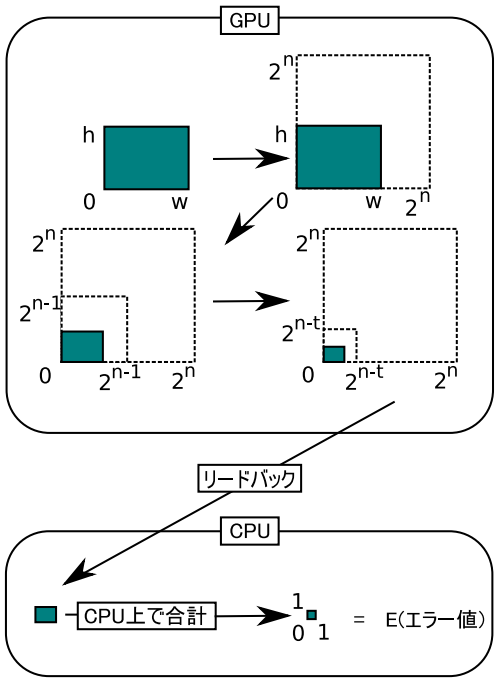


図 2 計算結果の合計

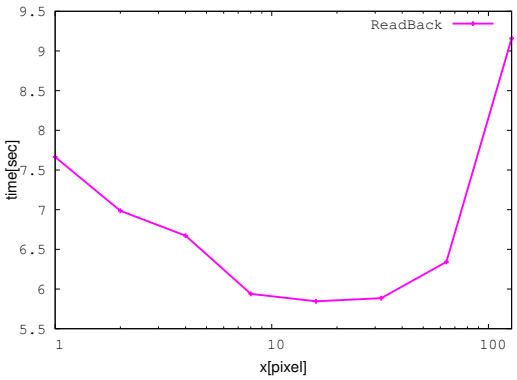


図 3 合計値計算における，リードバックのサイズによる計算時間の変化

処理を GPU に実装する．画像生成では多くの画像を扱うが，同じ画像を繰り返し使用するため，新規の画像のみを CPU から GPU へ転送し，GPU 上のメモリに保持しておくことで転送を最小限にして速度の向上を図る．GPU 上では，CPU の計算結果に従って変形したポリゴンを FBO に順次貼り付けることで，モザイキングを行った画像が得られる．生成された動画は，このまま GPU から直接ディスプレイに表示することも可能であるが，GPU から CPU へリードバックすることで，ハードディスクへの録画などの用途も可能にしている．

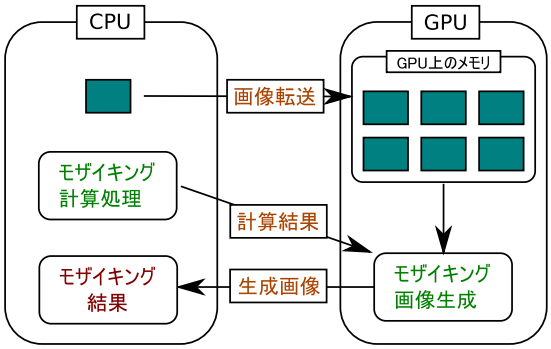


図 4 モザイキング

5. 結　　果

サイズ別のグローバルモーションの推定時間の，CPU での計算と，GPU で，オフスクリーンバッファに PBuffer[11] と FBO を使った場合の比較を図 5，同じ図の GPU の処理速度の詳細がわかるように，縦軸を拡大したものを図 6 に示す．

すべて最小値探索に BGFS 法を使っており，フレームレートが 30[fps]，10[sec] の正方形の映像である．GPU の並列計算を用いることで，CPU に比べて最大約 165 倍の速度向上を達成した．

図 5 に示したように，CPU のみを用いた場合，画像幅と計算時間がほぼ二次曲線になっている．この映像は正方形であり，画像の総ピクセル数と，計算時間が比例しているといえる．一方，GPU を用いた場合 (図 6)，画像幅が 100 から 200，200 から 300，500 から 600，1000 から 1080 の間で他と比べて処理時間の変化が大きい．これは，合計値計算の際の最初の FBO 確保量が，式 (6) にしたがい n の値が変化し，128 から 256，256 から 512，512 から 1024，1024 から 2048 に増え，それにより，合計値計算処理が 1 処理ずつ増えるためである．

次に，QVGA(320×240[pixel]), フレームレートが 30[fps], 10[sec] の映像を使用した場合の CPU と GPU のモザイキングの時間の比較結果を表 1 に，安定化結果を図 7 に示す．また，実行結果を本論文に付随するムービーファイルに示した．なお，実行環境は CPU が Intel® Core™2 Duo E6850，GPU が NVIDIA® GeForce® 8800 GTX，メモリが 4GB である．

表 1:比較結果

	CPU	GPU(FBO)
グローバルモーション推定	1.61	0.021
モザイキング	0.018	0.0020

[sec/frame]

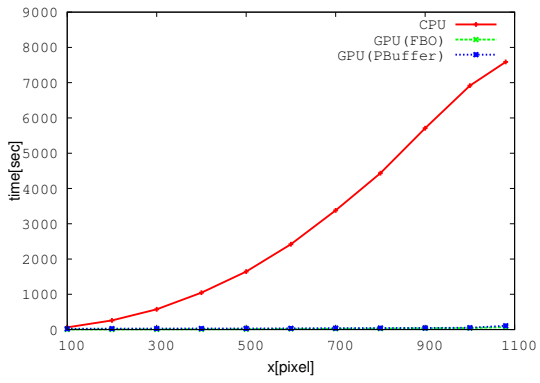


図 5 サイズ別のグローバルモーション推定時間

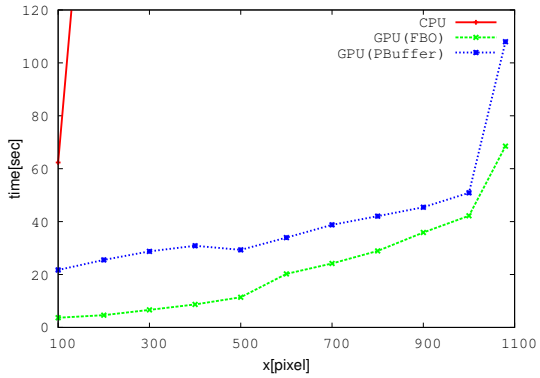


図 6 サイズ別のグローバルモーション推定時間 (拡大)

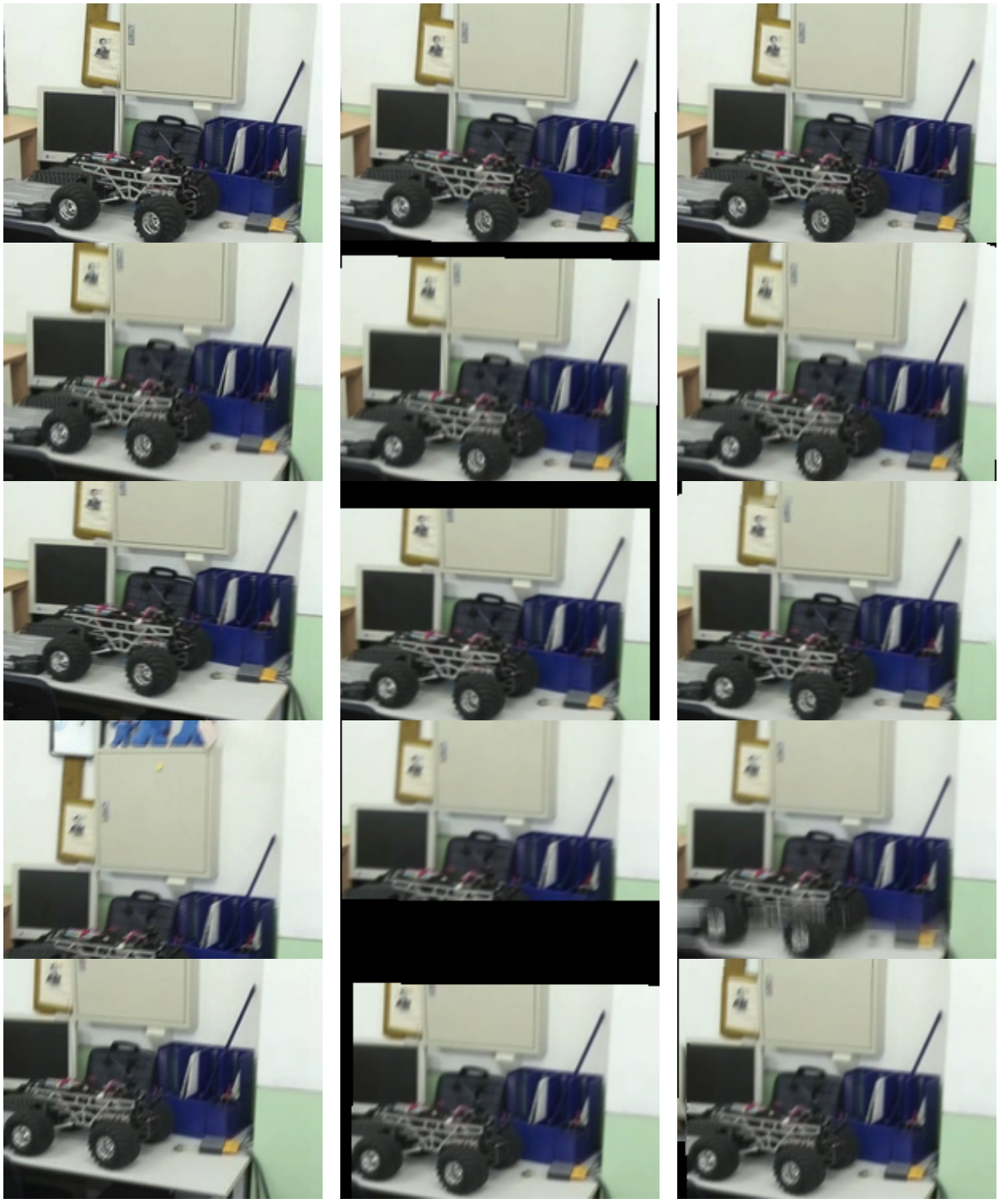
6. 結　　言

320×240[pixel] の画像において，GPU で計算処理を行ったグローバルモーション推定時間は平均 46.6[fps]，モザイキングなどを含めた全行程では 平均 40.3[fps] となった．一般的に PC の動画は 30[fps] 程度であるため，グローバルモーションとモザイキングに GPU を用いることでリアルタイムを達成した．GPU を数枚使用して高速化させる技術である SLI を使用することや，CPU をはるかに上回る速度で性能向上している GPU により，さらなる計算速度の向上も可能であると思われる．

今後の課題として，グローバルモーション推定においては，カルマンフィルタ[6] などによる初期値推測や，合計値計算方法の改善によるさらなる計算速度の高速化を行い，より大きな画像でもリアルタイムを実現させることなどが挙げられる．また，振動補正では，カルマンフィルタ[6] などを用いて過去の情報のみで振動補正ができれば，フレームの遅れなく振動補正ができ，十分実用的なものになると考えられる．モザイキングについては，本研究では高速化のために Litvin らの手法[6] を用いているが，ローカルモーション推定[8] の処理を GPU に組み込み，リアルタイム処理を保ったまま，動画の中に動物体が存在しても自然な補間を行えるようにすることが挙げられる．

参　考　文　献

1) Jargstorff,F.: *GPU Gems:Programming Techniques, Tips and Tricks for Real-Time Graphics*, chapter Chapter 27: A Framework for Image Processing, pp. 445–467, Addison-Wesley Pub. (2004).
2) Strzodka, R. and Garbe, C.: Real-Time Motion Estimation and Visualization on Graphics Cards, in *Proceedings IEEE Visualization 2004*, pp. 545–552 (2004).
3) Sinha, S. N., Frahm, J.-M., Pollefeys, M. and Genc, Y.: GPU-Based Video Feature Tracking and Matching, Technical report, Technical Report 06-012, Department of Computer Science, UNC Chapel Hill (2006).
4) Tomasi, C. and Kanade, T.: Detection and Tracking of Point Features, Technical report, Carnegie Mellon University Technical Report CMU-CS-91-132 (1991).
5) Ready, J. M. and Taylor, C. N.: GPU Acceleration of Real-time Feature Based Algorithms, in *IEEE Workshop on Motion and Video Computing (WMVC'07)*, pp. 8–9 (2007).
6) Litvin, A., Konrad, J. and Karl, W. C.: Probabilistic video stabilization using Kalman filtering and mosaicking, in *IS&T/SPIE Symposium on Electronic Imaging, Image and Video Communications*, pp. 663–674 (2003).
7) Krüger, J. and Westermann, R.: Linear algebra operators for GPU implementation of numerical algorithms, *ACM Transactions on Graphics (ACM SIGGRAPH 2003)*, Vol.22, No.3, pp. 908–916 (2003).
8) Matsushita, Y., Ofek, E., Ge, W., Tang, X. and Shum, H. Y.: Full-Frame Video Stabilization with Motion Inpainting, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.28, No.7, pp. 1150–1163 (2006).
9) Bergen, J. R., Anandan, P., Hanna, K. J. and Hingorani, R.: Hierarchical Model-Based Motion Estimation, in *ECCV'92 : Proceeding of the Second European Conference on ComputerVision*, pp. 237–252 (1992).
10) Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P.: *Numerical Recipes in C++*, Cambridge University Press (2002).
11) 藤澤誠, 天谷貴大, 三浦憲二郎 : GPU を用いたビデオ映像の安定化, 情報処理学会論文誌, Vol.49, No.2, pp. 1022–1030 (2008).



(a) 補正前　(b) 補正後　(c) モザイキング

図 7　安定化結果