Vol.2017-HPC-160 No.19
2017/7/27

高解像度画像対応ステレオマッチングの GPGPU による高速化

虞飛[†1]　佐藤裕幸[†1]

概要：本稿では，Coarse-to-Fine の手法を利用したステレオマッチング手法をベースに，GPGPU を利用するための CUDA 言語による高速化の手法を提案する．そのベース手法では，ソーベルフィルタにより特徴点を抽出し，ドロネー三角形分割で画像を分割し，エピポーラ幾何と最大事後確率で視差を段階的に計算する．このように，三つの段階に分けられているので，高解像度画像に対応し易く，精度も処理速度も良い．しかし，視差の計算はシングルスレッドの CPU コードしかないため，他の GPGPU を利用した手法よりも低速であるが，処理速度はまだ向上する余地があり，他手法に比べて高速になる可能性が高い．従って，本稿では，高並列な CUDA 処理と適切なデータ構造を用いて，視差計算部分の処理速度を向上させ，高解像度画像対応の並列化処理の手法を提案し，その有効性を示す．また，高速化を実現した上で，更に精度向上の可能性を検討する．以上のような個別の並列化と最適化の提案手法に基づくステレオマッチングを CUDA8.0 で実装した．評価実験により，並列化した部分それぞれで約 2~10 倍，全体で 2 倍以上の速度向上を確認することができた．

キーワード：ステレオマッチング，高速化，CUDA，Coarse-to-Fine

Fast efficient large-scale stereo matching using GPGPU

FEI YU[†1]　HIROYUKI SATO[†1]

Abstract: To accelerate a Coarse-to-Fine based stereo matching algorithm, in this study we propose a CUDA based parallel algorithm. In the base algorithm, feature points are picked out by the Sobel Filter, a picture is divided by the Delaunay Triangulation and parallaxes are calculated step by step by Epipolar geometry and Maximum a posteriori. Thus processing is three stages, so it's easy to correspond to a high-resolution picture and the precision and a processing speed are good. But the calculation of a parallax is more low-speed than the method for which other GPGPU was used because there is only a CPU cord of a single thread. Therefore there is a possibility which becomes high-speed compared with other methods. In the proposed algorithm, parallel computing in each stage of the base stereo matching algorithm is performed by using the compute unified device architecture to reduce the running time. Experimental results show that the proposed algorithm is at least 2 times faster than the base algorithm, and has good performance on high resolution pictures.

Keywords: Stereo matching, Acceleration, CUDA, Coarse-to-Fine

1. はじめに

ステレオマッチングとは，2 次元の画像から 3 次元の情報を得る手法である. 視点の異なる 2 台のカメラを用いて，三角測量の原理で距離を求める．実空間中の 3 次元点が撮影され，それぞれ投影した左図と右図の投影点の視差から距離を計測する手法であるため，画素一つ一つに対応する点を見つけ出す必要がある．これらの処理には，処理速度の速い Local method と，精度が高い Global method がある[1]．近年，GPGPU を利用する高速化ステレオマッチング手法が多くなってきており，視差改善処理[2]による高精度で処理速度も速いステレオ手法[3]も出てきた．しかし，これらの手法は，GPGPU を利用するため GPU の性能に高く依存する．また，画像の高解像度化に対し処理速度の低下について coarse-to-fine[4]を利用した手法も多い[5].

本稿では，coarse-to-fine の手法を利用したステレオマッチング手法[6]をベースに，GPGPU を利用するための CUDA 言語による高速化を実現する手法を提案する．そのベース手法では，視差値の信頼度が高い点を特徴点として抽出し，ドロネー三角形分割で画像を分割し，エピポーラ幾何と最大事後確率で視差を段階的に計算する．三つの段階で高解像度画像に対応し易く，精度も処理速度も良い．しかし, 視差の計算はシングルスレッドの CPU コードしかないため，他の GPGPU を利用した手法よりも低速であるが，処理速度はまだ向上する余地があり，他手法に比べて

†1 岩手県立大学大学院　ソフトウェア情報学研究科
Iwate Prefectural University　Graduate School of Software and Information Science

高速になる可能性が高い.

このような背景から，本稿では，高並列な CUDA 処理と適切なデータ構造を用いて，視差計算部分の処理速度を向上させ，高解像度画像対応の並列化処理の手法を提案し，その有効性を示す．また，高速化を実現する上，更に精度向上の可能性を検討する．

今回ベースとした手法は ELAS 法を呼ばれ，視差計算処理を三つの処理段階で行うので精度が高いが，GPU で並列化させるにはいくつかの問題点がある．本研究では，視差計算の基礎部分を CUDA 化する上で，三つの段階でそれぞれ異なる並列化を行った．

まず第一段階は，特徴点の抽出と有効な視差を得る処理である．この特徴点の抽出は，ソーベルフィルタで行うので画素単位で並列化し，その結果を画素に付加して特徴点とする．その特徴点毎に並列に視差計算を行う．なお，ここでは，他の特徴点抽出手法も利用可能であり，精度向上するため視差計算手法の改善余地も大きい．計算した視差値は，左右２つの図の特徴点の差の最小値と第二最小値を検索し，その比率が一定以上であれば後段の画像分割で用いる点として有効とする．

次に，ドロネー三角形分割で画像を分割し，画像を一定サイズの格子（優先点）の集合に分け，各格子に対して視差計算を行う．ドロネー三角形分割は CUDA 化可能だが，CPU 実行でも高速なので，最適化時に CUDA 化するかどうかを検討する．優先点の確定と格子集合の作成も同様である．

最後は全画素に対する視差計算である．前回計算した優先点の視差を利用し，分割した三角形の辺に対し最大事後確率で視差計算する処理を CUDA で並列化する．この部分にある視差の計算は，エピポーラ幾何法であり，エピポーラマトリックスの計算を CUDA で並列化する．

そして，各段階には，処理はそれぞれ違いがある．特徴点を用いた視差計算部分では，有効性の判定において，最小値と第二最小値の検索と比較があるため，計算した視差値データの集合は Structure of Array のような並列化に適したデータ構造が必要である．全画素の視差計算の部分では，画素が属する三角辺の確定と頂点の整列を行い，また最大事後確率に関する計算は，本研究で並列化の規模と構成を調整し，最適化を行う．

以上のような個別の並列化と最適化を提案手法に基づくステレオマッチングを CUDA 8.0 で実装した．評価実験により，並列化した部分それぞれで約 2~10 倍，全体で 2 倍以上の向上を確認することができた．

2 章でステレオマッチング手法における関連研究及びベース手法について説明したのち，3 章でそのベース手法を CUDA により高速化する提案手法について紹介する．4 章では提案手法の評価実験結果を説明して考察を行い，5 章で本研究の成果を纏める．

2. 関連研究

本章ではステレオマッチングにおける研究を説明し，ベースにする手法の選択及び手法の流れを説明する．

(1) ステレオマッチングの研究

Scharstein et al.[1]はステレオマッチングの研究を良く纏めて，評価体系も提案した．Scharstein らが作った評価用プログラムと評価用ステレオデータセットにより，評価対象のステレオマッチング手法の精度と汎用性が良く分かる．そして，近年，高解像度画像対応のニーズとリアルタイム化向け高速化研究のため，ステレオマッチング手法の処理速度も評価指標の一つに追加された．

(2) ステレオマッチング手法の比較

それぞれのステレオマッチング手法は，精度，汎用性と処理速度により，各自の適した分野ごとに使われている．新たな手法を含めて，手法間の比較が必要であるため，処理速度がトップクラスの手法を表 1 に示す．また，そして精度がトップクラスの手法を表 2 に示す．

表 1 と表 2 により，高精度ステレオマッチング手法の処理速度は高速手法より百倍以上低速であるが，高速手法には精度が良い手法がある．一番速い IDR 法[3]と処理速度と精度が共に良い MCSC 法以外では，ELAS 法は処理速度が 2 位で精度も比較的良いことが分かる．

そして，IDR 法と MCSC 法は GPU を利用しているが，ELAS 法はシングルコア実装のみであることから，ELAS 法は GPU 利用により改善余地があると思われる．

表 1　処理速度トップクラスのステレオマッチング手法

	処理時間/解像度 (Time/MP)	解像度 (5-6MP)	エラー率 (Average absolute error in pixels)	実行環境
IDR	0.36	Half	6.35	TITAN Black
ELAS	0.49	Half	9.52	1 core, i7@3.6 GHz
LS-ELAS	0.50	Full	15.2	1 core, i7@3.4 GHz
SED	0.50	Full	34.5	1 core, i7@2.1 GHz
MCSC	0.52	Full	3.73	GTX 1080
ELAS	0.56	Full	14.1	1 core, i7@3.6 GHz
SGBM1	0.68	Quarter	14.3	1 core, i7@3.3 GHz
MC-CNN-fst	1.26	Half	4.37	GTX TITAN X

表 2　精度トップクラスのステレオマッチング手法

	処理時間/解像度 (Time/MP)	解像度 (5-6MP)	エラー率 (Average absolute error in pixels)	実行環境
3DMST	130	Half	2.17	GTX TITAN X
MC-CNN+TDSR	124	Full	2.63	GTX TITAN X

(3) ELAS 法

Andreas Geiger らの ELAS 法[6]は，ステレオマッチングにおいて信頼度の高い点（サポート点）を抽出し，周辺の視差範囲を確定し，エピポーラ幾何と最大事後確率で周辺画素の視差を計算する手法である．

信頼度の高いサポート点と確率計算の利用で，他の手法より高精度と高速の視差計算ができる．ELAS 法は図 1 に示す通り三つの処理段階に分けられる．

第一段階はサポート点の抽出である．全画素の視差計算の基となるため，サポート点の視差の精度は最終結果に深い影響がある．関連する手法として，Šára [7]は，ステレオマッチングにおける視差の信頼度の高い部分の抽出法を述べた．ELAS 法では，その手法を利用し，ソーベルフィルタによりサポート点の抽出をする．

第二段階はサポート点周辺領域を利用した視差範囲の確定である．ELAS 法はドロネー三角形分割の手法で，サポート点により画像を分割し，サポート点を頂点とする三角錐で，周辺領域の視差範囲を確定する．

第三段階はサポート点の視差と確定した周辺範囲で，エピポーララインの制約と最大事後確率（Maximum a posteriori estimation 略称 MAP）の手法で範囲内の画素の視差を計算する．

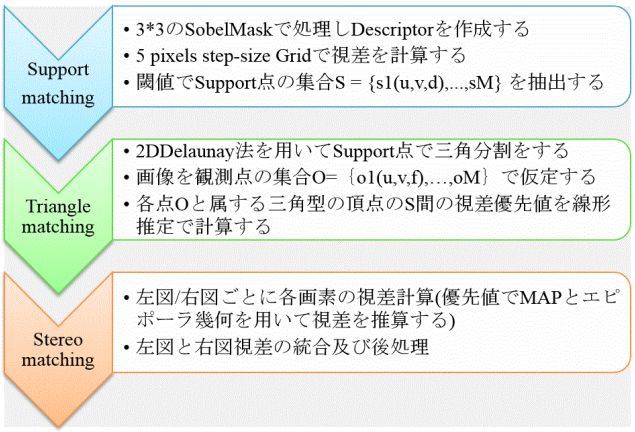

図 1　ELAS 法の手順

3. GPGPU による ELAS 法の高速化

本章では，ELAS 法の高速化手法について説明する．本研究では，図 2 に示す ELAS 法の各計算部の処理時間を分析し，それぞれの高速化のために，CUDA による並列化手法を提案する．図 3 では，本研究における GPU による ELAS 法の高速化の処理手順を示す．

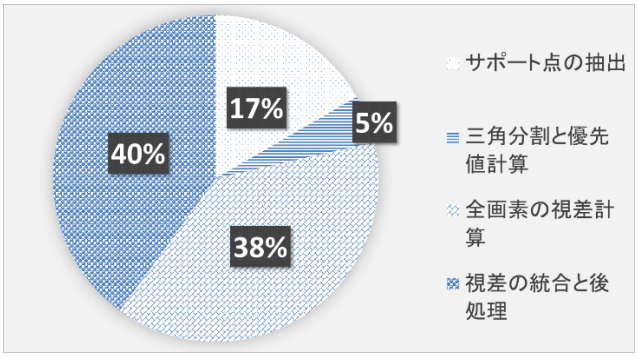

図 2　ELAS 法の処理時間比率

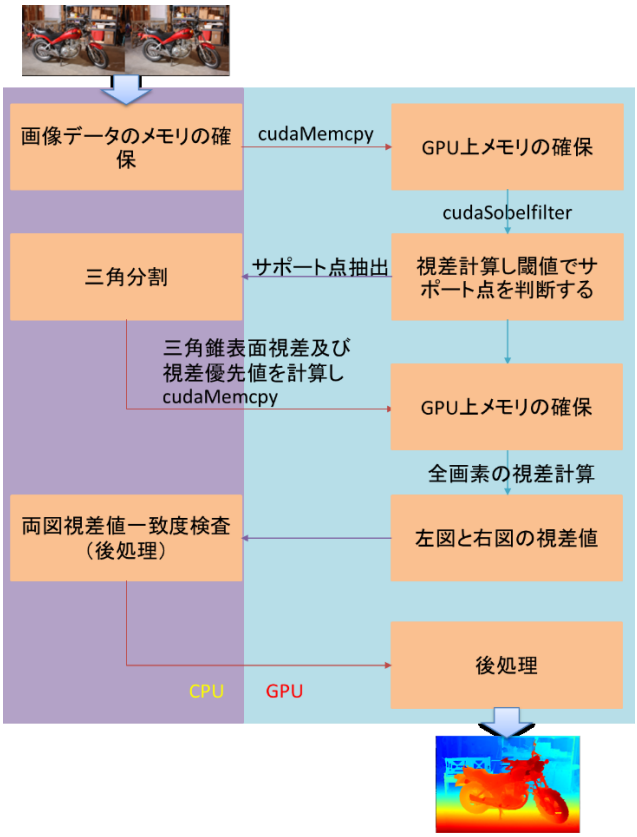

図 3　GPU による ELAS 法の手順

3.1 CUDA によるサポート点の抽出

ELAS 法においてサポート点の抽出は精度を左右する大きな要因となる．その手順は，全画素をソーベルフィルタで処理した後，視差を計算し閾値によりサポート点を抽出する．ここでのソーベルフィルタや着目画素周辺の視差計

算は，非常に高い並列性があるので， GPU 利用に適している．本研究では，以下の CUDA サンプルを参考として用いた．

(1) **SobelFilter[8]**

NVIDIA 社によるエッジ検出の CUDA サンプルコードであり，SobelMask の計算に，Texture メモリと Shared メモリの利用例が提供されている．本研究では，高速メモリを用いて画像データに関する処理を大幅に速度向上させている．

(2) **Stereo Disparity[9]**

NVIDIA 社によるステレオマッチングの CUDA サンプルコードであり，640x533 のステレオ画像の視差計算を行っている．本研究では，各画素に対応するスレッドでデータを連続アクセスし，ステップ数 5 で候補サポート点の視差計算を行う．そして，閾値で候補点を判断し，サポート点を抽出する

3.2 三角分割

この部分では，上記の処理により抽出したサポート点の座標と視差値により，ドロネー法で画像を数千以上の三角形区域に分割し，三角錐各平面の視差及び優先値を計算する．条件分岐が多いため，GPU 利用には不利であり，CPU でも処理速度が速いため，CPU で処理する．

3.3 CUDA による全画素の視差計算

ELAS 法では，各画素の視差計算が画素周辺のサポート点による事前に計算した視差の優先値でマッチング計算するため，画素を処理する時に必ず三角頂点とした周辺のサポート点を確定しておく必要がある．確定してから，周辺頂点の優先値により視差を計算する．

同一の三角形内画素の周辺頂点は同一なので，図 4 のように隣接する三角形ごとに逐次に処理を行う．このように隣接三角形を逐次に処理するため，周辺のサポート点の情報は容易に得ることができる．

本研究では，並列化処理のために，三角形を並列化の単位として，各三角形にスレッドを割り付ける並列化処理を提案する．三角形の処理が逐次に行われないため，三角形の周辺のサポート点の情報を事前に収集する必要がある．

図 5 で示すように，視差計算の findMatch()の CUDA カーネルへの処理のため，三角形内の画素を Loop 処理により抽出する．

図 6 で示すように，前記三角分割部で得た計算結果（plane_a,b,c）及び抽出した画素座標（temp_u,v）を配列で保存する．

図 7 で示すように，上記で保存したデータを GPU へ転送するため，GPU 上のメモリを確保する．これらの情報と 3.1 節で既に GPU メモリ上に確保した画素データとを一緒に視差計算の CUDA カーネルで処理し視差を算出する．

```
for (i=0; i<tri.size(); i++) {
      // get plane parameters
      p_i = i*3;
      plane_a = tri[i].t1a;
      plane_b = tri[i].t1b;
      plane_c = tri[i].t1c
      // triangle corners
      c1 = tri[i].c1;
      c2 = tri[i].c2;
      c3 = tri[i].c3;
      // c1,c2,c3 をソートして A,B,C とする
      … …
      //  直線 A,B の傾き AB_a, 高さ AB_b を求める
      //  直線 A,C の傾き AC_a, 高さ AC_b を求める
      … …
      // first part (triangle corner A->B)
      if (A_u!=B_u) {
         for (u=max(A_u,0); u<min(B_u,width); u++){
         v_1 = AC_a*u+AC_b;
         v_2 = AB_a*u+AB_b;
         for (v=min(v_1,v_2); v<max(v_1,v_2); v++)
            { findMatch(... …);}
      }
      // second part (triangle corner B->C)
      … …
}
```

図 4　三角形内 Loop 展開の疑似 C コード

```
//  画素座標抽出用配列
vector<int32_t> temp_u = vector<int32_t>();
vector<int32_t> temp_v = vector<int32_t>();
// first part (triangle corner A->B)
if (A_u != B_u) {
   for (u = max(A_u, 0); u<min(B_u, width); u++){
      v_1 = (AC_a*u + AC_b);
      v_2 = (AB_a*u + AB_b);
      // Loop 処理で三角形内画素座標抽出
      for (v = min(v_1, v_2); v<max(v_1, v_2); v++)
          {temp_u.push_back(u);
          temp_v.push_back(v);}
   }
}
// second part (triangle corner B->C)
… …
```

図 5　三角形内画素抽出の疑似コード

```
for (j = 0; j<temp_u.size(); j++) {
    planes_a[size_total] = plane_a;
    planes_b[size_total] = plane_b;
    planes_c[size_total] = plane_c;
    //画素座標抽出
    pixs_u[size_total] = temp_u.at(j);
    pixs_v[size_total] = temp_v.at(j);
    size_total++;
}
```

図 6　同三角形関連情報抽出の疑似コード

```
// 画素座標メモリ確保
int32_t* d_u, *d_v;
cudaMalloc((void**)&d_u,size_total*sizeof(int32_t));
cudaMalloc((void**)&d_v,size_total*sizeof(int32_t));
// 三角形関連情報メモリ確保
float* d_planes_a, *d_planes_b, *d_planes_c;
cudaMalloc((void**)&d_planes_a, size_total*sizeof(float));
cudaMalloc((void**)&d_planes_b, size_total*sizeof(float));
cudaMalloc((void**)&d_planes_c, size_total*sizeof(float));
… … … … … … …
findMatch_GPU << <DimGrid, DimBlock >> >(… …)
cudaDeviceSynchronize();
```

図 7　GPU 上メモリの確保

3.4 CUDA による視差の統合と後処理

3.3 節で述べたような視差計算の結果は，ステレオ画像の左図と右図それぞれの視差値であるため，両方の図の視差値の統合と視差図生成ための後処理が必要である．

統合した視差図にはエッジを多く含むので，平滑化を行うためバイラテラルフィルタが使用されている．研究[2]により，後処理の改善により視差の精度向上が有効で，研究[12][13][14]によりバイラテラルフィルタの高速化例も多い．

本研究は，ベース手法に処理時間 2 割占めたバイラテラルフィルタ処理部を CUDA による高速化を実施した．Shared メモリを使用し，図 8 に示すよう 8 画素フィルタ範囲を修正した．元プログラムにより約 10 倍高速化を確認した．

4. 評価実験

本章では前章で説明した ELAS 法の各計算部の CUDA による高速化について GPU 上の実装実験により評価を行う．まず，プログラムの実行時間を測定する．また，異なる解像度画像を処理する時間を分析し，評価を行う．最後に，高速化した上で精度向上の余地を検討する．なお，実験では，Middlebury Stereo Vision Page[10]から得た画像データセットを用いた．開発環境は表 3 のものを利用した，

表 3　本実験の開発環境

項目	仕様
OS	Windows 10 64bit
CPU	Intel® Core™ i5-4460 3.2GHz
GPU	GeForce GTX 960 1127 MHz
開発ツール	Microsoft Visual Studio 2013
CUDA バージョン	8.0
CUDA コア	1,024

4.1 ELAS 法各計算部の高速化

ここでは，本研究における ELAS 法の CUDA による高速化提案の高速化率を説明する．前記開発環境で，Middlebury Stereo 2014 datasets の Motorcycle（5.9MPixel）をシングルコアによる元プログラムと本研究の提案手法で処理し実行時間を計測した．また，それぞれの CUDA 化部分について高速化率を算出した．

表 4 と図 8 に示すように，本研究において CUDA による ELAS 法の高速化率は 2 倍以上で，それぞれの各計算部については約 2~4 倍であることを確認した．

表 4　実行時間(ms)と高速化率

	元プログラム	CUDA 化	高速化率
サポート点視差計算と抽出	590	140	4.2
三角分割と視差優先値計算	175	170	1.0
全画素視差計算	1,382	513	2.7
視差の合成(後処理)	1,458	670	2.2
Total	3,605	1,493	2.4

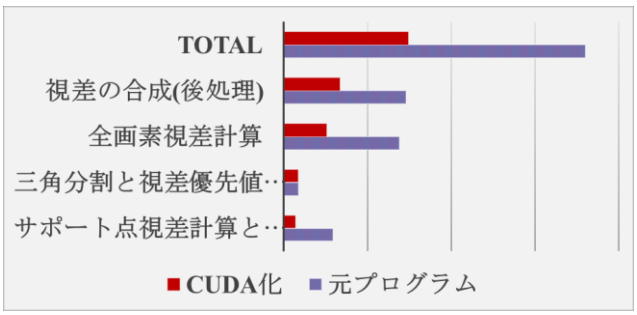

図 8　実行時間(ms)比較図

4.2 異なる解像度画像の処理

表 5 に，異なる解像度画像の提案手法による処理時間を示す．図 9 に示すように，画像解像度の増大に従って，高速化率も増大することが確認された．これにより，本研究の提案手法にでは，画像解像度の増大による処理時間の増加を抑えることができると考えられる．

表 5　異なる解像度画像の実行時間(ms)と高速化率

	画素 (MPixel)	元プログラム	CUDA 化	高速化率
Motercycle	5.928	3,605	1,493	2.41
Adirondak	5.726	3,298	1,402	2.35
Raindeer	1.490	828	375	2.20
aloe	1.423	810	356	2.27
Cones	0.675	365	185	1.97
Urban1	0.526	277	138	2.00
Urban2	0.526	275	132	2.08

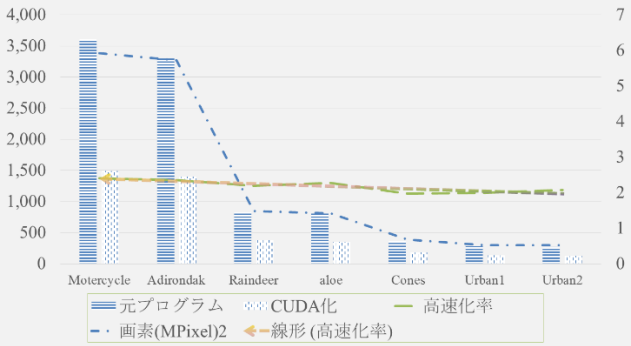

図 9　画像解像度による高速化率の推移

4.3 精度向上に関する考察

本研究の提案手法によると ELAS 法の 2 倍以上の高速化が確認された．従って，精度向上のための処理を追加しても，速度と精度が共に元プログラムより向上する可能性がある．G Li et al.[11]の研究などにより，サポート点数の追加により，サポート点ベースのステレオ法の精度向上ができると思われる．サポート点の追加を 2 倍以下にすれば，全体の処理時間の増加が 2 倍以下に抑えられるので，本提案手法の高速化率により，精度と速度が共に向上することが可能だと考えられる．

5. おわりに

本研究では，ELAS 法のようなサポート点ベースのステレオ法におけるサポート点の抽出部，サポート点による全画素の視差計算部のそれぞれの処理部分について CUDA による高速化を提案し実装した．また，視差画像後処理の高速化も提案し，一部を実装した．評価実験により，2 倍以上の高速化を確認した．更に，対象画像の解像度の増大による処理時間の増加を抑えられることも確認した．これにより，高解像度画像対応ステレオマッチングの GPGPU による高速化が実現された．

本研究の提案手法により，精度向上ための処理を追加しても，高速な処理が可能で，速度と精度ともに向上することが可能であると考えられる．

今後の課題としては，CUDA 化部分の拡大，最適化による高速化率の向上と精度向上が挙げられる．

参考文献

[1] Scharstein, D., Szeliski, R., & Zabih, R. (2001). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In Stereo and Multi-Baseline Vision, 2001.(SMBV 2001). Proceedings. IEEE Workshop on (pp. 131-140). IEEE.

[2] Yoon, K. J., & Kweon, I. S. (2005, June). Locally adaptive support-weight approach for visual correspondence search. In Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on (Vol. 2, pp. 924-931). IEEE.

[3] Kowalczuk, J., Psota, E. T., & Perez, L. C. (2013). Real-time stereo matching on CUDA using an iterative refinement method for adaptive support-weight correspondences. IEEE transactions on circuits and systems for video technology, 23(1), 94-104.

[4] Marr, David, and Tomaso Poggio. "A computational theory of human stereo vision." Proceedings of the Royal Society of London B: Biological Sciences 204.1156 (1979): 301-328.

[5] Alahi, A., Ortiz, R., & Vandergheynst, P. (2012, June). Freak: Fast retina keypoint. In Computer vision and pattern recognition (CVPR), 2012 IEEE conference on (pp. 510-517). Ieee.

[6] Geiger, A., Roser, M., & Urtasun, R. (2010, November). Efficient large-scale stereo matching. In Asian conference on computer vision (pp. 25-38). Springer, Berlin, Heidelberg.

[7] Šára, R. (2002). Finding the largest unambiguous component of stereo matching. Computer Vision—ECCV 2002, 900-914.

[8] Podlozhnyuk, V. (2007). Image convolution with CUDA. NVIDIA Corporation white paper, June, 2097(3).

[9] Stam, J. (2008). Stereo imaging with CUDA. OpenVIDIA, january.

[10] Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nešić, N., Wang, X., & Westling, P. (2014, September). High-resolution stereo datasets with subpixel-accurate ground truth. In German Conference on Pattern Recognition (pp. 31-42). Springer, Cham.

[11] Li, G., Zhang, X., Li, C., Jin, H., & Zhao, J. (2015). Design and application of parallel stereo matching algorithm based on CUDA. Microprocessors and Microsystems.

[12] Mattoccia, S., Giardino, S., & Gambini, A. (2010). Accurate and efficient cost aggregation strategy for stereo correspondence based on approximated joint bilateral filtering. Computer Vision–ACCV 2009, 371-380.

[13] Mattoccia, S., Viti, M., & Ries, F. (2011, June). Near real-time fast bilateral stereo on the GPU. In Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on (pp. 136-143). IEEE.

[14] Yang, Q. (2014). Hardware-efficient bilateral filtering for stereo matching. IEEE transactions on pattern analysis and machine intelligence, 36(5), 1026-1032.