1ZB-2

GPUを用いた拡張現実感処理の高速化 *

佐藤　一平 †　　石畑　宏明 ‡

† 東京工科大学大学院バイオ・情報・メディア研究科　　‡ 東京工科大学コンピュータサイエンス学部

1　はじめに

カメラから得られた画像情報に, コンピュータ内部の情報（仮想オブジェクト）を合成してディスプレイに出力する, 拡張現実感（AR）という技術がある. Parallel Tracking and Mapping(PTAM) [1] は, 拡張現実感を行うソフトで必須であった, マーカーという目印を必要としないために注目されている. 一方で, マーカーを使った場合よりも動作速度が低下することが問題になっている.

最近のCPUは, マルチコア化が進んでいる. グラフィックスプロセッサ（GPU）も, CPU以上にコア数を増やし, 計算速度を上げている. PTAMの動作速度を向上させる方法として, これらの複数コアを利用する方法が考えられる.

本研究では, PTAMの実行に, 最も時間のかかる部分をGPUの複数のコアを使用して並列処理をすることにより高速化する.

2　PTAM

PTAMでは, Mappingスレッドと Trackingスレッドが協調動作することで, マーカー不要のARを実現している. 図1に, PTAMの処理フローを示す.

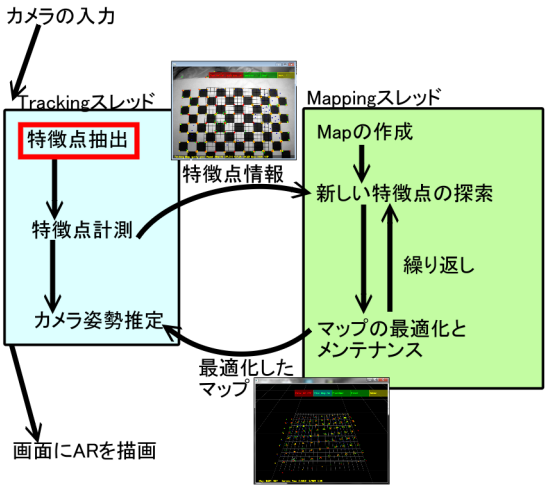

図1: PTAMの処理フロー

Mappingスレッドは, まず, ステレオ初期化の過程で特徴点のマップを作る. カメラからの画像に, 新しい特徴点が有ればマップに追加し, 使われなくなった特徴点が有れば削除を繰り返す.

Trackingスレッドは, カメラからの画像に前処理を施した後, 特徴点を抽出しマップと比較を行いカメラの姿勢を求めて, 仮想オブジェクトに反映を繰り返すことでARを実現する.

3　PTAMの調査

PTAMコードの中で, 実行に一番時間がかかっている場所（ホットスポット）を調査した.

3.1　PTAMのホットスポット分析

PTAMの全体の処理時間に対する, 各処理を図2に示す. PTAMの実行時間のうち, 画像処理部分が24 %を

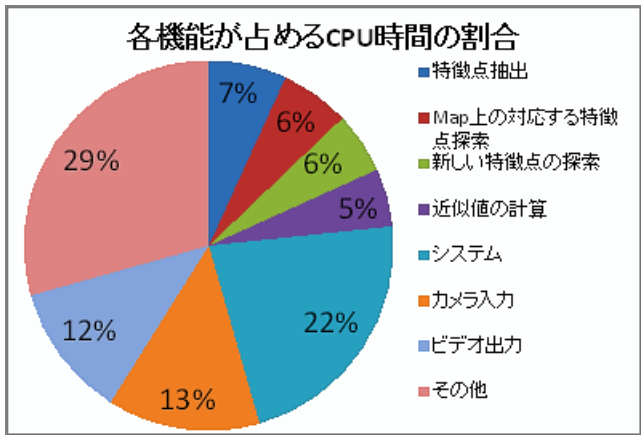

図2: PTAMの全処理時間に占める各処理の割合

有している. 画像処理部分で一番時間がかかっているのは, faster_corner_detect_10という特徴点抽出を行う関数であり, 全体の実行時間の7 %を占めている.

3.2　faster_corner_detect関数

Listing 1にfaster_corner_detect関数を示す. この関数は, 画像の4辺から中心に向かう3画素を除いた部分の画像から特徴点を抽出を行う. 入力画像データへのポインタと閾値を入力, 特徴点のXY座標を格納したベクトルのポインタを返す. if文中の, is_corner_10関数で, 特

---

* Fast Augmented Reality Processing by Using GPU

† Ippei Satou ・ Tokyo University of Technology Graduate School of Bionics,Computer and Media Sciences

‡ Hiroaki Ishihata ・ Tokyo University of Technology School of Computer Science

徴点か否かを判断し, 特徴点の XY 座標を, Vector に格納する.

Listing 1: ”faster_corner_detect”

```
1  void faster_corner_detect_10(
2  const BasicImage<byte>& I, //入力画像
3  std::vector<ImageRef>& corners, //特徴点の座標XY
4  const int barrier) //特徴点判断の閾値
5  {
6    //画像のほぼすべての画素を調べる
7    for(int y=3; y < I.size().y - 3; y++){
8      for(int x=3;x<I.size().x - 3; x++)
9        if( //実際に特徴点とするかを決める
10       is_corner_10_less(
11       &I[y][x],I.row_stride(),barrier) ||
12       is_corner_10_greater(
13       &I[y][x],I.row_stride(),barrier)
14       ) //特徴点が見つかればに追加するVector
15         corners.push_back(ImageRef(x,y));
16   }
17 }
```

is_corner_10 は, 調査する画素の周囲 48 ピクセルの明るさの変化量が, 引数で与えられた閾値と比較する関数である. 周囲の明るさの変化量が, 閾値を超えると特徴点と判断する. is_corner_10 は, 3 重の入れ子になった 125 個の if 文で構成される. 特徴点を決定するために, 最大 30 回の if 文を通過する場合もある.

3.3 faster_corner_detect_10 の CUDA 化

この部分を, GPU で動作するコード（CUDA）に書き換え実行する. 図 1 の for 文を展開し, 画素数と同数のスレッドを起動する. 画像の周囲 3 ピクセルは, 何もせずにスレッドを終了する. for 文の条件内の各スレッドは, is_corner_10 関数を呼び, 特徴点とするかどうかを判断する.

得られた特徴点は, 特徴点の Vector へ入れられるが, この部分は並列化できない. GPU で全画素に対する特徴点のデータを作成し, その後 CPU 側で Vector に取り込む処理をする.

4 実験

CPU でのホットスポット実行と GPU でのホットスポット実行を比較し, ホットスポット処理にかかる時間を比較する.

4.1 実験環境

実験に使用した環境を表 1 示す. CPU の複数コアを活用する手段として OpenMP を使用した. 実験に使用した画像は, 640 x 480 ピクセルの 8bit グレースケールの（307KB）のデータである.

表 1: 実験環境

CPU	Core i7 980X 3.33GHz
GPU	Geforce GTX 480 1.4GHz
Memory	DDR3-1600 12GB
OS	windows7 SP1 64bit
CUDA	4.0 BUG FIX Update
コンパイラ	Visual C++ 2010 (cl:16.00)

4.2 実験結果

表 2 に CPU と GPU 特徴点抽出処理に必要な時間を示す.

表 2: CPU と GPU の計算時間

CPU/GPU	コア	Time[μ 秒]	性能比	性能比
GPU	1	184,500	0.02	1.00
GPU	480	480	12.5	452
CPU	1	5,080	1.00	36.3
CPU	6	2,869	1.77	64.3

GPU は SIMD 処理であり, 分岐のすべてを実行するので, 1 コアでの性能は低い. 1 スレッド当たりの処理にかかる時間はすべて均一であるため, コア数に対してスケーラビリティーが良く 480 コアを使用した場合は, 452 倍高速化できた.

CPU は, 条件分岐の片方しか実行しないため 1 コア時の性能は良い. 並列処理の場合には, Vector への push_back 操作が並列化できないので, 特徴点判断と特徴点の追加を別ループにしている. 6 コア時の性能は, 1.77 倍にしかなっていない原因の一つと考えられる.

5 まとめ

PTAM の実行速度を上げるために, ホットスポットの特定を行った. 画像処理のホットスポットの 1 つである, 特徴点を抽出する関数, faster_corner_detect_10 を GPU で実行した. CPU による特徴点抽出と比較しても, GPU は 12.5 倍高速化出来た. if 文の多いプログラムでも, 並列性の高いプログラムであれば, GPU の性能を発揮できる事を示した.

参考文献

[1] Georg Klein,Parallel Tracking and Mapping for Small AR Workspaces,ISMAR’07,1-10pp,2007