Vol.2010-EC-17 No.7
2010/8/23

iPhone向けリアルタイム特徴点トラッキングライブラリ

近藤　誠[†1]　杉本麻樹[†1]　稲見昌彦[†1]

ゲームプラットフォームとしてもシェアを伸ばしつつある iPhone 上において, リアルタイムに動作する, 特徴点トラッキングライブラリを提案する. 本ライブラリは, OpenGl ES 2.0 の GLSL(OpenGL Shading Language) を用いて画像の処理を行い, iPhone 上での GPGPU(General-Purpose Computing on Graphics Processing Unit) を実現している. 本ライブラリにより, マーカレス AR を使ったゲームなどが開発できるようになる

Realtime Natural Feature Tracking Library for iPhone

Makoto Kondo,[†1] Maki Sugimoto[†1]
and Masahiko Inami[†1]

We propose real time natural feature tracking library for new gaming platform iPhone. Using GLSL(OpenGL Shading Language) in OpenGL ES 2.0, this library does GPGPU(General-Purpose Computing on Grpahics Processing Unit) on iPhone. This library enables users to make marker-less AR game application.

1. はじめに

携帯電話の高性能化に伴い, 今まではデスクトップコンピューターでしか実現できなかった処理が, iPhone や Android などのスマートフォン上で軽快に動作するということが可能になっている. また、AppStore, Android Market を始めとるするアプリケーションストアが次々にオープンし, そこでは様々なアプリケーションが販売されている. これらのプラットフォームは, NintendoDS や PSP と並ぶ, 新しいゲームプラットフォームとしても認知をされ始めている.

いままで, モバイルデバイスにおいて, この様に同じ OS が世界中で同時に利用され, かつ開発者が自由にアプリケーションを開発し, 販売することができるプラットフォームは存在しなかった. そのため, これからも iOS や Android の存在感は増していく物と考えられる.

一方で, これらのデバイスは, 高度なプロセッシングパワーと, 常時接続のネットワーク, そして外界の情報を取り入れるためのカメラを備え持っている. これらの環境を用いて, 商用ベースで実際に動作する, Augmented Reality アプリケーションがすでに AppStore や Android Market にいくつか登場している[1)2)]. しかしながら, これらのアプリケーションはどれもが位置情報と加速度, 及び方位情報を用いて, カメラプレビュー上に浮遊するタグのようなものを浮かせるにとどまっている. 本研究では, iPhone3GS, および iPhone 4 上で動作する, リアルタイム特徴点トラッキングライブラリを提案する.

2. 先行研究

モバイルデバイス上で動作する, マーカレス Augmented Reality 環境を提案する物としては, Georg らによる PTAM[3)] の iPhone 上での実装[4)] が挙げられる. これは既にデスクトップ PC 上で実現されていた, 同氏らによる PTAM を iPhone でもリアルタイム動作を可能にするために, 内部的な処理を簡略化した物である. このシステムにでは, 特徴点の抽出に Shi-Tomashi のコーナー検出器[5)] を用いており, これは本ライブラリでも利用している. また, Wagner らによるスマートフォン向け 6 自由度のカメラ姿勢推定システムでは, SIFT 特徴量[6)] を利用して特徴点のマッチングを行い, 特徴点の抽出には FAST[7)] と呼ばれる特徴点検出アルゴリズムを用いている.

どちらのシステムにおいても, すべての処理は CPU 上で行われている. そのため特徴点の抽出の際には処理の負荷を軽減するために, より負荷の低いアルゴリズムを利用したり, または対象画像の解像度をさげたものを用いて, あらかじめ大まかな場所を推定してる. そしてその情報をもとに実際の解像度で特徴点の抽出を行うなどしている.

3. iPhone のスペック

本ライブラリは, iPhone や iPad の OS である, iOS 4.0 の上で構築されている. そのため, 本セクションにおいてはまず本ライブラリの性質を理解する上で重要な, iOS プラットフォームの特徴を説明する.

†1 慶應義塾大学大学院メディアデザイン研究科
Graduate School of Media Design, Keio University

© 2010 Information Processing Society of Japan

3.1 カ メ ラ

iPhone にはモデルによって搭載されているカメラが異なっている. iPhone 4 には, フロントカメラ, バックカメラの二つがあり, また iPhone3GS にはバックカメラがある. それぞれのスペックを以下に示す.

カメラ名	最大解像度	オートフォーカス	形式
iPhone 4 フロントカメラ	640x480	×	BGRA32, 420YpCbCr
iPhone 4 バックカメラ	1280x720	○	BGRA32, 420YpCbCr
iPhone 3GS	640x480	○	BGRA32, 420YpCbCr

表 1 iPhone のカメラスペック表
Table 1 iPhone Camera Specification

本ライブラリにおいては, 処理能力, および各種カメラの性能の関係から, 入力画像のサイズを 640x480 とし, OpenGL 上で性能にあわせてサイズを下げて利用している. これにより, 本ライブラリは iPhone3GS および, iPhone4 のフロントカメラ, およびバックカメラの三種類の条件において動作が可能である.

デバイスのカメラから得られる画像の形式は, 420YpCbCr と BGRA32 のニ種類であり, 今回は OpenGLES を用いて画面に描画する都合上, BGRA32 形式を採用した.

カメラのフレームレートは最大で 30fps であり, 本システムにおいては, 30fps の最大値で画像を取得し, 処理しきれないフレームは適宜間引いて表示を行った.

3.2 OpenGLES 2.0

3.2.1 プログラマブルシェーダ

iPhone は, 3GS, 4 共に OpenGLES 2.0 をサポートする PowerVR SGX を搭載している. そのため, iPhone3G をはじめとする, 従来のモバイルデバイスでは利用できなかった, プログラマブルシェーダを利用することが可能となっている.

iPhone で利用可能なシェーダは, バーテックスシェーダとフラグメントシェーダの二種類である. バーテックスシェーダは, 入力された頂点情報や, テクスチャ座標を操作することによって, 形状の変更や明かりの計算などを行うシェーダであり, フラグメントシェーダは, バーテックスシェーダで計算された値を用いて, 実際に描画されるピクセルの色を決定するためのシェーダである. 本ライブラリにおける処理の大半は, フラグメントシェーダを用いて記述する.

3.2.2 テクスチャの形式

通常 OpenGL では, テクスチャの形式には 8bit 符号なし整数 (GL_UNSIGNED_BYTE) や, 固定小数点 (GL_SHORT), 浮動小数点 (GL_FLOAT) などの様々な形式が利用可能であるが, 注意すべき点としては, フレームバッファオブジェクトと接続された浮動小数点テクスチャが使えないため, フラグメントシェーダの入力, 出力に利用できる値は, 0-255 に制限されている. この制約は GPU を用いて画像処理をする際には考慮しなくてはならない.

4. アルゴリズム

GPU を用いて特徴点トラッキングをする物としては, Sudipta N Sinha らによる GPU-KLT 9) が存在する. GPU-KLT はハードウェアの仕様として, 浮動小数点テクスチャが利用できることをあげているため, 本ライブラリにおいては, これを浮動小数点テクスチャの利用をすることなしに実装を行った.

(1) **特徴点抽出**

特徴点の抽出には, 4), 9) と同様に Shi-Tomasi の特徴点検出器[5] を用いる.
Shi-Tomasi 特徴点検出器では, 以下に定義される 2x2 の行列 Z の最小固有値 (以下コーナー値) が充分大きい場合にそれを特徴点と見なすという物である.

$$Z = \begin{pmatrix} \sum_w {I_x}^2 & \sum_w I_x I_y \\ \sum_w I_x I_y & \sum_w {I_y}^2 \end{pmatrix} \tag{1}$$

I_x, I_y はそれぞれ画素値を x, y で偏微分した値であり w は対象がその近傍の領域である.

(2) **特徴点トラッキング**

特徴点のトラッキングには, まず検出された特徴点を, コーナー値が高い順に並び替え, 前のフレームにおいて得られた同じく整列された特徴点と照らし合わせる. はじめから順番にマッチングを行い, 最も距離が近く, かつ距離がしきい値よりも小さいものを同一の特徴点と見なしてトラッキングする. 一通り対応関係が決定した後, すべての対応における, 特徴点の移動距離の平均を算出し, そこから極端に大きな物は外れ値と見なし, トラッキングされた物のリストから取り除く.

5. 実 装

5.1 GPU の計算モデル

CPU から GPU へのデータは, glTexImag2D を用いて, テクスチャとして渡し, GPU か

© 2010 Information Processing Society of Japan

ら CPU へのデータは glReadPixels を用いて画素値の配列として受け取る.

テクスチャとして渡されたデータは, そのテクスチャと同じサイズの, テクスチャに接続されたフレームバッファオブジェクトに対して, シェーダを用いて描画する. このときに, 特定の処理が一回実行され, 結果が描画対象のフレームバッファオブジェクトに書き込まれることになる. この処理結果にさらに処理を適用する場合には, そのフレームバッファオブジェクトが接続されているテクスチャを, 同じサイズの, テクスチャに接続された別のフレームバッファオブジェクトに対して描画する. この二つのフレームバッファオブジェクトのペアが, 互いにシェーダを用いて描画し合うことによって, 複数の処理を適用する.

一回のフレームバッファオブジェクトへの描画では, それぞれの画素値に対して, 与えられたテクスチャの座標, 及びその近傍の値を用いて, 新しい画素値を計算して代入することができる.

近傍の値を用いる計算をする際には, その前の描画との同期をとらなければならないために, 一度の描画で一度までしか用いることができない.

5.2 特徴点抽出

特徴点を抽出する際に必要な処理は, 複数のシェーダからなりたっている. 以下に実際に利用されているシェーダのリストを挙げる.

- 前処理
- 微分 (水平)
- 微分 (垂直)
- 行列の基礎要素計算
- ウィンドウ内での集積 (水平)
- ウィンドウ内での集積 (垂直)
- コーナー値計算

図 1 は本ライブラリによって, 画像から特徴点が抽出される様子を示した画像であり, 図 1 a は入力画像である.

以下に, iPhone の OpenGL ES 2.0 の GLSL を用いて特徴点を抽出する処理の詳細を記述する.

(1) **前処理**

まず, 前処理として, カメラから得られた画像をグレースケール画像に変換する. カラー画像からグレースケール画像への変換は, YUV 色空間に変換後, その Y 成分を抽出することによって行う. 前述の通り, 画像の形式は BGR であり, また縦横が転置しているため, 転置

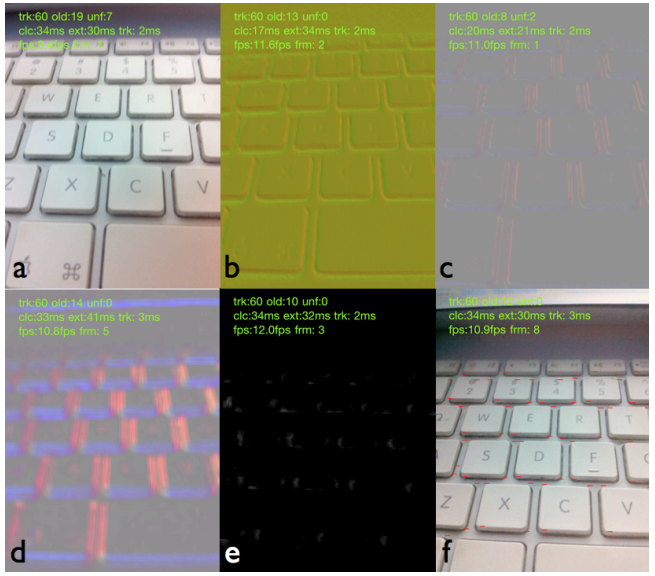

図 1 処理画像の流れ
Fig. 1 Image sequence

を戻す処理もここで行う. ここまでのグレースケール化と転置の処理をあわせて一つ目のシェーダで行い, すべての処理はフラグメントシェーダで行う.

(2) **微分**

微分は水平方向と垂直方向の二回にわけて適用する. そのため, それぞれのフィルタは一次元で表現され, それらを水平, 垂直の両方向に適用することで, 二次元のフィルタを適用した場合と同様の結果を得ることができる.

水平方向の微分を行う時には, 水平方向にはガウシアンフィルタを微分したものを, 垂直方向にはガウシアンフィルタをかけ, また同様にして垂直方向にも微分する.

フィルタは, 中心から両方向にに 3 つの値, 幅が 7 のウィンドウを用いており, 全体の総和が 1 になるように正規化されている. ガウシアンフィルタの σ は可変であるが, ここでは 1.0 をデフォルトの値としている.

フィルタ処理を行って得られた x 方向, y 方向の微分値は, RGBA の四つのコンポーネントのうちの二つのコンポーネントにそれぞれ格納する.

© 2010 Information Processing Society of Japan

また, フラグメントシェーダーで利用できる画素の値の範囲は 0-255 までであるので, 負の値を表現する為に計算結果の画素値に 0.5(128) を加えておく. 水平方向, 垂直方向のための微分で計二個のシェーダをここでは利用する.
図 1 b は入力画像に対してこの微分フィルタを適用した結果の画像である.

(3) **行列の基礎要素計算**

コーナー値は, (1) で示した行列の最小固有値である. そのため, まず各点においてそれぞれの値, $I_x{}^2, I_xI_y, I_y{}^2$ を計算する. I_x, I_y はテクスチャの R と G の成分に, 0.5 を基準として代入されているので, 0.5 を引いてそれぞれの値を計算し, その結果にまた基準の 0.5 を加算して, 結果を R, G, B のコンポーネント値として代入する. 図 1 c が b の入力に対して行列の要素計算を行った物である. これもまた一つのフラグメントシェーダである.

(4) **ウィンドウ内での集積**

計算された各々の行列の基礎要素を, 特定のウィンドウ内ですべて足し合わせる. ウィンドウの領域は 7x7 を利用し, 微分の際と同様に水平方向に足し合わせた後に, 垂直方向についても同様に足し合わせることによって, 実現する. 結果は前項と同様に R, G, B のコンポーネント値として, それぞれ $\sum_w I_x{}^2, \sum_w I_xI_y, \sum_w I_y{}^2$ の値を代入する. 水平, 垂直それぞれ一つずつ計二つのシェーダを利用する. 図 1 d が c の入力をウィンドウ内で集積した結果であり, それぞれの線が太く協調されていることがわかる.

(5) **コーナー値計算**

コーナー値は, (1) に示される行列の最小固有値に相当する. (1) の行列は実対象行列であるので, 固有値は以下の二次方程式の二つの解として求めることが可能である.

$$(\sum_w I_x{}^2 + \sum_w I_y{}^2)\lambda^2 + (\sum_w I_x{}^2 - \sum_w I_y{}^2)\lambda + \sum_w I_x{}^2 \sum_w I_y{}^2 - (\sum_w I_xI_y)^2 = 0$$

これは通常の二次方程式であるので, 解の公式を用いて解き, その小さい方をコーナー値として任意の RGBA のコンポーネントに代入する. 図 1 e が d のコンポーネントのそれぞの値からコーナー値を計算した物であり, 図中に白く映っている塊がコーナーを示している.

(6) **CPU への読み出し**

この様にして計算された値をコーナー値を glReadPixels を用いて GPU から読み出す. 得られる画像の形式は, RGBA であり, その中から前述のフラグメントシェーダを用いたときに代入したコンポーネントから参照することができる. この段階では, まだ極大値の抽出はされていないので, コーナー付近に高い画素値の塊がある.

(7) **極大値取得**

読み出された画像から極大値のみを抽出するために, すべての画素値とその座標の組を, 画素値の高い順に並べ替える.
次に, 解像度と同じサイズの配列を用意し, これをマップとして利用する. 画素値の高い順から, その座標を取得し, 特徴点として登録する. 極大値のみを取得するために, 選ばれた特徴点の近傍はマップに利用済みとしてセットされ, 以後新たな特徴点は, マップが空いている時にのみ登録する.

5.3 トラッキング

前節でも示した通りに, トラッキングの処理はすべて CPU 上で行われる. テクスチャに利用できる型の制約上, 9) で用いられている GPU 上の KLT トラッカーは動作しない. 本ライブラリにおいては, それぞれの特徴点の動きが十分に小さく, かつ極端に近くに複数の特徴点が存在しないこと, そしてコーナー値が高い特徴点は少し移動しても高いままであることを仮定して, 特徴点の高いものから順に, 前のフレームの座標から最も近いものを選ぶという独自の方法を採用している.
図 2 は実際に iPhone 上で本ライブラリが動作している様子を表しており, 画像中に表示されている線がトラッキングされた特徴点の軌跡を表している. 軌跡からわかる通り, この画像はカメラを右から左に移動させたときのものである.

6. 性　　能

図 3 は本ライブラリを iPhone4 上で用いた際の, 対象画像の解像度と各処理の時間の関係である. 解像度が 160px x 213px を超えたあたりから爆発的に処理時間が増大しており, また時間は主にシェーダーの処理と極大値の取得の二つの処理に費やされていることがわかる.
通常の利用の際には, 処理速度の関係から, 160px x 213px の解像度を用いることが望ましく, またそのときの平均のフレームレートは 10.05fps となっている.

7. 結　　論

本研究では, GPU を用いることで, iPhone 上でリアルタイムに特徴点を抽出, トラッキングするライブラリを開発した.
今後の課題としては, まず GPU 化されていないトラッキングの処理を GPU を用いて行う物に差し替えていくことが挙げられる. これと同時に, 毎フレーム特徴点抽出を行うことなくトラッキング可能なアルゴリズムを利用することで, 処理速度の向上を実現することが

© 2010 Information Processing Society of Japan

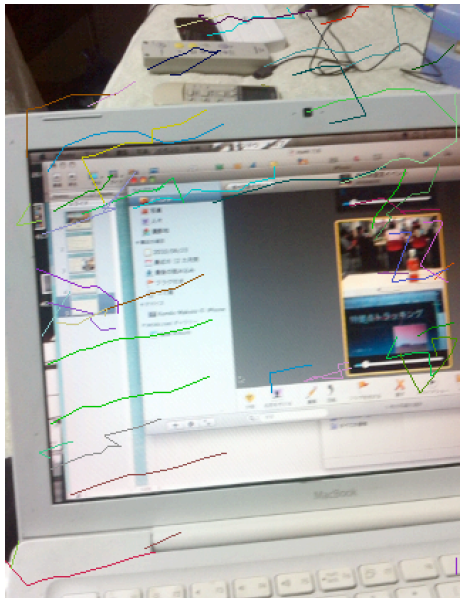

図 2 トラッキングの様子
Fig. 2 Tracking image

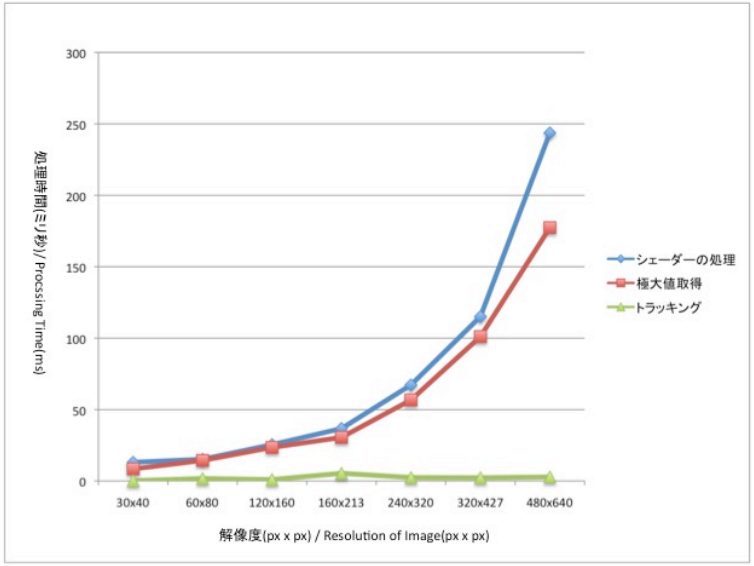

図 3 解像度と各処理の時間
Fig. 3 Resolution and Processing Time

できると考えられる. また, ボトルネックとなっている CPU へのデータ読み出し部の処理方法の改善も別な方法の検討が必要である. さらには, GPU, OpenGL ES 2.0 を搭載した Android などの別プラットフォームに移植し, クロスプラットフォーム化も進めていきたい.

参 考 文 献

1) Augmented Reality Browser:Layar http://www.layar.com/
2) セカイカメラ http://sekaicamera.com/ja/
3) G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In Proc 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR ' 07), October 2007
4) Georg Klein and David Murray Parallel Tracking and Mapping on a Camera Phone In Proc. International Symposium on Mixed and Augmented Reality (ISMAR'09, Orlando)
5) Jianbo Shi and Carlo Tomasi. Good Features to Track. IEEE Conference on Computer Vision and Pattern Recognition, pages 593-600, 1994.
6) D. Lowe, Distinctive Image Features from Scale-Invariant Key- points, Int ' l J. Computer Vision, vol. 60, no. 2, pp. 91-110, 2004.
7) E. Rosten and T. Drummond, Machine Learning for High-Speed Corner Detection, Proc. European Conf. Computer Vision (ECCV ' 06), pp. 430-443, 2006.
8) D. Wagner, G. Retimayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose Tracking from natural features on mobile phones. In Proc 7th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'08), Sept. 15-18 2008.
9) Sudipta N Sinha, Jan-Michael Frahm, Marc Pollefeys and Yakup Genc, GPU-Based Video Feature Tracking and Matching, Technical Report 06-012, Department of Computer Science, UNC Chapel Hill, May 2006.

© 2010 Information Processing Society of Japan