2005－CG－120（4）
2005／8／18

# EasyGPU : GPU を用いたコンピュータビジョン実験環境の開発

高橋　誠史[1]　宮田　一乗[2]　白井　暁彦[3]
[1]北陸先端科学技術大学院大学 知識科学研究科
[2]北陸先端科学技術大学院大学 知識科学教育センター
[3]CPNI Lab Laval / ENSAM
E-mail: { masa-t, miyata }@jaist.ac.jp, shirai@mail.com

**概要**　本論文では, コンピュータビジョンにおける各種演算をプログラマブル GPU 上で行うシステムの提案を行う．本システムは，GPU のプログラムコードが実行ファイルから独立していることに着眼して，画像処理のアルゴリズムのモジュール化を行う．そして，このモジュールの処理順序をグラフィカルなインターフェイスで組み換え可能な，コンピュータビジョンに関するアプリケーション開発の支援環境を試作した.

# EasyGPU : Development of experimental environment for computer vision by means of programmable GPU

Masafumi Takahashi[1], Kazunori Miyata[2], Akihiko Shirai[3]
[1]School of Knowledge Science, Japan Advanced Institute of Science and Technology
[2]Center for Knowledge Science, Japan Advanced Institute of Science and Technology
[3]CPNI Lab Laval / ENSAM
E-mail: { masa-t, miyata }@jaist.ac.jp, shirai@mail.com

**Abstract** This paper proposes a system which performs various operations for computer vision process on programmable GPU. Program codes of GPU are independent from an executable code, therefore, this system modularizes the algorithm of image processing. This paper also reports an experimental environment for developing computer vision applications.

## 1. はじめに

近年，リアルタイム 3DCG の高品質なレンダリングに利用されるグラフィックスハードウェア(Graphics Processing Unit, 以下 GPU と書く)は, レンダリングパイプラインのジオメトリ処理やピクセル処理に伴うベクトルや行列処理がプログラム可能なアーキテクチャになった. プログラマブル GPU は, 3DCG のジオメトリ処理やピクセル処理などの各工程がプログラマブルな設計になっており, 各プログラムの処理ユニットは並列的にデータを処理する能力を持っている. このような強力な計算能力を利用して, 本来の目的であるレンダリング以外のコンピュータサイエンスの諸問題を解決する研究が広がりを見せている[1].

本研究では, 以上述べたような背景に基づきプログラマブル GPU の強力なピクセル処理能力をコンピュータビジョンの分野に適用することを目的とする．そして, 高解像度のカメラでも実時間で動作するアプリケーションを開発するためのグラフィカルな実験環境を構築する. 本研究で開発するシステムは，GPU 上で画像処理アルゴ

リズムを簡単に利用ができることから，”EasyGPU”と名づける．

本研究の最終目標は，GPU ベースのコンピュータビジョンを含む画像処理のグラフィカルな開発環境とアプリケーション作成のためのライブラリの整備である．本論文では，その環境を実現する第 1 歩としての実験環境の開発について述べる．

## 2. 研究の背景

### 2.1 目的

コンピュータビジョンのリアルタイムアプリケーションでは，リアルタイム性を高めるために取り込む動画の処理に対する計算負荷をいかに抑えるかという課題がある．この課題に対する解決としては，カメラ側の解像度やフレームレートなどのスペックを落とす方法や精度の低いアルゴリズムの採用などが挙げられる．これらの方法は，比較的容易に実装ができるが，その反面高い効果が望めるアルゴリズムの採用が難しくなる．

その他に，マルチコアの CPU の採用やキャプチャ専用の PC を複数台設置し，ネットワークを介して負荷分散を行う方法などが挙げられるが，これらはアプリケーションの実装の難易度が上がると考えられる．

本研究では，アプリケーションにおける動画像処理の負荷の軽減のために取り込む画像をビデオメモリ上に転送して処理する．そして，プログラマブル GPU の並列計算能力を利用して高速処理をさせることで，アプリケーション上での CPU の負荷を下げる．

GPU のプログラムコードはアプリケーションの実行ファイルに対して独立したファイルでの記述が可能である．これは，実行ファイル側の変更を行わなくても GPU のプログラムコードを書き換えるだけで，アルゴリズムの調整や組み換えが可能となる．実行ファイル側の変更を行わないでアルゴリズムの調整や組み換えがエディタ上で即時に可能なことは，実行ファイルのビルドに伴う時間がほとんど発生しないため生産性を高めると考える．

画像処理アルゴリズムのモジュール化という点では DLL を使う方法もあるが，DLL 自体も実行ファイルと同様にビルドの必要性がある．一方，GPU 上のプログラマムは基本的にバイナリデータではなくテキストデータであり，読み込み後にリアルタイムでコンパイルされる，したがって DLL に比べて調整がしやすいものと考える．さらに DLL は実行する CPU に依存したバイナリであるためテキストデータである GPU のプログラムコードは他の CPU 環境への移植性も高いものと考える．

### 2.2 関連研究

GPU をベースとしたコンピュータビジョンライブラリの研究例には，OpenVIDIA[2]がある．OpenVIDIA は，NVIDIA 社のハードウェアを対象に，OpenGL と同社の開発した GPU 向けプログラミング言語の Cg を組み合わせで開発を行う．このライブラリは，複数の GPU を挿した PC で GPU の並列計算が行うこともできる．このライブラリを利用することで，高速処理が可能なコンピュータビジョンのための環境を構築することができる．

GPU 向けプログラミング言語の Sh[3]ではアルゴリズムをモジュール化して扱える機能がある．これは，アルゴリズムの実行順序や結合などを実行ファイル側のコードで指定できる機能であり，アルゴリズムの再利用性を高める仕組みである．Sh は，GPU 向けのシェーディングプログラム言語でコンピュータビジョンを目的にした言語では無いが，この仕組みは本研究においても応用できると考えた．ただ，Sh はあくまでも C++ からのアクセスを想定して設計されており，アル

ゴリズムをモジュール化してグラフィカルに操作する設計ではない.

グラフィカルなユーザーインターフェイスを備えたビジュアルプログラミング環境としては, Apple 社の Quarts Composer がある．また，このソフトウェアではグラフィカルなユーザーインターフェイス上で, ビデオや画像の処理の手順を指定してビジュアルエフェクトを構築していく．このソフトウェアでは，GPU を用いた処理を利用することも可能である.

本研究では, アルゴリズムの実行手順をグラフィカルなユーザーインターフェイスを用いて, 開発の省力化を目指す.

## 3. GPU 上での画像処理

### 3.1 作業データ

GPU 上で動画像処理を行うためのに，リアルタイムに取り込んだ動画像をテクスチャメモリに書き込む方法を用いる. テクスチャメモリに書き込むことで，GPU のピクセル処理機能からデータアクセスがしやすくなる.

### 3.2 有効な手法

GPU 上で実行するのに有効な画像処理アルゴリズムには, 並列計算が可能な処理が挙げられる. 具体的には，フィルタ処理，色空間変換処理，色認識処理，2 値化処理などがあげられる.

その他に, マルチテクスチャリングの機能などを使ったテクスチャ同士の論理演算などが挙げられる. 具体的には, マスク処理や背景差分などの手法に応用できると考える.

本研究では, これら有効な手法とその処理の仕組みを実装できるシステム設計を目指す.

### 3.3 実装例

著者らは, 本システムを試作する以前に, GPU を利用した 3 つのシステムの発表を行い評価をした.

ViewFrame[4]においては, GPU を用いたマーカーレスのリアルタイム顔移動追跡を行った. この研究では，取り込んだ動画像を RGB 表色系から CIE L*a*b 表色系に変換して肌色領域の抽出を行うところまでを GPU 側に行う．RGB 表色系から CIE L*a*b 表色系への変換は，DV フォーマットに準拠した 720x480 ピクセルの解像度に対して行う場合, 高い処理負荷があるが, その解消に GPU を用いることが有効であると確認した.

LuminaStudio[5]では，リアルタイム画像合成を行う方法を提示した. カメラでキャプチャした映像中のマスク領域に対して別の動画を合成して, バーチャルセットのための実装例を提示した.

RoboGamer[6]では，ビデオゲームをリアルタイムで撮影して画像解析を行い, それを元にゲームコントローラの自動操縦を行うシステムを開発した. このシステムでは，デジタルビデオカメラで撮影したビデオゲームの画面をリアルタイムで解析する必要性から GPU を用いた.

## 4. システム設計

### 4.1 処理

従来のコンピュータビジョンの処理は, 入力画像に対して画像処理アルゴリズムを順次に適用していく(図 1).

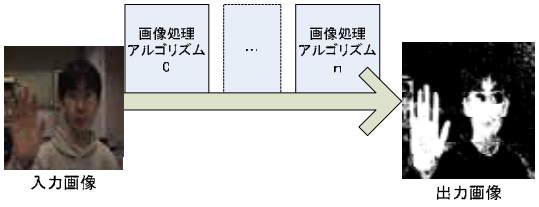


図 1.一般的な画像処理の流れ

一方で，GPU を利用するシェーディングプログラミングでは, 頂点処理を行う頂点シェーダプ

ログラミングとピクセル処理を行うピクセルシェーダプログラミングのセットを順番に処理する.

本研究では，GPU のピクセルシェーダプログラム上で画像処理プログラムミングのアルゴリズムを実行する．GPU プログラミングのコードは, アプリケーションの実行ファイルのコードとは別に外部に置いておくことが可能であるため, 個別のピクセルシェーダプログラムで実現されたアルゴリズムを画像処理モジュールとして捉える.

### 4.2 GPU 上の画像処理の流れ

EasyGPU での処理の流れは，通常の画像処理アルゴリズムの組み合わせをピクセルシェーダプログラムの組み合わせに置き換えた構造になる（図 2)．各ピクセルシェーダプログラムの出力結果はテクスチャデータにレンダリングされる. この出力結果を格納したテクスチャデータを次に処理するピクセルシェーダに渡す. このようにしてアルゴリズムを連続して処理できる構造にする.

画像処理アルゴリズムを実行するピクセルシェーダの他にアルゴリズムの処理単位で管理する必要性がある要素として, 入力画像以外のテクスチャデータがある. 入力画像以外のテクスチャとは, 例えば背景差分処理やマスク処理などのアルゴリズムで使う背景画像やマスク画像などが挙げられる.

EasyGPU では, 画像処理アルゴリズムを実行するピクセルシェーダと, 処理ごとに必要なテクスチャデータの管理をセットにして, モジュール化する.

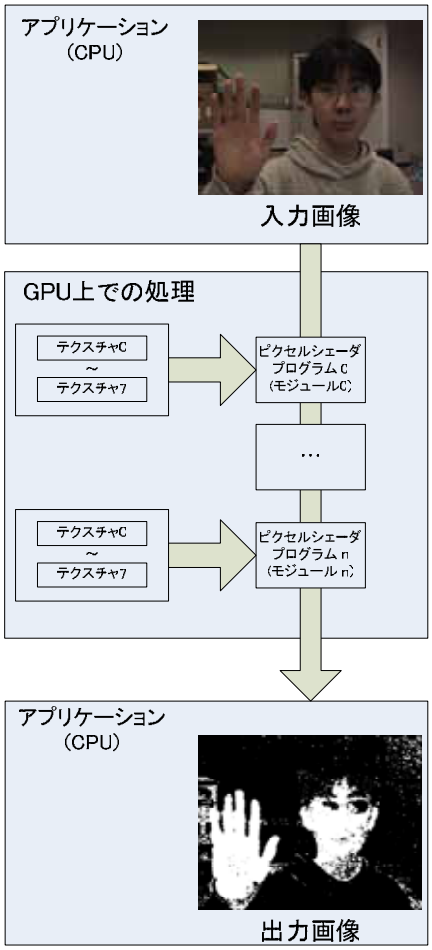


図 2.GPU 上画像処理の流れ

### 4.3 シェーディング言語

GPU プログラミング言語には，Microsoft 社 DirectX における HLSL を利用したエフェクトファイルフォーマットを利用する. HLSL とエフェクトファイルフォーマットを採用した理由は, アノテーションやセマンティクスと言う仕組みが利用できる点である.

セマンティクスとは変数に意味的な定義を与える仕組みである. 意味的な定義はアプリケーション側で自由に定義することができ, 異なった名称の変数でも機能的な意味で変数の呼び出しが可能となる.

アノテーションとは, 変数につける注釈機能である. ここで付けた注釈は実行ファイルから読み取りが可能である．この機能の利用例としては,

シェーダで取り扱うテクスチャにバインドするテクスチャファイルの関連性などの記述が挙げられる.

以上の仕組みにより, シェーダファイルからアプリケーションコードの制御をする仕組みが構築できる. こうした仕組みは実行ファイル側の処理に変更を加える際のビルドの手間を軽減できるという利点がある.

さらにアノテーションにはユーザーインターフェイスを記述するための共通の仕組みがある. すなわち, 変数の編集用のユーザーインタフェイスのタイプを指定できる. ここでいうユーザーインターフェイスのタイプとは, Windows のダイアログのボタンやスライダーなどである.

エフェクトファイルフォーマットは, NVIDIA 社の CgFX との互換性があるので Windows 以外の環境への移植が可能である.

## 5. システムの実装

### 5.1 システムの概要

EasyGPU では, 任意のキャプチャデバイス, ビデオファイル, 静止画像を入力画像に使うことが出来る. 入力画像はテクスチャメモリに書き込まれ, テクスチャサンプラにセットされる. テクスチャサンプラとは実行するピクセルシェーダが利用するテクスチャのことを指す. その後に画像処理アルゴリズムを実行するシェーダプログラムを実行する. この処理の流れを図 3 で示す.

図 4 に動作画面を示す. ここでの①は入力画像, ②は処理画像である.

アルゴリズムを格納したエフェクトファイルは, 図 4 中の③のボタンを押してシステム側にロードされる. EasyGPU ではシェーダプログラムのエディタは備えない. したがって, アルゴリズムのプログラミング自体は, Microsoft 社の Effect Edit[8] や ATI Technologies 社の RenderMonkey[9], NVIDIA 社の FX Composer[10]と言ったシェーダ開発ツールなどの利用を前提としている.

ロードされたエフェクトファイルは図 4 中の④のリストボックス上に処理順に表示される. 処理の順番の組み換えは, 図 4 中の⑤のボタンで行う. 図 4 中の⑥では, 入力画像から出力画像までの処理の流れが図示される.

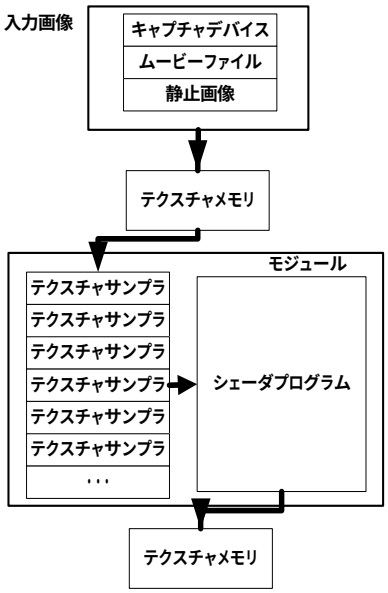


図 3.EasyGPU の処理の流れ

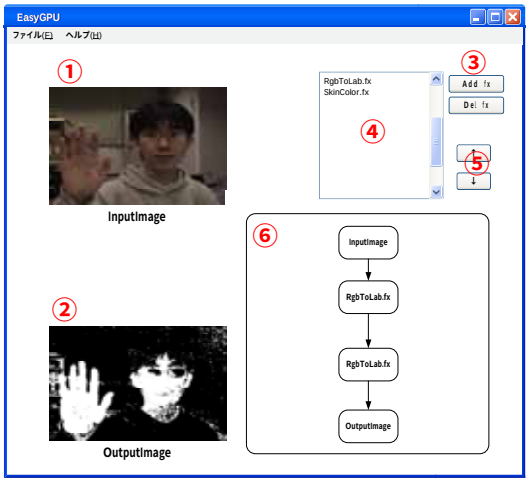


図 4.動作画面

### 5.2 シェーダプログラミングの仕様

アルゴリズムを実装するシェーダプログラムには HLSL を採用した.

入力画像以外のテクスチャを用いる場合には実行ファイル側でファイルをロードするが, この際のファイルパスは, エフェクトファイルのアノテーションを利用して記述する. アノテーション記法に関しては, 3D モデリングソフトウェアなどでも採用されている DXSAS(DirectX Standard Annotations and Semantics)を利用する.

モジュールの出力先であるテクスチャデータをバインドするテクスチャサンプラは, このシステムで定義したセマンティクスを用いて指定される.

## 6. まとめと今後の展望

GPU を用いたコンピューティングは, 本来のシェーディングプログラミングと異なった用途で利用する場合においてもシェーダ開発のプロセスが応用できるのではないかと考える. しかし, 現在のシェーダ開発の統合環境は GPU コンピューティングの多様な用途に対応しきれてはいない. したがって, コンピュータビジョンや動画像処理に特化した開発支援のシステムが必要ではないかと考える.

EasyGPU の開発は本稿の執筆段階においても続けられている. したがって今回の報告においては, 基本的な機能の実装と方向性の提示が中心になった. システムの評価や開発方法については今後発表予定である.

今後の展望として, アプリケーションプログラマが EasyGPU で作成した処理をそのまま組み込めるようにするためのランタイムライブラリの開発を検討している. また現状のエフェクトファイルでは, 複数ファイルの処理の順序を記述することが出来ない. この問題に対処するためには EasyGPU 上での作業結果を保持するメタファイルをデザインする必要があると考えている.

## 7. 参考文献


[1]GPGPU.org
http://www.gpgpu.org

[2]OpenVIDIA, http://openvidia.sourceforge.net/

[3]Sh, http://libsh.org/

[4]Quartz Composer,
http://developer.apple.com/documentation/GraphicsImaging/Reference/QuartzComposerRef/index.html

[5] 河原塚 有希彦, 高橋 誠史, 宮田 一乗," ViewFrame2-マーカレス顔部検出手法を利用した“ViewFrame”-", 芸術科学会論文誌 Vol.3 No.3 , DiVA 展特集論文, pp. 189-192

[6] A. Shirai, M. Takahashi, K. Kobayashi,H. Mitsumine, and S. Richir, " Lumina Studio: Supportive Information Display for Virtual Studio Environments", IEEE VR 2005 Workshop on Emerging Display Technologies, pp.17-20, Bonn Germany, 2005.

[7] Akihiko Shirai, Lionel Dominjon, Masafumi Takahashi, Kazunori Miyata, Makoto Sato, Simon Richir, "RoboGamer: A robotic TV game player", ACE 2005

[8]Effect Edit,
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/directxsdk/Tools/Content/EffectEdit.asp

[9]RenderMonkey,
http://mirror.ati.com/developer/rendermonkey/index.html

[10]FX Composer,
http://developer.nvidia.com/object/fx_composer_home.html