JAIST Repository
https://dspace.jaist.ac.jp/

Title	グラフィックスハードウェアを活用した画像処理手法の開発支援システムの提案
Author(s)	高橋，誠史
Citation	
Issue Date	2005-09
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/588
Rights	
Description	Supervisor:宮田　一乗，知識科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修　士　論　文

グラフィックスハードウェアを活用した画像処理手法の開発支援システムの提案

指導教官　　宮田一乗　教授

北陸先端科学技術大学院大学
知識科学研究科知識社会システム学専攻

350203　高橋 誠史

審査委員：　宮田　一乗　教授（主査）
國藤　進　　教授
西本　一志　助教授
金井　秀明　助教授

2005 年 8 月

Copyright © 2005 by Masafumi Takahashi

目　次

１　序論　1

１．１　研究の背景と目的　．　．　．　．　．　．　．　．　．　．　．　．　．　．　１

１．２　GPU の利用　．　．　．　．　．　．　．　．　．　．　．　．　．　．　３

１．３　関連研究　．　．　．　．　．　．　．　．　．　．　．　．　．　．　３

１．４　本研究の位置づけと効果　．　．　．　．　．　．　．　．　．　．　．　４

2　GPU を用いた画像処理　6

２．１　GPU に適した画像処理アルゴリズム　．　．　．　．　．　．　．　．　６

２．１．1　並列処理　．　．　．　．　．　．　．　．　．　．　．　．　．　6

２．１．2　算術合成　．　．　．　．　．　．　．　．　．　．　．　．　．　11

２．１．3　拡大・縮小　．　．　．　．　．　．　．　．　．　．　．　．　．　12

２．２　適用事例　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．　13

２．２．１　ViewFrame　．　．　．　．　．　．　．　．　．　．　．　．　．　１3

２．２．２　LuminaStudio　．　．　．　．　．　．　．　．　．　．　．　．　16

２．２．３　RoboGamer　．　．　．　．　．　．　．　．　．　．　．　．　．　21

3　システム設計　２４

３．１　画像処理アルゴリズムの GPU プログラミングへの組み込み　．　．　24

３．２　システムの構成要素　．　．　．　．　．　．　．　．　．　．　．　．　27

３．２．1　入力画像の取り込み　．　．　．　．　．　．　．　．　．　．　．　27

３．２．2　GPU プログラミング言語　．　．　．　．　．　．　．　．　．　27

３．２．3　ユーザーインターフェイス　．　．　．　．　．　．　．　．　．　29

４　システムの実装　３１

４．１　システム概要　．　．　．　．　．　．　．　．　．　．　．　．　．　．　31
４．２　画面レイアウト　．　．　．　．　．　．　．　．　．　．　．　．　．　33
４．３　ソフトウェア部の実装　．　．　．　．　．　．　．　．　．　．　．　34
４．３．1　画像データのテクスチャデータ化　．　．　．　．　．　．　34
４．３．２　画像処理モジュールの取り込み　．　．　．　．　．　．　35
４．３．3　手順の可視化　．　．　．　．　．　．　．　．　．　．　．　36

５　結論　３７
５．１　考察　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．　37
５．２　今後の課題と将来展望　．　．　．　．　．　．　．　．　．　．　．　38

参考文献　３９
謝辞　４０
本研究に関する研究発表　４１

図　目　次

１．１シェーディングプログラム適用のモデル　．　．　．　．　．　．　．　．　．　．２
１・２　画像処理プログラムの流れ.　．　．　．　．　．　．　．　．　．　．　．　．　．　．２
２．１　平滑化フィルタ処理の例　．　．　．　．　．　．　．　．　．　．　．　．　．　．8
２．２　ラプラシアンフィルタ処理の例.　．　．　．　．　．　．　．　．　．　．　．　．9
２．３　エンボス処理の例．　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．10
２．４　減算合成の例　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．12
２．５　乗算合成の例　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．12
２．６　縮小を用いた重心抽出例．　．　．　．　．　．　．　．　．　．　．　．　．　．13
２．７　ViewFrame システム概要　．　．　．　．　．　．　．　．　．　．　．　．　．15
２．８　ViewFrame の利用風景　．　．　．　．　．　．　．　．　．　．　．　．　．15
２．９　GPU での肌色領域抽出　．　．　．　．　．　．　．　．　．　．　．　．　．16
２．１０　LuminaStudio のシステムのイメージ　．　．　．　．　．　．　．　．　．19
２．１１　LuminaStudio の処理の流れ.　．　．　．　．　．　．　．　．　．　．　．20
２．１２　RoboGamer の設置風　．　．　．　．　．　．　．　．　．　．　．　．　．22
２．１３RoboGamer 画面．　．　．　．　．　．　．　．　．　．　．　．　．　．　．23
２．１４　解析画面　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．　．23
３．１　一般的な画像処理プログラミングの流れ　．　．　．　．　．　．　．　．24
３．２　一般的な GPU プログラミング　．　．　．　．　．　．　．　．　．　．25
３．３　パス間でデータを受け渡すモデル　．　．　．　．　．　．　．　．　．　．25
３．４　システムのデータフロー　．　．　．　．　．　．　．　．　．　．　．　．26
３．５　システムを構成する重要な３要素　．　．　．　．　．　．　．　．　．　．27
３．６　画像処理の流れ　．　．　．　．　．　．　．　．　．　．　．　．　．29
3. ７　可視化の例　．　．　．　．　．　．　．　．　．　．　．　．　．　．30
４．１　GPU を搭載した PC　．　．　．　．　．　．　．　．　．　．　．　．31
４．２　画面レイアウト　．　．　．　．　．　．　．　．　．　．　．　．33
４．３　入力画像のテクスチャデータ化　．　．　．　．　．　．　．　．　．　．35

第１章　序論

本章では，本論文で提案するシステムの目的とグラフィックハードウェアを取り巻く環境，および関連研究について述べ，本論文の位置づけを示す．

1.1　研究の背景と目的

近年，リアルタイムで３次元コンピュータグラフィックス（以下，3DCG）の高品質なレンダリングの処理には，グラフィックスハードウェア(Graphics Processing Unit，以下 GPU と書く)が利用されている．GPU は，３次元物体をディスプレイの２次元空間に射影するための演算と，物体表面の材質感表現，および光のシミュレーションを行う機能などがプログラムによりカスタマイズ可能なハードウェア構成になっている．このうち射影のためのプログラムを頂点シェーダと呼び，表面の材質と光のシミュレーションを行うプログラムをピクセルシェーダプログラム（または，フラグメントシェーダプログラム）と呼ぶ．図 1.1 は，入力された形状データに対して，ピクセルシェーダプログラムで記述された表面の質感表現を行うモデルを図示している．

一方，画像処理は画像の中から様々な情報を取得したり，芸術的な表現や効果を施す場合などに用いられる．画像処理プログラムでは，図 1.2 に示すように入力画像に対して様々な画像処理アルゴリズムを適応することで出力画像を得る．

本論文では，ピクセルシェーダの処理モデルと画像処理のモデルの類似性に着目し，シェーダ開発のモデルを取りいれた画像処理アプリケーションのためのグラフィカルな開発システムを提案する．本論文の目的は，GPU の高速な演算の機能を画像処理アルゴリズムに適用すること，およびアプリケーションの開発効率の向上である．

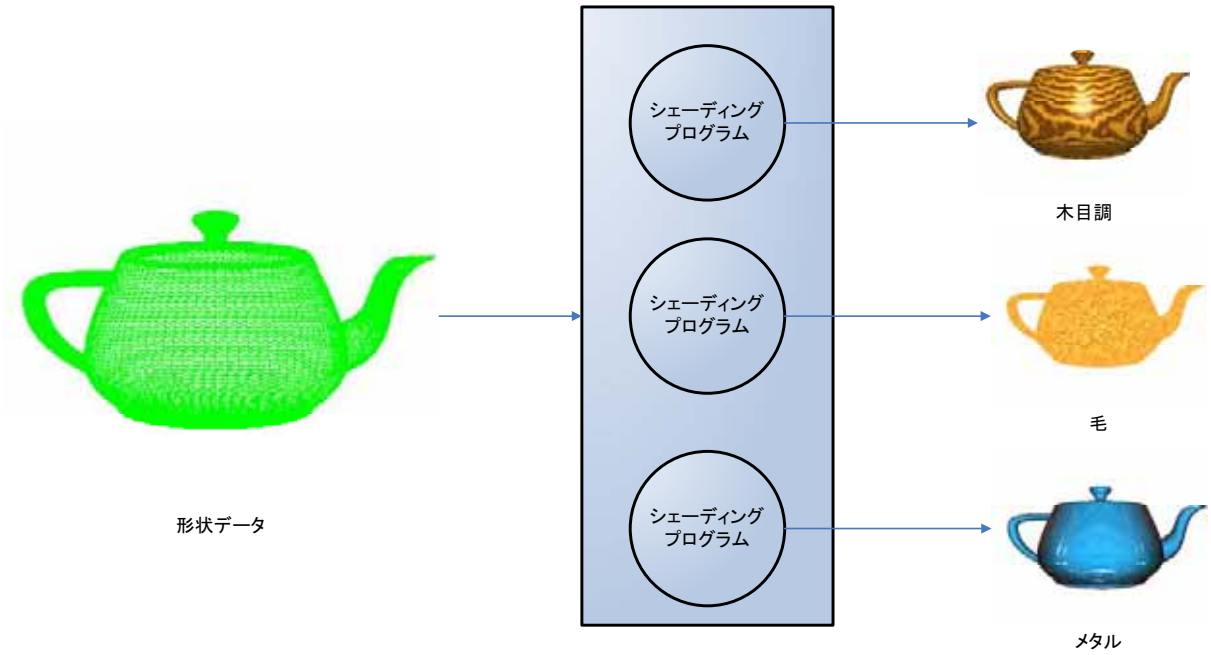


図 1.1 シェーディングプログラム適用のモデル

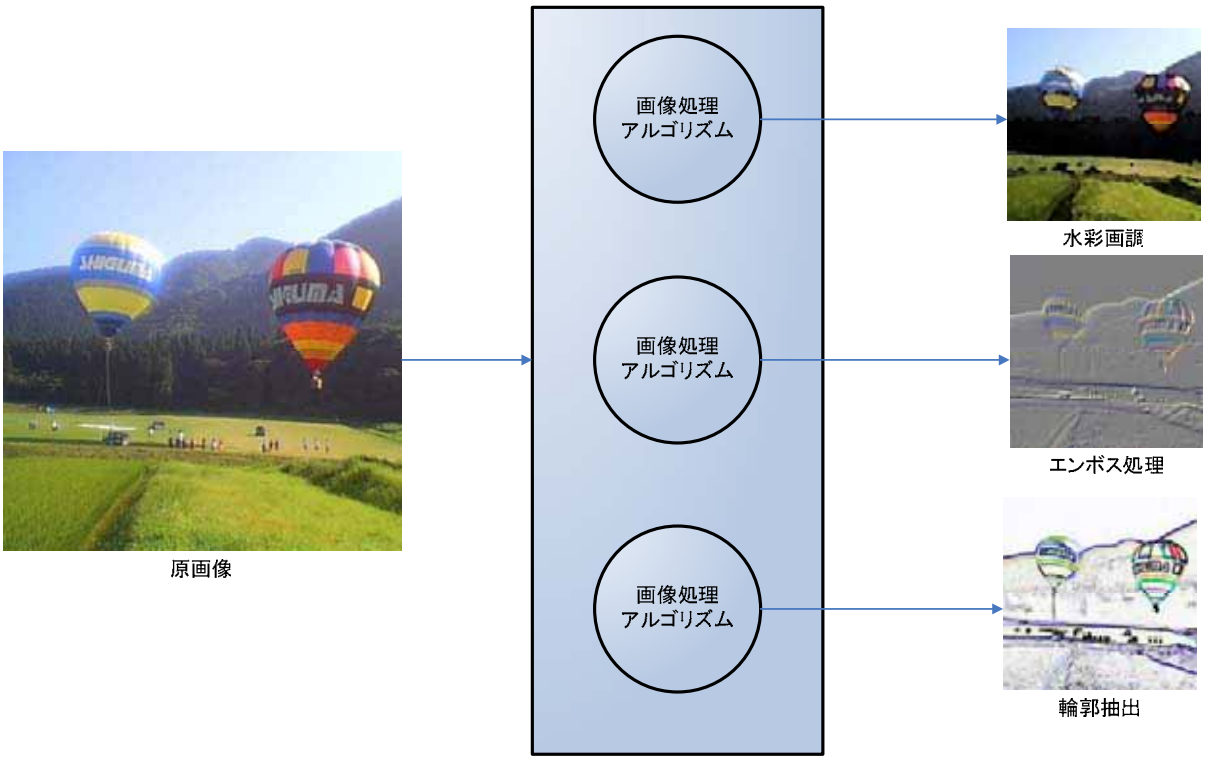


図 1.2 画像処理プログラムの流れ

1.2　GPU の利用

一般にリアルタイム 3DCG の処理では，ベクトルや行列の演算が多用されている．具体的には，3 次元空間内のオブジェクトを 2 次元空間へ射影する処理や，オブジェクトの表面の質感表現，および 3 次元空間内における光のシミュレーションなどの処理である．そのため，現在の GPU では，3DCG のジオメトリ処理やピクセル処理を効率的に処理するために，ベクトル型のプロセッサで並列処理する設計になっている．この処理設計が，リアルタイム 3DCG のレンダリング手法に多様性を持たせるとともに品質向上に寄与している．

さらにプログラム機能の進化により，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．こうしたプログラマブルな GPU を用いた汎用計算の研究分野を GPGPU(General-Purpose computation Graphics Processing Unit の略称)とよぶ[1].

1.3　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としては，OpenVIDIA[2]が挙げられる．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である Cg との組み合わせで開発を行う．OpenVIDIA を用いることで，複数の GPU を装備したコンピュータ上での並列計算も可能であり，コンピュータビジョンの高速な処理環境を構築することができる．OpenVIDIA は，GPU によって画像の処理を高速化する点で目的は同じである．本は，GPU を使って画像処理を高速に行う点では OpenVIDIA と同様のスタンスを持つが，グラフィカルな環境で開発を支援するという点が異なる．

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．この機能は複数のアルゴリズムを組み合わせた処理の開発に便利な機能で，事前に記述したアルゴリズム同士の実行順序や結合などの処理を簡単に記述できる．Sh は，GPU 向けのシェーディングプログラミング言語で画像処理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．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．この

機能は複数のアルゴリズムを組み合わせた処理の開発に便利な機能で，事前に記述したアルゴリズム同士の実行順序や結合などの処理を簡単に記述できる．Sh は，GPU 向けのシェーディングプログラム言語で画像処理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．

一方，グラフィカルなユーザーインターフェイス(GUI)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[4]などがある．Quarts Composer では，GUI により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．また，Quarts Composer では，GPU を用いた処理を利用することも可能である．このソフトウェアでは簡単に画像処理を用いた映像を作成することが可能であるが，画像処理アルゴリズム自体の開発を行うためのツールではない．

1.4　本研究の位置づけと効果

本手法の優位点としては，既存の画像処理ライブラリやコンピュータビジョンライブラリを用いた実装法とは異なり，ハードウェアで処理されるアルゴリズムの組み合わせをプログラムコード上で行わず，ビジュアルな環境を用いて行うことが出来る点が挙げられる．さらに既存の GUI の画像処理ソフトウェアと比較して，画像処理アルゴリズムだけではなく画像認識アルゴリズムにも適用可能な点も挙げられる．以下に，３つの大きな特長を列挙する．

(1) ハードウェアの処理を容易に使える

本研究で提案する開発支援システムは，画像処理アルゴリズムをハードウェア上で実行するため高速処理が可能である．従来，画像処理アルゴリズムの計算速度の向上のためには，高度なプログラミングスキルを必要とされていた．ここで，この高度なプログラミングスキルとは，高速化のための CPU 特有のアセンブラプログラミングやビデオを入力に使うためのファイル処理，およびキャプチャデバイスのプログラミングをさす．本システムでは，これらの実装上の作業負荷を軽減し，簡単に GPU を用いた並列計算処理を実行できるようにする．

さらに後述するアルゴリズムの部品化により，事前に必要な画像処理アルゴリズムが部品化されていれば，それを組み合わせるだけで，実行したい処理をプログラミン

グの作業を伴わず実装できる.

(2) 画像処理アルゴリズムの部品化

本システムでは，画像処理アルゴリズムの記述にシェーディングプログラミング言語を用いている．シェーディングプログラミング言語は，3DCG を用いたアプリケーションを構築する際の部品として，実行ファイルのコードとは別に記述される．実行ファイルと別に記述されることから，アルゴリズムの調整や変更が容易になるため，実行ファイルの開発者とアルゴリズムの開発者の間で分業がしやすくなる．以上の特徴は，画像処理アルゴリズムを組み合わせたアプリケーションの開発でも応用できると考えた.

(3) ビジュアルな環境における処理手順の可視化

本システムでは，部品化したアルゴリズムの組み合わせをビジュアルな環境で行い，処理手順を可視化する.

一般に，画像処理を用いたプログラムで使われるアルゴリズムは，すでに確立された手法の組み合わせで実現されることが多い．画像処理アルゴリズムが部品化されていれば，プログラムコード自体を書き換えずに開発が可能であると考える．したがって，ビジュアルな環境下で画像処理アルゴリズムの処理手順を編集できれば，従来のプログラミングスタイルと比較して，開発効率が向上すると考える.

第２章　GPU を用いた画像処理

本章では，本システムが開発支援する GPU を用いた画像処理アルゴリズムに対してどのようなものが有効であるかを述べるとともに，実際に GPU を用いた実時間での画像処理を用いたシステムの実装例を紹介する．

2.1 GPU に適した画像処理アルゴリズム

GPU に適したアルゴリズムの特長には，以下の 3 点が挙げられる．

- 並列処理
- 算術合成
- 拡大・縮小

これらは，GPU に備わっている 3DCG のレンダリング機能の特徴である．本研究ではこれらの機能を画像処理アルゴリズムに応用することで，処理の高速化を試みた．

2.1.1　並列処理

GPU には並列処理のための 2 種類の仕組みがある．1 つは，4 次元ベクトルの算術演算がそのまま処理できることである．画像のピクセルごとに赤，緑，青と透明度の４成分を持つため，4 次元ベクトルの算術演算が出来ることは画像処理にとって都合がよい．

もう 1 つは，シェーダプログラムを実行する GPU 上のパイプラインが並列化されている点である．これは，同じ処理を複数同時に実行することができるという利点がある．

GPU には，3 次元の物体の質感を表現するために，任意の画像を貼り付ける機能がある．すなわち，画像（テクスチャ）を物体上にマッピングする処理である．テクスチャマッピングと呼ばれるこの処理は，GPU 内で並列処理される．画像処理の GPU への実装は，平面の矩形へのテクスチャマッピング機能を応用する．

並列性の高いアルゴリズムは，テクスチャマッピング機能を用いて GPU 上に実装可能である．ここで，並列性の高い画像処理アルゴリズムとは，すべてのピクセルにおいて同一の処理をかけるアルゴリズムをさす．具体的には，フィルタ処理や色空間変換，および 2 値化処理があげられる．その例を図 2.1 から図 2.3 までに図示した．

図 2.1 は平均化フィルタの例で，各ピクセルに対してその周囲 8 ピクセルを含めた色の平均値を書き込む. 図中の(a)の原画像に, (c)の画像フィルタを適用することで(b)が出力される．これにより画像全体でぼやけた感じになるが，輪郭線のジャギーを目立たなくする利点がある．

図 2.2 は，ラプラシアンフィルタの例である．ラプラシアンフィルタは，各ピクセルに対して周囲の複数のピクセルとの２次微分を行う．図 2.2 では，図中の(a)の原画像に対して，周囲４近傍のピクセルに対する画像フィルタが(d)，周囲８近傍のピクセルに対する画像フィルタが(e)で，それぞれ結果が，(b)，(c)となる．この処理は主に，画像の先鋭化や輪郭線検出などに用いられる．

図 2.3 は，エンボス処理の例である．この処理は，各ピクセルの水平方向，および垂直方向に隣接する列同士を微分する．図 2.3 中の原画像(a)に対して，(d)の画像フィルタを用いて水平方向に微分したものが，(b)の結果画像である．垂直方向は，(e)の画像フィルタで微分して(c)の結果が得られる．この処理は，主に画像に対して凹凸情報を付加する特殊効果の目的で用いられる．

(a)

(b)

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

(c)

図 2.1 平滑化フィルタ処理の例.

(a) 原画像　(b) 処理後　(c) 平滑化フィルタ.

(a)

(b)

(c)

0	1	0
1	−4	1
0	1	0

(d)

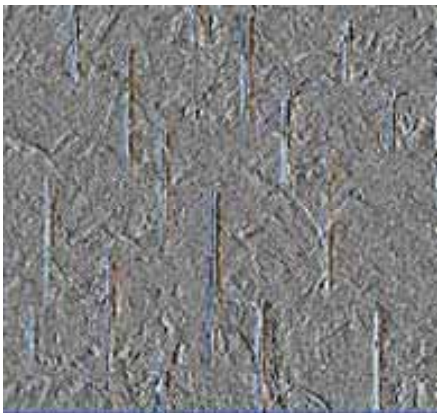
1	1	1
1	−8	1
1	1	1

(e)

図 2.2 ラプラシアンフィルタ処理の例.

(a) 原画像 (b) (d)の画像フィルタによる結果画像 (c) (e)の画像フィルタによる結果画像 (d) 4 近傍ラプラシアンフィルタ (e) 8 近傍ラプラシアンフィルタ.

(a)



(b)

(c)

−1	0	1
−1	0	1
−1	0	1

(d)

1	1	1
0	0	0
−1	−1	−1

(e)

図 2.3 エンボス処理の例

(a)原画像 (b) 水平方向のエンボス処理を施した結果画像 (c) 垂直方向のエンボス処理を施した結果画像 (d) 水平方向のエンボスフィルタ (e) 垂直方向のエンボスフィルタ.

2.1.2　算術合成

画像の算術合成とは，複数の画像の画素同士を加算，減算，乗算，除算することである．すなわち，式 2.1 のように四則演算を画素単位で行う．

加算：$(As, Rs, Gs, Bs) = (A_{p1} + A_{p2}, R_{p1}+R_{p2}, G_{p1} + G_{p2}, B_{p1} + B_{p2})$

減算：$(As, Rs, Gs, Bs) = (A_{p1} - A_{p2}, R_{p1} - R_{p2}, G_{p1} - G_{p2}, B_{p1} - B_{p2})$

乗算：$(As, Rs, Gs, Bs) = (A_{p1} \times A_{p2}, R_{p1} \times R_{p2}, G_{p1} \times G_{p2}, B_{p1} \times B_{p2})$

除算：$(As, Rs, Gs, Bs) = (A_{p1} / A_{p2}, R_{p1} / R_{p2}, G_{p1} / G_{p2}, B_{p1} / B_{p2})$

… (2.1)

ここで，画像の各ピクセルは，（A：透明度，R：赤成分，G：緑成分　B：青成分）の４つの要素で構成されるものとする．また，出力するピクセルを(As, Rs, Gs, Bs)，入力画像 1 および入力画像２のピクセルをそれぞれ，$(A_{p1}, R_{p1}, G_{p1}, B_{p1})$，$(A_{p2}, R_{p2}, G_{p2}, B_{p2})$で表すものとする．

GPU のマルチテクスチャリング機能を用いることで，以上の算術合成処理が GPU 上で実装可能となる．マルチテクスチャリングとは，テクスチャマッピング処理において複数の画像をテクスチャとして利用する機能である．例えば，自動車のボディを表現する場合，表面の質感を決める基本テクスチャ画像とロゴ画像の２枚の画像を用いて合成するという場面で利用される．

算術合成を用いた画像処理の例を図 2.4 と図 2.5 で示す．図 2.4 は，減算合成の例である．入力画像から背景画像を減算処理することで，右の差分画像を出力する．これにより，2 枚の画像の相違点が分かる．図 2.5 は，乗算合成の例である．原画像に対してマスク画像を乗算することで，右の出力画像を得る．

図 2.4　減算合成の例.

図 2.5 乗算合成の例.

2.1.3　拡大・縮小

GPU では，テクスチャの拡大や縮小の操作であるスケーリング処理をハードウェアで処理する機能が備わっている．これは，テクスチャを貼り付ける物体の画面上での大きさの変化に対応するためである．

この機能を用いて，画像内の任意の色集合に対する重心の座標位置を検出する手法が提案されている[5]．ピクセルシェーダプログラムの 1 回の処理では，検出対象の画像のピクセルすべてを走査することができないため，領域分割をして段階的に絞込みを行う．図 2.6 に，画像の縮小機能を用いた重心抽出の例を示す．

まず，図 2.6①に示すように，画像を 4 ピクセルごとのグループに分割して，グループ内における検出対象の色を持つピクセルの座標値の重心を算出する．算出結果は，解像度が縦横半分の画像に新たに書き込まれる．すなわち，①の 4 ピクセルの各グループの計算結果を，②の同色の画素位置に対応させて書き込む．②に値が書き込まれ

たあとは，③のように近傍４ピクセル領域分割を再度行い，同様の処理を施して，④の結果を得る．最終的に⑥に示すように１ピクセルの画像に収束した段階で，色集合の重心座標は画像データに出力される．

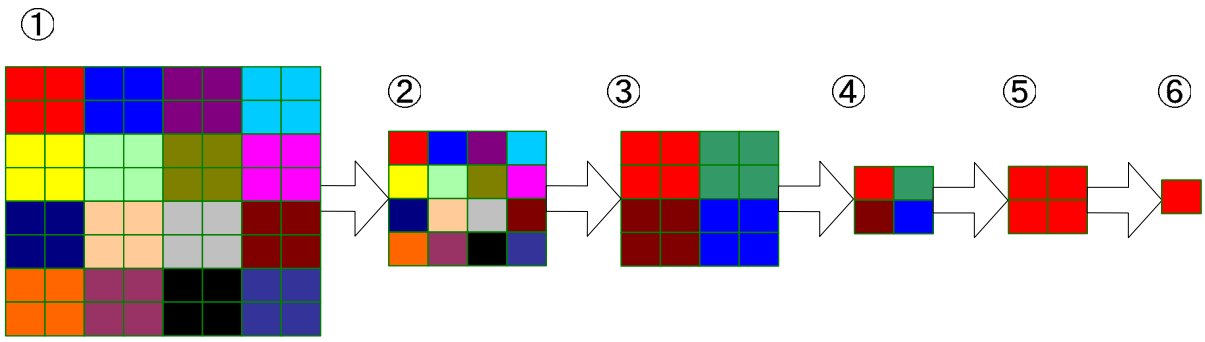


図 2.6 縮小を用いた重心抽出例.

2.2 適用事例

画像処理アルゴリズムを，実時間動作が目的のアプリケーションに実装する際の問題点には，計算負荷が大きいことが挙げられる．一般にグラフィックスを表示するアプリケーションにおいて，実時間動作となる計算速度は，ディスプレイのリフレッシュレートへの同期（計算回数 60 回/秒以上）を基準にしている．コンピュータビジョンや動画像処理アプリケーションにおいては，入力データ（カメラからの入力画像や動画ファイル）を遅延無く処理できるかどうかが基準になる．動画像の処理では，連続的に静止画像を取り出す処理の計算負荷が高いため，CPU による実装では実時間処理の維持が困難と考えられる．

以降，GPU を用いた画像処理を実装したシステムとして ViewFrame[6]，LuminaStudio[7]，RoboGamer[8]の 3 例について述べる．

2.2.1 ViewFrame

ViewFrame は，顔追跡をマーカレスで行い，その結果を用いて任意の視点から見た映像を提示するシステムである．ViewFrame は，図 2.7 に示すように IEEE1394

接続の DV カメラ 1 台と PC，赤外線測距センサ，そしてディスプレイにより構成されている．DV カメラを体験者に正対したディスプレイの上部に設置し，赤外線測距センサとの組み合わせでユーザ顔部の位置検出を行う．そして，検知された顔の位置に応じた画像を実時間で生成し表示する．

画像処理による物体の追跡では，カメラで取得した映像内の物体抽出処理の計算負荷が高いため，抽出がしやすいような特徴的な色をマーカとして用いるケースが多い．一方，ViewFrame では，利用者にマーカを装着させずに，実時間で動作する追尾の仕組みを提案した．図 2.8 に，システムの利用状態を示す．

このシステムでは，２つの画像処理アルゴリズムを実装した．まず，入力される画像のデータを RGB 表色系から CIE L*a*b 表色系へ変換する．CIE L*a*b 表色系は，類似した色の抽出がしやすいという特徴があるため，顔認識の画像フォーマットとして扱いやすい．しかし，色空間の変換アルゴリズムが複雑であるため，画像内の数十万の画素すべてに適用するには計算負荷が極めて高い．そのため CPU での実時間処理が困難であると考えられる．以上のことから，CPU で肌色抽出を行う場合には，CIE L*a*b 表色系よりも変換アルゴリズムが単純で計算負荷の軽い HSV 表色系を用いる場合がある．しかし，この手法の場合，室内環境で蛍光灯の光を肌色と判断することがあるため，精度面での信頼性が低くなる．

CIE L*a*b 表色系への色空間の変換後，画像から肌色の近似色とそうでない色を全ピクセルで判別して，その結果を元に画像の 2 値化処理を行う．最終的に，2 値化処理後の色領域の重心座標を，顔追跡のための値として取り出す．この処理結果の例を図 2.9 に示す．

ViewFrame で実装した肌色抽出法に対し，GPU による処理と CPU での処理のそれぞれの実行時間を比較した．検証方法は 1 秒間の処理回数を比較するという方法を採用した．実験には，Pentium 4 3.0GHz(HT テクノロジ対応)，RAM 1GB，RADEON 9800 XT 256MB を搭載した Windows PC を用いた．その結果，GPU による実装が毎秒 148 回，CPU での実装が毎秒 38 回と，処理速度に 4 倍近い差が出た．このことから，本手法の有用性が確認できた．

図 2.7 ViewFrame システム概要

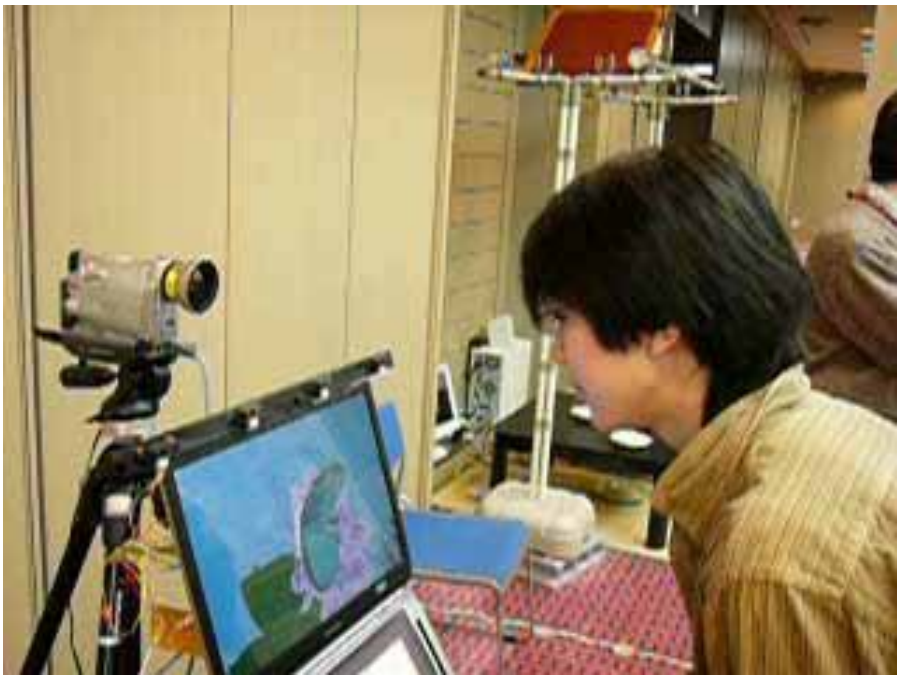


図 2.8 ViewFrame の利用風景

図 2.9 GPU での肌色領域抽出

2.2.2 LuminaStudio

LuminaStudio は，バーチャルスタジオのためのシステムである．バーチャルスタジオとは，テレビ番組などのセットを CG で製作し，人と合成して映像を作る技術である．映像の合成は，カメラで撮影した役者を背景から切り出し，CG で構築したセットに合成する．

LuminaStudio では，天気予報番組のような，役者と情報提示用の巨大スクリーンの合成を実時間で行うものを対象とする．図 2.10 にシステムの処理の流れを示す．図 2.10 の図中の①が実際のスタジオで，②はカメラが撮影した映像である．図 2.10 ①の“BLUE SCREEN”（ブルースクリーン）と書かれたエリアは，青の単一色のスクリーンである．このブルースクリーンの領域を④の視聴者向けの映像に置き換える．さらに②の撮影された映像の中のプロジェクタ投影面領域を，③の画像に置き換えるという流れになる．

バーチャルスタジオでは，一般にクロマキー処理と呼ばれる手法が用いられる．これは背景の壁を単色にして，背景の壁と同じ色の部分に映像の合成処理を行う手法である．クロマキー処理では，役者が背景と似た色の服を着ないなどの注意が必要であるが，容易に実現が可能である．しかし，この手法では，役者は合成対象の映像を見ながら説明や演技をすることができないという問題がある．

一方，巨大スクリーンを設置して，バーチャルセットをそのまま表示して，役者が映像を見ながら演技が出来るようにして撮影する方法もある．この場合，プロジェクタの輝度や環境光の反射などが問題となり，撮影した映像の情報提示領域が不鮮明になることが多い．そこで，LuminaStudio では投影面の領域の特定を行い，その部分に画像合成を行う方法を提案した．これにより役者が合成対象の映像を見ながら，かつ映像を鮮明に合成することが可能になった．

LuminaStudio では，情報提示領域の抽出には入力した RGB 表色系の画像を YCbCr 表色系に変換して，プロジェクタの光による明度とスクリーンに利用した素材の彩度を元に行う．この YCbCr 表色系は，Y が明度で，Cb が緑から青への色差で，Cr が緑から赤への色差である．この変換は，式 2.2 で与えられる．

$$
\begin{aligned}
Y &= 0.29900 * R + 0.58700 * G + 0.11400 * B \\
Cb &= -0.16874 * R - 0.33126 * G + 0.50000 * B + 128 \\
Cr &= 0.50000 * R - 0.41869 * G - 0.08131 * B + 128
\end{aligned}
\quad \cdots (2.2)
$$

ここで，R，G，B はそれぞれ変換前の画像の赤，緑，青色成分である．

プロジェクタを情報提示に用いた場合，その投影面は周囲よりも明度が高くなることから，Y 値を元に事前にプロジェクタ投影時の明度変化を取得して閾値処理を行う．ただし，プロジェクタの光量が強い場合には，投影面の周囲も明るくなる影響を受けるため，明度だけでの絞り込みは難しい．
そこで，Cb と Cr の情報を用いて，スクリーンに利用した素材の色差を元にした領域抽出も適用する．LuminaStudio では，投影に使うスクリーンを任意の単一色のシートにする．例えば，青いシートをスクリーンとして使う場合，その部分は青みが強くなるため，Cb が高い数値を出す領域を投影面の領域として抽出することができる．
図 2.11①は入力画像の明度情報 Y を表した画像であり，図 2.11②は入力画像の Cb を表した画像である．これを式 2.3 で与えられる画像の乗算合成を行った結果が図 2.11 の③である．

$$
(As, Rs, Gs, Bs) = (A_Y \times A_{Cb}, R_Y \times R_{Cb}, G_Y \times G_{Cb}, B_Y \times B_{Cb}) \cdots (2.3)
$$

ここで，出力するピクセルを(As, Rs, Gs, Bs)，明度情報を元に作成した画像を(A_Y, R_Y, G_Y, B_Y), Cb の情報に元に作成した画像を(A_{Cb}, R_{Cb}, G_{Cb}, B_{Cb})で表すものとする．

LuminaStudio では, IEEE 1394 接続のデジタルビデオカメラで撮影した映像に対して，320×240 の解像度の MPEG1 ビデオを合成するのに，PentiumeM 1.7GHz Mobility RADEON 9700 のノート PC で秒間 300 回以上の合成処理を行うことができた．利用したカメラの画像入力の速度は，秒間約 30 コマのため，LuminaStudio は生放送のように即時性が必要な場合にも利用できるものと考える．

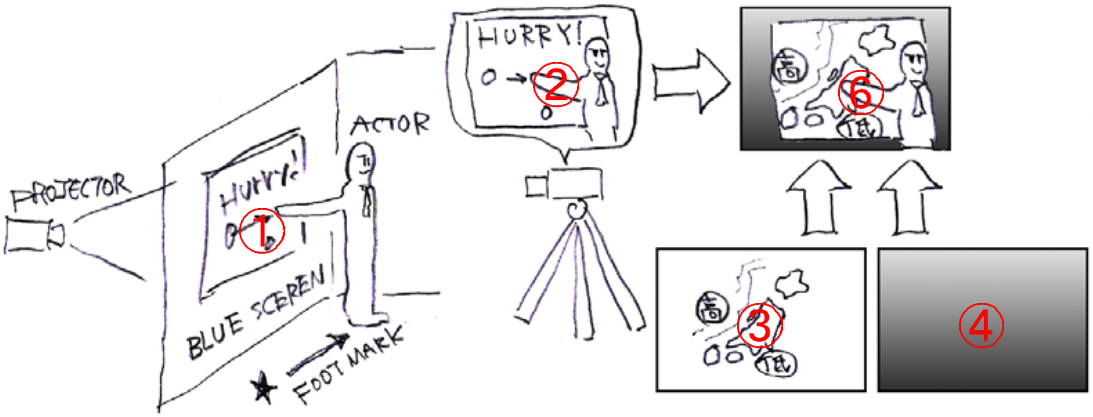


図 2.10 .LuminaStudio のシステムのイメージ

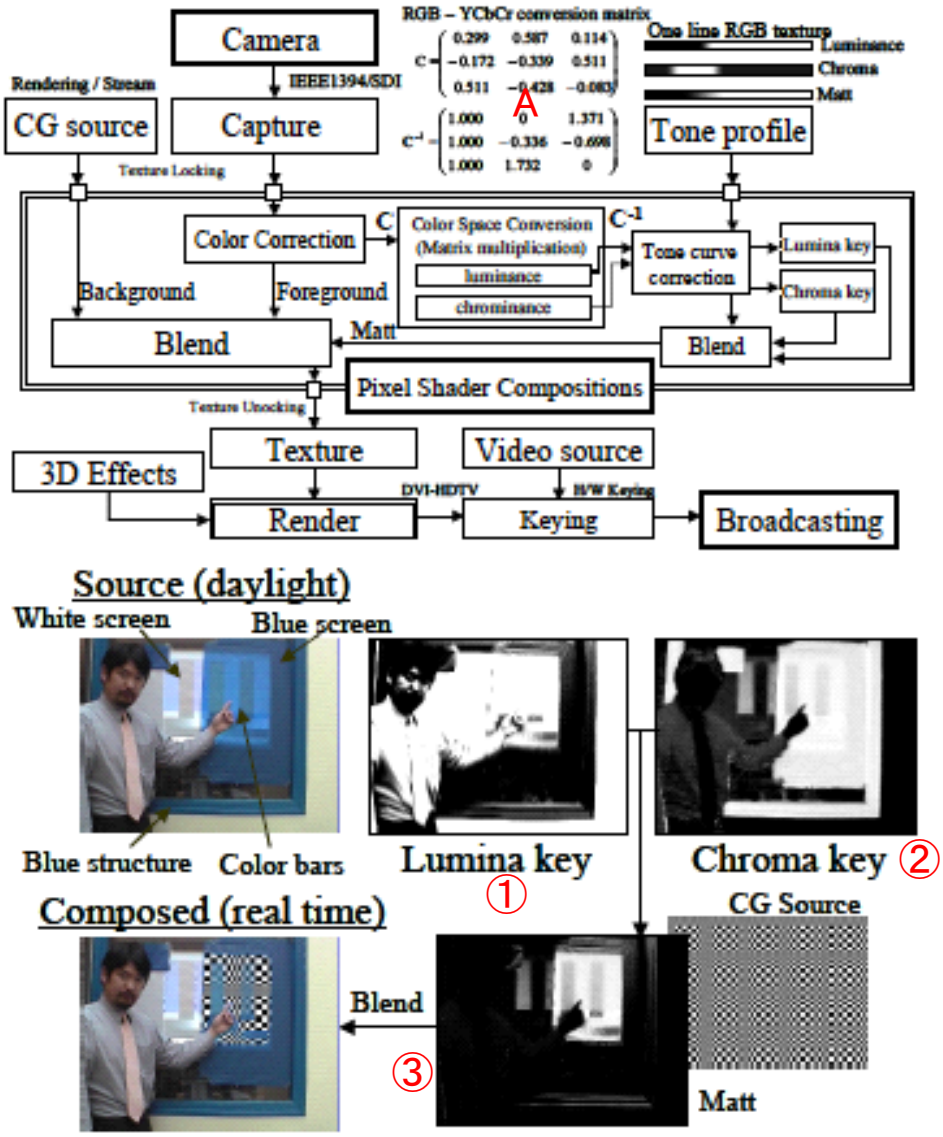


図 2.11 LuminaStudio の処理の流れ

2.2.3 RoboGamer

RoboGamer は，ビデオゲームをロボットが自動操縦するシステムである．このシステムでは，ビデオゲームの画像解析の結果に基づいて，SPIDAR[9]と呼ばれるハプティクスデバイスを用いて，ビデオゲームの入力装置の操作を行う．図 2.12 に，実際の設置の様子を示す．SPIDAR は，図 2.12 中の①のジョイスティックの先端に接続される．

RoboGamer での処理の流れは，プロジェクタに出力した画面をデジタルビデオカメラで撮影し，撮影された画像をコンピュータで即時に解析し，SPIDAR に制御信号を送る．

ビデオゲームの操作には，画面の変化に対して即時応答性が必要であるため，実時間での画像解析が必要である．RoboGamer では，GPU を用いることで入力に対して遅延のないシステムを実装した．

RoboGamer では，アタリ社の PONG™ [10]を対象に自動操縦システムを構築した．PONG は，画面上の左右に棒状の上下に動くパドルが配置され，左側をコンピュータのプレイヤー，右側を人間のプレイヤーが操作する．それぞれのプレイヤーが操作するパドルで，画面上に出てくるボールをお互いが打ち返す．ボールを自分のパドルの後ろへそらしたプレイヤーが負けとなり，相手に得点が入るという卓球に似たシステムのビデオゲームである．

図 2.13 に，実際に動作中の画面を示す．図 2.13①が入力画像で，②が画像処理後の画像である．図 2.13②の画像処理結果を用いてプレイヤーのパドルの位置とボールの位置を取得する．画面の下側には，SPIDAR に出している指示を Up や Down などのテキスト情報として出力するとともに，状況チェックのための各種パラメータを表示する．

PONG のゲームの状態は，SPIDAR が操作するプレイヤーのパドル位置と弾き返すボールの位置を取得して判断する．パドルとボールの位置は，カメラから入力された画面から色認識をし，座標抽出を行って検出した．

図 2.14 にビデオゲームの解析画面を示す．入力画像に対して，パドル A の領域を a に示すように緑色((R,G,B)=（0, 255, 0）)で塗りかえる．B のボールは，b に示すように白色((R,G,B)=（0, 255, 0）)に塗りかえる．

パドルとボールの色認識に使う基準となる色は，設営時に事前に画面を撮影して取

得する．これは，PONG の画面を出力する装置（プロジェクタやモニタ）の発色の違いや，設営環境の環境光に対して対応するためのキャリブレーションの意味を持つ．

RoboGamer では，以上の様な手法で GPU に映像解析を行わせることで，Pentiume M 1.73GHz，RAM 1MB，GeForce Go 6200TC(128MB)のシステム構成のノート PC で秒間 150 回以上の画像解析が可能になった．これにより，画面遷移の早いビデオゲームに即時応答できるシステムを実現できた．

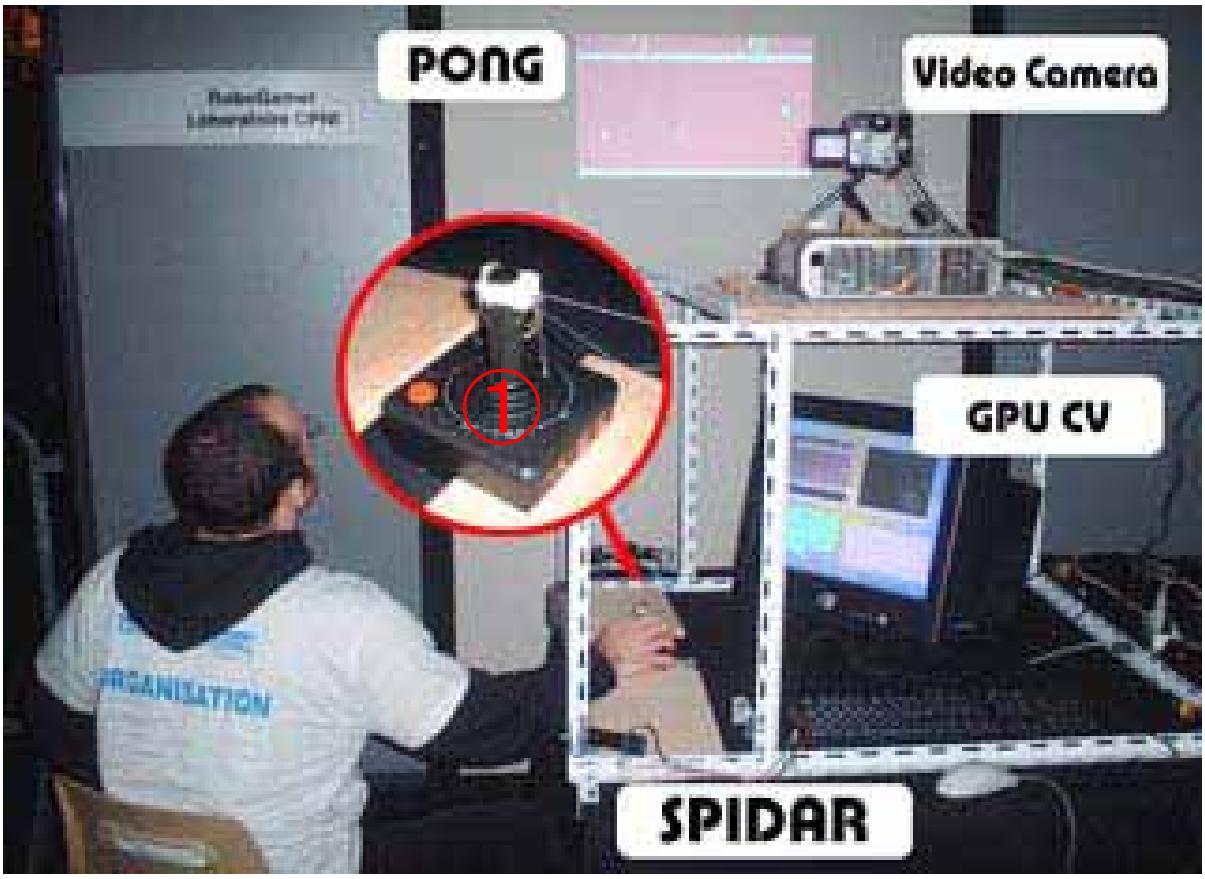


図 2.12 RoboGamer の設置風景

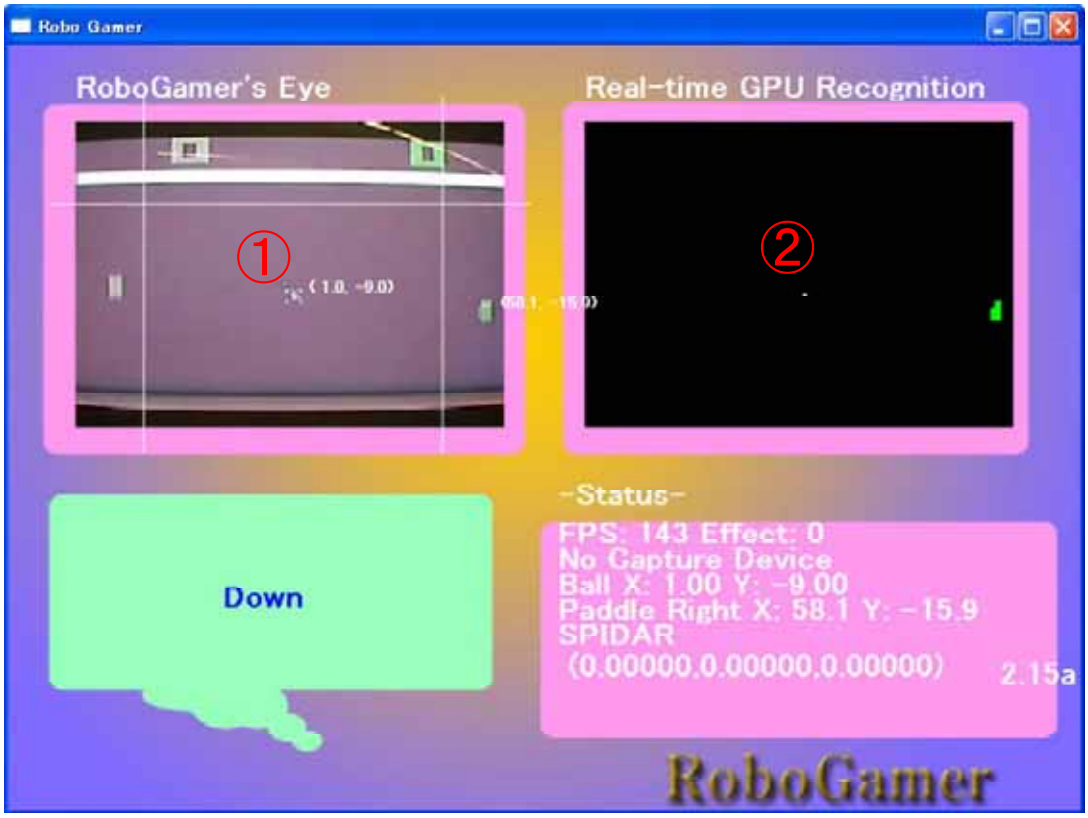
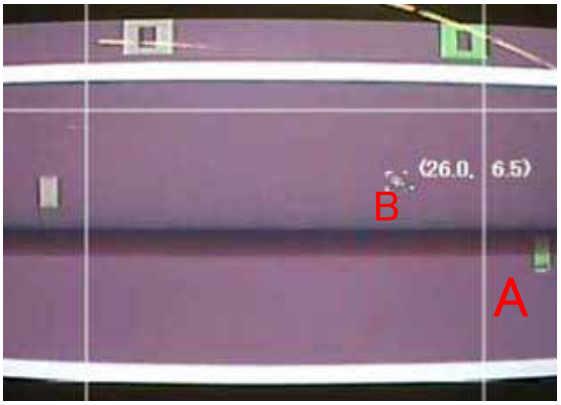
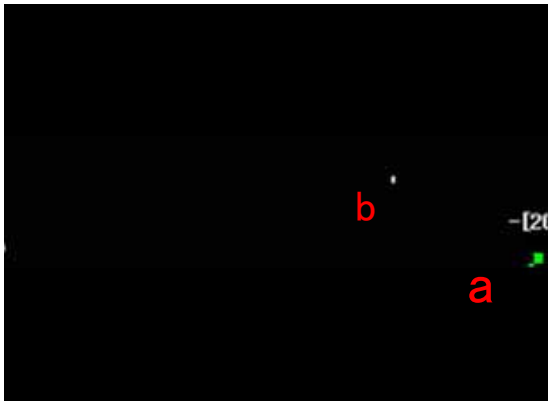


図 2.13 RoboGamer 画面



入力画像



解析画像

図 2.14 解析画面

第３章　システム設計

本章では，システムを設計する上で前提となる画像処理プログラミングの処理の流れを確認し，GPU 上で実装する方法について述べる．

3.1　画像処理アルゴリズムの GPU プログラミングへの組み込み

画像処理アルゴリズムを用いたアプリケーションでは，通常１つの入力画像に対して複数の画像処理アルゴリズムを順次適用する．

図 3.1 に ViewFrame での処理の例を示す，ここでは，入力画像を RGB 表色系から CIE L*a*b 表色系へと変換する処理をアルゴリズム 0，続いて，ある閾値で判定した肌色の近似色を持つ画素を白に，そうでないものを黒に塗る画像の二値化処理をアルゴリズム 1 とする．

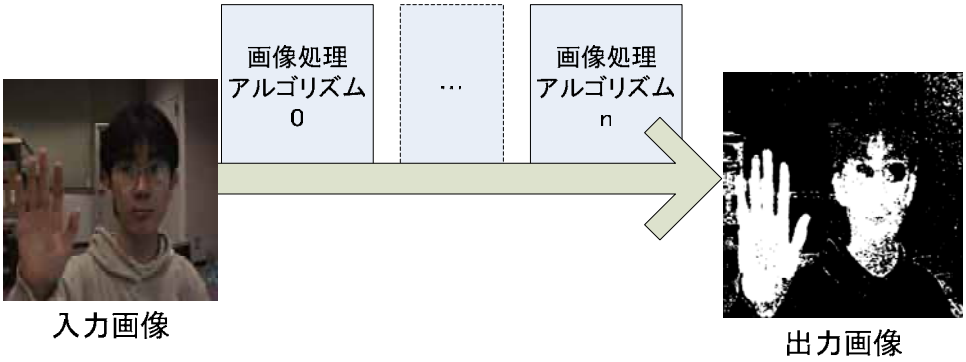


図 3.1 一般的な画像処理プログラミングの流れ

一方，GPU を利用するシェーディングプログラミングでは，3 次元頂点データの処理を行う頂点シェーダプログラムとピクセル処理を行うピクセルシェーダプログラムのセットを処理して，フレームバッファにオブジェクトをレンダリングする(図 3.2)．この２つのシェーダプログラミングのセットをパスと呼ぶ．

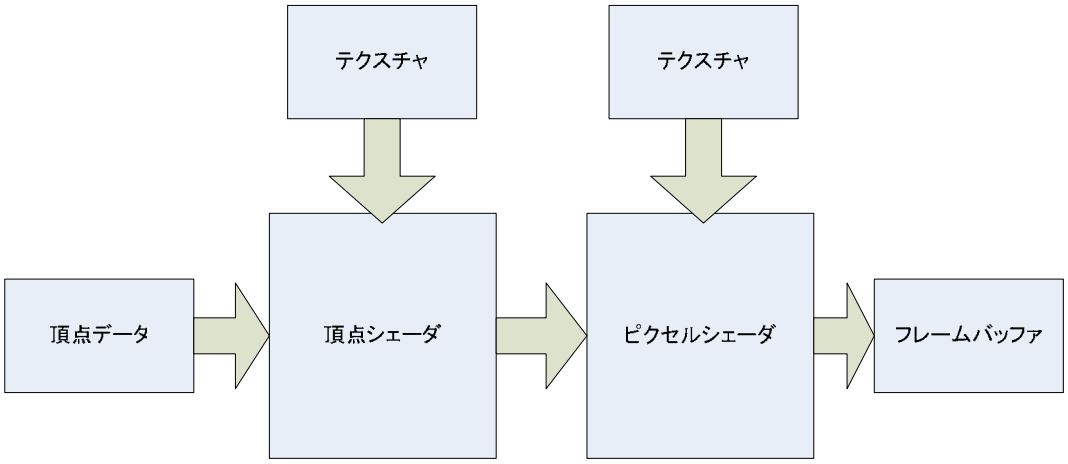


図 3.2 一般的な GPU プログラミング

出力先のフレームバッファは，入力データとしてのテクスチャバッファに置き換えることが出来ため，図 3.3 のようにパスの出力結果を別のパスに渡すことが出来る．この仕組みを用いることで，テクスチャバッファのデータを連続処理するサイクルが構築できる．本論文では，この画像処理アルゴリズムを実行するパスを画像処理モジュールと呼ぶことにする．

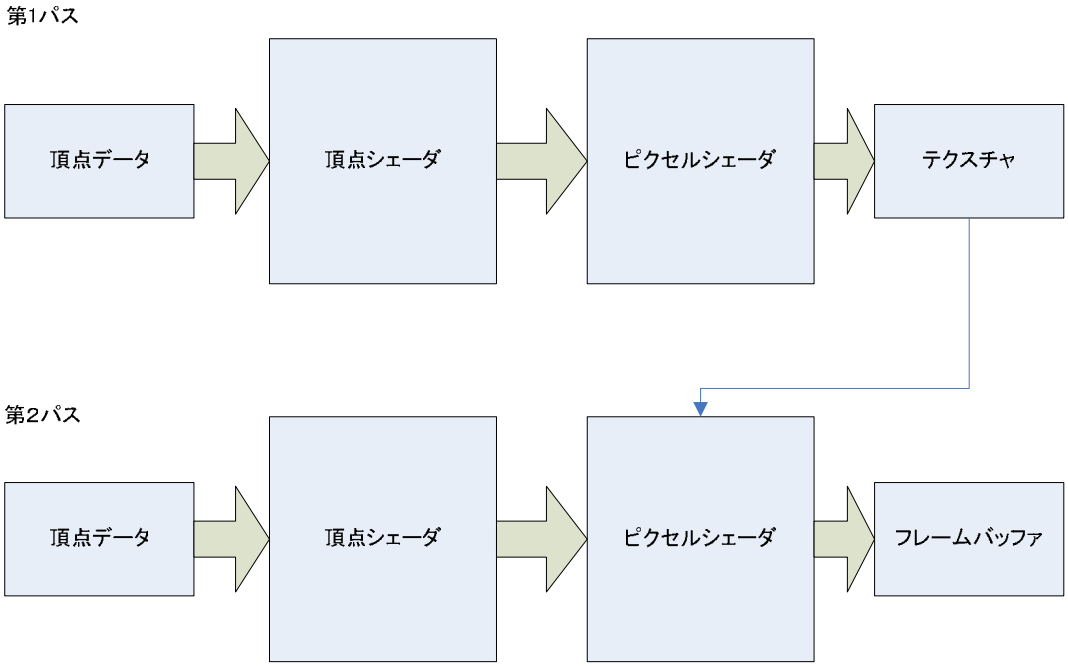


図 3.3 パス間でデータを受け渡すモデル

本研究で設計したシステムでは，カメラなどのキャプチャデバイスからの入力画像や，動画および静止画像ファイルをテクスチャバッファにコピーすることでピクセルシェーダプログラムに画像データを渡し，画像処理を行う．さらに複数の画像処理アルゴリズムを処理する場合には，ピクセルシェーダプログラムによる処理結果をテクスチャバッファにレンダリングすることで次のモジュールに渡す．その処理の流れを，図 3.4 に示す．

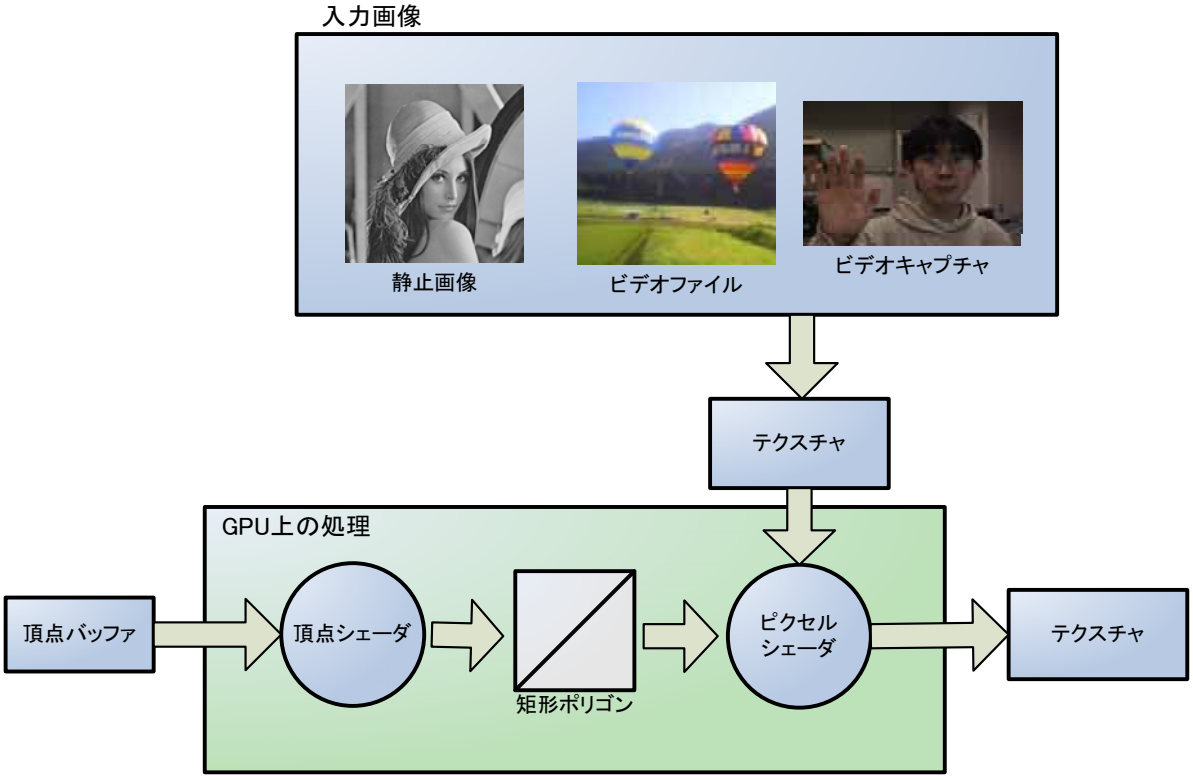


図 3.4 システムのデータフロー

3.2　システムの構成要素

本節では，システムを構築する上で必要な要素や具体的な方法について述べていく．システムを構成する重要な要素として，図 3.5 に示すような，(1)入力画像の取り込み，(2)画像処理を行う GPU プログラム，(3)画像処理アルゴリズムを組み合わせ，かつ，処理の流れを可視化するユーザーインターフェイスがあげられる．以降，各構成要素について述べる．

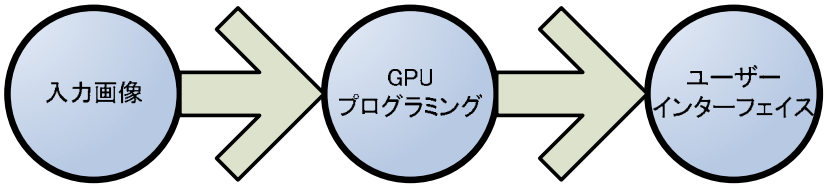


図 3.5　システムを構成する重要な３要素

3.2.1　入力画像の取り込み

本システムでは，動画および静止画像，ビデオキャプチャデバイスからの入力画像を, 入力画像として利用できる. 入力画像は, すべてテクスチャバッファに展開され, GPU 上のピクセルシェーダプログラムから、画像データにアクセスできるようになる．

静止画像の取り込み方法は, OpenGL や Direct3D に代表される 3DCG API が標準的にサポートをする機能である．ビデオキャプチャと動画ファイルに関しては, 3DCG API がテクスチャとしての利用を想定していないことや, 標準のメディアストリームのための API が十分に整備さていないため，OS ごとに実装の方法が変わる．本システムでは, Microsoft Windows 上のマルチメディア API である DirectShow を用いて，動画ファイルやビデオキャプチャデータを Direct3D のテクスチャバッファにバインドする処理を行った．

3.2.2　GPU プログラミング言語

GPU プログラミングで用いられる高級言語には，NVIDIA 社の Cg(C for graphics)[11]や, Microsoft 社の HLSL(High Level Shading Language)[12], OpenGL

の GLSL(OpenGL Shading Language)[13]などの企業や業界団体が策定したものと，Stanford 大学の Brook[14]や Waterloo 大学の Sh のように大学が主導になって策定されたものとがある．前者のプログラミング言語は，GPU そのものや GPU の制御を行う API の仕様を決める決定権を持つため言語自体は CPU と GPU のプログラミング言語を別々な言語として認識して開発が進めれらている．一方，後者の大学発の GPU プログラミング言語は，前者の企業や業界団体と異なりハードウェアの設計自体が出来ないが，CPU 側のプログラミングで使えるソース上での利用を想定して開発が進められている．

本システムでは，以下の観点から HLSL を GPU プログラミング言語として採用した．

(1) 動画ファイルやキャプチャデバイスを制御する API と相性がよい

前節で述べたように，本システムでは，動画ファイルおよびビデオキャプチャデータをテクスチャバッファに書き出すのに，DirectShow を利用している．HLSL は，3DCG API のひとつである Direct3D の GPU プログラミング言語であるが，DirectShow は，Direct3D と相性がよいため，結果として HLSL との相性もよい．る．

(2) Cg と互換性があるため Cg での開発者でも問題なく使える

Cg は，NVIDIA 社が開発した GPU 向けプログラミング言語である．一方，HLSL は，Microsoft 社の Direct3D の GPU プログラミング言語であるが，NVIDIA 社の開発協力があるため，両者の言語間の仕様は非常に互換性の高い設計になっている．両者の違いは，コンパイラとランタイムが異なると言う点である．そのため言語自体の仕様に関しては，どちらか一方が使えることができれば，もう一方を扱うことは容易に出来る．

(3) 利用者が多い．互換性のある Cg の開発者を加えると，GPU プログラミング言語としては，最も利用者が多い．

前述のとおり，Cg と HLSL の２つの言語の互換性が高いことから，両方の言語が使える開発者にターゲットを広げることが出来る．

(4) エフェクトファイル形式と呼ばれるメタファイル形式が存在し，GPU プログラムのコード以外に実行ファイルの制御命令が記述できる

エフェクトファイル形式とは，HLSL のメタファイル形式である．エフェクトファイル形式は，HLSL の頂点シェーダとピクセルシェーダのコードの記述の他に，開発者が自由に定義できるパラメータを記述できる．このパラメータは，実行ファイルからの参照が可能で，アプリケーションの設定ファイルのように利用することが可能になる．

(5) エフェクトファイル形式を利用できるツールが多い

エフェクトファイル形式の記述ファイルは，３次元形状データを作成するモデリングソフトウェアやシェーダ開発ツールなどで編集ができ，相互にデータのやり取りが可能である．そのため，開発者はそれぞれが慣れ親しんだツールを使うことが出来る．

3.2.3　ユーザーインターフェイス

本システムで考慮するユーザーインターフェイスは，大きく２つの性質のものに分かれる．１つは画像処理の流れを構築するためのインターフェイスで，もう１つは処理の流れを可視化するインターフェイスである．

本システムでは，入力画像やアルゴリズムなどの処理のフェイズをブロックとして表示し，そのブロックをリンクさせることで全体の処理を構築する．

画像処理の流れは，図 3.6 に示すように，入力画像に対する処理結果を次のアルゴリズムの入力画像に使うストリーム処理である．操作方法として，各画像処理アルゴリズムを記述した GPU プログラムコードの実行順序を，ビジュアルに操作できるようにする．

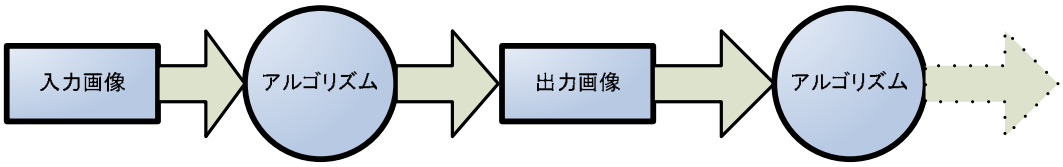


図 3.6　画像処理の流れ

図 3.6 を元に，画像処理を組み立てた場合の例を図 3.7 に図示した．図 3.7 は，背景差分と肌色認識の画像処理アルゴリズムを重ねた例である．

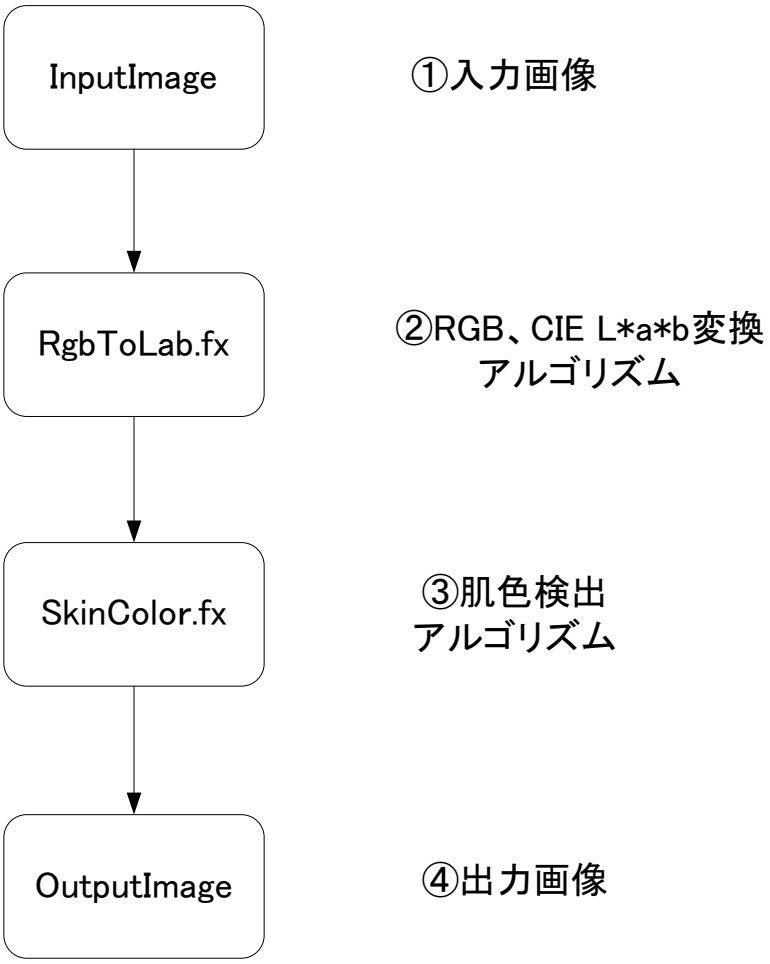


図 3.7 可視化の例

第 4 章　システムの実装

本章では，画像処理アルゴリズムを GPU プログラム上で実行および確認できるシステムの実装について述べる．

4.1　システム概要

システムは，図 4.1 に示すような，1 台の Windows PC（Windows XP / 2000）上に実装される. 実装する PC には, ピクセルシェーダ 2.0 以上の GPU プログラムが, 実行できる構成が必須である．ただし，頂点シェーダプログラムに関しては，GPU 上で動作しなくてもよい.

入力に使うことが出来るデータは，静止画像，動画ファイル，およびビデオキャプチャデータである.

システムの開発には，Visual Studio.NET 2003 環境下で，C++言語と DirectX を用いた．DirectX のうち，GPU の利用のために 3DCG のコンポーネントである Direct3D を，動画ファイルの読み込みやキャプチャデバイスの制御に DirectShow を利用した．なお，GPU プログラミング言語には，HLSL を用いる.

図 4.1 GPU を搭載した PC

(1) 静止画像ファイルフォーマット

システムで利用できる静止画像ファイルフォーマットは, BMP, JPG, PNG, TGA, TIFF，および DDS である.

(2) 動画ファイルフォーマット

システムで利用できる動画ファイルフォーマットは，AVI，MPEG1，MPEG2，WMV である．ただし，PC にあらかじめコーデックが入っていなければならない.

(3) ビデオキャプチャデバイス

ビデオキャプチャには，USB 接続のカメラや，IEEE 1394 接続のデジタルビデオカメラなどの動画像入力デバイスを用いる.

4.2　画面レイアウト

図 4.2 に開発したシステムの画面の例を示す．以下，各機能を説明する．

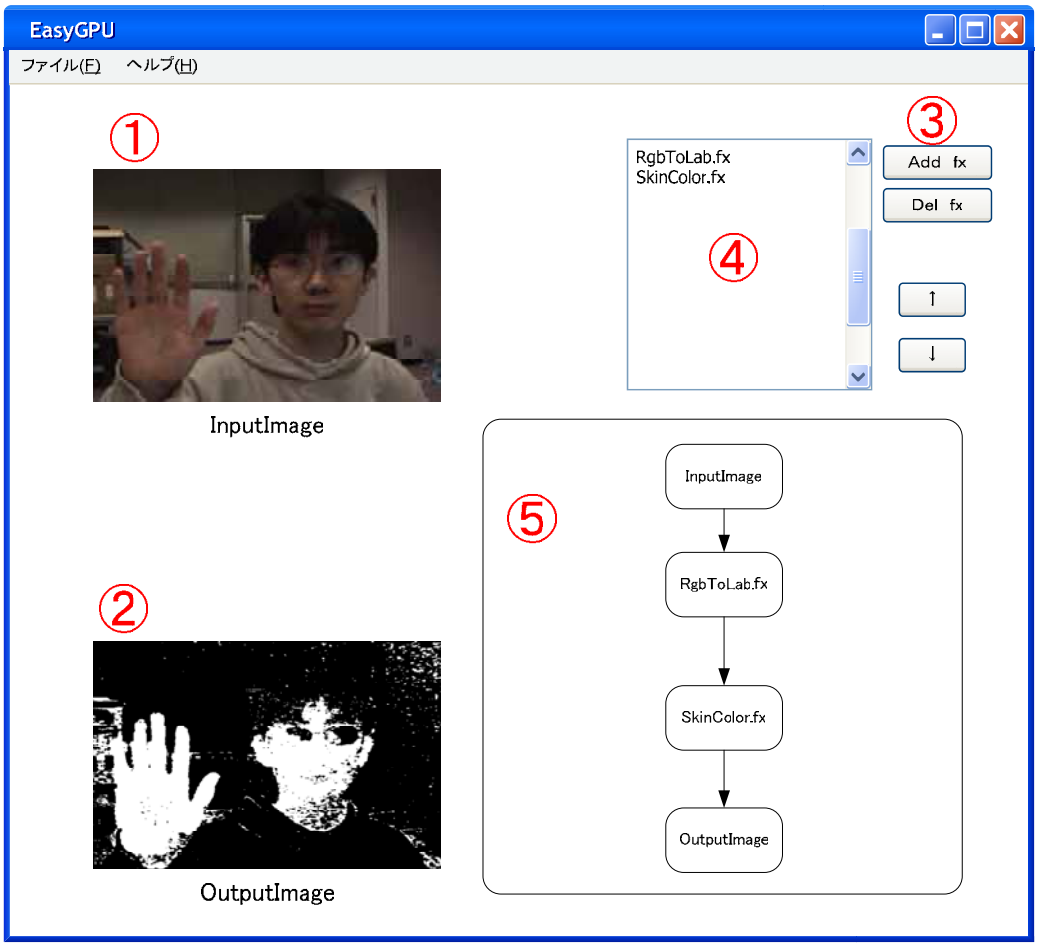


図 4.2 画面レイアウト

① <u>入力画像</u>

図 4.2①には，メニューで選択された入力に使うデータの種類に応じて，入力画像が表示される．入力画像が動画ファイルの場合には自動的にループ再生が行われ，キャプチャデバイスを利用した場合には撮影している画像がリアルタイムで更新される．

② 出力画像

図 4.2②には，画像処理後の出力画像が表示される．入力画像が動画ファイルの場合は，再生画像がリアルタイムに処理されて出力され，キャプチャデバイスを利用した場合には，リアルタイムに撮影画像が処理されて出力される．

③ ボタン

図 4.2③には，4 つのボタンが配置されている．【Add fx】ボタンを押すと，画像処理アルゴリズムを記述したエフェクトファイルを読み込み，ファイル名を図 4.2④の領域に表示する．【Del fx】ボタンは，④のエディットボックスで選択しているアルゴリズムを削除して，実行処理から外すことが出来る．

【↑】ボタンは，④のエディットボックスで選択したアルゴリズムの実行順序を 1 つ前に入れ替えるボタンである．【↓】ボタンは，実行順序を 1 つ後に入れ替える

④ エディットボックス

図 4.2④のエディットボックスでは，実行するアルゴリズムのプログラムコードを格納したファイルの一覧を表示する．アルゴリズムは，上から順番に実行される．

⑤ アルゴリズム実行順の可視化

図 4.2⑤には，アルゴリズムの実行順序がフロー図として表示される

4.3 ソフトウェア部の実装

本節では，本システムのソフトウェア上の実装について述べる．

4.3.1 画像データのテクスチャデータ化

4.1 節で述べた，静止画像ファイルや動画像，ビデオキャプチャデバイスからの入力画像は, データの種類に関わらず, 図 4.3 に示すようにテクスチャデータ化される．静止画像に関しては, 使用する Direct3D の API によってテクスチャデータ化される．動画像とビデオキャプチャデバイスからの入力画像に関しては，DirectShow による再生と同時に，ビデオメモリ上のテクスチャデータを格納するバッファに画像がコピ

ーされ，テクスチャデータ化される．

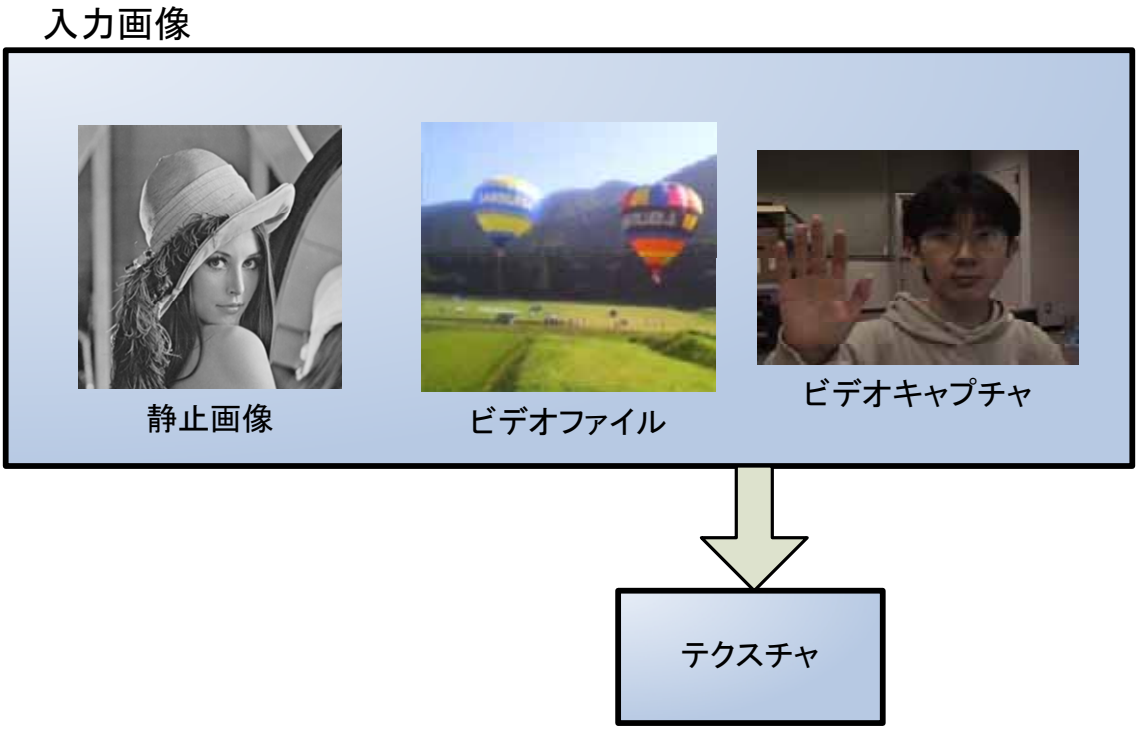


図 4.3　入力画像のテクスチャデータ化

4.3.2　画像処理モジュールの取り込み

画像処理モジュールは，DirectX エフェクトファイル形式に記述する．画像処理モジュールには，以下の記述が出来る．

① 画像処理アルゴリズム

画像処理モジュールは，DirectX エフェクトファイル形式で記述される．画像処理モジュールには，以下の各記述が可能である．

② アルゴリズムで使う補助画像の有無

マスク処理や背景差分処理のように入力画像のほかに画像を利用する場合，補助画像にどのファイルを使うかを指定する必要がある．エフェクトファイルでは，実行ファイル側が読み取りできる形で，任意のパラメータの埋め込みが可能であり，補助画像

として利用する画像ファイルは，そのパスを記述することで指定される．本システムでは，エフェクトファイルのデータ型であるテクスチャオブジェクト型の定義時に，画像ファイルのパスの記述がされていれば，指定された画像をテクスチャデータ化する仕組みを実装した．

③ GPU の組み込み機能のスイッチング

エフェクトファイルでは，GPU に組み込まれた機能のスイッチングが記述できる．組み込み機能には，レンダーステートとテクスチャステージステート，およびサンプラステージステートがある．レンダーステートとは，頂点処理およびピクセル処理のための設定である．具体的には，透過処理や深度バッファの取り扱いなどである．テクスチャステージステートでは，マルチテクスチャ利用時の合成に関しての設定を扱う．サンプラステージステートでは，テクスチャのスケーリング時の画像補間処理などを扱う．

4.3.3 手順の可視化

画像処理アルゴリズムの手順の可視化では，入力画像から出力画像まで，画像処理モジュールの実行順序を図示する．今回のシステムの仕様では，画像処理で用いる入力画像を複数にしたり，出力結果が複数に分岐したりはしないため，直線的な有向グラフのような形で，手順が可視化される．

第 5 章　結論

本章では，システムを開発しての考察と，本研究の課題や将来展望について述べる．

5.1　考察

本システムには，大きく分けて以下の２つの成果がある．

(1)　高速化を容易に行える

本システムでは，GPU 用のプログラミング言語を利用することで，マルチスレッド化やアセンブラ言語と比較して，画像処理アルゴリズムの高速化を容易に行える．

(2)　開発効率の向上

ビジュアルな開発環境を用いることで，プログラムコードを記述する作業よりも開発の時間が短縮できた．また，複数のアルゴリズムを組み合わせた処理手順を可視化することで，開発者が作業のつながりをイメージしやすくなった．さらに，コード化された画像処理手法を部品として扱うことで，コーディング作業を減らすことが出来た．

本研究を通して，シェーダ開発手法が画像処理アルゴリズムの開発に適用できることが明らかになった．従来の 3DCG に関するプログラム開発では，アプリケーションのコードとシェーダのコードが同じソースツリー上で作業される．したがって，どちらかのソースを更新した場合，両方のコードをビルドする手間が発生する．しかし，現在では，アプリケーションのコードとシェーダのコードを分離して開発するため，片方の作業の更新で済む．したがって，3DCG アプリケーションの開発において作業の分業作業をしやすくなる．

この開発の効率化は，画像処理プログラム開発でも同様である．本来，動画像の入力プログラムを開発するスキルと画像処理アルゴリズムを開発するスキルは別のものであるが，両者のコードが混在した環境では，異なるプログラムの知識を必要とされる．本システムを利用することで，画像処理のプログラマが画像処理の部分だけに

専念することが可能になる．

5.2　今後の課題と将来展望

本研究では，ビジュアルな開発環境による画像処理アルゴリズムの開発支援ができた．しかし，開発したアルゴリズムのアプリケーションへの組み込み作業は，開発者自身の手でおこなわなければならない．したがって，開発者の負担を下げるために，開発支援ツールで作成した結果を，アプリケーションに簡単に取り込むためのランタイムライブラリの整備が必要であると考える．

ユーザーインターフェイスに関しては，現段階ではシンプルで扱いやすいが，モジュールごとのパラメータ変更などが不可能であるという問題点がある．さらに複数の入力ソースや，処理した画像を別の画像処理に対する補助画像として利用できない点なども，現在の課題である．

したがって，アプリケーションに組み込むためのランタイムライブラリの整備とユーザーインターフェイスの洗練化が課題になると考える．

今後は，画像処理アルゴリズムの開発環境だけでなく，アプリケーションへの組み込みを考慮した総合的な開発支援環境を目指したい．

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] GPU による重心抽出, http://tpot.jpn.ph/t-pot/program/113_Center/index.html

[6] 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

[7] 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.

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

[9] SPIDAR, http://www.cyverse.co.jp/jp/Products/3dGrip/

[10] PONG, http://www.pong-story.com/

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

[12] HLSL, http://www.microsoft.com/japan/msdn/directx/

[13] GLSL, http://www.opengl.org

[14] Brook, http://graphics.stanford.edu/projects/brookgpu/

謝辞

本研究は，さまざまな方々のご協力により成り立っています．まず，指導教官である，本学知識科学教育研究センター宮田一乗教授に心より感謝いたします．研究に対して大変親身にご指導いただき，また，研究環境の整備をしていただきました．さらに，早くから研究発表や学会参加の機会を与えてくださったことはたいへん刺激になりました．

論文審査員である，國藤教授，西本助教授，金井助教授には，中間審査の段階からさまざまなご助言をいただき，感謝いたします．

そして研究の過程，ゼミにおいて，貴重な意見やアドバイスをくださった，宮田研究室の皆さんに感謝いたします．公私共々おせわになりました．

Appendix　本研究に関する研究発表

[1] H.Yabu,Y.Kamada,M.Takahashi,Y.Kawarazuka,K.Miyata, “Ton2: A VR Application With Novel Interaction Method Using Displacement Data”, ACM SIGGRAPH E-Tech, 2005

[2] A.Shirai, M.Takahashi, K.Miyata, M.Sato, S.Richir, “Development of Robotic TV Game Player Using Haptic Interface and GPU Image Recognition”, ACM SIGGRAPH, Poster #126, 2005

[3] 薮,鎌田,高橋,河原塚,宮田, ”変位情報を用いた VR アプリケーションの実装 –バーチャル紙相撲“トントン””, 芸術科学会論文誌、Vol.4, No.2, pp.36-46 (2005)

[4] 宮田一乗,高橋誠史,黒田篤, “GPU コンピューティングの動向と将来像”, 芸術科学会論文誌、Vol.4, No.1, pp.13-19 (2005)

[5] A.Shirai, M.Takahashi, K.Miyata, M.Sato, S.Richir, ”RoboGamer?: Development of Robotic TV Game Player using Haptic Interface and GPU Image Recognition”, Proceedings of SIGCHI International Conference on Advances in Computer Entertainment Technology, pp.471-472, 2005

[6] 薮,鎌田,高橋,河原塚,宮田, ”変位情報を用いた新たなインタラクション手法の提案-VR アプリケーションへの応用例”, インタラクション 2005 論文集, pp.93-94, 2005

[7] M. Takahashi, K. Miyata, “GPU based interactive displacement mapping”, Proc. of IWAIT2005, pp.477-480, 2005

[8] 薮,鎌田,高橋,河原塚,宮田, “ジャンピングインタラクションを用いた VR アプリケーション”, NICOGRAPH2004 秋季大会論文集, pp.101-106 (2004)

[9] 高橋誠史, 河原塚有希彦, 桑村宏幸, 宮田一乗, “UoQA –ジェスチャ認識と簡易なモーションベースを用いたVRアプリケーション“, 芸術科学会論文誌, Vol.3, No.3, pp.200-204 (2004)

[10] 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, pp.189-192 (2004).

[11] 高橋誠史, 河原塚有希彦, 宮田一乗, “GPU による肌色認識処理の高速化に関する一手法”, 第 3 回 NICOGRAPH 春季大会, pp.55-56 (2004)

[12] 高橋誠史, 河原塚有希彦, 桑村宏幸, 宮田一乗, “UoQ –ジェスチャ認識を用いた映像体験環境-“, 芸術科学会論文誌、Vol.2, No.4, pp.123-127 (2003)

謝辞

本研究は，さまざまな方々のご協力により成り立っています．まず，指導教官である，本学知識科学教育研究センター宮田一乗教授に心より感謝いたします．研究に対して大変親身にご指導いただき，また，研究環境の整備をしていただきました．さらに，早くから研究発表や学会参加の機会を与えてくださったことはたいへん刺激になりました．

論文審査員である，國藤教授，西本助教授，金井助教授には，中間審査の段階からさまざまなご助言をいただき，感謝いたします．

そして研究の過程，ゼミにおいて，貴重な意見やアドバイスをくださった，宮田研究室の皆さんに感謝いたします．公私共々おせわになりました．