I-043

# 動画像処理のパラメタ決定支援ツールとその高速化の検討

## Decision Support Tool for Parameter Tuning in Video Processing and Its Acceleration

豊住 耕一† 熊谷 一樹† 高橋時市郎†
Kouichi Toyozumi Kazuki Kumagai Tokiichiro Takahashi

## 1. まえがき

我々は動画像処理プログラム開発環境(以下本環境)を開発している．本環境は，Common Lisp を拡張した言語を使用している(以下本言語)．本言語は，少ない記述量で動画像処理モジュールを効率的に開発できる利点がある．画像処理に特化した最適化機能により，他の動的プログラミング言語を用いた画像処理フレームワークよりも，ユーザが実装した処理が高速に動作する[1]．

しかしながら，動画像処理は，用いるアルゴリズムやフィルタを適用する順番，適切なパラメタ等を論理的に定めることが難しい．このため，動画像処理は多くの試行錯誤を伴う．

本稿では，本環境に整備した，必要な操作量を抑えた動画像処理環境について報告する．具体的には，動画像処理に使用するパラメタの決定を支援し，試行錯誤の回数を削減するツールを開発した．同時に，GPU を用いた高速化機能も実装したので，併せて報告する．

## 2. 研究背景

動画像中の処理対象となる要素が多い場合，用いるアルゴリズムやフィルタを適用する順番，パラメタ等を試行錯誤して決定することが多い．この試行錯誤に費やす労力や処理時間は膨大なものになり，効率化が求められる．

三原らはこの問題を解決するために，大量にバリエーションを生成して利用者に選ばせるという手法を提案している[2]．しかしながら，三原らの手法は，ファイルに画像を書き出す構成となっているため，画像の入出力のオーバヘッドが大きく，対話性が十分に活かされているとは言い難い．

一方，我々が先に提案した開発環境では，Read-Eval Print Loop を拡張した，処理結果の画像を逐一表示する機能が備わっている．これにより，利用者は対話性を活かして動画像処理プログラムの開発を効率的に行うことができる．

## 3. 提案手法

### 3.1. パラメタの決定支援機能

動画像処理で最適な内部パラメタを決定する作業は試行錯誤を伴う時間のかかる作業である．我々は，この最適パラメタ決定作業を支援するために，動画像処理関数／フィルタのパラメタを自動的に変える機能(パラメタ決定支援関数)を実装した．パラメタ決定支援関数は，簡潔な記述でパラメタ設定を行うことができる特長がある．

同時に，処理結果を，パラメタ空間と対応するように，規則的に配置して画像として出力する関数も実装した．この関数は，静止画像を返す関数となっているため，動画像処理結果を一覧表示する時に，動画像処理関数／フィルタはそれぞれの処理結果をディスクに書きこんで保持しておく必要がない．

### 3.2. GPU による高速化

高速化のために，動画像処理を GPU 上で行うための機能を新たに追加した．Common Lisp をベースとする本言語で，直接 GPU 上のプログラムを記述することはできない．そのため，次のようにして GPU 上でのプログラミングを可能としている．

OpenCL を介してプログラムを実行するためにはプログラムをプログラミング言語 OpenCL C (以下 OpenCL C)で実装する必要がある．我々は，本言語のサブセットを定義し，それを OpenCL C に変換し，OpenCL プログラムをビルドすることで，OpenCL による GPU プログラミングを可能とした．

## 4. 使用例と評価

### 4.1. パラメタの決定支援

#### 4.1.1. フィルタによるチャネル可視化

本節と次節では,図 1 に示す画像の空の色をより濃くし，左下に位置する木をより青くする処理を例に．本言語の有用性を説明する．そのためには，先ず，空と木を分離した後，それぞれを加工し，元の画像と合成する必要がある．

リスト 1 は，入力画像の色相，彩度，輝度を求めた後，それぞれを擬似カラー表示するプログラムである．結果を図 2 に示す．

図 2 から，空の部分は隣接する他の部分に比べて色相の差異が大きいこと，木の部分も他の部分に比べて色相の差異が大きいが，隣接する看板や幌も似た色相になっているため，領域の指定や彩度も併用してキーイングを行うといった工夫が必要であるということがわかる．

入力画像から．キーイングに使用する値や閾値を元の画像から推測することが難しい場合でも，提案手法によって，このようなパラメタの設定の参考になる情報を得るためのプログラムを，短いコードで記述することができた．

図 1. 入力画像

図 2. チャネルの可視化の結果画像

† 東京電機大学大学院 未来科学研究科
Graduate School of Science and Technology for Future Life, Tokyo Denki University

```
(defvar *layer* (make-layer
 :image (image<-file "Src1.png")
 :filters (list (pa$ map-image1-i))))
(defvar *image* (make-co-image :size (new-size 1920
1088) :layers (list (cons "Layer" *layer*))))
(screen<-images (render-co-image-with-candidate
 *image* (list (cons "Layer" (list #'visualize-pixel-hue
 #'visualize-pixel-saturation
 #'visualize-pixel-brightness))))  1920 3)
```

リスト 1. チャネルの可視化

### 4.1.2. 色調変更

リスト 2 は空と木を別レイヤに分離して，”空”領域に施すガンマ補正パラメタを 0.6 から 4.2 まで 0.4 間隔で変化させ，”木”領域に施す色相補正パラメタを 10.0 から 64.0 まで 18.0 間隔で変化させる．その結果を，横方向に空の色の変化が，縦方向に木の色の変化が現れるように処理結果を表示するプログラムである．その結果を図 3 に示す．

提案手法により，短いコードで，2 つのパラメタの変化を直感的に把握しやすい配置で出力することができた．

図 3. 色調変更の結果候補画像

### 4.1.3. カスタムフィルタの適用

任意の関数を用いて画素の値を取り出し，その値を用いて画素をソートし，任意の順位の画素を出力する 5×5 画素の大きさの空間フィルタを実装した．リスト 3 の例では，前節で得られた全処理結果画像のうちの 1 つに対してこのフィルタを適用し，画素値を取り出す関数の候補として，画素の色相，彩度，輝度を取り出す関数，順位の候補として，1 番目，13 番目，25 番目を与え，その組み合わせから得られる画像を並べて表示する．その結果を図 4 に示す．

以上述べたように，提案手法により，パラメタの変化と処理結果の対応関係を予測することが極めて困難な動画像処理について，全処理結果を比較するだけで，所望の動画像処理結果をもたらすパラメタを決定することができた．

## 4.2. GPU による高速化の評価

我々は GPU の高速化効果の性能測定をベンチマークテストによって行った．計測に使用したシステムは，Intel Core i5 2.4Ghz, RAM4GB, Mac OS X 10.6.7 であり，対象となる画像のサイズは縦 1088 画素，横 1920 画素である．結果を図 5 に示す．

図 5 から，SUBTRACT や ADD などの画素間演算では，GPU による高速化の効果が現れていることがわかる．

## 5. むすび

動画像処理の効率よく行うために，パラメタ決定を支援する機能を開発した．また，高速化のために演算を GPU で行う仕組みと環境を開発した．

## 参考文献

[1] 豊住耕一，熊谷一樹，高橋時市郎 ：“動画像処理のための動的プログラミング環境の開発”，映情学技報, vol. 35, no. 14, AIT2011-20, pp. 29-32, 2011 年 3 月．

[2] 三原洋平，川村正祥，中島克人 ：“メディアコンテンツ作成支援のためのバリエーション生成／表示方法の提案，第 20 回人工知能学会全国大会, 2C2-5, 2006.

```
(defparameter *sample3* (image<-file "Src3.png"))
(defparameter *layer3* (make-layer :image *sample3*))
(set-layer-filters *layer3* (pa$ rank-filter-image 2 2))
(screen<-images
 (render-layer-with-params *layer3* (list 0 12 24)
    (list #'pixel-hue #'pixel-saturation #'pixel-brightness))
 2560 2)
```

リスト 3. カスタムフィルタの適用

図 4. カスタムフィルタ適用の結果候補画像

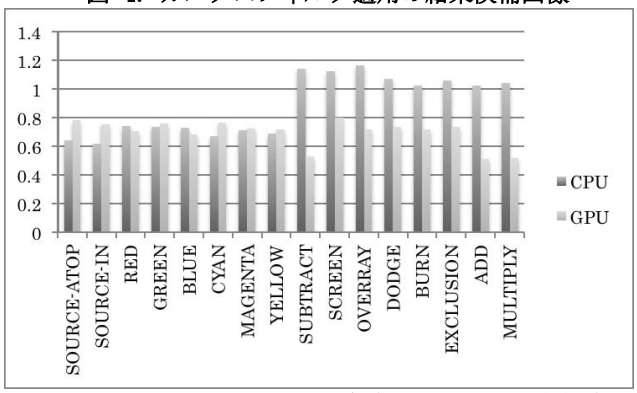


図 5, CPU と GPU における合成処理用関数の実行時間の比較(単位:秒)

```
(defparameter *sample1* (image<-file "Src1.png"))  (defparameter *sky* (make-layer :image (image<-file "Src2.png")))
(set-layer-filters *sky* (pa$ band-pass-filter-image-o 260.0 180.0 #'pixel-hue-component) #'adjust-gamma)
(defparameter *tree-layer* (make-layer :image *sample1*))
(set-layer-filters *tree-layer* (pa$ high-pass-filter-image-o 0.4 #'pixel-saturation-component)
                                (pa$ band-pass-filter-image-o 100.0 40.0 #'pixel-hue-component)
                                (pa$ mask-image-cl-o :out 298 739 257 349) #'adjust-hue)
(defparameter *back-layer* (make-layer :image *sample1*))
(defparameter *co-image* (make-co-image :size (new-size 1920 1088)))
(add-layer *co-image* "Background" *back-layer*) (add-layer *co-image* "Sky" *sky*)
 (add-layer *co-image* "Tree" *tree-layer*)
 (screen<-images (render-co-image-with-candidate *co-image*
    (list (cons "Sky" (loop for param from 0.6 to 4.2 by 0.4 collect param))
          (cons "Tree" (loop for param from 10.0 to 68.0 by 18.0 collect param)))) 5760 9)
```

リスト 2. 色調変更