RA-004

ループレス画像処理記述言語の提案と実装

桜井寛子† 岡田慎太郎†
津邑公暁† 松尾啓志†

本稿では，省電力化を意識した画像処理をより簡単かつ効率よく記述することができるプログラミング言語を提案する．携帯端末における画像処理は，適切な処理精度を選択しつつ出来るだけ処理を低減することが求められるが，一般にそのようなプログラムは記述が困難である．そこで我々は，プログラマから画像の解像度や構成要素数を隠蔽することで，そのようなプログラムの記述性を向上させるライブラリ RaVioli を提案している．プログラマが記述する画像処理プログラムは構成要素あたりの処理のみに限定され，解像度や繰り返し回数を意識せずにプログラムを記述出来るようになった．本稿では，RaVioli のフロントエンドとして繰り返し記法や暗黙的変数定義を提供することで，より直感的に画像処理を記述する言語を提案する．さらに顔検出，エッジ抽出プログラムを実際に本言語で記述することにより，コードサイズと可読性の評価を行った．

Proposal and Implementation of Loopless Image Processing Description Language

Hiroko Sakurai,† Shintaro Okada,† Tomoaki Tsumura† and Hiroshi Matsuo†

This paper proposes a programming language for image processing. Programmers can describe an energy-aware image processing program more easily and efficiently with this language. In mobile devices, it is demanded for image processing to choose appropriate processing precision and reduce processing amount as much as possible. However, it is difficult for a programmer to describe such a program. Therefore, we have proposed a library: RaVioli. RaVioli conceals resolution or components number of target images from the programmer. Hence, programmer can describe programs without considering resolution and iteration. In this paper, a new programming language is proposed as a front-end of RaVioli. With this language, programmers can describe image processing programs more intuitively. A face detection and an edge detection program written by this language show the good code size and readability.

1. はじめに

携帯端末の高機能化に伴い，組み込みデバイスにおける動画像処理などの高度な情報処理の重要性が増大している．このようなデバイスでは，そのプロセッサパフォーマンスがあまり高性能ではないことを前提としなければならず，またバッテリー駆動式という制限から消費電力を抑えることも必須であり，動画像をリアルタイム処理するために有限リソースをいかに効率よく扱うかが非常に重要な問題となる．一方，このような携帯端末向けに動画像処理プログラムを記述する場合，PC 等で使用していたプログラムをそのまま移植するだけでは不十分であり，場面に応じた処理精度を適切に選択しつつ，必要に応じて処理量を軽減する機能を実装することが必要である．これはプログラムを複雑化させバグ混入の可能性が増大するが，特に携帯端末の場合，ひとたび製品に組み込んだプログラムを修正・更新することは容易ではなく，これは開発プロセスのコスト増大にもつながる．

これに対し我々は，動的に解像度 (時間解像度，空間解像度) を自動調整することで処理量を変動させ，プログラマが意識せずともリアルタイム性の保持および省電力化を実現するライブラリ RaVioli を提案し，その実現を目指している．

RaVioli は解像度を動的に変更するという目的から，解像度をプログラマから隠蔽することを実現している．本稿ではこの RaVioli のフロントエンドとして，繰り返し記法の統一や暗黙的変数定義を提供することで，より直感的に画像処理を記述できる言語を提案する．

2. 関連研究

動画像処理を抽象化することで記述性を向上させるアプローチとして，STL を基本とするテンプレートを用いてフィルタ処理や分析処理などの高度な処理ま

† 名古屋工業大学
Nagoya Institute of Technology

```
Gray_Img(Image){
  for(y=0; y<Hight; y++){
    for(x=0; x<Width; x++){
      Image[x][y]=Gray_Pix(Image[x][y]);
    }
  }
}
```

図 1 従来のグレースケール化

```
Gray_Img(Image){
  //Image は IMAGE クラス
  Image->proc(Gray_Pix);
}
```

図 2 高階メソッドを使用したグレースケール化

で抽象化したライブラリ VIGRA[1] や，動画像処理の一般的なアルゴリズムを C 関数や C++メソッドとして提供する OpenCV[2] 等がある．しかしこれらは処理精度の動的変更には対応していない．そこで我々は解像度非依存型ライブラリ RaVioli を開発している．

2.1 解像度非依存型ライブラリ RaVioli

解像度非依存型ライブラリ RaVioli[3] は動的に解像度 (時間解像度，空間解像度) を調節することで処理量を変動させ，プログラマが意識せずともリアルタイム性の保持および省電力化を実現することを目標としている．また入力画像を C++のクラスの概念を用いて抽象化することで，解像度や画像の構成要素等を隠蔽し，プログラマがより本質的な処理内容にのみ注力することができる環境を目指している．

RaVioli では，画像に対する処理は全て画像クラスの持つメソッドを通じて行う．画像クラスの持つメソッドは，グレースケール化等の具体的な機能を持つものではなく，ユーザが記述した関数を引数にとり，それを画像全体あるいは指定された部分に適用するという機能を持つ高階関数の形で実現している (以下，**高階メソッド**)．

グレースケール化を例に挙げて，従来の記述と RaVioli の高階メソッドを使用した記述の違いについて解説する．通常のグレースケール化の場合，プログラムは**図 1** のようになる．画像の幅と高さをイタレーション回数として，1 つ 1 つのピクセルに対して彩度を 0 にするという処理を施す必要がある．またこの場合，解像度が変わるとループイタレーション部分を変更しなくてはならない．一方高階メソッドを使用したグレースケール化の場合，プログラムは**図 2** のようになる．プログラマは 1 つのピクセルに対する処理を記述し，それを高階メソッド proc に渡すことで画像全体に処理を施すことができる．IMAGE クラスは画像の幅，高さの情報を内部に持っており，高階メソッドが自動的にその値に応じた繰り返し処理を適用するため，プログラマはイタレーションを意識することなく画像処理プログラムを記述できる．解像度が変更された場合でもそれはライブラリにより吸収され，自動的に必要な画素にのみ処理が適用される．

2.2 RaVioli の問題点

様々な画像処理の中で，今回本稿で取り上げる 3 つの処理を以下に示す．

(1) ピクセルの 1 対 1 写像
1 つのピクセルを参照してそのピクセルを処理する．(例:グレースケール化，2 値化，肌色検出)

(2) ピクセルの多対 1 写像
処理対象のピクセルとその近傍画素を参照して中心のピクセルを処理する．(例:ぼかし，エッジ抽出)

(3) ボックス処理
対象とするピクセルの集合 (部分画像) を処理する．(例 : テンプレートマッチング)

高階メソッドは，ユーザが記述した 1 つのピクセル (または部分画像) に対する上記のような処理を画像全体に施す．しかしこれらの画像処理を記述する際，プログラマはそれぞれの繰り返しパターンに応じて異なる高階メソッドを使用する必要がある．そのためプログラマは，目的の画像処理がどの高階メソッドに当てはまるかを意識してプログラムを記述しなければいけない．そこで本稿では，どの繰り返しパターンに当てはまるかを意識する必要のない，より直感的なプログラミングパラダイムをプログラマに提供する言語を提案する．

3. 記述方式の提案

3.1 ループレス記述方式

一般に画像処理は，1 つの構成要素に対する処理の繰り返しである．そのためプログラマは，最低限，この 1 つの構成要素に対する処理と，それを繰り返し適用する処理の範囲だけ記述すればよい．この考えは RaVioli と同様であが，2.2 節で述べたように RaVioli を使用する場合，プログラマは目的の画像処理がどの高階メソッドに当てはまるかを考え，使い分ける必要があった．そこで本言語では，ピクセルのような処理の単位を処理単位，その処理を施す対象を処理対象全体として以下のような記述方式を提案する．

提案する記述方式

処理単位@処理対象全体

これを使用することでプログラマは，多対 1 写像の場合も 1 対 1 写像のときと同様に,「ピクセル@イメージ」のように記述することができ，統一的に処理の単位，及び処理を施す対象を指定できる．処理単位がピクセル集合 (部分画像) の場合でも，先に挙げた記述方式の先頭に括弧で囲んで処理単位の大きさを指定すればよく，汎用的にこの記述方式を使用できる．

また本言語では，基本的に変数の型を宣言する必要はない．しかし関数を定義する際には，返り値の型を記述する代わりに関数名の前に Func と記述する．

3.2 言語仕様

この節では，2.2 節で取り上げた 3 つの画像処理に対する具体的な例を 1 つずつ挙げ，さらにその処理を本記述方式にしたがって記述したプログラムを示すことにより，本記述言語の仕様を解説する．

3.2.1 1 対 1 写像

グレースケール化 (**図 3**) を例に，1 対 1 写像の本言語での記述を説明する．2 行目でピクセル p1 の RGB 値の平均を求め，3 行目で，求めた平均値をピクセル p1 の RGB 値にそれぞれ代入している．

基本的な構造は上から順に，関数名の定義と引数の宣言，施したい処理の記述，返り値の宣言と，ほぼ C 言語と同様のプログラム構造を有している．「pn (n は正数)」は Pixel 変数である．2 行目の「p1.R」のように「.」の後ろに扱いたい色情報 (RGB 値，HSV 値) の記号を記述することで，ピクセル p1 の R 値を扱うことができる．また 3 行目のようにブレースを使用することで，タプル単位で値を操作することができる．「Img_」から始まる変数は Image 変数である．これは画像の全てのピクセルの情報と画像の幅，高さなどの情報を，プログラマが抽象的に扱うために用いる変数である．ここで関数の引数に「p1@Img_in」とある．これはこの関数の引数が画像 Img_in であること，および Img_in の処理単位はピクセル p1 であることを表している．本言語の仕様により，関数本体で記述されているピクセル p1 に対する処理を，画像 Img_in 全体に適用されるまで繰り返し行う．このように本記述方式を用いることで，プログラマは処理の単位と処理を施す範囲を簡潔に指定できるようになる．さらに，関数定義内でこの 1 つの処理単位ピクセル p1 に対する処理を記述するだけで，画像全体に処理を施すことができるようになる．

3.2.2 多対 1 写像

2 値化画像のエッジ抽出を行うプログラムを**図 4** に示す．2〜5 行目で，p1 の周囲 8 画素の R 値の和から，p1 の R 値を 8 倍した値を引いて tmp に代入している．7，8 行目では tmp の値が 50 より大きかった場合に p1 に黒を，それ以外の場合には白を代入する．

「$< x, y >(x, y$ は整数)」は Coord 表現である．プログラマはこの記法を用いることで，相対座標を扱うことができる．例えば <-1,-1>p1 は，現在処理対象となっているピクセル p1 の左上のピクセルを指す表記である．

3.2.3 ボックス処理

ボックス処理の本言語での記述を，テンプレートマッチングを例にして説明する．本記述言語で記述したテンプレートマッチングのプログラムを**図 5** に示す．ここで，引数で与えられている「(Img_TP)Img_Small@Img_in」というのは，処理単位が Img_TP の大きさの Img_Small であることと，処理の対象が Img_in であることを表している．

```
1 :  Func Gray(p1@Img_in){
2 :    Ave = (p1.R+ p1.G+ p1.B) / 3;
3 :    p1.{R, G, B} = {Ave, Ave, Ave};
4 :  }(Null)
```

図 3 グレースケール化

```
1 :Func edgedetect(p1@Img_in){
2 : tmp=(<-1,-1>p1.R+ <0,-1>p1.R+ <1,-1>p1.R
3 :     +<-1, 0>p1.R            + <1, 0>p1.R
4 :     +<-1, 1>p1.R+ <0, 1>p1.R+ <1, 1>p1.R)
5 :     -8*p1.R;
6 : tmp=abs(tmp);
7 : if(tmp > 50) p2@Img_out=#black;
8 : else          p2        =#white;
9 :}(Img_out)
```

図 4 エッジ抽出

```
1 : Func SAD(p1@Img_a, p2@Img_b){
2 :   sum+=abs(p1-p2);
3 : }(sum)

4 : Func TPmatching(Img_TP,
                    (Img_TP)Img_Small@Img_in){
5 :   min=INT_MAX;
6 :   @{
7 :     sad=SAD(Img_Small,Img_TP);
8 :     if(min>sad){
9 :      min=sad;
10:      {x,y}=Img_Small.{x,y};
11:      }
12:     sad=0;
13:   }
14: }(x,y)
```

図 5 テンプレートマッチング

テンプレートマッチングでは，テンプレート画像 Img_TP と部分画像 Img_Small の差分を計算し，それが最小となるところを探している．7 行目の SAD が差分を計算している関数であり，現在までの最小値と比較する．この SAD 関数は 1〜3 行目で定義されている．

これまで説明してきた画像処理プログラムでは，関数内すべてを繰り返し適用してきた．しかしテンプレートマッチングでは，変数 min の初期化のように，関数を呼び出した直後にだけ実行すればよい部分がある．このように，関数内に処理単位ごとに繰り返したい処理とそうでない処理が混ざっている場合は，画像全体に施したい処理を@{ と } で囲めばよい．図 5 では 6 行目から 11 行目がこれに該当する．これがない場合は，関数本体に記述されている処理全てを繰り返し適用する．

4. 実装と評価

本言語で記述したプログラムを，RaVioli を使用した C++のプログラムに変換するトランスレータの実装を行った．トランスレータは perl を用いて作成しており，これによりグレースケール化や顔検出，エッジ抽出プログラムなどの変換が可能となった．以降で，

表 1　コードサイズの比較

プログラム名	比較対象		本言語	RaVioli	C++
顔検出	行数	(lines)	57	62	162
	バイト数	(bytes)	1084	1472	4029
エッジ抽出	行数	(lines)	22	25	48
	バイト数	(bytes)	582	840	1475

```
void Gray(Img* img){
 int i,j,Ave;
 for(i=0;i<img->height;i++){
  for(j=0;j<img->width;j++){
   Ave=(img->D[i][j].r
        +img->D[i][j].g
        +img->D[i][j].b)/3;
   img->D[i][j].r=Ave;
   img->D[i][j].g=Ave;
   img->D[i][j].b=Ave
  }
 }
}

Gray(&Image);
```

図 6　C++：グレースケール化

```
void Gray_Pix(RV_Pixel* p){
 int R,G,B,Ave;
 p->getRGB(R,G,B);
 Ave=(R+G+B)/3;
 p->RGBabs(Ave,Ave,Ave);
}

Image->proc(Gray_Pix);
```

図 7　RaVioli：グレースケール化

顔検出，エッジ抽出プログラムによるコードサイズの比較及び記述性・可読性の比較により，本言語の評価を行う．

4.1　プログラムサイズの比較

顔検出[4] とエッジ抽出のプログラムを本言語，RaVioli を使用したプログラム，C++のみで記述したプログラムの 3 つで記述し，プログラムの行数とバイト数がどの程度削減できるか評価した．なお，この評価対象のプログラム群において画像の入出力を行っている部分は，あらかじめ定義された関数を使用しておりコードサイズの評価対象としない．**表 1** から，本言語のプログラムは，RaVioli を使用したプログラムおよび C++のみで記述したプログラムから，行数とバイト数を削減したことが確認できる．

4.2　記述性・可読性の比較

RaVioli 及び C++のみで記述されたグレースケール化のプログラムをそれぞれ**図 6**，**図 7** に示し，図 3 と比較する．本言語と RaVioli では，解像度や構成要素数をプログラマから隠蔽することで，イタレーション回数を記述する必要をなくしている．また処理単位に対するユーザ定義関数を適用する際，RaVioli では高階メソッドを使い分ける必要があるが，本言語と C++のみで記述されたプログラムでは，それを意識する必要はない．このように本言語では，プログラマは本来記述すべき処理だけを記述出来るようになり，さらにコードの記述量が削減したことで，プログラムの記述性・可読性も向上した．

5. おわりに

本言語の提案により，プログラマが記述する部分は構成要素あたりの処理のみに限定された．これによりプログラマはピクセル数や解像度を意識せずにプログラムを記述できるようになり，処理の繰り返し回数を意識する必要がなくなった．また，本言語で記述したプログラムを RaVioli を使用した C++のプログラムに変換するトランスレータを実装した．トランスレータは perl を用いて作成しており，このトランスレータの実装によって，グレースケール化や顔検出，エッジ抽出プログラムなどの変換が可能となった．

今後の課題としては，まず静止画像だけでなく動画像に対する画像処理プログラムの記述を対象として，本言語の拡張を行うことである．本稿ではグレースケール化や顔検出，エッジ抽出プログラムなどの静止画像に対する画像処理プログラムの記述を対象としてきた．今後はさらに動画像に対する画像処理プログラムの記述を対象として本言語の拡張を行う．次に自動並列化への拡張が挙げられる．近年では 1 チップ上に複数のプロセッサコアを集積し，それらを並列に動作させることによって高い性能を実現するマルチコアの技術が，広く一般に普及してきた．しかし並列処理プログラムを開発するには，プログラムを実行するプロセッサやアーキテクチャに関する深い知識，またそれを活かすプログラミング技法が必要である．そこで本言語を拡張することにより，メモリアーキテクチャに依存する処理の記述を隠蔽し，プログラマが容易に並列処理プログラムを開発できるようにしていきたい．

参　考　文　献

1) Köthe, U.: Generic Programming for Computer Vision: The VIGRA Computer Vision Library, http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/ (2006).
2) Intel Corp.: *Open Source Computer Vision Library* (2001).
3) 岡田慎太郎，津邑公暁，松尾啓志：解像度非依存型動画像処理ライブラリの提案と実装，情報科学技術フォーラム 一般講演論文集第 3 分冊，情報処理学会，pp.335–338 (2007).
4) Yao, H. and Gao, W.: Face Detection and Location Based on Skin Chrominance and Lip Chrominance Transformation from Color Images, *Pattern Recognition*, Vol. 34, pp. 1555–1564 (2001).