社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

動画像処理ライブラリ RaVioli における領域別処理量調整の実現

近藤　勝彦†　　大野　将臣†　　津邑　公暁†　　松尾　啓志†

† 名古屋工業大学

あらまし　近年，汎用 PC の高性能化により高度な画像処理が実行可能になってきた．しかし，汎用 OS を用いたシステムでは常に十分な CPU リソース量を確保できるという保証がないため，リアルタイムに動画像を処理できるとは限らない．そこで，解像度を動的に変動させて処理量の調整を行うことで，この問題を解決する動画像処理ライブラリ RaVioli が提案されている．しかし，RaVioli は単純に画像の解像度を低減させるので，動画像処理の精度が低下する．処理のリアルタイム性を保つために解像度を低減させるので，処理精度まで保証することはできないが，できるだけ高い解像度で処理を行いたい．そこで，画像の領域別に処理量を調整することを提案し，RaVioli に追加実装した．サンプルプログラムを用いて評価を行い，提案手法を用いた際に解像度の低減が抑えられることを確認した．

キーワード　リアルタイム処理，負荷調整，動画像処理ライブラリ

Tiling with Different Spacial Resolutions for Pseudo Realtime Video Processing Library RaVioli

Katsuhiko KONDO†, Masaomi OHNO†, Tomoaki TSUMURA†, and Hiroshi MATSUO†

† Nagoya Institute of Technology

Abstract　The performance of general purpose computers is increasing rapidly, and now they are capable of running video processing applications. However, on general purpose operating systems, realtime video processing is still difficult because there is no guarantee that enough CPU resources can surely be provided. A pseudo realtime video processing library RaVioli has been proposed for solving this issue. RaVioli conceals two resolutions, frame rate and number of pixels, from programmers and provides a dynamic and transparent resolution adjustabiliity. Using RaVioli, pseudo realtime video processing can be achieved easily, but output precision may be roughened for reducing processing load. To prevent this situation, this paper proposes a method for dividing whole video frame into several subframes and adjusting the resolutions of each subframes automatically and individually. We have implemented the method on RaVioli and have made some evaluations with a sample program. The result shows that the proposed method can keep the resolution of video frames higher than traditional RaVioli.

Key words　real-time processing, load adjustment, image and video processing library

1. はじめに

近年，侵入者検知システムや衝突回避システムなどリアルタイム性の重要な動画像処理システムの開発が盛んに行われている．また，汎用 PC の高性能化と低価格化により，高性能な計算機環境を容易に手に入れることが可能となった．そのため今後，汎用 PC および汎用 OS 上でリアルタイム動画像処理を行うことが多くなると予想される．

しかし，汎用 OS 上で，1/30 または 1/60 秒毎に処理を行うリアルタイム動画像処理の実現はいまだに難しい．その主な理由として，1 フレームあたりの処理量の変動や，複数プロセスの並行実行による利用可能な CPU リソース量の変動が挙げられる．これらは 1 フレームあたりにかかる処理時間に影響し，この処理時間の増減がリアルタイム性の保証を難しくしている．

そこで，汎用システム上で擬似的なリアルタイム性を保証する動画像処理ライブラリ RaVioli [1] [2] が提案されている．RaVioli は前フレームまでの処理時間に応じて処理対象フレームの解像度を自動的に変動させて処理量を調整する．この方法では，厳密に実時間に処理されるわけではないが，処理の大幅な遅れを回避し，リアルタイム性を擬似的に保証する．

しかし，RaVioli の問題点として，解像度を低減させることによる処理精度の低下が挙げられる．RaVioli は解像度を低減させることでリアルタイム性を保証するため，処理精度を保証することは難しいが，できるだけ解像度の低減を抑えて処理精

度を高く保ちたい．

そこで，リアルタイム動画像処理の入力フレームに注目する．リアルタイム動画像処理の入力フレームには詳細に処理する必要がない領域が存在する．この領域に対する処理量を減らすことで，フレーム全体にかかる処理量を削減することができ，その他の領域を高い解像度で処理することが可能になる．これを実現するためにフレームを複数の領域に分割し，領域別に解像度を変動させて，処理量を調整することを提案する．

2. 背　　景

2.1 従来の動画像処理

リアルタイム動画像処理では，使用可能な CPU リソース量に応じてフレームの処理時間に応じて処理量を調節することが重要である．しかし，これまでは複数の予め定義されたルーチンを切り替えることが，これを実現する唯一の方法であった．例えば Imprecise Computation Model [3] は計算時間の長さに応じて処理精度を変化させるモデルである．またこのモデルに基づき，複数のルーチンから適切なルーチンを動的に選択する信頼度駆動アーキテクチャ [4] も提案されている．しかし，このモデルは処理精度を変化させるために，予め複数のルーチンを定義しなければならないためプログラマの負担が大きくなる．

一方で，良く知られる動画像処理ライブラリに VIGRA [5] や OpenCV [6] がある．これらのライブラリは動画像処理の抽象化を目的としたもので，VIGRA では C++の STL と同様にテンプレートを用い，プログラマに抽象的な処理を提供している．また，OpenCV では多くの動画像処理アルゴリズムを C 言語の関数や C++のメソッドとして提供している．しかし，これらのライブラリを用いて実装されたプログラムで処理量を動的に調整したり，リアルタイム性を保証することは難しい．

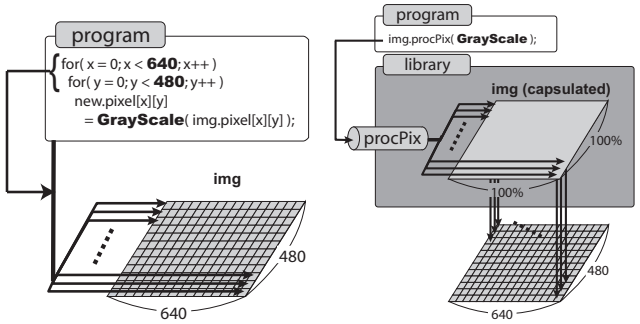
これに対し，RaVioli は 1 フレームあたりにかかる処理時間の増加によりリアルタイム処理が困難になった場合，解像度を変動させることで処理量を調整し，処理がリアルタイムに行われることを擬似的に保証する．すなわち，処理された出力フレームが入力フレームに対して大幅に遅れないことを保証する．また，動的に解像度を変動させるためにプログラマから解像度を隠蔽するため，プログラマは解像度を意識することなく処理を記述できる．次節で RaVioli について詳細に説明する．

2.2 RaVioli

2.2.1 解像度非依存性

処理量を調整するために動的に解像度を変動させる場合，1 フレームあたりの画素数やフレームレートの変動に対応したプログラムの記述が不可欠である．そこで，RaVioli はプログラマから動画像のフレーム数や画像の幅および高さを隠蔽し，解像度をライブラリ内で制御している．解像度には空間解像度と時間解像度の 2 つがあり，空間解像度は 1 フレームを構成する画素数を意味し，時間解像度はフレームレートを意味する．この 2 つの解像度を隠蔽することで，プログラマは動画像のフレーム数や画像の幅や高さを意識した記述を省略できる．

RaVioli を使用しない一般的な動画像処理には，量子化された動画像データを扱うためにループがよく用いられる．例えば，画像をグレースケールに変換する処理は，図 1(a) に示す通り，



(a) 一般的な場合　　(b) RaVioli を用いる場合

図 1　グレースケール処理

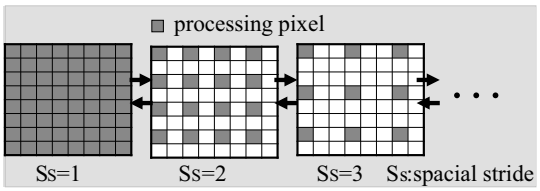


図 2　空間解像度ストライドの変更

各画素を変換する処理は最も内側のループに記述され，この処理が画像中の全ての画素に繰り返し適用される．このように，ループを用いて行う画像処理では，プログラマは画像の幅と高さを意識した記述を行わなければいけない．

一方，RaVioli では動画像の構成要素である画素またはフレームに対する処理のみを関数として記述し，その関数を RaVioli が提供しているメソッドに渡すことで，画像中の全ての画素を処理することが可能である．RaVioli ではこの構成要素に対する処理を記述した関数を**構成要素関数**と呼び，その構成要素関数を引数とするメソッドを**高階メソッド**と呼ぶ．ここで，RaVioli を用いてグレースケールに変換する処理の様子を図 1(b) に示す．RaVioli では画像情報をもつクラスのインスタンスである img の高階メソッド procPix() に構成要素関数 GrayScale() を渡すのみでよい．ここで，高階メソッドの procPix() はそのインスタンス img がもつ画像の全ての画素に，GrayScale() を繰り返し適用する．このような処理構造を用いることで，プログラマは解像度や繰り返し処理を意識することなく画像処理プログラムが記述できる．なお，RaVioli には様々な繰り返しパターンに対応する高階メソッドが用意されているが，それら詳細については文献 [1] を参照されたい．

2.2.2 リアルタイム性の保証

RaVioli は解像度をプログラマから隠蔽したことにより，負荷に応じて 2 つの解像度を動的に変動させることを可能にしている．RaVioli は空間解像度と時間解像度を制御するための，空間解像度ストライド (S_S) および時間解像度ストライド (S_T) を持ち，各ストライドを増減させることにより解像度を変動させている．例えば，空間解像度ストライドの値を増減させることで処理量を調整する場合は図 2 に示す通り，処理画素数を増減させる．一方，時間解像度ストライドの値を増減させることで処理量を調整する場合は図 3 に示す通り，処理フレーム数を増減させる．

ここで，どちらの解像度を低減させるべきかは処理内容よっ

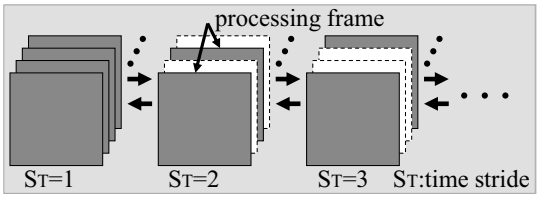


図 3　時間解像度ストライドの変更

て異なる．例えば，2 つの解像度の内，時間解像度をより重要とする侵入者検知や衝突回避などと，空間解像度をより重要とする顔認識や QR コードリーダなどがある．そのため，RaVioli はどちらの解像度をどれだけ優先したいかをプログラマが指定できるようにしている．この指定には優先度セットと呼ぶ 2 つの値 (P_S, P_T) の組を用いる．P_S は空間解像度に対する優先度を表し，P_T は時間解像度に対する優先度を表す．例えば，$(P_S, P_T) = (1, 3)$ の場合，時間解像度を 3 倍優先したいということを表し，RaVioli は空間解像度ストライドと時間解像度ストライドを 3 : 1 の割合で維持しようとする．

3. 提　　案

3.1 提案手法の着眼点

2.2 節で説明したように，RaVioli は解像度を動的に変動させて処理量を調整している．しかし，解像度を低減させることには限界があり，解像度が大幅に低減することでプログラマが期待した動画像処理が行えなくなることが考えられる．そこで，リアルタイム動画像処理の入力フレームの特徴に着目した処理量調整方法を提案する．

リアルタイム動画像処理はリアルタイムに入力フレームをキャプチャし処理するが，入力によっては詳細に処理する必要がない領域が存在する．例えば，侵入者検知システムでは侵入者が現れず入力フレームに変化がない場合，そのフレームを詳細に処理する必要はない．また，侵入者が現れた場合でも，侵入者が存在する領域以外の大半の領域は変化がなく，詳細に処理する必要はない．これと同様にその他の動画像処理にも詳細に処理する必要がない領域が存在すると考えられる．これらの領域を詳細に処理することで CPU リソースが無駄に使用される．RaVioli は CPU リソースが十分に確保できないとき，解像度ストライドを増加することで処理量を削減するため，この無駄な処理によって画像全体の解像度低下を引き起こすことがある．

そこで，フレームを均等な大きさの領域に分割し，詳細な処理が必要な領域と必要でない領域で別々に解像度を変動させることを提案する．従来のフレーム全体に対して設定される解像度ストライドに加えて，領域毎に解像度ストライドを設定する．入力フレーム中の詳細な処理が必要な領域は，フレーム全体の解像度ストライド値 (以降，ベースストライドと呼ぶ) を領域のストライドに設定し，詳細な処理が必要でない領域はそれよりも低い解像度ストライド値 (以降，ラフストライドと呼ぶ) を領域に設定する．これにより，リアルタイム動画像処理時の無駄な処理を削減できる．

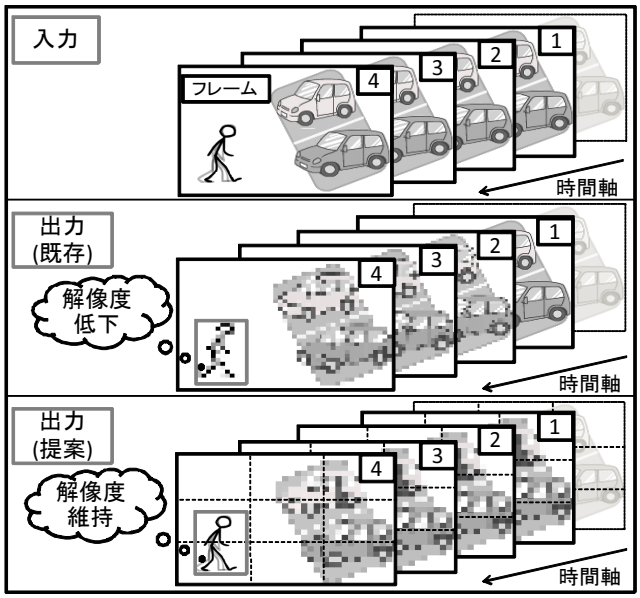


図 4　2 手法を用いた侵入者検知の様子

3.2 動作モデル

提案手法の動作モデルを説明するために侵入者検知システムを例として，既存の RaVioli での処理と提案方式の RaVioli での処理を比較する．なお，この侵入者検知システムでは，侵入者を見逃さないために全てのフレームを処理すると仮定する．そのため，RaVioli は処理量調整のために空間解像度を変動させる．

2 手法を用いて侵入者検知を行う様子を図 4 に示す．図 4 の上段はフレーム 1 から 4 の順にキャプチャされた入力フレームを示している．この例では，フレーム 1 から 3 は数フレーム前から変化がなく，フレーム 4 で初めて侵入者が現れたとする．また，図 4 中段は既存の RaVioli を用いて得られる出力，下段は提案手法を用いて得られる出力を示している．

まず，図 4 中段に示した既存の RaVioli での侵入者検知の様子について述べる．既存の RaVioli は，全ての入力に対して，できるだけ高い解像度で処理しようとするため，フレーム 1 のような前入力から変化がないフレームに対しても通常通り処理する．ここで，利用可能な CPU リソース量が減少し処理が間に合わなかったとすると，次のフレーム 2 は解像度を低減させて処理される．RaVioli は処理が間に合うまで，解像度ストライドを徐々に大きくするため，解像度低下が数フレームに渡って続くことがある．この例でもフレーム 3，フレーム 4 で解像度低下が起こったとする．ここで，フレーム 4 の処理では，出力フレームの空間解像度が低減しているため侵入者を詳細に検出することは難しい．

一方提案手法では，まず入力フレームをプログラマが指定した分割数に分割する．プログラマはフレームを行と列にそれぞれ何分割するかを指定する．そして，その各領域で詳細な処理が必要かどうかを自動的に判定する．その結果，詳細な処理が必要な場合はベースストライドをその領域に設定し，詳細な処理が必要でない場合はラフストライドをその領域に設定する．そして，設定された解像度ストライドで各領域を処理する．

図 4 下段に示した提案手法での侵入者検知の様子について述

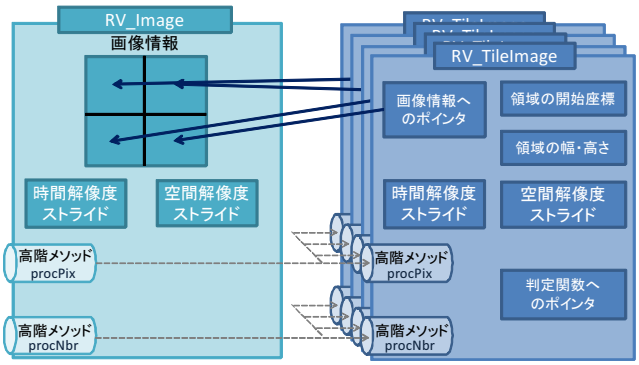


図 5　RV_TileImage の概略

べる．図 4 下段のフレーム内の垂直，水平方向の破線は分割領域の境界線を表している．この例では，3 行 3 列に領域を分割したとする．フレーム 1 から 3 に対して，既存の RaVioli では常に詳細に処理していたが，提案方式ではラフストライドで詳細な処理が必要でない領域を処理する．これによりフレーム全体の処理量が削減できるため，利用可能な CPU リソース量が減少しても，ベースストライドを増加せずに済む．ここで，侵入者が現れるフレーム 4 の処理では，侵入者がいる左下の領域を詳細な処理が必要だと判定し，ベースストライドをその領域の空間解像度ストライドに設定する．フレーム 1 から 3 の処理時にフレーム全体にかかる処理量を削減できるため，このとき設定されるストライド値は既存の RaVioli を用いた時より小さい．そのため侵入者を詳細に検出することが可能である．

このように，提案手法は詳細に処理する必要がない領域に対する処理量を削減することを可能にする．従来よりも処理量を削減することが可能になるため，詳細な処理が必要な領域の処理精度の低下を緩和できる．

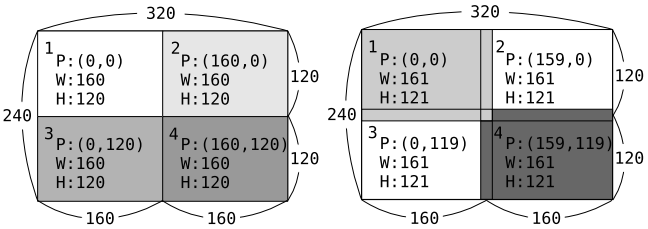
4. 実　　装

領域別処理量調整を実現するために，新たに分割領域を管理する RV_TileImage クラスを実装した．また，分割領域で画像処理を行うために，高階メソッドを変更し，詳細な処理が必要かどうかを判定する関数を追加した．本章では，これらを順に説明し，最後にプログラムがどう実行されるのかを説明する．

4.1　RV_TileImage クラス

提案手法では領域別に処理量を調整するために，画像を均等な大きさの領域に分割して処理する．そこで，その分割領域を管理するための RV_TileImage クラスを新たに追加する．RV_TileImage クラスの概略を図 5 に示す．RV_TileImage クラスはメンバとして，領域の左上を示す開始座標，領域の幅と高さ，各解像度ストライド，判定関数へのポインタを持つ．

また，RV_TileImage は画像情報の領域を確保せず，1 フレームの情報を持つ RV_Image クラスのインスタンスが確保した画像情報の領域へのポインタを持つ．RV_TileImage インスタンスは画像処理時に分割領域と同じ数生成される．その各インスタンスが RV_ Image インスタンスの持つ画像情報の一部分を処理することによって，画像全体が処理される．このとき，RV_TileImage インスタンスの処理範囲は領域の開始座標および幅と高さによって決まる．RV_TileImage インスタンスはその処理範囲に対して，従来と同様に高階メソッドを使って構成要素関数を繰り返し適用することで，分割領域を処理する．



(a) procPix,ProcImgComp　　(b) procNeighbor

図 6　各高階メソッドの分割領域

4.2　高階メソッド

本提案ではプログラマが既に記述した構成要素関数を変更しなくても領域別に処理量を調整できるようにする．そのため，RV_Image インスタンスの持つ高階メソッドを使って，構成要素関数を画像全体に対して適用している動作を，ライブラリ内で RV_TileImage インスタンスの高階メソッドを使って処理するように変更する必要がある．

そこで，RV_Image クラスの高階メソッドの内部処理を変更する．まず，既存の高階メソッドは領域別に処理量を調整する場合とそうでない場合で処理内容を分岐させる．領域別に処理量を調整する際には，分割数に応じて RV_TileImage インスタンスを生成し，前節で述べたようにフレーム全体を処理する．一方，領域別に処理量を調整しないときは従来通り処理を行う．

このように全ての高階メソッドを変更する．しかし，高階メソッドの繰り返し処理の単位には 1 画素や近傍画素集合，テンプレート画像などがあり，各高階メソッドごとに 1 つの RV_TileImage インスタンスが担当する処理範囲が異なる．そのため，メソッドごとに領域の分割方法を変える必要がある．各高階メソッドを実行する際の領域の分割方法について説明する．以降の説明ではフレームを 4 つ (2 行 2 列) に分割する場合を想定する．

procPix

procPix は 1 画素に対する処理を記述した構成要素関数を受け取り，その関数を画像全体に繰り返し適用する高階メソッドである．そのため，RV_TileImage を用いてフレームを領域別に処理する時，フレームは図 6(a) のように 1，2，3，4 の領域に単純に分割される．なお，図 6(a) 中の P は開始座標，W は領域の幅，H は領域の高さを表している．これ以降の他の高階メソッドの説明においても同じ意味で用いる．

procImgComp

procImgComp は 2 枚の画像の同じ位置を示す画素に対する処理を記述した構成要素関数を受け取り，その関数を画像全体に繰り返し適用する高階メソッドである．差分検出などに用いられる．procImgComp の繰り返し処理の単位は procPix と同じ 1 画素なので，2 枚の画像とも procPix 同様に図 6(a) のように分割される．

procNeighbor

procNeighbor は 1 画素とその近傍画素 (8 近傍) に対する処理を記述した構成要素関数を受け取り，その関数を画像全体に

```
1.  void Prog2{RV_Pixel *p}{…}
2.  void Prog1{RV_Image *f}{
3.     f->procPix(Prog2);  //画像の高階メソッド
4.  }
5.  int  UserDP(RV_Image *N,*B){
6.     //判定処理
7.  }
8.  int main(int  argc, char* argv[]){
9.     RV_Streaming video;
10.    video.setPriority(7,3);  //優先度設定
11.    video.setDetFunc(UserDP); //判定関数の設定
12. // video.setDetFunc(FrameDiff);  //判定関数の設定
13.    video.setTileNum(3,4);  //領域の分割
14.    video.procStream(Prog1);  //動画像の高階メソッド
15. }
```

図 7　プログラム記述

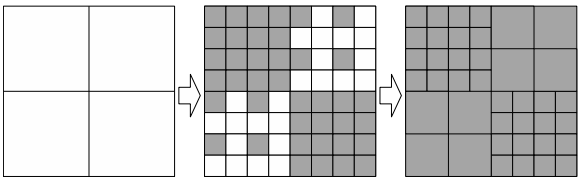


図 8　画像処理の流れ

表 1　評 価 環 境

CPU	Intel Core2 Ex. Quad(3.0GHz)
Memory	8GB
OS	Solaris10
Compiler	Sun Studio 12 (Sun C++ 5.9)

繰り返し適用する高階メソッドである．分割の様子を図 6(b) に示す．図 6(b) の 1 の領域は図 6(a) の 1 の領域の右端と下端の近傍画素を含むように分割される．同様に 2 の領域は左端と下端，3 の領域は右端と上端，4 の領域は左端と上端の近傍画素を含むように分割される．

4.3　判 定 関 数

領域別に処理量調整を行うために，各領域で詳細に処理する必要があるかどうかを判定しなければいけない．本提案では判定関数の処理構造に RaVioli の構成要素関数と同じ処理構造を用いる．プログラマは 1 フレームまたは 2 フレームを用いて，領域を詳細に処理するかどうかを判定する関数を定義し，高階メソッドを使ってライブラリに渡す．RaVioli はその判定関数を各領域での詳細な処理が必要かどうかの判定に用いる．また，いくつかの使用頻度が高いと思われる判定関数を予め定義し，これをユーザに提供する．これにより，プログラマが判定関数を記述しなくても領域別に処理量を調整できる．どちらの場合においても，判定関数は詳細に処理する必要があるとき 1，そうでないとき 0 を返す関数とする．

ここで，提案方式の RaVioli での動画像処理プログラムの記述例を図 7 に示し，それを用いて判定関数の記述や設定の方法を説明する．プログラマは main 関数と構成要素関数 (Prog1,Prog2)，判定関数 (UserDP) を記述する．main 関数では，まず動画像を管理する RV_Streaming クラスのインスタンス video を生成し (9 行目)，そのインスタンスの持つメソッド setPriority を用いて優先度の設定を行う (10 行目)．設定された優先度に応じて video は時間解像度と空間解像度を調整する．次に，判定関数を設定する．プログラマが記述した判定関数 UserDP を設定する (11 行目)，または，予め用意された関数 FrameDiff を設定する (12 行目)．FrameDiff は 2 フレームの差分を取り，変化がある領域を詳細に処理する必要があると判定する関数である．そして，フレームを何分割するかを指定し (13 行目)，動画像処理の高階メソッドに 1 フレームを処理する構成要素関数 Prog1 を渡す (14 行目)．Prog1 では 4.2 節で説明したように変更した RV_Image クラスの高階メソッド procPix を用いて，各領域ごとに UserDP または FrameDiff を用いて判定し，それにより決まるストライドで領域を処理する (3 行目)．このように記述することで，動画像中のすべてのフレームに対して処理を施すことが可能である．

更に，判定関数を引数とする高階メソッド setDetFunc とその引数である判定関数の詳細な仕様について述べる．

void setDetFunc(int(*DP)(RV_Image* Fn))
void setDetFunc(int(*DP2)(RV_Image* Fn,*Fb))

現在の処理フレーム Fn を用いる判定関数 DP または Fn と 1 つ前の処理フレーム Fb を用いる判定関数 DP2 を引数とする高階メソッドである．この高階メソッドはプログラマが記述した判定関数へのポインタを RV_TileImage の持つ判定関数用のポインタ変数に設定する．また，プログラマは 1 つまたは 2 つフレームを用いて詳細な処理が必要かどうかを判定し，詳細な処理が必要な時は 1，必要でないときは 0 を返す判定関数を記述する．そして，setDetFunc にその判定関数を引数として渡す．このようにして判定関数を設定する．

4.4　画像処理の流れ

ユーザが記述したプログラムが提案手法を用いた RaVioli でどのように処理されるのかを述べる．まず，プログラマが指定した分割数に従って画像を分割する．ここで，プログラマが 2 行 2 列の分割数を指定したときの様子を図 8 左に示す．この時，4 つの RV_TileImage インスタンスが生成され，それぞれを通じて各分割領域が処理される．各分割領域では，詳細な処理が必要かどうかを判定し，画像処理が行われる．図 8 中央にその様子を示す．図中の色が変化した箇所が処理された画素を表し，左上と右下の領域が解像度を低減させずに処理され，右上と左下の領域が解像度を低減させて処理されている．このとき，右上と左下の領域は処理されていない画素が存在することが図からわかる．そのため，提案手法ではこのように処理されなかった画素に処理された画素を補う．この処理が行われると図 8 右に示す画像が出力される．このように画像を処理することで，領域別に解像度を設定し処理量を調整することが実現できる．

5.　評　　価

表 1 に示す評価環境で，既存の RaVioli と提案手法を追加した RaVioli を用いて動画像処理を行い，2 手法の出力画像と解像度の変動を比較した．結果をそれぞれ図 9，図 10 に示す．2 つの手法に対して同じ入力を用いるため，カメラから取り込んだ動画像を処理するのではなく，ファイルから読み出した画像を 1/30 秒毎に処理することで動画像処理を擬似的に再現し，それぞれ 50 フレームを処理した．また，提案手法の判定関数には 4.3 節で述べた 2 フレームの差分を判定基準にする FrameDiff を用い，分割領域数による違いを調べるために分割

(a) original　(b) diff(3,4)　(c) diff(8,8)

図 9　既存 RaVioli と提案手法の出力結果

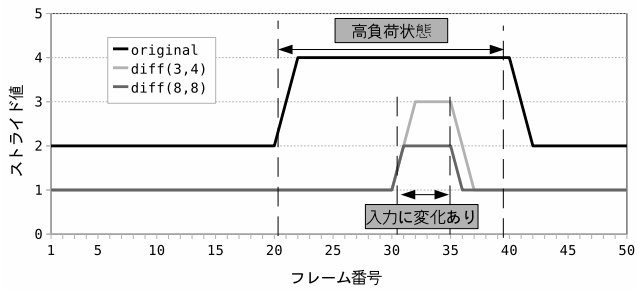


図 10　空間解像度ストライド値の変動

方法を 3 行 4 列，8 行 8 列と変化させた．これを図 9，図 10 では diff(3,4)，diff(8,8) とそれぞれ示す，また既存の RaVioli を original と示す．提案手法の効果を確かめるため，入力には 30 番目のフレームから 5 フレームのみ画像内に変化がみられるものを使用した．また，利用可能 CPU リソース量の減少による影響を調べるために，20 番目のフレームから 20 フレームの間，通常の処理を 5 回繰り返すことでフレームにかかる処理量を 5 倍とし，利用可能 CPU リソース量の減少を擬似的に表現した．

図 9 に最も負荷の高かった時刻における出力画像を示す．図 9(a)，図 9(b)，図 9(c) はそれぞれ original，diff(3,4)，diff(8,8) で得られた出力画像である．original が画像全体を同じストライドで処理し，詳細な処理が必要な領域の処理精度が低減してしまっているのに対して，提案手法では変化のあった領域を詳細に処理できていることが確認できる．特に，diff(8,8) は粗く処理された領域が画像全体に占める割合が diff(3,4) に対して大きい．そのため，diff(8,8) を用いた際に最もベースストライドの増大を抑えることができている．

次に，時間経過に伴うベースストライドの変化を図 10 に示す．この結果より提案手法では入力に変化がある 30 から 34 番目のフレームのベースストライドを，既存手法での空間解像度ストライド値に対して 1 または 2 低く保てたことが確認できた．処理画素数が 4 倍に増え，処理精度が向上した．

6. おわりに

本稿では，動画像処理ライブラリ RaVioli に対し画像を分割し領域別に処理量を調整することを提案した．領域別に処理量を調整することによってフレーム全体の処理量を削減できることを確認した．また，動画像処理中に領域別に処理量を調整することにより，詳細な処理が必要な領域の空間解像度の低下を抑えられることを確認した．

今後の課題として，分割領域単位での並列化が考えられる．並列に画像を処理することで処理時間を削減し，動画像処理中の解像度低下をさらに抑えることができると考えられる．また，分割数を増やすことで，各領域の処理対象画素が減少するため判定関数の精度が低下する可能性がある．そのため，最適な分割数を自動的に求める機能の開発も考えられる．その他にも，メモ化 [7] 手法の実装， Cell/B.E. [8] 等の様々なプラットフォームへの対応， CUDA [9] を用いた GPU の活用などによるさらなる高速化が課題としてあげられる．さらに，RaVioli に適した新しい動画像処理言語の開発も併せて検討していきたい．

謝辞

本研究の一部は，文部科学省科学研究費補助金（若手研究（B）21700028）による．

文　献

[1] 岡田, 桜井, 津邑, 松尾：“解像度非依存型動画像処理ライブラリ ravioli の提案と実装”, 情報処理学会論文誌 コンピュータビジョンとイメージメディア（CVIM）, **2**, 1, pp. 63–74 (2009).

[2] H. Sakurai, M. Ohno, T. Tsumura and H. Matsuo: “Ravioli: a parallel video processing library with auto resolution adjustability”, Proc. IADIS Int’l. Conf. Applied Computing 2009, Vol. 1, pp. 321–329 (2009).

[3] J. Liu, W.-K. Shih, K.-J. Lin, R. Bettati and J.-Y. Chung: “Imprecise Computations”, Proceedings of the IEEE, Vol. 82, pp. 83–94 (1994).

[4] H. Yoshimoto, N. Date, D. Arita and R. Taniguchi: “Confidence-Driven Architecture for Real-time Vision Processing and Its Application to Efficient Vision-based Human Motion Sensing”, Proc. of the 17th Int’l. Conf. on Pattern Recognition (ICPR’04), Vol. 1, pp. 736–740 (2004).

[5] U. Köthe: “VIGRA - Vision with Generic Algorithms”, 1.6.0 edition (2008).

[6] Intel Corp.: “Open Source Computer Vision Library” (2001).

[7] T. Tsumura, I. Suzuki, Y. Ikeuchi, H. Matsuo, H. Nakashima and Y. Nakashima: “Design and evaluation of an auto-memoization processor”, Proc. Parallel and Distributed Computing and Networks, pp. 245–250 (2007).

[8] Sony Computer Entertainment: “Cell Broadband Engine Architecture”, 1.01 edition (2006).

[9] NVIDIA Corp.: “NVIDIA CUDA Programming Guide”, 2.0 edition (2008).