卒業研究論文

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

指導教員　　松尾 啓志 教授
　　津邑 公暁 准教授

名古屋工業大学 工学部 情報工学科
平成 18 年度入学　18115066 番

近藤 勝彦

平成 22 年 2 月 8 日

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

内容梗概

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システムの開発が盛んに行われている．また，ビデオカメラなどの入力装置からリアルタイムに画像をキャプチャ可能な環境が整ってきたことや，汎用計算機の高性能化により高度な画像処理が実行可能となってきた．そのため汎用システム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，汎用システムでは並行実行プロセスなどの外乱により，リアルタイム動画像処理に必要な CPU リソースの確保が困難である．

そこで，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 がある．RaVioli では確保可能な CPU リソースの減少によりリアルタイム動画像処理が困難になった場合，解像度を低減させることで処理量を調節する．こうすることで，擬似的なリアルタイム性を保証している．

しかし，RaVioli は単純に画像の解像度を低減させるので，画像処理の精度が低下するという問題がある．これを解決する方法として，詳細な処理が必要でない画像や領域に対して解像度を低減させて処理し，処理量を削減することが考えられる．

そこで，画像を部分画像に分割し，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．これにより，動画像処理中の無駄な処理を削減し，解像度の低減を抑えることが可能となる．

実際にサンプルプログラムを用いて提案手法を評価した．既存の RaVioli と提案手法を実装した RaVioli でそれぞれサンプルプログラムを実行し，解像度の変化と出力画像を比較した．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．

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

目次

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，高性能なコンピュータ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困難である．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，他のプロセスによる使用可能な CPU リソースの変動があげられる．

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 が提案されている．RaVioli では CPU 利用率や並行実行プロセスによる負荷に応じて，処理対象画像の解像度を自動的に変動させる．これによって処理量を調節し擬似的なリアルタイム動画像処理を実現している．

このよう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必要になる．そこで，RaVioli ではプログラマから 1 フレームあたり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．こうすることで人間の映像認識過程に存在しない画素およびフレームといった概念を排除することが可能となり，より直感的な動画像処理プログラミングが実現できる．

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，処理精度まで保証することはできないが，できるだけ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に注目する．これらの入力には詳細な処理が必要でない画像や領域が存在する．この画像や領域に対する処理量を減らすことで，画像全体にかかる処理量を削減することができる．これによって詳細な処理が必要な画像や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画像を領域に分割し，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．

本論文では，2 章で本研究の背景と動画像処理ライブラリ RaVioli について述べ，3 章で領域別処理量調整方法を提案し，4 章でその実装について述べる．5 章で提案の評価とそれに対する考察を述べる．最後に 6 章で本論文のまとめと今後の課題を述べる．

2 背景

本章では従来の動画像処理に存在する問題点を述べ，それらに対する既存研究について説明する．また，本提案の基盤となる動画像処理ライブラリ RaVioli の特徴と問題点を述べる．

2.1 従来の動画像処理

ビデオカメラなどの入力機器と計算機間のデータ転送が高速になり，リアルタイムに画像を取り込む環境が整ってきている．また，汎用 PC の性能は向上し，価格も低下しているので，高い演算能力をもつ PC を手に入れることが容易になっている．そのため，今後汎用 OS や汎用 PC 上でリアルタイムに動画像を処理するシステムの開発がより盛んに行われると予想される．

しかし，汎用 PC 上でリアルタイム動画像処理に必要な計算資源を常に確保することは困難である．その原因として，1 フレーム当たりの処理量が入力によって変化することや他のプロセスの並行実行による使用可能 CPU リソース量の減少が挙げられる．例えば，顔検出を行うあるプログラムでは，まず背景画像とキャプチャした画像との差分をとり，エッジ抽出を行う．そしてその結果に対してハフ変換を行う．この時，入力であるキャプチャした画像と背景画像との間に差分があるかどうかで，エッジ抽出後のハフ変換の処理量が変動する．そのため，動画像処理中の各フレームに対する処理量は変動する．また，汎用 OS 上では複数の他プロセスが動画像処理プロセスと並行に実行されている．それらのプロセスによって，使用可能 CPU リソース量の変動が起こるため，動画像処理に必要なリソースを常に確保できるという保証はない．

この問題を解決するには，使用可能な CPU リソース量に応じて処理量を調整する必要がある．Imprecise Computation Model [1] 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．しかし，このモデルは処理精度を変化させるために，あらかじ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．

また，計算機を使って画像を処理する際には解像度という概念が不可欠である．従来の動画像処理では画像の大きさを考慮して，画像全体のピクセルに対して，処理を記述する．しかし，画像処理プログラムを記述する人間には，解像度という概念は不自然である．そもそも人間が物体を認識する過程には解像度という概念は存在しないため，解像度を意識した画像処理プログラミングは直感的でない．

そこで，擬似的にリアルタイム性を保証する解像度非依存型動画像処理ライブラリ

RaVioli[2][3] が提案されている．RaVioli では使用可能 CPU リソース量の減少によりリアルタイム処理が困難になった場合，解像度を自動調節す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．

一方で，動画像処理を抽象化するためのライブラリはこれまでにも提案されている．その中でもよく知られた動画像処理ライブラリに VIGRA[4] や OpenCV[5] がある．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，これらは単純に処理内容を抽象化してライブラリの形で提供しているものであり，画像に対する処理内容を，解像度を意識することなく記述することはできない．そのため，RaVioli の行う画像処理の抽象化はこれらのライブラリとはまったく異なっている．次節で RaVioli を詳細に説明する．

2.2 RaVioli

まず，RaVioli の特徴である解像度非依存性と擬似的なリアルタイム性の保証について述べる．そして，RaVili の問題点について述べる．

2.2.1 解像度非依存性

RaVioli ではプログラマから動画像の解像度を隠蔽することで，プログラマが直感的にプログラムを記述することを可能にしている．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している．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．

また，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** に示す通り，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に対して処理を施すことが可能である．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いい，その構成要素関数を引数とするメソッ

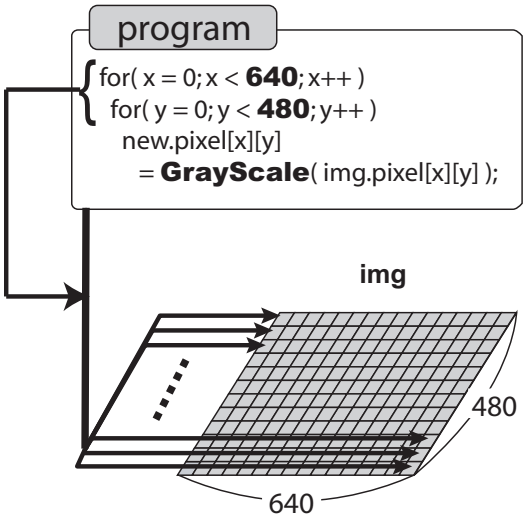

図 1: 一般的な動画像処理プログラム記述例

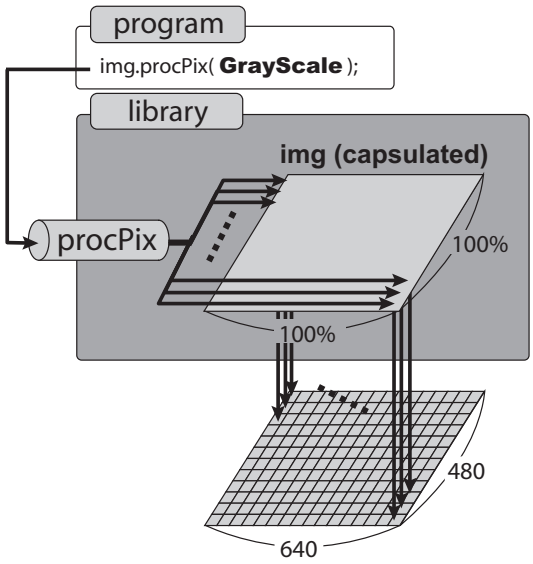

図 2: RaVioli 使用時プログラム記述例

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 2** 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．

2.2.2 リアルタイム性の保証

2.1 節で説明したように，汎用 OS 上で動画像処理に必要な CPU リソースを常に確保することは困難である．そこで，これを解決する方法として，動画像の解像度を低減させて処理量を減らすことが考えられる．RaVioli は解像度をプログラマから隠蔽したことにより，ライブラリ内で負荷に応じて処理解像度を動的に変動させることが可能になった．RaVioli で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．

ここで，空間解像度を変動させるときの処理方法を**図 3** に示す．RaVioli で空間解像度の変更を行う場合，1 フレーム上で処理する画素の間隔を示す空間解像度ストライドを大きくするか小さくすることで空間解像度の変更を行う．例えば，空間解像度を低減させる場合には，空間解像度ストライドを大きくし，処理対象画素の間隔を大きくする．具体的には図 3 に示すように，空間解像度ストライド $S_S = 1$ のとき，画像

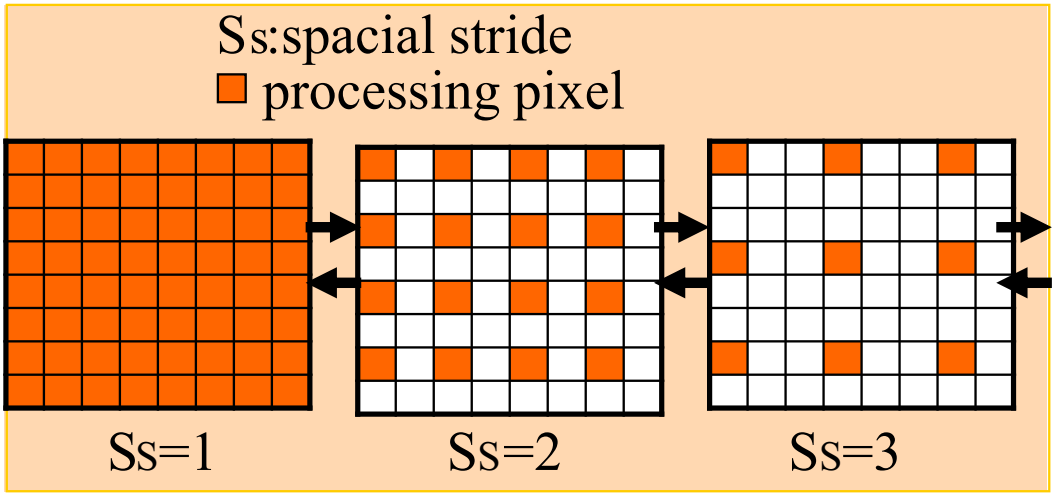

図 3: 空間解像度ストライドの変更

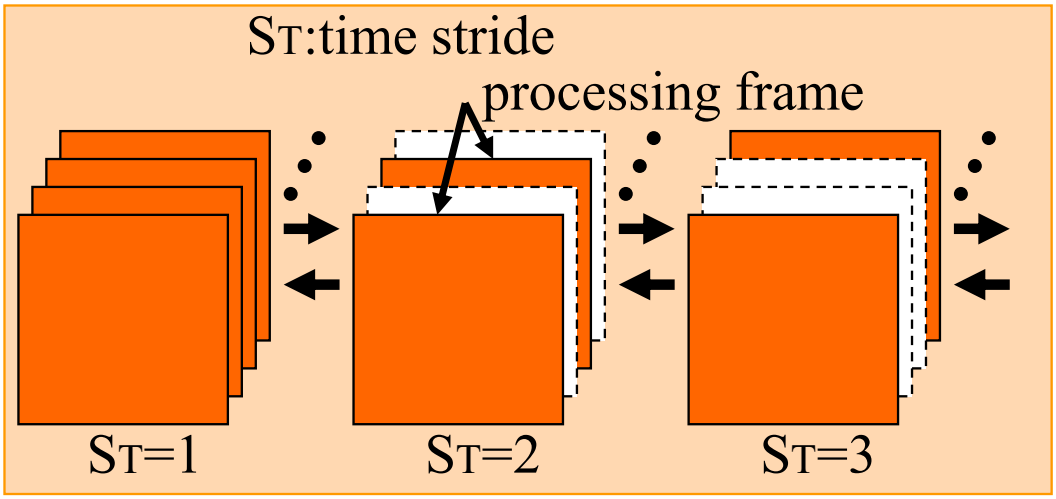

図 4: 時間解像度ストライドの変更

中の全ての画素を処理する．空間解像度ストライドを増加させ $S_S = 2$ にすると，処理対象画素は 1 つおきとなり，空間解像度が低減する．このとき，全体の処理画素数は $S_S = 1$ のときの 1/4 となる．さらに空間解像度ストライドを増加させ $S_S = 3$ とすると，処理画素数は 1/9 となる．

一方で，時間解像度を変動させるときの処理方法は**図 4** に示す通りである．空間解像度の変更と同様に，処理対象のフレームの間隔を示す時間解像度ストライドを変化させることで時間解像度の変更を行う．例えば，時間解像度を低減させる場合には，時間解像度ストライドを大きくし，処理対象フレームの間隔を大きくする． 図 4 の場合，時間解像度ストライド $S_T = 1$ のとき，入力フレーム全てを飛ばさずに処理する．時間

解像度ストライドを増加させ $S_T = 2$ にすると，処理対象フレームは 1 つおきとなり，時間解像度が低減する．このとき，全体の処理フレーム数は $S_T = 1$ のときの 1/2 となる．さらに時間解像度ストライドを増加させ $S_T = 3$ とすると，処理フレーム数は 1/3 となる．

また，プログラマは空間解像度および時間解像度に対する優先度を指定することができ，RaVioli は指定された優先度の比に応じた解像度の維持を行う．これにより，プログラマは処理内容に応じて優先度を設定するだけで目的のプラットフォームに適したアプリケーションの作成が可能となる．例えば，時間分解能の重要な処理では，時間解像度が優先されるように設定することで，RaVioli は空間解像度を重点的に低減させる．一方，顔認証などの空間分解能の重要な処理では，空間解像度が優先されるように設定することで，時間解像度が重点的に低減され，精細さの確保を行いつつリアルタイム性を実現することができる．

解像度の優先度は 2 つの値 (P_S, P_T) の組である優先度セットを指定する事で設定することができる．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，$(P_S, P_T) = (3, 7)$ と設定したとき，空間解像度ストライドと時間解像度ストライドを 7 : 3 の割合で維持しようとする．動画像処理アプリケーションを異なるプラットフォームに移植する際，プログラマはプログラムを書き直す必要がなく，優先度セットの値をそのプラットフォームに適した値にするだけでよい．その結果，プログラマはプラットフォームに対する要求性能ポイントを満たし，実時間処理を実現する動画像処理アプリケーションを容易に実装することが可能となる．

2.2.3 問題点

RaVioli では負荷に応じて，解像度を低減させて処理量を適切な量に調整する．このとき，処理する画素数やフレーム数が削減されるので，動画像処理の精度は低下する．これは解像度を低減させて，リアルタイム性を保証する際には避けられないことであるが，2 つの解像度をできるだけ高く保持することが望まれる．

そこで，RaVioil では前項で説明した優先度を設定することで，時間解像度ストライドと空間解像度ストライドの変動を制御することができる．例えば，優先度を $(P_S, P_T) = (1, 0)$ とすることで空間解像度ストライドを $S_S = 1$ に保つので，空間解像度を低減させることなく画像を処理できる．そのため，動画像処理の内容に応じて，どちらかの解像度の低減を抑えることは可能である．しかし，処理が間に合わないとき，優先度を低く設定した解像度が大幅に低減させられるため，プログラマの期待する処理結果を得ることが困難になると予想できる．

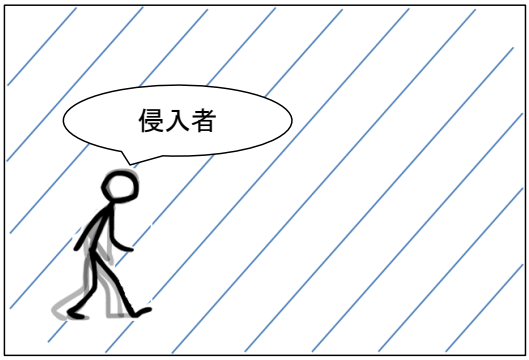

図 5: 入力フレーム

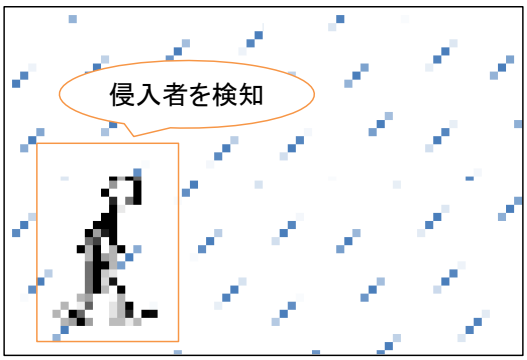

図 6: 出力フレーム

ここで，時間解像度に高い優先度を設定する場合を，侵入者検知システムを例に説明する．ここで想定する侵入者検知システムとは，入力画像から侵入者を検出し，侵入者が検出された時刻の画像をユーザに提示するシステムである．このシステムの目的は素早く行動する侵入者を見逃すことなく，かつ侵入者の顔をより詳細な画像で検出することである．このシステムでは，侵入者を見逃すことを避けるために，すべてのフレームを処理するように優先度を設定する．そのため，RaVioli はフレームの空間解像度ストライドを増加することで処理量を調整する．

ここで，このシステムを実際に動作させた場合を考える．このシステムへの入力を**図 5** に示す．図 5 は左下の領域に侵入者が現れた時のフレームである．また，左下の領域以外には侵入者はいないものとする．この入力フレームを空間解像度が大幅に低減した状態で処理すると，**図 6** のような出力が得られる．図 6 は左下の領域にいる侵入者を検知し，枠で囲う処理をした出力フレームである．空間解像度が大幅に低減しているため，侵入者の顔を詳細に検出することは困難であり，システムの目的を果たしていない．

また，空間解像度に高い優先度を設定する場合を，携帯電話などに搭載されている QR(Quick Response) コード読み取りシステムを例に説明する．まず，このシステムの動作を説明する．利用者は携帯電話のカメラを使って，QR コードを読み取る．このシステムでは，カメラ撮影の様にシャッターを押して画像を取り込むのではなく，ビデオ撮影のように読み取りたい QR コードにカメラを向けて一定間隔ごとに画像を取り込む．これには，シャッターを押すことによる画像のぶれを解消する目的がある．このように取り込まれる画像から QR コードを捉えた時に，QR コードから情報を取り出す．

このシステムの目的は，QR コードを正確に読み取り，なおかつ読み取りにかかる

時間をできるだけ短くすることである．そこで，QR コードを正確に読み取るために全ての画素を処理するようにする．そのため，RaVioli は時間解像度ストライドを増加することで処理量を調整する．しかし，時間解像度を大幅に低減させて処理を行うことで，QR コードから情報を取り出すまでに時間がかかり，リアルタイムに処理が行われていないようにユーザが感じてしまうかもしれない．

このように空間解像度や時間解像度を低減させることで，プログラマが期待した動画像処理が行えなくなる場合が存在する．そのため，処理量調整方法を変更することで，この問題を改善する．

3　提案

3.1　提案手法の着眼点

RaVioli はユーザから解像度を隠蔽することにより，動画像処理プログラムの実行時に動的に解像度を変動させることを可能にした．処理間隔である解像度ストライドを変動させて処理量を適切な量に調整し，擬似的なリアルタイム性を保証している．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には，時間解像度を空間解像度より重要とする侵入者検知システムや衝突回避システムなどと，空間解像度を時間解像度より重要とする顔認識システムや QR コード読み取りシステムなどがある．これらの動画像処理システムはリアルタイムに入力画像をキャプチャし処理を施すが，入力によっては詳細な処理を行う必要がない画像や領域が存在する．

例えば，2.2.3 項で述べた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．その他の動画像処理にも詳細な処理が必要でない画像や領域が存在すると考えられる．また，それらの画像や領域に対して詳細に処理を施すことは CPU リソースの有効活用とは言えない．RaVioli では 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調整するため，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．

また，RaVioli はユーザが記述した構成要素関数を画像全体に適用する際に，画像全体に対して空間解像度ストライドを設定しているため，全ての領域を同じレベルの詳

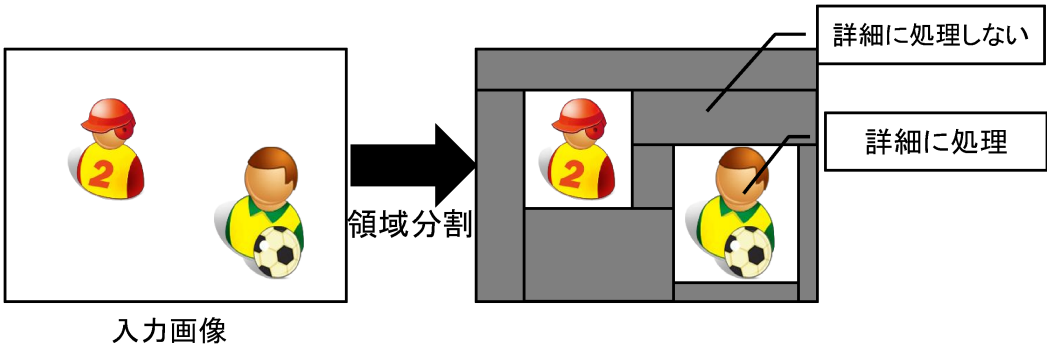

図 7: 動的な領域分割

細さで処理している．そのため，詳細な処理が必要かどうかによって領域ごとに処理量を調整することはできない．

そこで，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．入力フレーム中の詳細な処理が必要な領域は現在の解像度で領域を処理し，詳細な処理が必要でない領域は現在の解像度よりも低い解像度で領域を処理する．これにより，リアルタイムに動画像を処理する際の無駄な処理を削減することができる．

3.2 実現方法

3.2.1 領域の分割方法

リアルタイム動画像処理の入力の特徴を考慮して，入力フレーム中の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ために，1 フレームを分割して領域ごとに動画像処理を行う必要がある．ここで，領域の分割方法として，プログラム実行時に動的に領域を分割する方法と静的に領域を分割する方法の 2 通りが考えられる．また，どちらの方法で分割する場合も，各分割領域は長方形になるようにする．これは画像の大きさを幅と高さで表すことができ，各分割領域を画像処理しやすい．以下では入力画像から人物を検出するプログラムを例に 2 つの分割方法を比較する．

まず，詳細な処理が必要な領域と必要でない領域を動的に分割する場合を考える．動的に領域を分割した様子を**図 7** に示す．図 7 は入力画像が詳細な処理が必要な領域と必要でない領域に分割されている．人物が存在する領域だけを，詳細な処理が必要な領域としている．そのため，詳細な処理が必要でない領域が多く，無駄な処理を最大限に削ることができている．

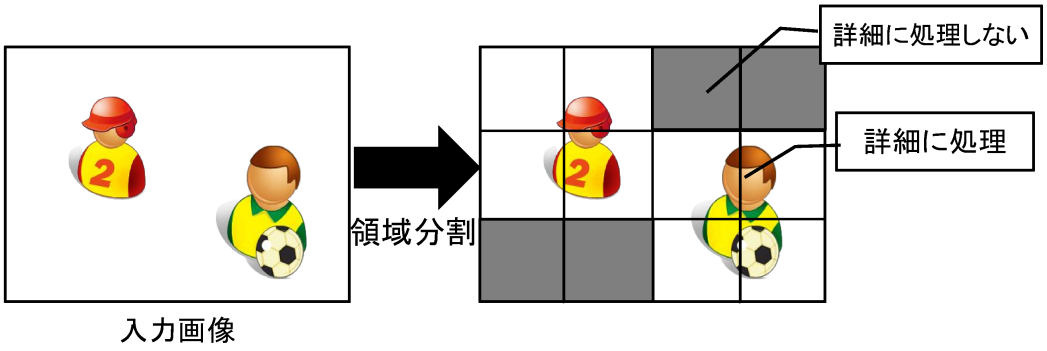

図 8: 静的な領域分割

しかし，これを実現するには詳細な処理が必要な領域と必要でない領域を正確に判定しなければいけないため，判定のためにコストがかかり過ぎてしまい，処理量を削減するという目的を達成することが困難になる．また図 7 の分割された領域は領域の大きさが不揃いであり，各領域の処理量を判断しにくい．これは今後，分割領域単位で処理を並列化する場合に問題になりうる．

一方で，静的に領域を分割する場合は**図 8** に示す通り，画像を均一な大きさに分割する．このとき，領域の大きさが決まっているため領域の判定コストはかからない．そのため，各領域ごとに詳細な処理が必要かどうかを判定するだけでよい．また，各領域で詳細な処理が必要かどうかを判定する方法として，構成要素関数のように判定のための関数を記述するか，ライブラリが提供する判定関数を用いるかをプログラマが選択できるようにする．これらの判定関数については 4.4 節で詳細を述べる．

しかし，動的に分割した場合と比べると無駄な処理を十分に削減できていない．これは，人物が複数の領域を跨いで存在しているからである．静的に領域を分割する場合，この問題は避けられない．しかし，均一な大きさに領域を分割することには，各領域にかかる処理量を容易に計算できるという利点もある．

このように，詳細な処理が必要な領域と必要でない領域をプログラム実行時に動的に決めることは困難である．一方，静的に領域を分割した場合，処理を並列化する際に処理の分担を行いやすくなる．そのため本提案では，大きさが均一な領域になるように，静的に画像を分割する．そして，各領域で詳細な処理が必要かどうかを判定し，領域の解像度を設定する．

3.2.2 動作モデル

前項では，動画像処理時の無駄な処理を削減するために，画像をあらかじめ均一な領域に分割して，領域ごとに解像度を変動させて処理量を調整することを提案した．そ

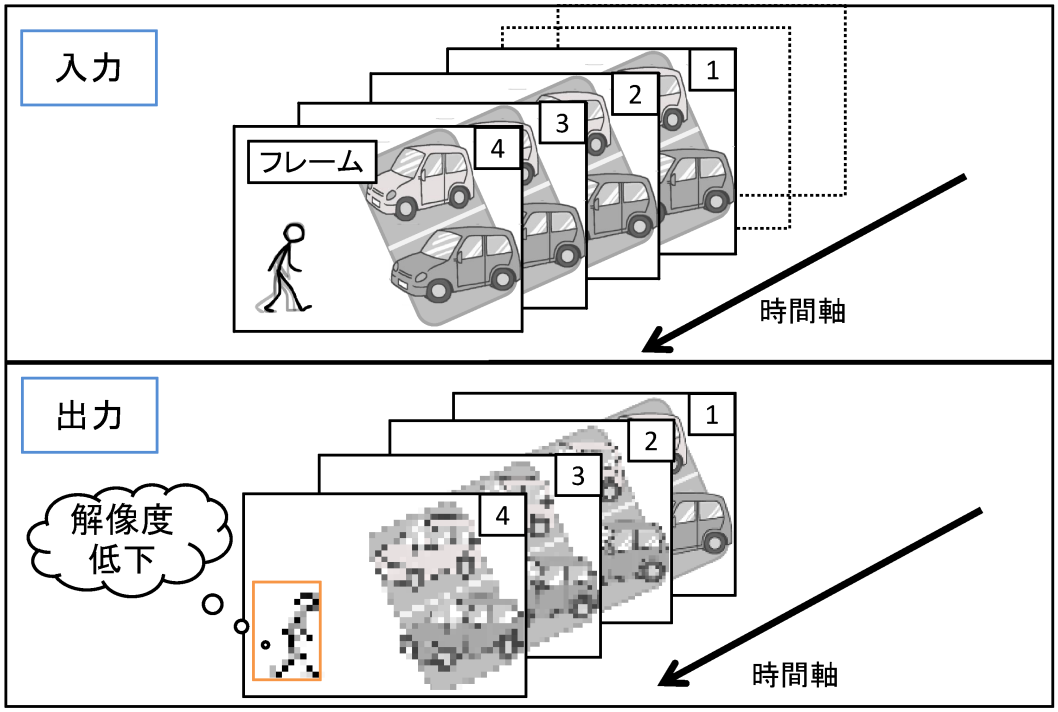

図 9: RaVioli を用いた侵入者検知

こで，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ことにより，提案手法の動作モデルを説明する．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全てのフレームを処理する．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．

まず，既存の RaVioli を用いて侵入者検知を行う様子を**図 9** に示す．図 9 の上段はフレーム 1，フレーム 2，フレーム 3，フレーム 4 の順にキャプチャされた入力を示している．また，下段はその入力に対して画像処理を施した出力を示している．フレーム 1，フレーム 2，フレーム 3 は前のフレームから変化がなく，フレーム 4 で初めて侵入者が現れたとする．

既存の RaVioli では，全ての入力に対して，常に詳細な処理を行うため，フレーム 1 のような前入力からの変化がないフレームに対しても詳細に処理を行う．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 で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 を処理すると，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

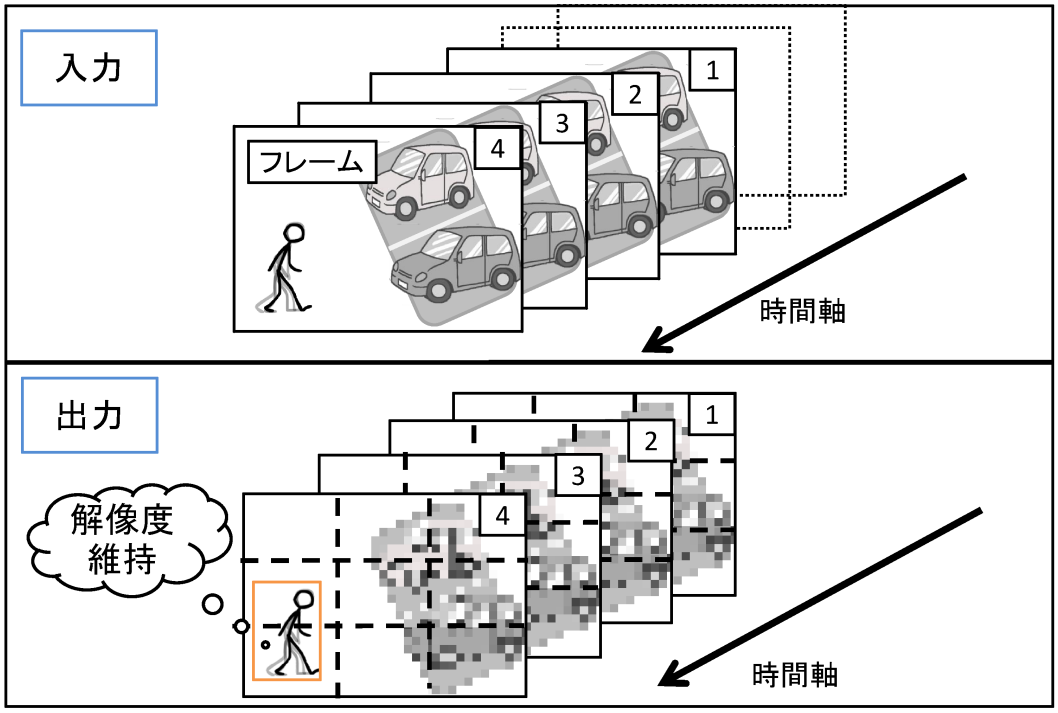

図 10: 提案手法を用いた侵入者検知

することは難しい.

次に，提案手法である領域別に処理量を調整する RaVioli を用いて侵入者検知を行う場合，まずカメラからの入力画像をプログラマが指定した分割数に分割する．そして，その各領域で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場合は現在の解像度ストライドを領域に設定し，詳細な処理が必要でない場合は現在の解像度ストライドよりも大きな値を領域に設定する．そして，設定された解像度ストライドで各領域に処理を施す.

提案手法を用いた侵入者検知の様子を**図 10** に示す．図 10 は先程の図 9 と同じ入力に対して，画像処理を施した様子を示している．入力フレーム内の垂直，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，フレーム 2，フレーム 3 に対して，既存の RaVioli では常に詳細な処理を施していたが，提案方式ではフレームに設定されている空間解像度ストライドよりも大きな値で，詳細な処理が必要でない領域を処理する．これにより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，空間解像度ストライドを増大させることなく処理できる．ここで，侵入者が現れるフレーム 4 を処理すると，左下の詳細な処理が必要な領域の解像度が低減していないため，侵入者を詳細に検出するこ

とが可能である．

このように，提案動作モデルでは詳細な処理が必要でない領域に対する処理を削減することにより，利用可能な CPU リソース量が減少した時の解像度低減をできるだけ抑えることを目指している．例で述べたような効果を得るためには動画像処理にかかる時間に対して，詳細な処理が必要かどうかを判定する処理にかかる時間が十分に小さくなければならないため，判定関数にはできるだけ軽量かつ正確な処理が求められる．

4　実装

まず，領域別処理量調整を行う際にどのようにプログラムを記述するのかを述べる．次に，領域別処理量調整を実現するために実装する，分割領域を管理する RV_TileImage クラスについて説明する．そして，分割領域で画像処理を行うために必要である，高階メソッドの変更について説明し，詳細な処理が必要かどうかを判定する関数の扱いについて述べる．最後にユーザが定義したプログラムがどのように実行されるのかを説明する．

4.1　ユーザインタフェース

領域別に処理量を調整する方法を用いてプログラムを記述するために，プログラマは既存の RaVioli でのプログラム記述方法との変更点を理解しなければいけない．そこで，まずは RaVioli でのプログラム記述方法を説明する．そして，領域別に処理量を調整する際のプログラム記述方法について説明する．また，プログラム記述に関して，ユーザが既に記述した main 関数や構成要素関数をできるだけ変更しないようにインタフェースを実装する．

まず，既存の RaVioli を用いたプログラムの記述例を**図 11** に示す．プログラマは main 関数と構成要素関数 Program1，Program2 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 setParam 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，動画像のキャプチャを開始する (11 行目)．カメラからの入力フレームは video が持つリングバッファに一旦格納される．次に，動画像用の高階メソッド StreamProc に構成要素関数 Program1 を渡す (12 行目)．StreamProc ではリングバッファから画像を管理する RV_Image クラスのインスタンスを生成し，そのインスタンスに各解像度ストライドを設定し，構成

```
 1: void Program2(RV_Pixel* pixel){
 2: ...
 3: }
 4: void Program1(RV_Image* Frame){
 5:  Frame->procPix(Program2);
 6: }
 7:
 8: int main(int argc, char* argv[]){
 9:  RV_Streaming video;
10:  video.setParam(7,3); //優先度の設定
11:  video.RunCapture(); //キャプチャ開始
12:  video.StreamProc(Program1); //動画像の高階メソッド
13:  return 0;
14: }
```

図 11: 既存の RaVioli を用いたプログラム記述例

要素関数 Program1 を呼び出す．また，Program1 では RV_Image クラスの高階メソッド procPix を用いて，Program2 の処理を画像全体に施す (5 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．

次に，領域別に処理量を調整する際のプログラムの記述例を**図 12** に示す．既存の RaVioli を用いて記述するプログラムとの違いは図 12 の 10 行目で行う判定関数の設定およびその判定関数 Program3 の定義，図 12 の 11 行目で行う領域数の設定である．プログラマは分割領域数として行数と列数を指定する．このとき，指定された行数と列数に応じて画像を分割する．

このように，ユーザが記述した画素やフレームに対する処理を記述した構成要素関数を変更することなく，領域別に処理量を調整するインタフェースを実装する．ここで，領域を分割し，判定関数によって各領域に設定された解像度ストライドで画像全体を処理するにはライブラリ内に必要な実装がある．以降，分割領域を扱うための新たに追加する RV_TileImage クラスについて次節で述べ，そのクラスを用いて画像処理を行う際に変更される高階メソッドについて 4.3 節で述べる．また，判定関数に関するライブラリ仕様についても 4.4 節で述べる．

```
 1: void Program2(RV_Pixel* pixel){...}
 2: void Program1(RV_Image* Frame){...}
 3: int  Program3(RV_Image* N, RV_Image* B){
 4:  ...//判定処理
 5: }
 6:
 7: int main(int argc, char* argv[]){
 8:   RV_Streaming video;
 9:   video.setParam(7,3); //優先度の設定
10:   video.setDetFunc(Program3);//判定関数の設定
11:   video.setTileNum(3,4);//3行4列の領域に分割
12:   video.RunCapture(); //キャプチャ開始
13:   video.StreamProc(Program1); //動画像の高階メソッド
14:   return 0;
15: }
```

図 12: 提案方式を用いたプログラム記述例

4.2 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，静的に均一な領域に分割して処理を行う．そこで，その分割領域を管理する RV_TileImage クラスを新たに追加する．RV_TileImage クラスの概略を**図 13** に示す．RV_TileImage クラスはメンバとして，領域の左上の位置である開始座標，領域の幅と高さ，判定関数のポインタ，各解像度ストライドを持つ．

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，画像全体を処理する．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，解像度ストライドを設定し，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．RV_TileImage クラスの各高階メソッドは RV_Image クラスの各

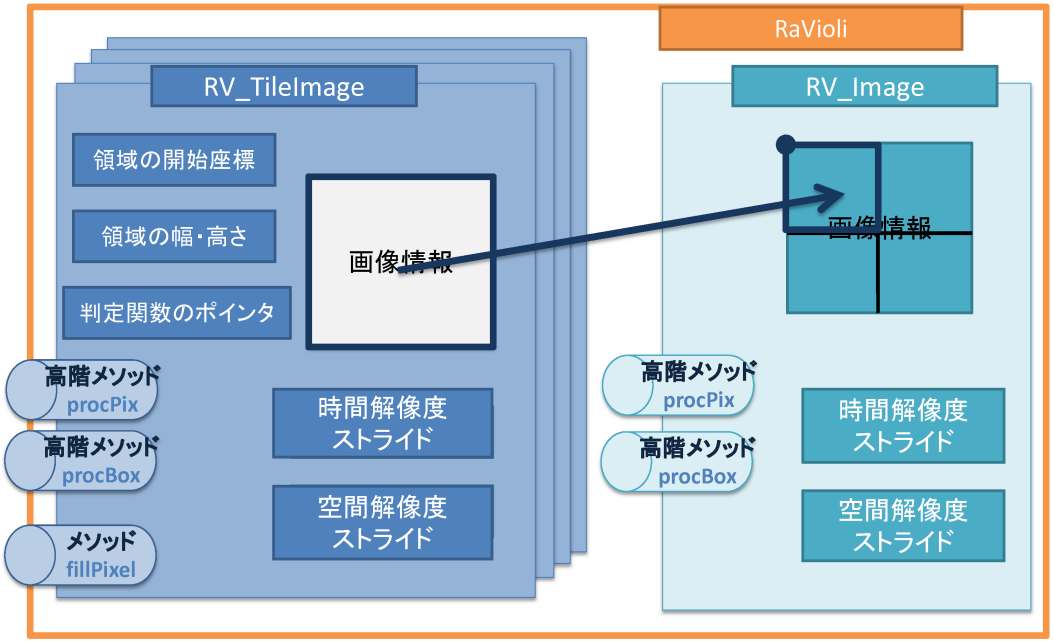

図 13: RV_TileImage の概略

高階メソッドと同様の処理を行う．

さらに，メソッド fillPixel を RV_TileImage に実装する．このメソッドは RV_Image インスタンスに設定されている空間解像度ストライド値よりも大きな値で領域を処理した際に使用する．以下に詳細を示す．

void fillPix(int ImageGRAIN,int TileGRAIN)

メソッド fillPixel は RV_Image インスタンスの持つ空間解像度ストライド ImageGRAIN と RV_TileImage インスタンスの持つ空間解像度ストライド値 TileGRAIN を引数にとる．TileGRAIN の値が ImageGRAIN の値と等しい時，画像中の処理対象となるピクセルが全て処理されているため，何も処理をしない．TileGRAIN の値が ImageGRAIN より大きい時，画像中の処理対象となるピクセルが全て処理されていないため，処理されたピクセルを使って，処理されていないピクセルを補間する．TileGrain の値が ImageGRAIN の値よりも小さい場合は存在しないため，エラーとなる．

4.3 高階メソッドの変更

4.1 節で述べた通り，本提案ではプログラマが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な仕様にする．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のを，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しなければならない．

そこで，RV_Imageクラスの高階メソッドの変更方法を高階メソッドprocPixを用いて説明する．まず，現在のprocPixと同じ処理をするメソッド_procPixを作成し，procPixの処理内容を変更する．procPixではまず分割領域別に処理を行うかを判定する．行わない場合は先程説明したRV_Imageインスタンスの_procPixを呼び出す．一方で，分割領域別に処理を行うときは，各RV_TileImageインスタンスの処理範囲を適切に指定し，分割領域数と同じ数のRV_TileImageインスタンスの高階メソッド_procPixを呼び出す．さらに，各RV_TileImageインスタンスでメソッドfillPixelを実行する．

このようにその他の高階メソッドも変更する．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，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．

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を用いて画像を領域別に処理する時，画像は**図14**のように1，2，3，4の領域に単純に分割される．なお，図14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．そのため，procPixと同様に2枚の画像は**図15**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繰り返し適用する高階メソッドである．分割の様子を**図16**に示す．図16の1の領域は図14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ている．

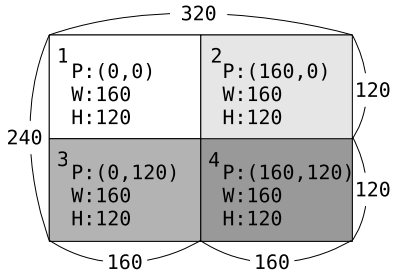

図 14: procPix の分割領域

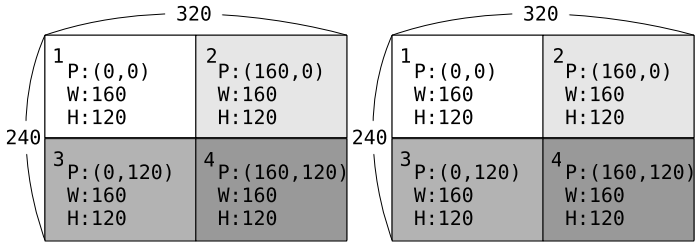

図 15: procImgComp の分割領域

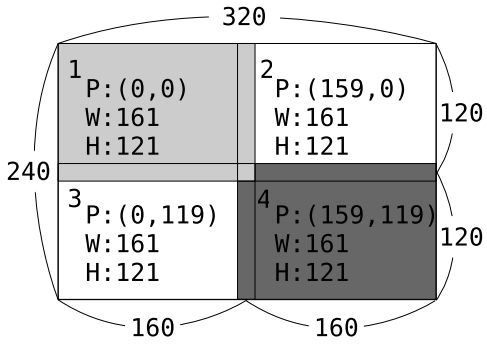

図 16: procNeighbor の分割領域

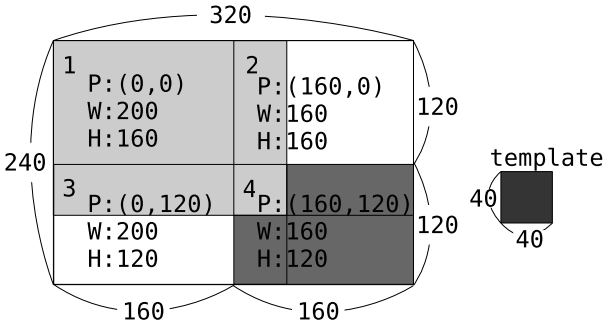

図 17: procBox の分割領域

procBox

procBox はテンプレート画像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．分割の様子を**図 17** に示す．図 16 の 1 の領域は図 14 の 1 の領域の右端と下端からテンプレート画像を含むように分割される．同様に 2 の領域は下端，3 の領域は右端からテンプレート画像を含むように分割されている．

4.4 判定関数

領域別に処理量調整を行うために，各領域で詳細な処理が必要な領域であるかどうかを判定しなければいけない．本提案では RaVioli の構成要素関数の様に判定関数をプログラマが記述し，その判定関数を引数とする RV_Streaming クラスの高階メソッド setDetFunc を実装する．この時プログラマは，判定関数に渡される引数によって 3 通りの判定関数を記述することができる．ここで，3 種類の判定関数とは，現在の処理フレームとそのひとつ前の処理フレームを引数とする判定関数，現在の処理フレームのみを引数とする判定関数，領域の座標 x,y を引数とする判定関数である．また，RaVioli は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．このとき，ライブラリ内のどの判定関数を使用するかを指定する際にもメソッド setDetFunc を用いる．

まず，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．先程述べたように判定関数には 3 種類の記述方法があるので，その判定関数を引数とする高階メソッド setDetFunc は 3 種類存在する．以下に各高階メソッドを示す．

void setDetFunc(int(*DetProgram)(RV_Image* Fnow,RV_Image* Fbfr))

現在の処理フレーム Fnow と 1 つ前の処理フレーム Fbfr を用いる判定関数 DetProgram 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のポインタを RV_TileImage の持つ関数ポインタに設定する．また，プログラマは 2 枚の画像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．

void setDetFunc(int(*DetProgram)(RV_Image* Fnow))

現在の処理フレーム Fnow を用いる判定関数 DetProgram 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のポインタを RV_TileImage の持つ関数ポインタに設定する．また，プログラマは 1 枚の画像を用いて，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．

void setDetFunc(int(*DetProgram)(int x,int y))

分割領域の座標を用いる判定関数 DetProgram 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のポインタを RV_TileImage の持つ関数ポインタに設定する．また，プログラマは領域の位置を用いて，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．この判定関数は，あらかじめどの領域を詳細に処理する必要があるかわかっている場合に用いる．例えば，固定カメラを用いた監視システムで，入力画像の中で出入口が存在する領域を詳細に処理する場合などが考えられる．

また，プログラマは判定関数を記述しなくてもライブラリが提供する判定関数を用いることもできる．そこで，ライブラリが提供する判定関数を使用する際の指定方法と用意されている判定関数について説明する．

void setDetFunc(int func_num)

ライブラリが提供する判定関数を使用する際に指定するためのメソッドは，先程

と同じ setDetFunc を用いる．プログラマはライブラリが提供する判定関数に対応する整数 func_num を引数としてメソッド setDetFunc に与える．ライブラリ内には現在以下の判定関数が実装されている．

int FrameDiff(RV_Image* Fnow, RV_Image* Fbfr)

setDetFunc を引数 $func\_num = 1$ で呼び出すと，この FrameDiff が RV_TileImage の持つ関数ポインタに設定される．この判定関数は現在の処理フレーム Fnow と 1 つ前の処理フレーム Fbfr の差分をとり，フレーム間に変化があるかどうかを調べる．処理フレームに変化がある場合 1 を返し，変化がない場合 0 を返す．入力フレームにあまり変化が起こらない場合に有効な判定関数である．

int right(int x, int,y)

int left(int x, int y)

int top(int x, int y)

int bottom(int x, int y)

int center(int x, int y)

setDetFunc を引数 $func\_num = 2, 3, 4, 5, 6$ で呼び出すと，順に right，left，top，bottom，center が RV_TileImage の持つ関数ポインタに設定される．これらの判定関数はそれぞれ分割領域の右端の列，左端の列，上端の行，下端の行の領域と，一番外側の領域以外の領域を詳細に処理が必要であると判定する．あらかじめ詳細に処理したい領域が決まっている場合に用いる．

4.5 ユーザプログラム処理の流れ

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．プログラマが指定した分割数に従って画像を分割する．この様子を**図 18** に示す．プログラマが 2 行 2 列の分割数を指定したときの様子である．分割数と同じ数の RV_TileImage インスタンスが生成される．このとき，各分割領域は 4 つの RV_TileImage インスタンスによって処理される．次に，各分割領域で詳細な処理が必要かどうかを判定し，画像処理を行う．**図 19** にその様子を示す．図 19 は左上と右下の領域は解像度を低減させずに画像処理し，右上と左下の領域は解像度を低減させて処理している．このとき，右上と左下の領域は処理が行われていないピクセルが存在することが図 19 からわかる．そのため，提案手法ではこのように処理が行われなかったピクセルを処理が行われたピクセルを用いて補間する．この処理が行われると**図 20** 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

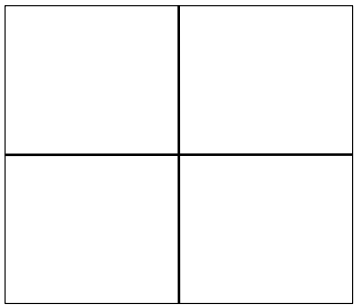

図 18: 画像の分割

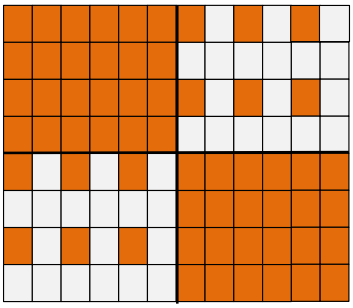

図 19: 領域別画像処理

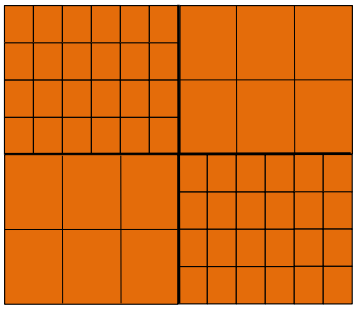

図 20: 最終的な出力

表 1: 評価環境

CPU	Intel Core2 Ex. Quad
クロック周波数	3.0GHz
Memory	8GB
OS	Solaris10
Compiler	Sun Studio 12 (Sun C++ 5.9)

し処理量を調整することが実現できる．

5 評価

表 1 に示す評価環境で提案手法を評価した． まず，提案手法を用いて画像を処理する際の処理削減による高速化の効果を確かめる．そこで，1 枚の画像に対してグレースケール化の処理を施すプログラムを使用して，既存の RaVioli を用いた場合と提案手法を用いた場合を比較した．画像には幅が 320 ピクセル，高さが 240 ピクセルであるものを用いた．結果を**図 21** に示す．図 21 は左から既存の RaVioli を用いて画像全体を空間解像度ストライド $S_S = 1$ で処理するのにかかった時間，提案手法を用いて領域別に解像度を変動させて処理するのにかかった時間を示している．このとき，提案手法では詳細な処理が必要でない領域数を 0∼12(領域数) まで変化させて処理にかかった時間を計測した．なお，詳細な処理が必要でない領域には現在の解像度ストライド値 ($S_S = 1$) より 4 大きいストライド値 ($S_S = 5$) を設定した．図 21 では詳細な処理が必要でない領域数が 0∼3 のとき，提案手法の処理時間が既存の RaVioli よりも大きくなった．また，処理を削減する領域を大幅に増加させても，処理時間はあまり変わっていない．これは，評価に用いたグレースケール化の処理が軽量であるため，削減できる処理時間に対して，領域毎に処理をするためにかかるオーバヘッドが大きいため

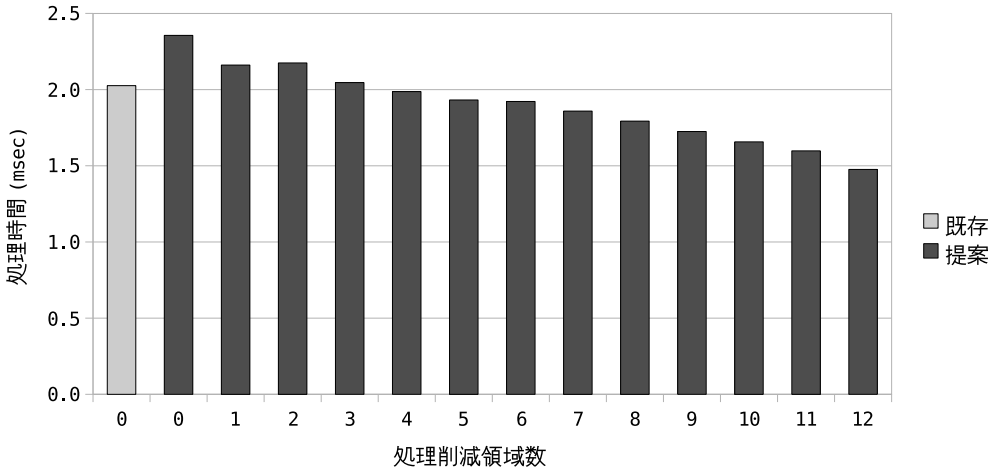

図 21: 処理量の少ない画像処理

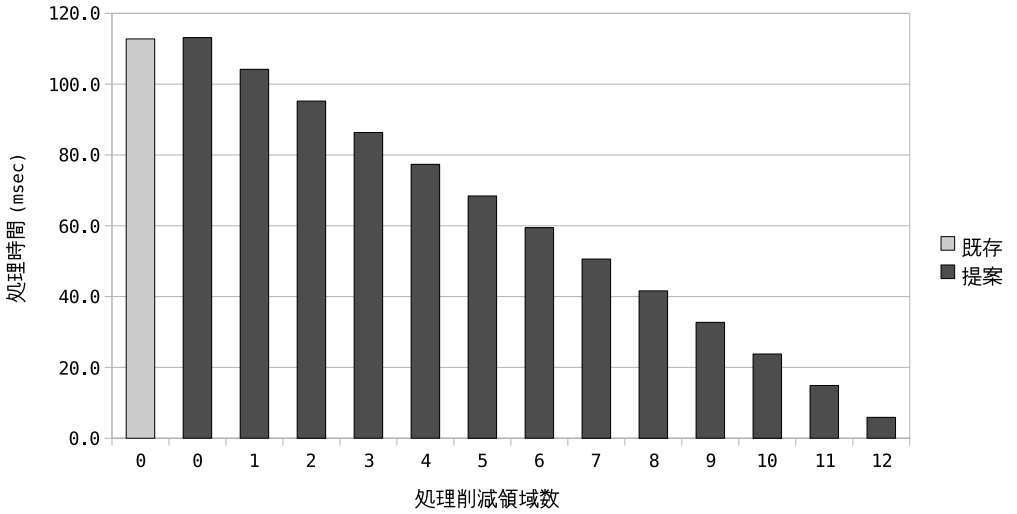

図 22: 処理量の多い画像処理

であると考えられる．

そこで，処理量を増加させて評価を行なった結果を**図 22** に示す．図 22 では詳細な処理が必要でない領域数が増えるにつれて，処理にかかる時間が十分に減少していることがわかる．この時，図 22 中の左の 2 つ処理時間を比べるとほぼ差はないため，領域毎に処理をするためにかかるオーバヘッドは処理にかかる時間に対しごく僅かであることがわかる．図 21 と図 22 より，処理にかかる時間に対して領域別に処理するためにかかるオーバヘッドが小さい時，全体の処理時間を削減できることを確認した．また，図 21 の画像処理では 1 秒間におよそ 500 フレームを処理できるため，RaVioli を

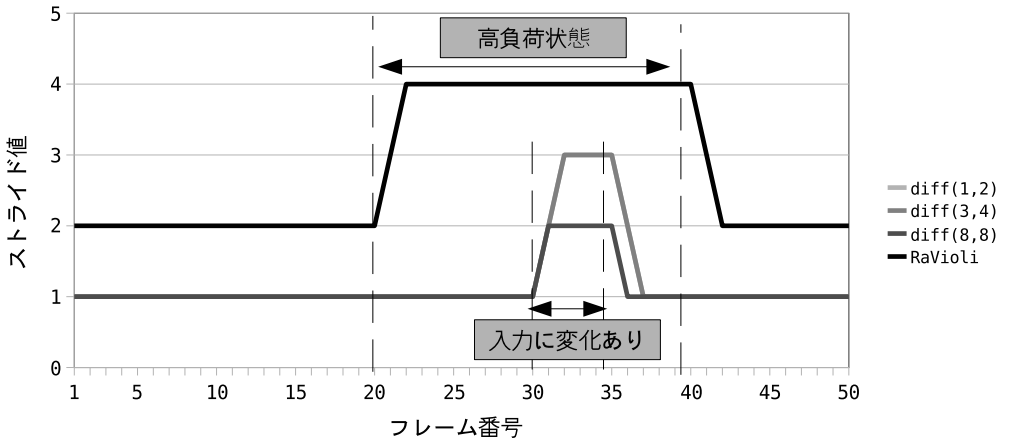

図 23: ストライド値の変動

用いて動画像処理を行う際にフレームの解像度は低減しない．本提案の目的は解像度低減をできるだけ抑えることであるので，図 21 のような場合に処理時間が増加してしまうことは問題ではない．このような場合には，ユーザは分割せずに画像を処理することを選択すると考えられる．

次に，実際に動画像を処理して解像度の変動と出力画像を比較した．今回，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で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に対して，1/30 秒ごとに処理を施すことで動画像処理を擬似的に再現した．この方法を用いて，50 フレームを処理した．提案手法を用いた RaVioli と既存の RaVioli での空間解像度ストライド値の変動の様子を**図 23** に示す．このとき，提案手法の判定関数にはライブラリが提供する FrameDiff を用いた．また，分割領域数による違いを調べるために，分割方法を 1 行 2 列，3 行 4 列，8 行 8 列と変化させた．図 23 中ではそれぞれ，diff(1,2)，diff(3,4)，diff(8,8) とした．また，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．さらに，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 フレームを処理する際の処理量を通常の 5 倍とした．これは利用可能 CPU リソース量の減少を処理時間の増加で表現している．

この結果より提案手法では入力に変化があるとき (30~34 番目のフレーム) の詳細な処理が必要な領域の空間解像度ストライド値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る．なお，diff(1,2) と diff(3,4) は全く同じ解像度変動を示したため，図 23 では 2 つのグラフは重なっている．

さらに，提案手法と既存手法での出力画像 (34 番目のフレーム) を確認した．提案手

図 24: diff(1,2) を用いた時の出力

図 25: diff(3,4) を用いた時の出力

図 26: diff(8,8) を用いた時の出力

図 27: 既存 RaVioli の出力

法を用いて処理された画像を分割数の少ない順に**図 24**，**図 25**，**図 26** に示し，既存の RaVioli を用いて処理された画像を**図 27** に示す．図 27 が画像全体を同じレベルの詳細さで処理しているのに対して，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，画像を 8 行 8 列に分割した図 26 は，粗く処理された領域が画像全体に占める割合が図 24 や図 25 に対して大きい．そのため，8 行 8 列に分割した際に最も解像度の低減を抑えることができている．しかし，分割数を増やすことで，各領域の処理対象ピクセルが減少するため判定関数の精度が低下すると考えられる．この問題を解決することは今後の課題である．

6　おわりに

本論文では，まず，動的に使用可能なリソース量が変動するような環境において動的に解像度を変動させることで処理量を減らし，リアルタイム性の保証を行う解像度

非依存型動画像処理ライブラリ RaVioli について述べた．また，RaVioli の処理量調整方法には問題点があることを示した．

そして，画像を分割し領域別に処理量を調整することを提案し，RaVioli に実装した．分割領域を処理するためのクラス RV_TileImage クラスを新たに追加し，RV_TileImage を用いて処理するために RV_Image クラスの高階メソッドを変更した．また，プログラマが記述した画像処理のプログラムの内容を変更することなく，領域別に処理量を調整することを可能にした．領域別に処理量を調整することによって画像処理にかかる処理時間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，プログラム実行時に分割領域数や分割するかどうかを変更させることが考えられる．処理内容によっては画像を領域に分割して処理を行わない方が処理を高速に行えることがあるので，プログラム実行中に処理に適した方法に切り替えることで最良の結果を得ることを可能にする．

また，分割領域単位での並列化も考えられる．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抑えることができると考えられる．

謝辞

本研究のために，多大な御尽力を頂き，御指導を賜わった名古屋工業大学の松尾啓志教授，津邑公暁准教授，齋藤彰一准教授，松井俊浩助教に深く感謝致します．また，本研究の際に多くの助言，協力をして頂いた松尾・津邑研究室および齋藤研究室の方々に深く感謝致します．特に，研究に関して貴重な意見を下さった大野将臣氏，桜井寛子氏，稲葉崇文氏，浅井宏樹氏に感謝致します．

参考文献

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

[2] 岡田慎太郎, 桜井寛子, 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 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）, Vol. 2, No. 1, pp. 63–74 (2009).

[3] Sakurai, H., Ohno, M., Tsumura, T. and Matsuo, H.: RaVioli: a Parallel Video Processing Library with Auto Resolution Adjustability, *Proc. IADIS Int'l. Conf.*

Applied Computing 2009, Vol. 1, pp. 321–329 (2009).

[4] Köthe, U.: *VIGRA - Vision with Generic Algorithms*, 1.6.0 edition (2008).

[5] Intel Corp.: *Open Source Computer Vision Library* (2001).