卒業研究論文

動画像処理ライブラリ RaVioli における
入力の変化を考慮した領域別処理量調整手法

指導教員　　津邑 公暁 准教授
松尾 啓志 教授

名古屋工業大学 工学部 情報工学科
平成 21 年度入学　21115135 番

松永 拓也

平成 25 年 2 月 12 日

動画像処理ライブラリ RaVioli における
入力の変化を考慮した領域別処理量調整手法

松永 拓也

内容梗概

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が盛んに開発されている．また，汎用計算機の性能向上や価格低下により，高性能な計算機環境を容易に入手可能になってきている．そのため今後，汎用計算機上でリアルタイム動画像処理システムが盛んに開発されると予想される．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 リソース量を常に確保することは困難である．

この問題を解決するため，動画像処理ライブラリ RaVioli が提案されている．RaVioli では利用可能な CPU リソース量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，自動的に解像度を低減させることで処理量を調整し，リアルタイム性を保証している．しかし，解像度の低減により，プログラマが求める処理結果が得られなくなる可能性がある．つまり，動画像処理の処理精度が低下してしまうという問題がある．この問題はリアルタイム性を保証するために避けられないが，処理精度は可能な限り高く保ちたい．

そこで，リアルタイム動画像処理における，入力動画像の各領域に対する重要度に応じて解像度を自動的に変動させる新しい処理量調整手法を提案する．提案手法では，入力動画像に対しプログラマが詳細に処理したい重要な領域における処理精度の低下を抑制する．リアルタイム動画像処理における入力動画像には，詳細に処理せずとも処理精度に大きな影響を与えない領域が部分的に含まれている場合がある．このため，そのような領域は低い解像度で処理することで処理量を削減できる．そこで，動画像ストリームを部分ストリームに分割し，部分ストリーム毎に解像度を保持，変動できるように RaVioli を拡張する．さらに，解像度低下を一定の割合で抑制する領域を優先領域として設定する．これにより時々刻々と変化する入力動画像における重要な領域の移り変わりに対応する．

提案手法を実装し，ベンチマークプログラムを用いて評価した．既存の RaVioli と提案手法を実装した RaVioli でそれぞれベンチマークプログラムを実行し，解像度の変動と出力画像を比較した結果，提案手法を用いた際に既存の RaVioli と比べ重要な領域の解像度低下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
入力の変化を考慮した領域別処理量調整手法

目次

1 はじめに

空港や工場などで侵入者や不審物を検出するシステムや，炎や煙などを認識して火災を検知するシステム，自動車走行時の衝突回避システムなどリアルタイム性が重要となるシステムが盛んに開発されている．一方で，汎用計算機の高性能化と価格低下により，高性能な計算機環境を容易に入手可能になってきている．これらのことから，今後汎用計算機上でリアルタイム動画像処理システムが盛んに開発されることが予想される．

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システムの実現はいまだに困難である．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 リソース量の変動が挙げられる．

そこで，汎用 OS 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**（**R**esolution-**A**daptable **V**ideo and **I**mage **O**perating **Li**brary）[1, 2] が提案されている．RaVioli では利用可能な CPU リソース量に応じて，**空間解像度**(1 フレームあたりの画素数) または**時間解像度**(フレームレート) を動的に変動させることで処理量を調整する．この方法では，処理精度を犠牲にして処理の大幅な遅れを回避し，リアルタイム動画像処理を擬似的に保証している．

一般に，このように動的に解像度を変動させる場合，プログラマは 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を記述しなければならない．しかし，これらの変動を意識しながら動画像処理アプリケーションを開発することはプログラマにとって大きな負担となる．そこで，RaVioli はプログラマから画像の幅や高さ，動画像のフレームレートを隠蔽する．これにより，ライブラリ内で処理対象とする画素数やフレームレートを制御し，出力画像の解像度を動的に変動可能とする．また，人間の映像認識過程には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，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，出力画像の解像度を変動させて処理量を調整することには限界がある．解像度を大幅に低減させてしまうと，動画像処理の処理精度も大幅に低下してしまい，プログラマが期待する処理結果が得られなくなる可能性がある．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，処理精度の低下は避けられないが，可能な限り処理精度を高く保ちたい．

そこで本研究では，入力の特徴に基づく処理量調整手法を提案する．提案手法では，リアルタイム動画像処理の入力に注目する．この入力には，プログラマが施したい処

理にとって重要な領域と重要でない領域が存在し，それらの領域は時々刻々と移り変わっていく．ここに着目し「重要な領域」,「重要でない領域」，そして「重要な領域へと変化することが予想される領域」のそれぞれの領域で別々に解像度を変動させることで，重要な領域における処理精度の低下を抑制する．

本論文では以下，2 章で本研究の関連研究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 章でその実装方法について説明する．5 章では提案手法の評価とそれに対する考察を述べる．最後に 6 章で本論文の全体をまとめる．

2 背景

本章では，まず関連研究について取り上げる．そして，本研究の対象となる動画像処理ライブラリ RaVioli についての概要を説明する．

2.1 関連研究

本節では関連研究として，まずリアルタイム動画像処理について取り上げる．その後，動画像処理のためのライブラリおよびプログラミング言語について取り上げる．

2.1.1 リアルタイム動画像処理

リアルタイム動画像処理アプリケーションが普及し，盛んに研究されている．例えば，Garcia-Martin ら [3] は動画像監視システムで用いられる動物体を検出するアルゴリズムを提案している．また，Kim ら [4] は火災の早期検出手法を提案し，Lin ら [5] は目の位置のリアルタイム検出アルゴリズムを提案している．

前章で述べたように，これらのリアルタイム動画像処理を汎用計算機上で実現することは困難である．これは，汎用 OS 上では複数プロセスの並行実行により利用可能な CPU リソース量が変動するためである．この問題の解決策の 1 つとして，利用可能な CPU リソース量に応じて処理量を調整するという方法がある．このような手法の 1 つとして，Imprecise Computation Model[6] がある．これは計算時間の長さに応じて動画像の処理精度を変化させるモデルである．また，このモデルに基づいて処理精度および処理時間に関する知識を利用し，適切なアルゴリズムを動的に選択するアーキテクチャも提案されている [7]．しかしこの方法では，計算負荷の異なるアルゴリズムに基づく処理を複数実装する必要があり，プログラマの負担は大きくなる．一方，RaVioli を用いる場合，この利用可能な CPU リソース量に応じた処理量の調整は，任意のアプリケーションに自動で適用できる．

2.1.2 動画像処理のためのライブラリやプログラミング言語

リアルタイム動画像処理アプリケーションが盛んに研究される一方で，多くの画像処理および動画像処理用ライブラリがこれまでに提案されている．例えば，VIGRA[8], OpenCV[9, 10] はよく知られた画像処理ライブラリである．VIGRA は C++の STL と同様にテンプレートを用い，プログラマに画像処理を抽象的に記述できる環境を提供している．また，OpenCV 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．これらのライブラリは画像処理プログラムの記述を容易にするが，これらを用いて処理量を調整可能なプログラムを記述することは困難である．

また，金井ら [11] は数式エディタを用いて記述できる独自の言語を提案し，画像処理プログラミングを抽象化している．この記述言語では処理単位となる画素配列の大きさを定義し，その配列の要素に処理を施すことでループレスな記述ができる．この点は RaVioli と似ている．一方で，この記述言語はプログラマが処理したい画像の画素数を明示的に指定する必要があるため，動的に解像度を変動させ処理負荷を調整するようなプログラムの作成は困難である．他にも，Halide[12] という画像処理プログラミング言語が提案されている．Halide のプログラムでは画像処理アルゴリズム部分と並列化手順部分を分けて記述する．これによりプログラマは画像処理アルゴリズムに変更を加えることなく，最適な並列化スケジューリングを模索できる．一方で，これは画像処理の知識だけでなく，並列化に関する知識がプログラマに必要であることを意味する．

これに対し，動画像処理ライブラリ RaVioli[1, 2] のアプローチはこれまでに説明した画像および動画像処理のためのライブラリやプログラミング言語とは全く異なる．RaVioli はプログラマから解像度という概念を隠蔽することで，より直感的な画像処理プログラミングを実現する．また，プログラマから解像度を隠蔽しているため，動的に解像度を変動させ，処理量を自動調整することが可能である．これにより RaVioli は処理のリアルタイム性を保証している．

2.2 動画像処理ライブラリ RaVioli

本節では，まず，本研究の対象となる動画像処理ライブラリ RaVioli がどのように動画像処理の抽象化とリアルタイム性の保証を実現しているか説明する．その後，RaVioli が抱える問題点を述べる．

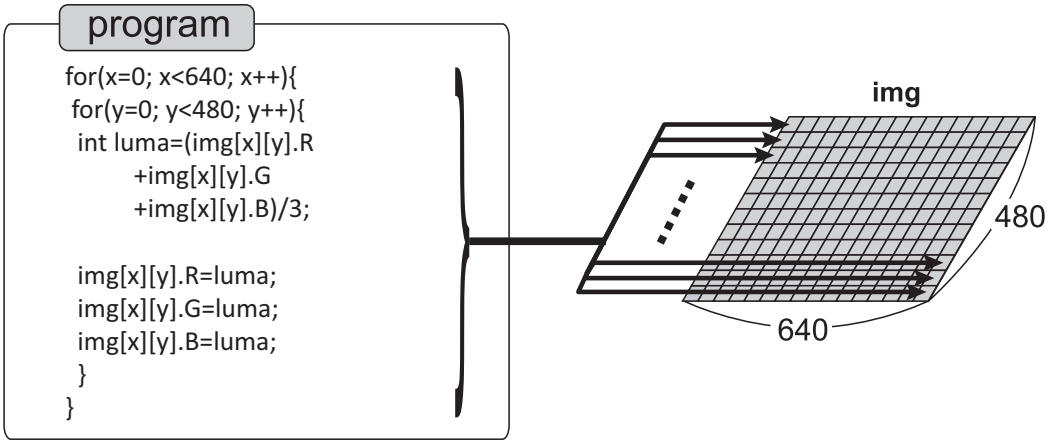


図 1: 一般的な画像処理プログラム

2.2.1 動画像処理の抽象化

動画像を構成する「画素」や「フレーム」は画像や動画像を計算機上で扱うために導入された概念であり，そもそも人間の脳内における視覚情報の認識過程には存在しない．しかし量子的に情報を扱う必要のある計算機上では，画像を画素の集合として，動画像をフレームの集合として扱わなければならない．また，このように量子化されているが故に，動画像処理プログラムを記述する際は for 文などのループ文を用いて，これらの量子化された構成要素に対し，繰り返し処理を適用する必要がある．しかし，この繰り返し処理は動画像処理の本質ではない．

これらの問題に対し RaVioli は，プログラマから解像度の概念を隠蔽するプログラミングパラダイムを提供している．ここで解像度とは**空間解像度**と**時間解像度**の 2 つを意味しており，空間解像度は 1 フレームを構成する画素数を，また時間解像度はフレームレートを意味している．RaVioli は，1 フレーム中の画像の各画素情報を格納している配列や画像の幅・高さ，フレームレート等をプログラマから隠蔽し，RaVioli 内でこれら全てを管理する．これにより，プログラマは解像度を意識せずに動画像処理を記述できる．

一般に画像処理では，画像の構成要素に対する処理を，画像全体または任意の範囲に繰り返し適用するものが多い．例えばカラー画像からモノクロ画像への変換や色の反転などの処理では処理単位は画素であり，ぼかしやエッジ強調などの近傍処理では，処理単位は画素およびその近傍画素である．また，テンプレートマッチング等の処理では処理単位は小さなウィンドウである．そしてこれらの処理は，一般的に**図 1** のようにループイテレーションを用いて記述される．例えば，カラー画像をグレースケー

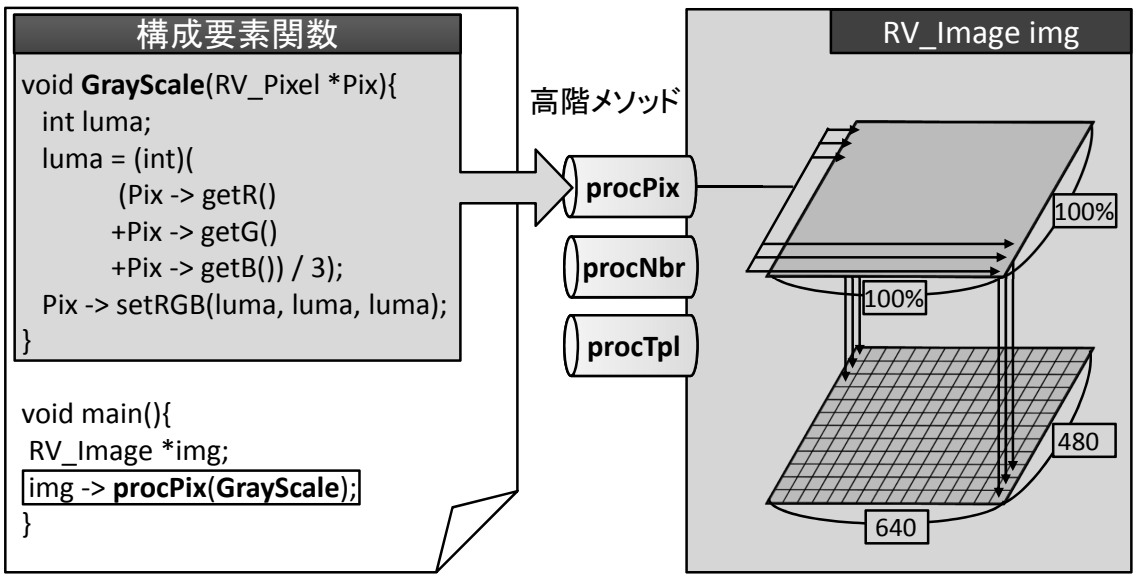


図 2: RaVioli の画像処理プログラム

ルに変換する場合，各画素を変換する処理は最も内側のループ内に記述され，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る場合，そのイタレーション回数は画像の幅と高さに応じて決まるため，プログラマは画像の幅と高さを意識してプログラムを記述しなければいけない．

一方，RaVioli では画像の構成要素である画素，または動画像の構成要素であるフレームに対する処理のみを関数として定義し，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にとるメソッドを**高階メソッド**と呼ぶ．

ここで，RaVioli の画像処理プログラムとその処理の様子を**図 2** に示す．なおこのプログラムはカラー画像をグレースケールに変換するプログラムである．RaVioli は画像中の幅や高さといった画像に関する情報を RV_Image クラスのインスタンスにカプセル化している．プログラマは画像の構成要素である画素に対する処理を構成要素関数として定義し，その関数を RV_Image の高階メソッドへ渡すことで画像中の全画素対して処理を適用することができる．そのため RaVioli の画像処理プログラムでは図 2 に示すように，RV_Image のインスタンス `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．この高階メソッド procPix() は `img` が持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

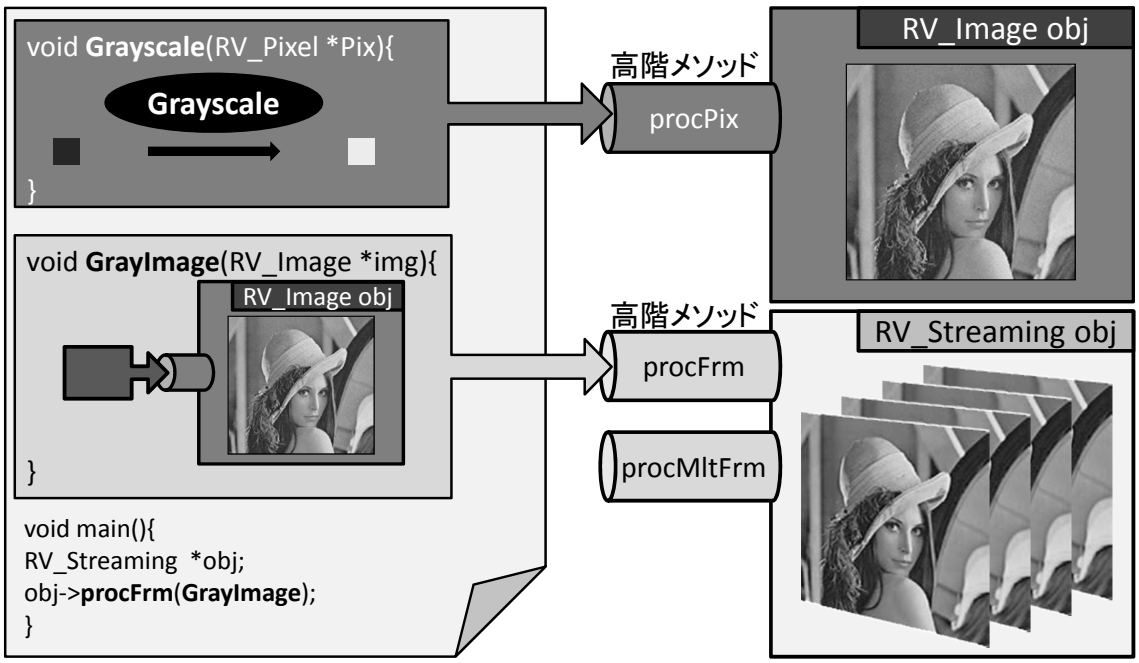


図 3: RaVioli の動画像処理プログラム

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．

次に，RaVioli の動画像処理プログラムとその処理の様子を**図 3** に示す．画像情報を RV_Image クラスのインスタンスにカプセル化したのと同様に，RaVioli は動画像中のフレームや，フレーム数，フレームレートといった動画像に関する情報を RV_Streaming クラスのインスタンスにカプセル化している．プログラマは動画像の構成要素であるフレームに対する処理を構成要素関数として定義し，その関数を RV_Streaming の高階メソッドに渡すことで，動画像中の全てのフレーム対して処理を適用することが可能である．ここで，フレームに対する処理とは先ほどの画像に対する処理そのものである．すなわち，RV_Image インスタンスに対する処理である．そのため，図 3 に示すように，フレームに対する処理を記述した構成要素関数 GrayImage() 内には，画像情報を持つ RV_Image インスタンスの高階メソッド呼出しが含まれる．このような処理構造を用いることで，プログラマは動画像の構成要素であるフレームの幅や高さ，フレームレートなどを意識することなく動画像処理プログラムを記述可能である．

2.2.2 リアルタイム性の保証

複数のプロセスが並行に実行される汎用 OS 上では，動画像処理に必要な CPU リソース量を常に確保できる保証はない．そのため，汎用 OS 上でリアルタイム動画像

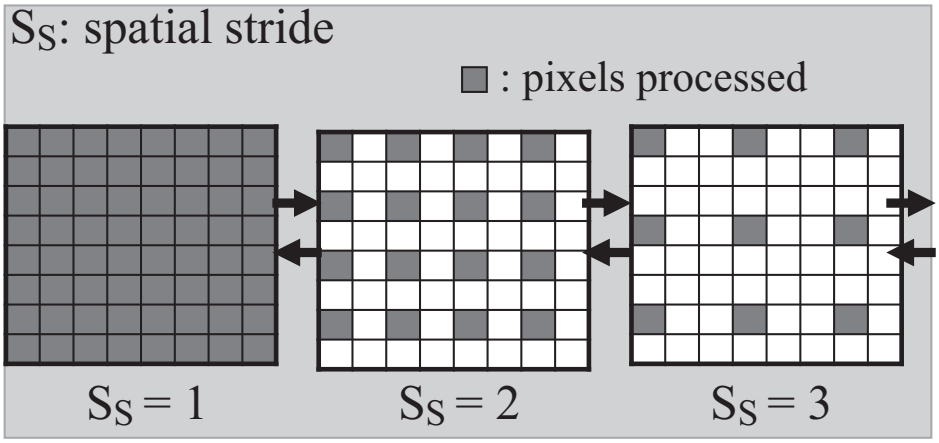


図 4: 空間解像度ストライドの変更

処理システムを実現することはいまだに困難である．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 はプログラマから解像度を隠蔽しているため，利用可能な CPU リソース量に応じて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に，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 (S_S) と，処理対象フレームの間隔を示す時間解像度ストライド (S_T) を持っている．これらのストライドを増減させることにより空間解像度と時間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 4** に示す．空間解像度ストライド $S_S = 1$ のとき，画像中の全ての画素が処理される．空間解像度ストライドを増加させ $S_S = 2$ となると，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．すなわち，処理画素数はストライド値の 2 乗分の 1 となる．

一方で，時間解像度を変動させるときの処理方法を**図 5** に示す．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処理する．時間解像度ストライドを増加させ $S_T = 2$ となると，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て解像度を維持する．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，アプリケーションの目的に適し

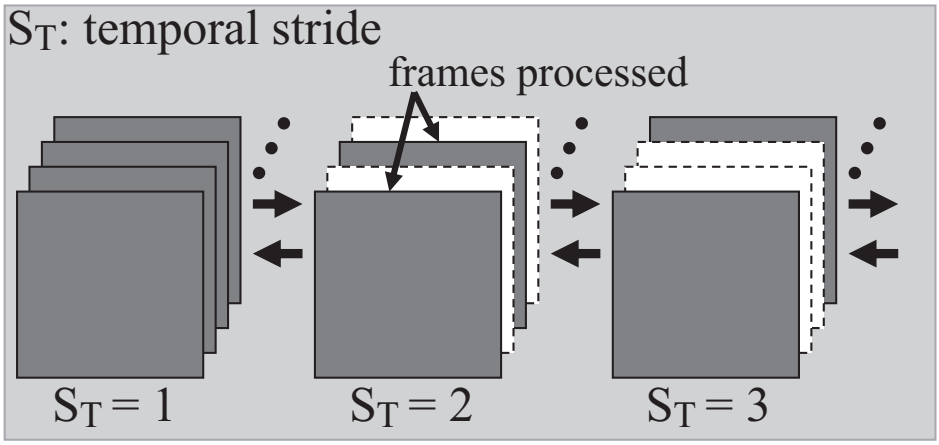


図 5: 時間解像度ストライドの変更

た解像度変動が可能となる．例えば，動物体の検出などの時間解像度が重要な処理では，時間解像度が優先されるように設定することで，空間解像度が優先的に低減され，より現実時間に近いリアルタイム処理を実現することができる．一方，顔認証などの空間解像度が重要な処理では，空間解像度が優先されるように設定することで，時間解像度が優先的に低減され，高い解像度を確保し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 $(\mathrm{P_S}, \mathrm{P_T})$ で構成される**優先度セット**を指定することで設定できる．$\mathrm{P_S}$ は空間解像度に対する優先度を表し，$\mathrm{P_T}$ は時間解像度に対する優先度を表す．例えば，$(\mathrm{P_S}, \mathrm{P_T}) = (1, 3)$ と設定した場合，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．

2.2.3 RaVioli の問題点

RaVioli は利用可能な CPU リソース量に応じて，解像度を低減させて適切な処理量に調整する．このとき，処理する画素数やフレーム数が低減されるため，動画像処理の精度は低下する．これは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．この問題に対して，プログラマは優先度を設定することで，一方の解像度の低減を抑えることが可能である．

しかし，処理が間に合わない場合，優先度を低く設定した解像度が大幅に低減されてしまう．この解像度の低減により，プログラマが期待した処理結果を得られない場合が存在すると考えられる．

ここで，侵入者検知システムを例に，解像度の低減によりプログラマの期待した処理結果が得られない場合を説明する．ここで想定する侵入者検知システムとは，入力

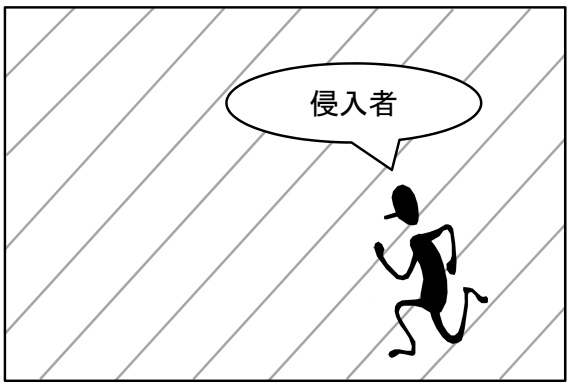


図 6: 入力フレーム

図 7: 出力フレーム

画像から侵入者を検知し，侵入者が検知された時刻の画像をプログラマ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．つまり，空間解像度，時間解像度どちらも高く維持したい．しかし，実際には利用可能な CPU リソース量に限界がある．このため，既存の RaVioli でこのシステムを実現する場合，侵入者を見逃すことだけは避けるために，全てのフレームを処理するように優先度を設定する．これにより，RaVioli は空間解像度を低減させ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 6** に示す．図 6 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 7** のような出力が得られる．侵入者を検知することはできているが，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．

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処理結果が得られない場合が存在する．そのため，処理量調整方法を変更することで，この問題を解決する．

3　領域別処理量調整手法

本章では，リアルタイム動画像処理における入力の特徴に注目し，領域別処理量調整手法を提案する．まず，3.1 節で提案手法の着眼点について述べる．その後，3.2 節で領域分割方法について説明し，3.3 節で入力の変化を考慮した解像度設定について説明する．

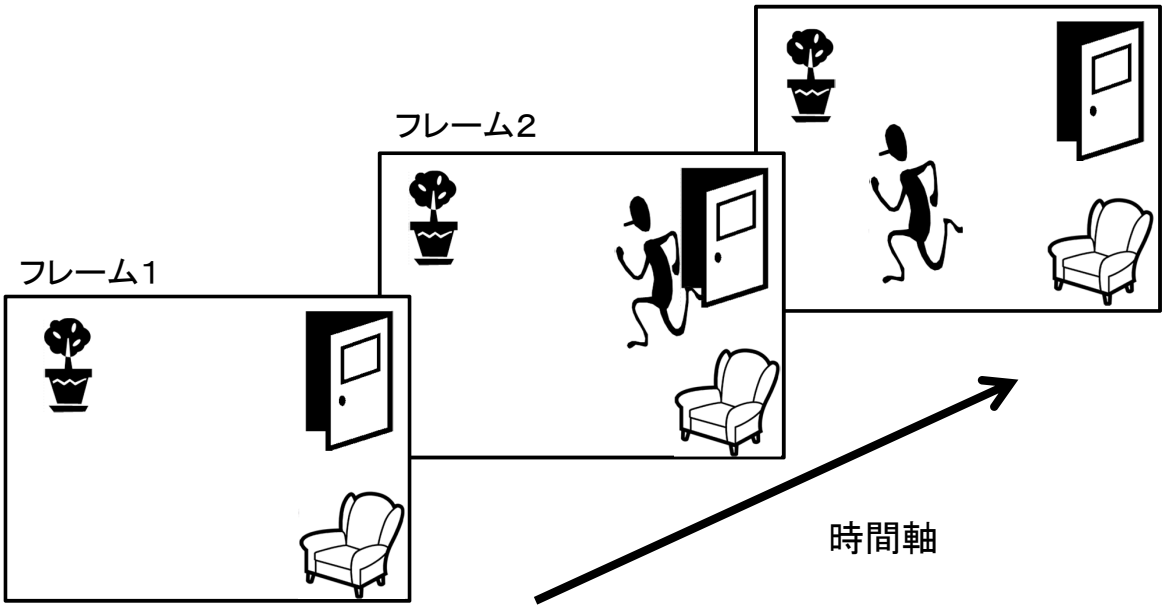


図 8: 侵入者検知システムへの入力とその特徴

3.1 着眼点

2.2 節で説明したように，RaVioli はプログラマから解像度を隠蔽し，動画像処理時に動的に解像度を変動させることで処理量を適切な量に調整する．これにより，動画像処理のリアルタイム性を擬似的に保証している．しかし，解像度を低減させることには許容範囲があり，解像度が大幅に低下することでプログラマの期待した処理結果が得られなくなる可能性がある．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，処理量調整手法を提案する．

リアルタイム動画像処理の入力には，重要な領域と重要でない領域があると考えられる．ただし動画像であるため，その重要な領域と重要でない領域は常にフレーム内で固定されているわけではなく移り変わっていくという特徴がある．ここで**図 8** のような入力フレームを侵入者検知システムが処理する場合を例にこれらの特徴を説明する．

まず重要な領域と重要でない領域があるという点に注目する．図 8 のフレーム 1 のように侵入者が存在せず，入力フレームに変化がないとき，そのフレームは重要ではないため詳細に処理しなくてもよいと考えられる．また，フレーム 3 のように侵入者が存在する場合でも，侵入者が存在する領域以外の大半の領域は重要ではなく，それらの領域は詳細に処理する必要がない可能性が高い．すなわち，リアルタイム動画像処理の入力には，詳細に処理する必要がない領域が存在する．

次に重要な領域と重要でない領域は移り変わっていくという点に注目する．これら

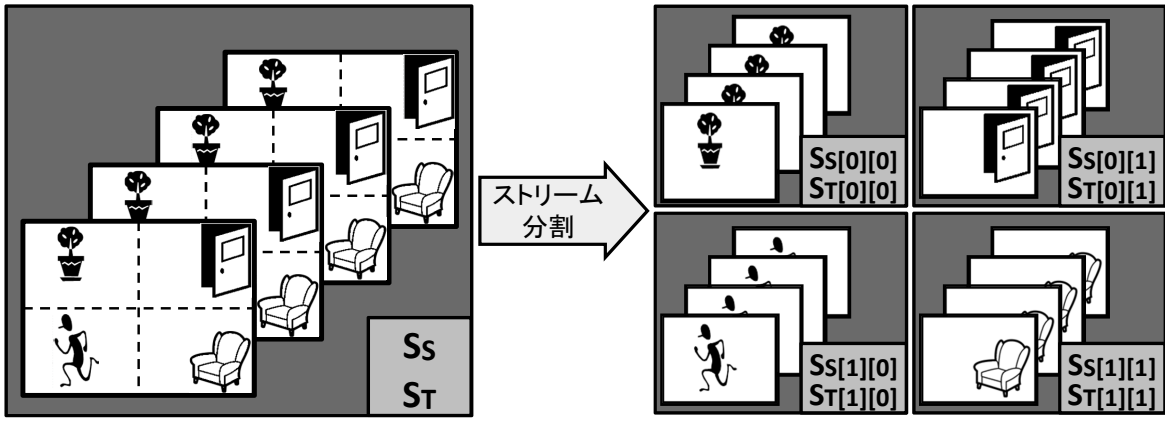


図 9: 動画像ストリームの分割

の領域の移り変わり方は全くのランダムではなく，ある傾向がある．例えば図 8 のフレーム 2 のように侵入者はドアから入ってくるということをプログラマはあらかじめ予想できる．また，侵入者は急に消えて全く別の領域に現れることはなく，連続する領域を移動していく．これらのことから，リアルタイム動画像処理の入力において，重要な領域へと変化する領域はある程度予測可能と考えられる．

以上の点を踏まえ，処理量調整手法を提案する．まず提案手法では，リアルタイム動画像処理の入力を領域分割し，重要でない領域の処理量を削減する．処理量の削減は，その領域における画素を間引いて処理し解像度を低くすることで実現する．これにより，重要な領域の処理精度を高く保つことを可能にする．

さらに提案手法では，重要な領域へと変化することが予想される領域を**優先領域**と定義し，その領域における解像度低減を抑制する．これにより，重要でない領域が重要な領域に変化する際，その変化に迅速に対応可能にする．

3.2 領域分割方法

提案手法では，領域別に処理量を調整するために，入力である動画像ストリームをいくつかの部分ストリームに分割する．ここで，動画像ストリームを 4 つの部分ストリームに分割する様子を**図 9** に示す．

図 9 の左側は分割前の動画像ストリーム，右側は分割後の各部分ストリームを表している．提案手法では，動画像ストリームを均等な大きさの部分ストリームに分割する．このように分割することで，1 フレームを処理する際の各部分フレームの処理量の見積もりを容易にする．これは将来的に，マルチスレッドで処理する際に，各スレッドの処理負荷を均衡化するために重要となる．

図 9 中の S_S，S_T は空間解像度ストライド，時間解像度ストライドをそれぞれ表している．提案手法では，部分ストリーム毎に空間解像度ストライドと時間解像度ストライドを管理し．この両解像度ストライドを部分ストリーム毎で別々に変動させる．このようにして提案手法では領域別処理量調整を可能にする．

3.3　入力の変化を考慮した解像度設定

本節ではまず，領域別に処理量を調整する方法について説明する．次に，優先領域の位置を考察する．そして，それを踏まえ，優先領域の設定方法について説明する．

3.3.1　領域別解像度変動方法

3.2 節で述べたように入力の動画像ストリームを分割することで，領域別に処理量を調整する．本研究では，各部分ストリームをいくつかの種類に分類し，それぞれの部分ストリーム毎に解像度を変動させることでこれを実現する．その分類の種類とは，「重要な領域」,「重要でない領域」,「優先領域」の 3 種類である．重要な領域に該当する部分ストリームは詳細に処理する必要があるため，その時点で設定可能な，できる限り高い解像度で処理する．そして，重要でない領域は低い解像度で処理する．これにより処理量を削減し，重要な領域の解像度を高く保つことを可能にする．しかしこれでは，重要でない領域が重要な領域に変化する際に素早く対応することができないと考えられる．これは，重要でない領域は低い解像度で処理されフレームレートが下がるため，領域が重要になるような入力の変化が起きた瞬間から，それを検知する瞬間に時間差が生じてしまうと考えられるからである．また，重要でない領域は出力画像が粗くなるため，その領域が重要となるような入力の変化が起きた瞬間に処理精度が悪くユーザが求める処理結果が得られないということも考えられる．

そこで，優先領域は解像度の低下を抑制する．これにより入力の急激な変化にも対応可能になる．

3.3.2　優先領域の位置

リアルタイム動画像処理における重要な領域とは，変化のある領域であると考えられる．ただし動画像であるため，その変化のある領域は移り変わっていく．しかしこの移り変わり方には 2 つの特徴がある．ここに着目し優先領域の位置を予測する．

まず 1 つ目は，初めに変化の起きる領域はある程度予想できるという特徴である．ここで，**図 10** のような画像を入力とする場合について考える．この時点では，入力フレームに重要な領域は存在していない．このためすべての領域の解像度を低減しても，プログラマの施したい処理にとって問題はないと考えられる．しかし，解像度を低減

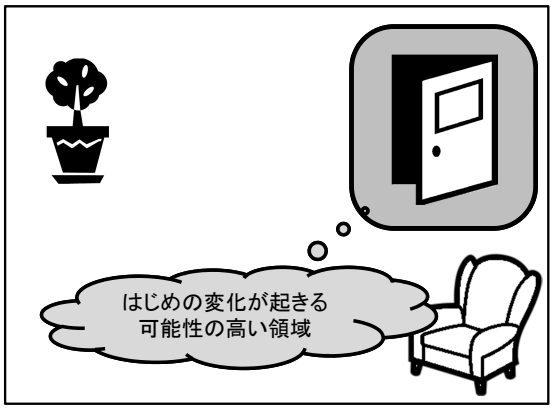


図 10: 初めの変化が起きる領域

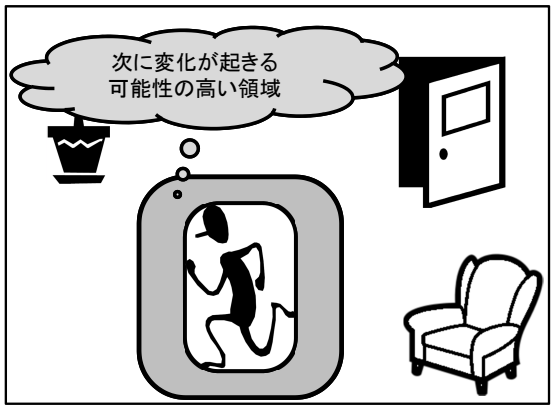


図 11: 変化のあった領域の隣接領域

しすぎると，領域が重要となる入力の変化が起きた際，その変化に素早く対応できないという問題が起こる．ただ，この入力フレームにおいて，ドアから何者かが入って来ることにより初めの変化が引き起こされるということをプログラマは容易に予想できる．このように，リアルタイム動画像処理システムにおいてプログラマは初めの変化が起きる領域というのをあらかじめ予想できる場合が多いと考えられる．

2 つ目は，変化のあった領域の隣接領域は次に変化が起きる可能性が高いという特徴である．ここで，**図 11** のような画像を入力とする場合について考える．この入力フレームにおいて，走っている人物が映っている領域が重要である．そのため，人物が映っている領域を詳細に処理する必要がある．ただ，人物は移動しているためそれに伴って重要な領域は移り変わっていく．しかし，人物は急に消えて全く別の領域に現れることはないと考えられるため，重要な領域の隣接領域が次に重要な領域となる可能性は高い．

以上の 2 つの特徴を踏まえて，変化がない場合でも解像度の低下を抑制する領域として優先領域を設定する．領域の特徴を考慮してプログラマが手動で設定する方法と，RaVioli が自動で設定する方法の 2 種類を用意する．次項でこの優先領域の設定方法について説明する．

3.3.3 優先領域の設定方法

優先領域を手動設定するために，変化のない場合でも解像度低下を抑制する領域をプログラマが指定できる機能を RaVioli に追加する．プログラマはこの機能を用いて，分割した入力で初めに変化が起きることが予想される領域を指定する．これにより，入力内で初めに発生する，重要な領域への変化に対応可能となる．例えば**図 12** のように入力を分割した場合では，初めに変化が起きることが予想されるドア付近の領域を優

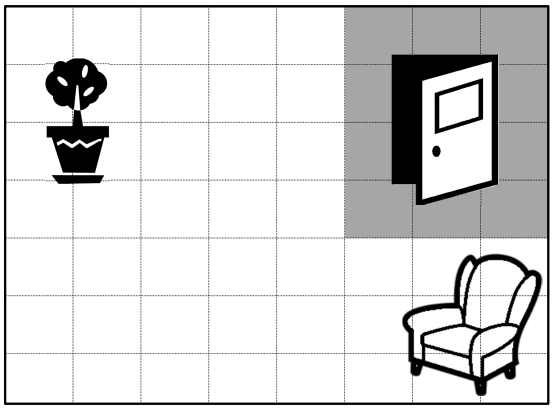


図 12: 優先領域の手動設定

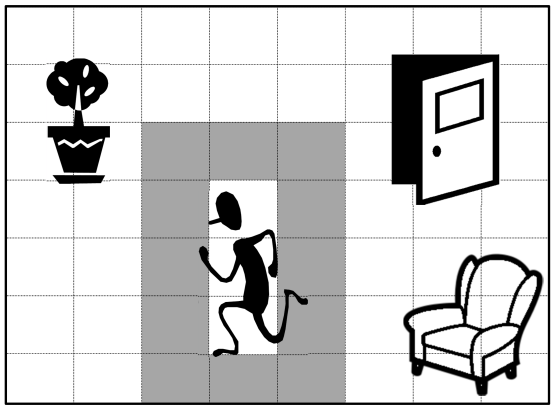


図 13: 優先領域の自動設定

先領域として指定する．

また，優先領域を自動設定するために，重要な領域の隣接領域を優先領域とみなし，その領域の解像度低下を抑制する機能を RaVioli に追加する．これにより，対象物体が分割した領域間を跨いだ移動をする場合でも重要な領域を詳細に処理することが可能となる．例えば**図 13** のように入力を分割した場合では，詳細に処理すべき対象物体である走っている人物が映っている領域の隣接領域を優先領域とみなす．以上のように，優先領域を設定することで入力の急激な変化に対応する動画像処理を実現する．

4　領域別処理量調整手法の実装

本章では領域別処理量調整手法の具体的な実装方法について説明する．

4.1　RV_TileImage クラス

提案手法では，領域別に解像度を変動させるために，動画像ストリームを部分ストリームに分割する．このとき各フレームは均等な大きさの部分領域に分割される．そこで，この部分領域を管理するための新しいクラス RV_TileImage を RaVioli に追加する．RV_TileImage クラスの概要を**図 14** に示す．RV_TileImage クラスは，部分領域が含まれる画像を保持する RV_Image クラスを継承している．図 14 に示す通り，RV_Image クラスのインスタンスは画像の幅や高さ，空間解像度ストライドと時間解像度ストライド，画素配列などの画像情報，および高階メソッドをメンバとして持つ．一方，RV_TileImage クラスのインスタンスは自身が担当する領域の幅や高さ，両解像度ストライド，高階メソッド，領域の左上を表す開始座標，判定関数へのポインタ，RV_Image インスタンスが確保した画像情報へのポインタを持つ．

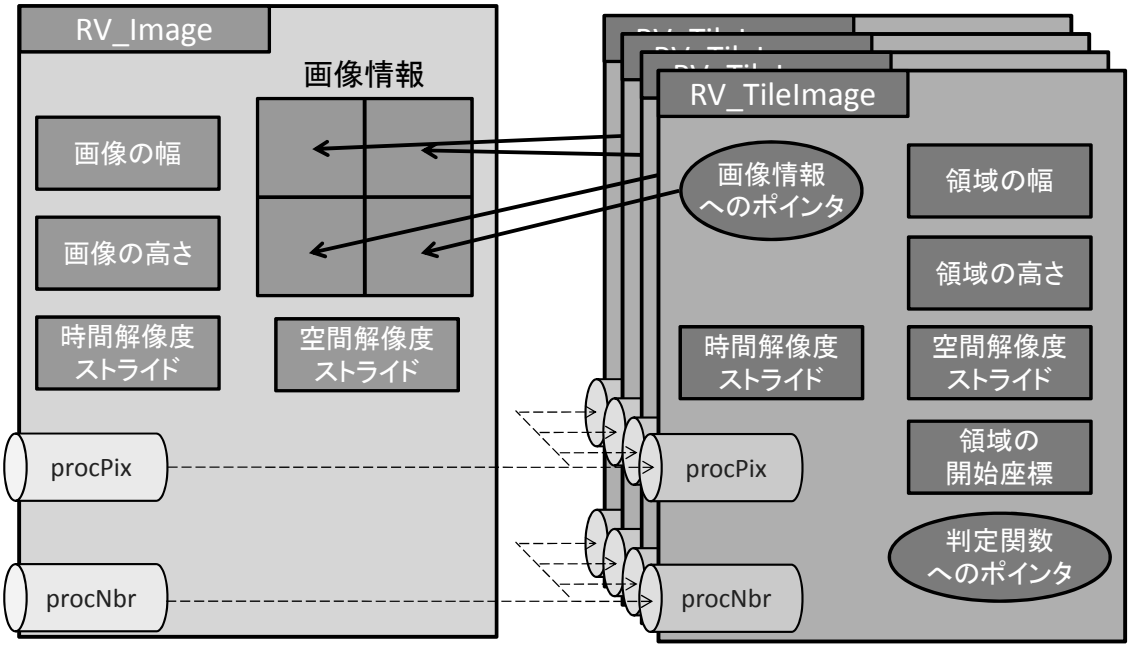


図 14: RV_TileImage クラスの概要

提案手法を用いて，動画像のフレームを処理する場合，RV_TileImage インスタンスが部分領域と同じ数だけ生成される．各 RV_TileImage インスタンスは担当範囲を持つため，全 RV_TileImage を処理することでフレーム全体を処理できる．ここで，各 RV_TileImage インスタンスの担当範囲は開始座標，領域の幅と高さによって決まる．そして，適切な高階メソッドを使って，その担当範囲に構成要素関数を適用する．このようにすることで，RV_TileImage インスタンスは RV_Image インスタンスの持つ画像の対応する領域に処理を適用できる．

4.2 処理量調整方法

動画像処理のリアルタイム性を維持しつつ，重要な領域を処理する際の解像度低下を抑制するためには，各部分ストリームが保持している空間解像度と時間解像度に設定できるストライドの選択肢を，複数種類用意する必要がある．本研究では，重要な領域と重要でない領域，優先領域の 3 つに注目するため 3 種類の解像度ストライドを用意する．そして，この 3 種類のストライドをそれぞれ，**ベースストライド**、**ラフストライド**，**中間ストライド**と呼ぶ．

この節では，まず 4.2.1 項でベースストライドとラフストライドについて説明する．次に 4.2.2 項で中間ストライドについて説明する．最後に 4.2.3 項でストライドを複数

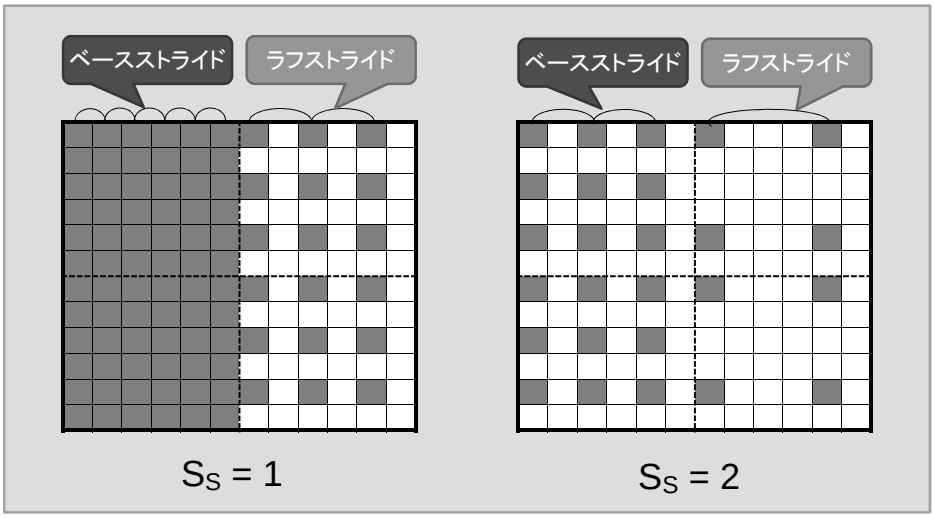


図 15: 空間解像度ストライドに対するベーススストライドとラフストライドの設定

設定可能としたことで新しく必要となる判定関数について述べる．

4.2.1　ベーススストライドとラフストライド

ベーススストライドは重要な領域に設定されるストライドである．重要な領域はできる限り高い解像度で処理する必要があるため，ベーススストライドはその時点で設定可能なできる限り小さいストライド値を保持する．

一方，ラフストライドは重要でない領域に設定されるストライドである．重要でない領域は処理量を低減するために低い解像度で処理する必要があるため，ラフストライドはベーススストライドよりも一定の量だけ大きいストライド値を保持する．

ここで，空間解像度および時間解像度に対するベーススストライドとラフストライドの設定について，それぞれ**図 15**，**図 16** を例に説明する． 二つの図は共に，動画像ストリームを 2×2 に分割したときの例を示しており，各フレーム内の破線は分割領域の境界線を表している．この例では説明を簡単化するために，ラフストライドの値はベーススストライドの値の 2 倍とし，各部分領域は左側の 2 つの領域が重要な領域，右側の 2 つの領域が重要でない領域とする．まず，フレーム全体の空間解像度ストライド $S_S = 1$ のとき，図 15 の左側に示す通り，重要な領域にはベーススストライド値として 1 を設定し，重要でない領域にはラフストライド値として 2 を設定する．このとき，フレーム全体にかかる処理量は，全体を $S_S = 1$ で処理する場合と比べて，5/8 となる．空間解像度ストライド $S_S = 2$ の場合は，図の右側に示す通り，ベーススストライド値は 2，ラフストライド値は 4 であり，それらのストライド値をそれぞれの領域に設定する．このとき，フレーム全体にかかる処理量は，全体を $S_S = 1$，$S_S = 2$ で処理する場合と

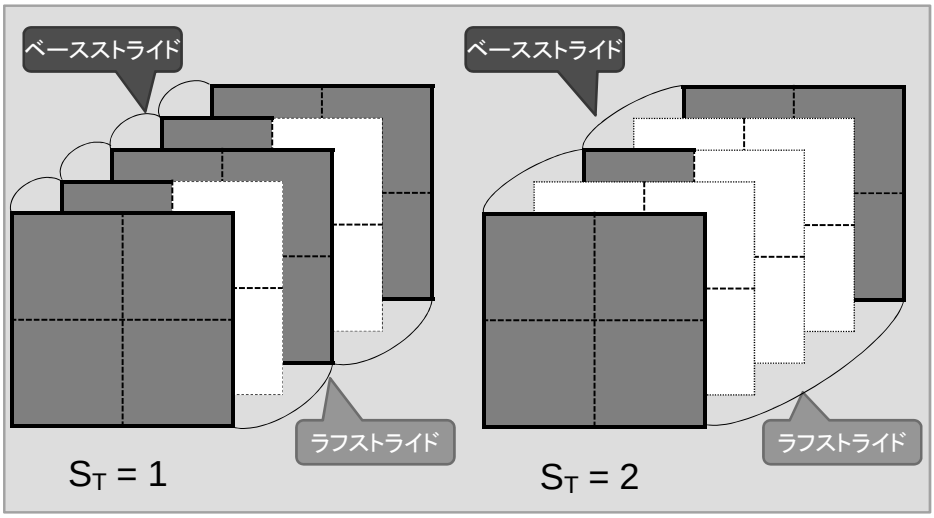


図 16: 時間解像度ストライドに対するベーススストライドとラフストライドの設定

比べて，それぞれ 5/32，5/8 となる．

一方，動画像全体の時間解像度ストライド $S_T = 1$ のとき，図 16 の左側に示す通り，重要な領域を含む部分ストリームにベーススストライド値として 1 を設定し，重要でない部分ストリームにはラフストライド値として 2 を設定する．このとき，動画像全体にかかる処理量は，全体を $S_T = 1$ で処理する場合と比べて，3/4 となる．時間解像度ストライド $S_T = 2$ の場合，図の右側に示す通り，ベーススストライド値は 2，ラフストライド値は 4 であり，それらのストライド値を各部分ストリームに設定する．このとき，動画像全体にかかる処理量は，全体を $S_T = 1$，$S_T = 2$ で処理する場合と比べて，それぞれ 3/8，3/4 となる．

以上のようにして，提案手法では重要でない領域に対する処理量を削減することで，利用可能な CPU リソース量を調整し，重要な領域に対する処理精度の低下を抑制する．また提案手法では，空間解像度ストライドと時間解像度ストライドのベーススストライドにできるだけ小さい値を設定し，優先度セットに基づいてそれらの値を変動させる．これにより，既存の RaVioli の機能である動的な処理量調整によるリアルタイム性の維持機能を損なうことなく，提案手法を実現できる．

4.2.2 中間ストライド

3.3 節で述べたように，本研究では，入力の急な変化に対応できるように解像度の低下を抑制する優先領域を設定する．そのため，このような優先領域にはベーススストライドやラフストライドではなく中間ストライドを設定する．中間ストライドはベーススストライドとラフストライドの間のストライド値を保持する．つまり中間ストライド

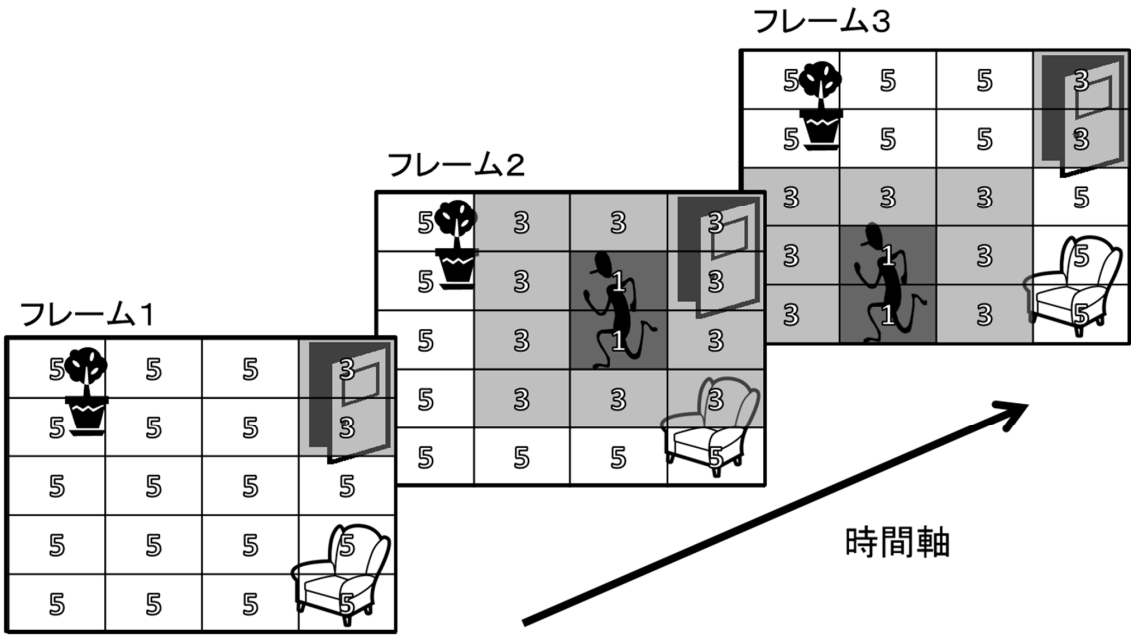


図 17: 中間ストライド設定位置の例

はベースストライドよりも処理量を削減する一方，入力フレームに変化が発生した際に，ラフストライドよりも素早くその変化を検知可能にする．

ここで，**図 17** を用いてどのように中間ストライドが設定されるか説明する．図 17 では，初めの変化が起きると予想されるドア付近の領域に手動で優先領域を設定しており，各部分領域内の数字は，その領域のストライド値を表している．さらに，この例ではベースストライド値は 1，ラフストライド値は 5，中間ストライド値は 3 であるとする．フレーム 1 では変化が無いため，優先領域として指定したドア付近の領域のストライド値には中間ストライド値の 3 が設定され、それ以外の領域のストライド値にはラフストライド値である 5 が設定される．フレーム 2 ではドアから人物が現れ変化が発生したため，人物が映っている領域のストライド値にベースストライド値の 1 が設定され，人物が映っている領域の隣接領域のストライド値に中間ストライド値の 3 が自動で設定される．その後，人物が移動した場合にはフレーム 3 のようにストライド値が設定される．

4.2.3 判定関数

領域別に処理量を調整するためには，各領域が重要かどうか，すなわち詳細に処理する必要があるかどうか判定しなければならない．そこで領域を詳細に処理すべきかどうか判定する判定関数を RaVioli に導入する．前述したように，ストライドは 3 種

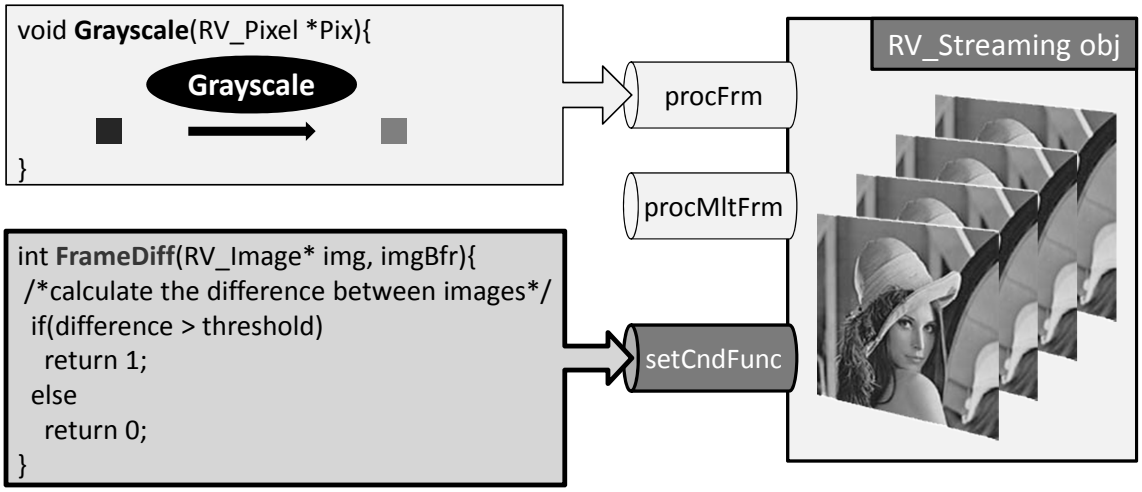


図 18: 判定関数の記述例と使用方法の概要

類用意する．しかし中間ストライドの設定方法は，プログラマが手動で設定位置を指定する，もしくは重要な領域の隣接領域に自動で設定する．このため判定関数は，詳細に処理すべきか，そうでないかの判定をする．

判定関数には，構成要素関数と同じ処理構造を用いる．このためプログラマは，入力の 1 フレームまたは 2 フレームを引数として判定関数を定義し，高階メソッドを使って RaVioli に渡す．これにより，プロブラム実行時に判定関数が適用される．

判定関数は，領域を詳細に処理すべきときは 1，そうでないときは 0 を返すように定義する．また, あらかじめ RaVioli 内に，汎用性が高いと思われるいくつかの判定関数を定義しておき，プログラマに提供する．これにより，プログラマが判定関数を定義しなくても領域別に処理量を調整できる．

ここで判定関数の記述例と使用方法の概要を**図 18** に示す．この FrameDiff は RaVioli 内であらかじめ提供されている判定関数であり，連続する 2 フレーム間の同じ位置の画素の差分を計算する．差分が閾値よりも大きい場合は詳細に処理すべきと判定し，1 を返す．一方，差分が閾値よりも小さい場合は詳細に処理する必要がないと判定し，0 を返す．前述したように，この判定関数を高階メソッドへ渡すことでプログラム実行時に判定関数が適用され，各領域にどちらかのストライドが割り当てられる．

具体的に，判定関数には引数の種類に応じて 3 種類の定義方法がある．その 3 種類とは，現在の処理フレームとその 1 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，入力フレーム内における領域の座標 x,y を引数とする判定関数である．このため，判定関数を引数とする高階メソッド setCondFunc も

3種類用意する．なお，この3種類の高階メソッドはC++のオーバーロードを使い実装しているため，プログラマは使い分けを意識する必要はない．以下に各setCondFuncの概要を示す．

void setCondFunc(int(*CondProgram)(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を用いて判定を行う関数CondProgramを引数とする高階メソッドである．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する判定関数をプログラムに記述する．

void setCondFunc(int(*CondProgram)(RV_Image* Fnow))

現在の処理フレームFnowを用いて判定を行う関数CondProgramを引数とする高階メソッドである．プログラマは1枚の画像を用いて，詳細な処理が必要かどうかを判定する判定関数をプログラムに記述する．

void setCondFunc(int(*CondProgram)(int x,int y))

入力フレームにおける分割領域の座標を用いて判定を行う関数CondProgramを引数とする高階メソッドである．プログラマは入力フレーム内における領域の位置を用いて，詳細な処理が必要かどうかを判定する判定関数をプログラムに記述する．この判定関数は，あらかじめどの領域を詳細に処理すべきか分かっている場合に用いる．

4.3 ユーザインターフェース

提案手法を用いてプログラムを記述するために，既存のRaVioliを用いたプログラムの記述方法に変更を加える．プログラマビリティが低下することを避けるため，既存のRaVioliのmain関数や構成要素関数の記述をできるだけ変更しないようにインタフェースを実装した．まずは既存のRaVioliを用いる際のプログラム記述方法を説明する．その後，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．

まず，既存のRaVioliを用いたプログラムの記述例を**図19**に示す．プログラマはmain関数と構成要素関数ForPixel，ForImage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(8行目)，そのインスタンスの持つメソッドsetPriorityを用いて優先度を設定する(9行目)．設定された優先度に応じて`video`が持つ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(10行目)．カメラからの入力フレームは`video`が持つバッファに一旦格納される．次に，動画像処理用の高階メソッドStreamProcに構成要素関数ForImageを渡す(11行目)．StreamProcではまず，バッファ内の入力フレーム数に応じ

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

図 19: 既存の RaVioli を用いたプログラム記述例

て，画像を管理するクラスである RV_Image クラスのインスタンスを生成する．次に，そのインスタンスに各解像度ストライドを設定し，構成要素関数 ForImage を呼び出す．また，ForImage では RV_Image クラスの高階メソッド procPix を用いて，ForPixel 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 20** に，そのプログラムでの領域分割と優先領域指定の様子を**図 21** に示す． 図 20 と既存の RaVioli を用いて記述するプログラムとの違いは 13 行目の判定関数の設定およびその判定関数 FleshDetect の定義，14 行目の領域分割数の指定，15 行目の優先領域の指定である．

では，図 20 に示す main 関数の記述に沿って処理内容を説明する．既存の RaVioli と同様にインスタンス `video` を生成し，優先度を設定する．次に判定関数がセットされ (13 行目)，その後プログラマが指定した分割数に従って画像を分割する (14 行目)．この際，分割数と同じ数の RV_TileImage インスタンスが生成される．ここでは，分割数を 5×6 と指定したため，図 21 のように入力の動画像ストリームは 5 行 6 列に分割される．その後，プログラマの指定に基づいて優先領域が設定される (15 行目)．優先領域は優先領域指定用関数 setTilePriority の引数に左上と右下の領域の行列番号を渡すことで指定できる．ここでは，図 21 のように 0 行 4 列目から 1 行 5 列目の領域が優先領域となる．最後に，既存の RaVioli と同様にキャプチャの開始を宣言し，高階メソッドへ構成要素関数を渡すことで動画像処理を実行できる．

```
1  void ForPixel(RV_Pixel *pixel){
2    /* 画素に関する処理 */
3  }
4  void ForImage(RV_Image* Frame){
5    Frame->procPix(Program2);
6  }
7  int FleshDetect(RV_Image *Curr, RV_Image *Prev){
8    /* 詳細に処理すべきかどうかを判定する */
9  }
10 int main(int argc, char* argv[]){
11   RV_Streaming video;
12   video.setPriority(7,3);              // 優先度の設定
13   video.setCondFunc(FleshDetect);      // 判定関数の設定
14   video.setTileNum(5,6);               // 領域分割数の指定
15   video.setTilePriority(0,4,1,5);      // 優先領域の指定
16   video.RunCapture();                  // キャプチャ開始
17   video.StreamProc(Program1);          // 動画像の高階メソッド
18   return 0;
19 }
```

図 20: 提案方式を用いたプログラム記述例

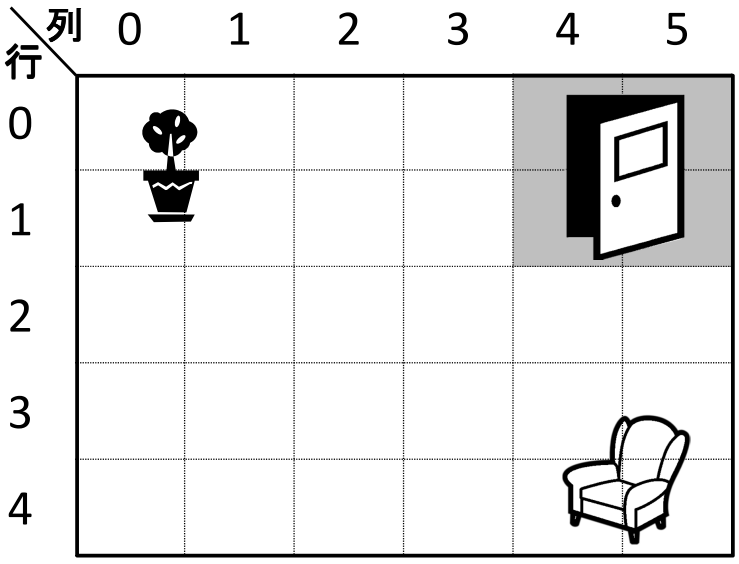


図 21: 領域分割と優先領域指定の様子

表 1: 評価環境

OS	Fedora15
CPU	AMD Phenom II X4 965
Frequency	3.4GHz
Memory	8GB
Compiler	gcc 4.6.0
Compile options	-O3

5 評価

既存の処理量調整手法を用いた RaVioli と提案手法を実装した RaVioli で動画像処理時の解像度ストライドの変動と出力画像を比較した.

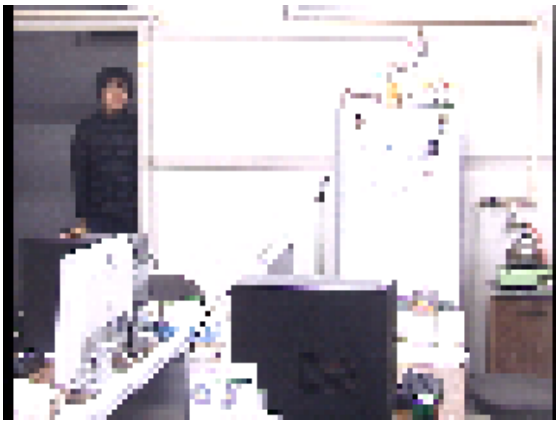
5.1 評価環境

評価環境を**表 1** に示す．CPU には 4 コア構成である AMD Phenom II X4 965 を用いた．また，コンパイラオプションには最適化オプションの`-O3` を指定した.

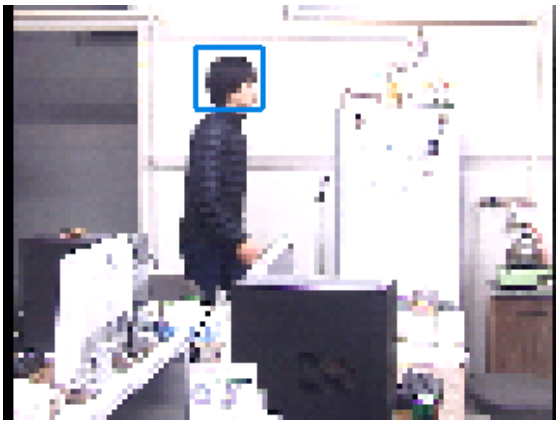
処理量調整手法の評価には，ベンチマークプログラムとして，テンプレートマッチングプログラムを使用した．なお，プログラムのマッチング画像には人物の横顔を用い，一定の閾値を越えた場合にのみマッチング位置を示すように実装した．入力には解像度が 320×240 画素の 120 フレームで構成される動画像を使用した．なお，領域の分割数は 9×9 とし，判定関数には 4.2.3 項で述べた 2 フレームの差分を判定基準にする FrameDiff を用い，優先度セットは $(\mathrm{P_S}, \mathrm{P_T}) = (1, 1)$ とした.

5.2 評価結果

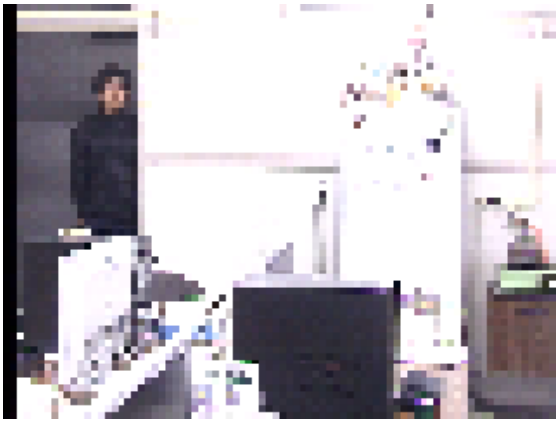
領域別処理量調整により，重要な領域の処理精度が高く維持されることを確認するために，既存の RaVioli と提案手法を実装した RaVioli のそれぞれの出力を比較した．なお，提案手法を実装した RaVioli については優先領域設定を用いないモデルと優先領域設定を用いるモデルのそれぞれを評価した．**図 22** に初めの変化が起きた瞬間（36 フレーム目）における各モデルの出力画像を, **図 23** に処理対象物体が分割領域間を跨いだ移動をした瞬間（80 フレーム目）における各モデルの出力画像を示す．どちらの図も画像は上から順に，(a) が既存の RaVioli を用いた既存モデル，(b) が RaVioli に優先領域設定以外の提案手法を実装した提案モデル（優先領域なし）, (c) が優先領域



(a) 既存手法



(a) 既存手法



(b) 提案手法（優先領域なし）

(b) 提案手法（優先領域なし）

(c) 提案手法（優先領域あり）

図 22: 36 フレーム目の出力

(c) 提案手法（優先領域あり）

図 23: 80 フレーム目の出力

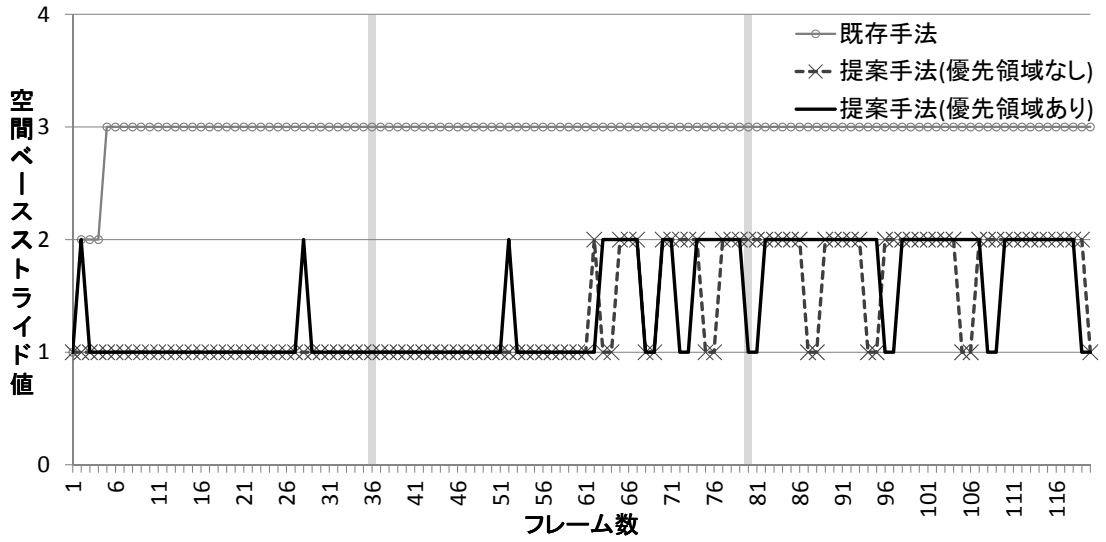


図 24: 空間ベーススストライド値の変動

設定を含む提案した全ての処理量調整手法を実装した提案モデル（優先領域あり）の出力結果である．なお，提案モデル（優先領域あり）は画像内のドア周辺の領域を優先領域として手動設定している．また，図 23 の各画像内に出力されている枠線はテンプレートマッチング処理の結果を示している．

まず図 22 について考察する．既存モデル，提案モデル（優先領域なし）ではドアから入ってきた人を詳細に処理することが出来ていない．一方，提案モデル（優先領域あり）では詳細に処理できている．これは，優先領域を手動設定したことにより，入力の初めの変化に対応できたためであると考えられる．

次に図 23 について考察する．既存モデルでは，処理量を調整するために画像全体の解像度を下げてしまっている．このため歩いている人が詳細に処理できていない．また提案モデル（優先領域なし）では，歩いている人がちょうど分割領域間を跨いで存在しているため，詳細に処理する必要のある顔の領域が処理できておらず，テンプレートマッチングも正しく行えていない．一方，提案モデル（優先領域あり）では，歩いている人の顔の領域すべてを詳細に処理することができている．これは詳細に処理すべき領域の隣接領域に優先領域を設定したためであると考えられる．以上の出力結果から，提案手法は既存の RaVioli と比べて，リアルタイム性を保証しつつ，重要な箇所の処理精度を高く保つことが実現できることが確認できた．

最後に，時間経過に伴う空間ベーススストライドの変動結果を**図 24** に，時間ベーススストライドの変動結果を**図 25** に示す． グラフの横軸は入力動画像のフレーム番号，縦

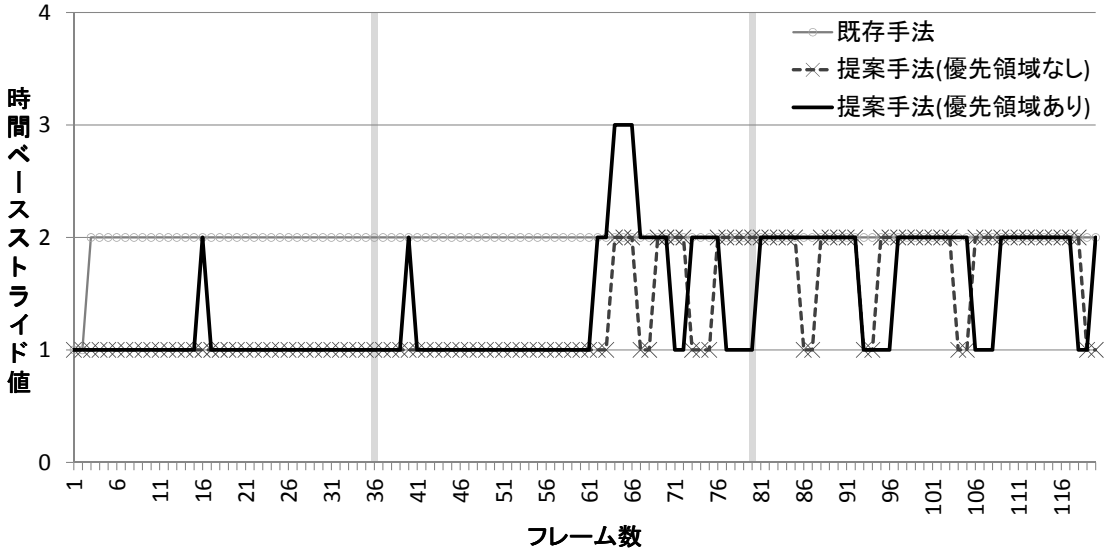


図 25: 時間ベーススストライド値の変動

軸はストライド値である．また，グラフ内の 2 本の縦線は図 22，図 23 で参照したフレームを示している．提案手法ではどちらも両ベーススストライド値を，既存の RaVioli の両解像度ストライド値に対して低く保つことができている．各モデルの空間ベーススストライドと時間ベーススストライドの平均値は，既存モデルが 2.95 と 1.98，提案モデル（優先領域なし）が 1.38 と 1.36，提案モデル（優先領域あり）が 1.40 と 1.40 であった．この結果から，提案手法により重要な領域における処理精度の低下を抑制できることが確認できた．

6　おわりに

本論文では，まず，利用可能な CPU リソース量が変動する汎用計算機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，リアルタイム性を擬似的に保証する動画像処理ライブラリ RaVioli について述べた．また，RaVioli には解像度低減による処理精度低下の問題があることを示した．

その問題を解決するために，リアルタイム動画像処理の入力の特徴に基づいて領域別に処理量を調整する手法を提案した．提案手法を実現するために，動画像ストリームを分割し，その部分ストリーム毎で処理量を調整する機能を RaVioli に実装した．また，分割領域を処理するためクラス RV_TileImage を新たに追加した．さらに優先領域を設定可能にすることで入力の急な変化にも対応可能とした．そして，提案手法を評価し，動画像処理中に処理量を領域別に調整することにより，詳細な処理が必要な領

域の空間解像度および時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，プログラマが分割領域をより直感的に扱えるGUIの開発が考えられる．これにより，プログラマの負担をさらに減らすことができる．また，分割領域単位で画像処理を並列化することが考えられる．並列に画像を処理することで処理時間を削減し，解像度低下をさらに抑えることができる.

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，松井俊浩准教授，梶岡慎輔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祖父江宏祐氏，大平真司氏，澤田晃平氏，内山寛章氏に感謝致します.

参考文献

[1] 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).

[2] 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).

[3] Garcia-Martin, A. and Martinez, J. M.: Robust Real Time Moving People Detection in Surveillance Scenarios, *Proc. 7th IEEE Int'l Conf. on Advanced Video and Signal Based Surveillance (AVSS'10)*, IEEE Computer Society, pp. 241–247 (2010).

[4] Kim, C., Han, Y., Seo, Y. and il Kang, H.: Statistical Pattern Based Real-time Smoke Detection Using DWT Energy, *Proc. Int'l Conf.on Information Science and Applications*, IEEE Computer Society, pp. 1–7 (2011).

[5] Lin, K., Huang, J., Chen, J. and Zhou, C.: Real-time Eye Detection in Video Streams, *Proc. 4th Int'l Conf. on Natural Computation*, Vol. 06, IEEE Computer Society, pp. 193–197 (2008).

[6] 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).

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

[8] Köthe, U.: *Generische Programmierung für dir Bildverarbeitung*, PhD Thesis, Universität Hamburg (2000).

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

[10] Bradski, G. and Kaehler, A.: *Learning OpenCV: Computer Vision With the OpenCV Library*, O'Reilly & Associates Inc (2008).

[11] 金井達徳, 瀬川淳一, 武田奈穂美: 組み込みプロセッサのメモリアーキテクチャに依存しない画像処理プログラムの記述と実行方式, 情報処理学会論文誌 : コンピューティングシステム, Vol. 48, No. SIG 13(ACS 19), pp. 287–301 (2007).

[12] Ragan-Kelley, J., Adams, A., Paris, S., Leboy, M., Amarasinghe, S. and Durand, F.: Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines, *ACM Transactions on Graphics (TOG) - SIGGRAPH 2012 Conference Proceedings*, ACM (2012).