解像度非依存型動画像処理ライブラリの提案と実装

岡田 慎太郎† 津邑公暁† 松尾啓志†

計算機の高性能化により，認識アルゴリズム等に代表されるある程度高度な画像処理を，汎用 PC でも行うことが可能となりつつある．一方で，高速なインターフェースも普及し，カメラ等からリアルタイムで動画像を取り込むことも容易となった．このため，Linux などの汎用 OS でのリアルタイム動画像処理が今後広く一般に行われると予想される．しかし，汎用 OS 上では限られたリソースを他のプロセスと共有しているため，必要なだけの CPU リソースを常に確保できるわけではない．このように使用可能リソースが時々刻々変化する汎用環境において，リアルタイム動画処理を保証することは一般に困難である．

本稿では，汎用環境でもリアルタイム性を保証する動画像処理を実現するにあたり，システムとインターフェースの両面から有用となるライブラリを提案する．システム面においては，解像度やフレームレートを状況に応じて動的に変更することで，処理に必要となる演算量を自動調整する機能を提供する．またインターフェース面においては，画像の構成画素や動画の構成フレームを隠蔽する事により，人間が本来持つ視覚イメージに近い記述が可能となるフレームワークを提供する．顔認識およびフレーム間差分検出プログラムを用いた検証の結果，本ライブラリが十分な記述能力を持つこと，およびプログラムの変更なく解像度やフレームレートが変更可能であり，また汎用環境において，実時間で動画像が処理できるよう負荷が自動調整されることを確認した．

Proposal and Implementation of a Resolution-Independent Video Library

Shintaro Okada,† Tomoaki Tsumura† and Hiroshi Matsuo†

General purpose computers are becoming capable for complex image processing through performance improvement. On the other hand, high-speed universal interface such as IEEE1394 is now popular and it is easy to capture video from camcorders. Therefore, it is expected that real-time video processing on general-purpose system such as Linux will become usual in the near future. On general purpose systems, many processes share the resource of the system. The amount of available computation resource fluctuates every moment on these systems. Hence, it is difficult to guarantee realtime video processing.

In order to implement real-time video processing on general-purpose system, this paper discribes about a video processing library which provides dynamic functions and good interface for programmers. This library adjusts the amount of operation automatically by modifying resolution or framerate dynamically. Besides, this library hide the framerate and the image resolution from programmers. The result of the experiment with face-detection program shows that our library has good capability for writing programs and the program works correctly with some different resolutions without any modification. Through the experiment with the program of detecting interframe difference, it is found that the framerate and resolution are automatically modified .

1. はじめに

計算機の高性能化により，認識アルゴリズム等に代表されるある程度高度な画像処理を，汎用 PC でも行うことが可能となりつつある．一方で高速なインタフェースも普及し，カメラ等からリアルタイムで動画像を取り込むことも容易となった．このため，汎用 PC および汎用 OS 上でのリアルタイム動画像処理が今後広く一般に行われると予想される．

一方で Linux などの汎用 OS では，使用可能なリソースが時々刻々変化するため，1/30 秒もしくは 1/60 秒毎に処理を行う必要のあるリアルタイムストリーミング処理を実現することは困難である．Linux をリアルタイム OS に拡張するプロジェクトも存在するが，一般に毎フレーム処理量が変動する認識処理においてリアルタイム処理を実現するためには，ワーストケースに基づく処理量削減が必要となる．

そこで我々は，汎用 OS でもリアルタイムに動画像処理が動作することを保証できる解像度非依存型動画

† 名古屋工業大学
Nagoya Institute of Technology

像処理ライブラリを実現するために，システムとユーザインターフェースの面で有用な手法を提案する．

まずシステム面においては，ユーザが指定した優先度に応じて自動的に処理量を軽減する機能を提供する．一般に，動画像処理プログラムにおいては出力フレームレートと 1 フレームに割くことのできる処理量はトレードオフであり，使用可能リソースが限られる場合，時間および空間における解像度のいずれかを低減させる必要がある．静的または動的にユーザから指定された優先度に沿うよう，自動的に出力フレームレートもしくは画素数を調整することのできるライブラリを提案する．

次にユーザインターフェース面においては，人間と計算機におけるイメージの扱い方の違いに基づいた新しいプログラミングパラダイムを提案する．そもそも視覚情報を認識・処理する際，人間の脳には,「イメージは画素の集合である」という意識は存在していない．この考え方を計算機上でも実現するため画素や解像度を隠蔽することで，プログラミングを簡易化，抽象化するライブラリの作成を目標とする．また，時間軸上においても解像度と同様にフレームレートを隠蔽することで，単位時間あたりの絶対フレーム数を意識する必要のない環境を提供する．

以上この 2 つのアプローチを組み合わせることにより，キャプチャした動画像を処理する際にユーザプログラムを変更することなく，解像度や処理フレームレートを自動的に調整するライブラリを実装した．

以下，2 章では本ライブラリの基本的な概念について述べ，3 章で具体的な実装方針とライブラリ仕様について述べる．次に 4 章では画像および動画に対する処理を通じた評価結果を示す．5 章では関連研究について述べ，最後に 6 章でまとめを行う．

2. 提 案 手 法

本研究で提案する解像度非依存型動画像処理ライブラリは，常にリソースが変動する状況下においても動画像のリアルタイム処理が可能である一方で，プログラマに対し対象画像の解像度および画素を隠蔽して，抽象度の高い記述を容易とする．

2.1 演算量の自動調整

高速なインターフェースが一般に普及し，カメラ等から実時間で動画像を取り込むことができるようになった．しかし，計算機の使用可能リソースが動的に変動する汎用 OS 上では，1/30 秒もしくは 1/60 秒の出力フレームレートを保ちつつリアルタイムストリーミング処理を実現することは困難である．

この状況を擬似的に解決するために，フレームあたりの演算量に応じて，動的に処理解像度を低減させる事が考えられる．一方で，使用可能リソースが十分ある場合には，十分高い処理解像度を維持することができる．このように出力フレームレートを維持しつつ処理できる最大の処理解像度になるように自動調整することで，使用可能リソースが変動する環境においても，常に実時間の動画像処理を可能にする．

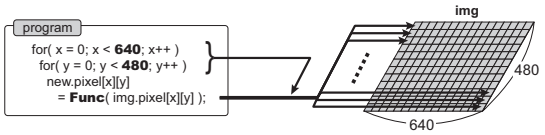
一方で，リアルタイム性よりも画像の精度が重要であるような処理も存在する．そのような場合には逆に，解像度を維持しつつ出力フレームレートを自動的に低減させることで演算量を調整する．

2.2 動画像処理の抽象化

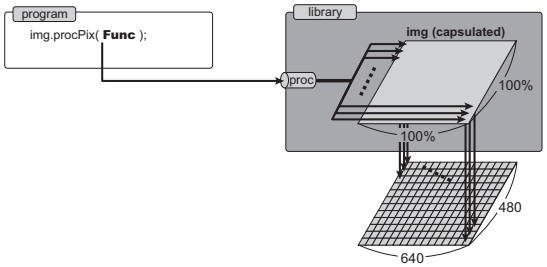
人間の脳内における認識過程には，画像に対して「これは画素の集合である」という意識は存在しない．しかし，量子的に情報を扱う計算機では，画像を点の集合として扱い，画像に変化を与える場合は，その点 1 つ 1 つに対して処理を施さざるをえない．

本稿で提案する解像度非依存型動画像処理ライブラリでは，人間と計算機のイメージの扱い方の違いを吸収し，人間が視覚イメージに対して行うような処理方法をそのまま計算機上に反映できる新しいプログラミングパラダイムを提供する．計算機上で画像から解像度という概念を隠蔽すると，画像の構成画素数を意識しなくても画像処理を記述できる．しかしその代償として画像処理の自由度が低下する可能性がある．ライブラリはこの問題を解決するために，画像に対するさまざまなインターフェースを提供する．そもそも従来の画像処理は，一般的に**図 1**(a) のように 1 つの画素に対する処理 `Func` を画像の任意の範囲に繰り返し適用する形で行われるが，今回の手法においてこの 1 画素分の単位処理は，人間が本来画像に対して持つ処理イメージに対応すると言える．本ライブラリでは，各画素に対する処理のみをプログラマに記述させ，それを全画像や特定範囲に適用する部分はライブラリが自動的に行う．また，解像度を隠蔽した環境において画像内の位置を指定するために，画像全体の大きさを 1 として正規化した座標（以下，**抽象座標**）を用いる．例えば画像の中心は抽象座標を用いて (0.5, 0.5) と表すことで，解像度に依らず常に中心を指示することができる．

以上 2 次元平面の抽象化について述べてきたが，この手法を時間軸上にも適用する．人間は動作している物体を観察するとき，やはりその視覚情報が複数枚のフレームで構成されていると認識しないことは明らかである．計算機上での動画像処理におけるフレームは，



(a) Traditional image processing.



(b) Image processing with our library.

図 1 画像処理の記述
Fig. 1 Image Processing

静止画像における画素に対応すると言える．本ライブラリが解像度と同様にフレームレートをユーザから隠蔽することで，ユーザは隣接する 2 フレームの時間間隔や単位時間あたりのフレーム枚数を意識することなくプログラムを記述することができる．

3. 実　　装

3.1 方　　針

2.1 節では，動画像の構成要素を増減させることで動的な演算量を自動的に調整するシステムを提案したが，この制御の際，出力フレームレートを優先するか解像度を優先するかをユーザに選択させる．例えば，高速で動く物体に対し動画像処理を施す場合には，多少画質が粗くなったとしても出力フレームレートの維持が必要になる．逆に精度の高いパターンマッチングなどを行う場合には，高い解像度が必要になる．このように，同じ動画像処理でもユーザが重きを置く対象は場合によって異なる．このトレードオフの関係にある 2 つの方法をユーザ側で選択可能にすることにより，リアルタイム動画像処理の自由度を高める．

解像度を調整する手法を説明するにあたり，例えばグレースケール化のように画像の構成画素全てに対して同じ処理を適用する場合を考える．従来の画像処理プログラミングでは，まず入力画像の構成画素数に応じたループを構成し，そのループ内で各座標に対応する画素に対してユーザが定義した関数を適用することでこれを実現する方法が一般的である（図 1(a)）．ここで，例えば解像度を 1/4 に変更したい場合，対象画像を縦横の 1 画素置きにアクセスする形で画像処理を行おうとすると，処理対象画素の座標を考慮する必要がある．

これに対して本ライブラリでは（図 1(b)），入力画像をカプセル化した画像インスタンスに対し，そのインスタンスが持つ「構成画素全てに処理を適用する」高階メソッド（図中 `procPix()`）に，各画素に対して行うべき処理を定義した関数を渡すことで，この処理を可能とする．この場合，プログラマは入力画像の構成画素数を用いた記述をする必要がなく，解像度が変更された場合でもその差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．

またフレームの扱いも解像度と同様である．フレーム間の差分検出のように，全てのフレーム間に対し同じ処理を適用する場合を考える．従来では，キャプチャデバイスから取り込んだ画像と，事前に保持しておいた 1 つ前の画像の 2 枚を引数とした関数を適用することでこれを実現する．処理するフレームレートを下げ，何枚かフレームをスキップして処理を行う場合，処理対象フレームのインデクスを調整する必要がある．

これに対して本ライブラリを使用した場合，カメラからキャプチャした動画はインスタンスを形成する．そのインスタンスは，1 枚のフレームもしくは 2 枚のフレームを引数に取る高階メソッドを持つ．例えば，隣接する 2 フレームに対する処理を記述する場合，処理対象のフレームインデクスは，現在の処理フレームレートに応じてライブラリにより自動的に計算される．このためユーザは，具体的な処理内容を書いた関数をその高階メソッドに渡すだけでよい．

2.1 節および 3.1 節で述べたように，処理量を自動調整するために解像度および処理フレームレートは動的に変化する．一方で，上記のように解像度およびフレームレートを隠蔽することで，ユーザはそれらを全く意識することなくプログラムを記述することができる．

3.2 主なクラスと高階メソッド

前節で述べた機能を実現するにあたり，PIXEL,IMG および STREAM の 3 つのクラスといくつかの高階メソッドを持つライブラリを C++で実装した．以下それぞれのクラスについて述べる．

PIXEL クラス

画像を構成する 1 画素の情報を保持するクラスであり，R，G，B のそれぞれの値を，8 ビットのメンバ変数として持つ．また，RGB 値を操作するためのメソッドを持ち，プログラマはこのメソッドを通じて画素の色をパーセンテージで変更することができる．

IMG クラス

画像の実体を表すクラスであり，メンバ変数として PIXEL インスタンスを構成画素数ぶん配列の形で保持する．また，現解像度に対応したパラメータ stride を持ち，画素アクセスの際の粒度を表す．IMG クラスのもつ高階メソッドは，一般に構成画素全体に対しプログラマから指定された関数を適用するが，例えば stride が 2 のときは画像の構成画素へのアクセスは 1 画素置きとなり，1/4 の解像度で処理が行われる．以下に実装した高階メソッドを示す．

procPix(*Func*, *Cs*, *Ce*)：抽象座標 *Cs* および *Ce* を対角頂点とする範囲内の，現 stride でアクセスされる全ての画素に対して指定された関数 *Func* を適用し，処理結果の画素を出力する．*Func* は 1 つの PIXEL を引数にとる関数である．*Cs*，*Ce* が省略された場合は画像全体を対象範囲とする．グレースケール化などに利用する．

procNbr(*Func*, *Cs*, *Ce*)：抽象座標 *Cs* および *Ce* を対角頂点とする範囲内の，現 stride でアクセスされる全ての画素に対して当該画素およびその近傍画素を入力として *Func* を適用し，処理結果の 1 画素を出力する．*Func* は当該画素，その近傍画素の複数 PIXEL へのポインタ，および近傍画素数を引数にとる関数である．*Cs*，*Ce* が省略された場合は画像全体を対象範囲とする．畳み込み演算などに利用する．

procCoord(*Func*, *Cs*, *Ce*)：抽象座標 *Cs* および *Ce* を対角頂点とする範囲内の，現 stride でアクセスされる全ての画素に対して，当該画素およびその抽象座標を入力として *Func* を適用し，中間データを生成する．*Func* は 1 つの PIXEL とその抽象座標を引数にとる関数である．*Cs*，*Ce* が省略された場合は画像全体を対象範囲とする．ハフ変換などに利用する．

transCoord(*Func*)：現 stride でアクセスされる全ての画素に対して，定められた対応規則 *Func* に従って当該画素の座標を変換する．*Func* は変換前，変換後の 2 つの抽象座標を引数にとる関数である．画像の回転、拡大縮小などに利用する．

procBox(*Func*, *Width*, *Height*)：現 stride でアクセスされる全ての画素を始点とする，指定された幅 *Width*，高さ *Height* を持つウィンドウに対し，そこに含まれる画素に対し処理を行う．*Width* および *Height* は全体画像に対する割合 (0〜1) で指定する．*Func* はマッチパターン等に用いる IMG インスタンス，およびウィンドウの始点・終点抽象座標を引数にとる関数である．テンプレートマッチングなどに利用する．

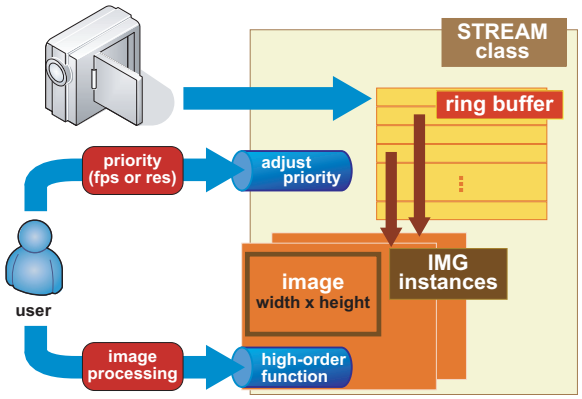


図 2　STREAM クラスの構造
Fig. 2　Structure of STREAM class

compImg(*Func*, *IMG *img*)：**現 stride でアクセスされる全ての画素と，引数で受け取った *img* の対応する位置の画素との比較を行う．*Func* は変換前，変換後の 2 つの画素を引数にとる関数である．2 枚の画像間の差分検出などに利用する．

STREAM クラス

動画像を処理するクラスであり，代表的なものにカメラから動画像をキャプチャするメソッドと，数枚のフレームを配列で格納しているリングバッファおよびそのメソッドがある．概略を**図 2** に示す．カメラの動画をキャプチャする部分は，Video4Linux2 (V4L2) ☆ ドライバを用いて実装し，取り込んだフレームをリングバッファに一時的に保存する．また pthread を使用し，サブスレッドでキャプチャ処理を行うことで，画像処理の演算量に依らず常に指定したキャプチャフレームレートを維持する．画像処理時にはリングバッファにあるフレームを IMG クラスのインスタンスに格納し，動画像用の高階メソッドを使って処理を記述する．1 フレーム分の演算時間がキャプチャ間隔より大きくなると，解像度優先の場合は処理対象フレームの間隔をあけ，フレームレート優先の場合は処理対象フレームのアクセス画素 stride の初期値を変更する．つまり，前者はフレームを複数枚スキップしてアクセスをすることで，後者は画像の処理する画素数を減らすことで，演算量を調整する．代表的な高階メソッドを以下に示す．

proc1Frm(*Func*)：全ての処理対象フレームに対し，指定された関数 *Func* を適用する．*Func* は 1 つの IMG インスタンス（フレーム）を引数にとる関数である．

☆ http://linux.bytesex.org/v4l2/

procAdjFrm(Func*)**：全ての処理対象フレームとその隣接フレームに対して指定された関数 *Func* を適用する．*Func* は 2 つの IMG インスタンスを引数にとる関数である．本ライブラリを使用した環境では，状況に応じて処理フレームをスキップすることで自動的に処理フレームレートが変動する．この際隣接フレームがどのキャプチャフレームにあたるかはライブラリによって自動的に判断され，当該フレームと共に *Func* に渡される．この高階関数は動画のフレーム間差分などに利用する．

4. 評　　価

4.1 画像処理の評価

1 枚の画像に対する処理の評価として顔検出プログラムを用い，ライブラリの記述能力や解像度変更への対応の評価を行った．

4.1.1 評価プログラム

本ライブラリの画像処理能力を評価するにあたり，サンプルプログラムとして色テンプレートを用いた顔検出プログラムを実装した．用いたアルゴリズムは肌色情報を使用する手法[1] に基づく単純なものである．

まず入力画像の肌色の部分とそれ以外の部分で 2 値化した中間画像を得る．ここで HSV 表色において色相値 H が 0〜30 の場合を肌色と判定した．次に，指定した大きさでかつ肌色部の占める割合が最大となるようなウィンドウを画像全体から検索することにより，画像内の顔の位置を検出する．

4.1.2 記 述 性

本ライブラリ使用の有無によってユーザが記述するプログラムがどのように変化するかを**図 3** に示す．

ライブラリを使用しない場合，まず顔として検出するウィンドウの大きさを，現解像度に応じて変更する必要がある．次に画像内をウィンドウをずらしつつ，ウィンドウに含まれる各画素に対し肌色か否かを判定し，その数を数える．`skinp()` は，当該画素が肌色である場合に `sum` をインクリメントするユーザ定義関数である．ここで，画像をウィンドウでスキャンする際にも，ウィンドウ内の画素に対し肌色判定を行う際にも，現解像度に応じた差分値 `stride` を用いてループを記述する必要がある．

これに対しライブラリを使用する場合，IMG クラスに定義されている高階メソッド `procBox()` を用いる．プログラマは IMG インスタンス `img` が持つ幅や高さを意識する必要がなく，高階メソッド `procBox()` に，プログラマが別途定義したウィンドウ内に対する処理を行う関数 `countSkin()`，および画像全体の大きさに対する比で指定したウィンドウの大きさを渡すだけでよい．`procBox()` は，比で指定されたウィンドウの大きさを現解像度に合わせて画素数に読みかえ，その大きさのウィンドウを画像内で現解像度に沿った stride でスキャンしつつ，`countSkin()` で定義されたウィンドウ内に対する処理を行う．

このように，プログラマが現解像度や画像の真のサイズを使用することなく，ライブラリを使用しない場合と全く同じ処理を記述することができる．また，本来現解像度を強く意識して記述する必要のある部分であるループなどの繁雑な箇所が省かれることで，バグの混入可能性を低下させるだけでなく，プログラムの

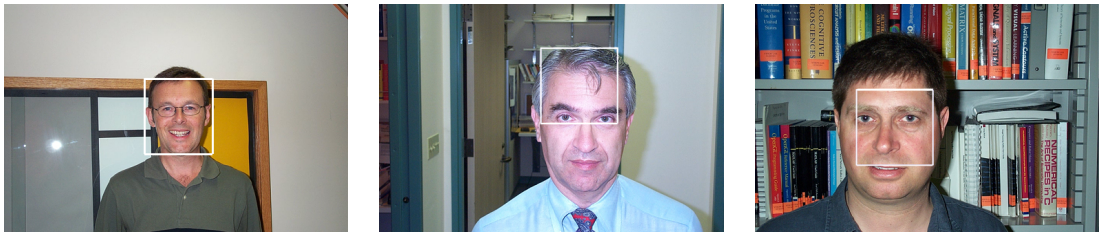
w/o library

```
// 解像度に応じたウィンドウサイズを設定
box.W = img.W * 0.2;
box.H = img.H * 0.3;
int max = 0;
int sum = 0;
Coord target;
// 現解像度に応じた増幅でイタレーション
for(y = 0; y < img.H-box.H-stride; y += stride){
  for(x = 0; x < img.W-box.W-stride; x += stride){
    sum = 0;
    for(by = 0; by < box.H-stride; by += stride)
      for(bx = 0; bx < box.W-stride; bx += stride)
        skinp( img.get_pixel(x+bx, y+by) );
  }
  if( max < sum ){
    max = sum;
    target.x = x;
    target.y = y;
  }
}
```

w/ library

```
int max = 0;
int sum = 0;
Coord target;
void countSkin( IMG* img, Coord cs, Coord ce ){
  sum = 0;
  img->procPix( skinp, cs, ce );
  if( max < sum ){
    max = sum;
    target = cs;
  }
}
int main(){
  // 画像全体に対し幅 20%，高さ 30% の box を
  // ずらしながら countSkin を適用
  img.procBox( countSkin, 0.2, 0.3 );
}
```

図 3　ライブラリ非使用/使用の場合のプログラム（部分）
Fig. 3　Difference of code between w/o library and w/ library.



(a) Output image #1. (b) Output image #2. (c) Output image #3.

図 4 顔検出の出力画像
Fig. 4 Outputs of face detection program.

(a) stride = 1 (b) stride = 2 (c) stride = 3

図 5 各解像度における出力画像
Fig. 5 Difference of outputs with various resolutions.

記述自体も少なく済む.

4.1.3 出 力 結 果

4.1.1 項で示した顔認識プログラムを本ライブラリを用いて記述し，カリフォルニア工科大の Computational Vision ☆で配布されているサンプル画像に対して処理を行った結果を**図 4** に示す．これら出力画像から分かるように，さまざまな入力画像に対し正しく顔の位置が検出できており，ライブラリの記述能力および処理の正当性が確認できた.

次に，解像度を 1/4 (stride = 2) および 1/9 (stride = 3) に変更した場合の結果を**図 5** に示す．この結果から，プログラムを一切変更することなく解像度を変更に対応できており，期待される結果が得られることを確認した．なお，(a) に比べて (c) の出力結果に若干の違いがあるが，これは画像が低解像度になったために精度が落ちた事によるものであり，本ライブラリの実装に起因するものではない.

4.2 動画像処理の評価

次に動画像処理の評価としてフレーム間差分を用い，解像度と処理フレームレートの調整による疑似リアルタイム処理の評価を行った．用いたシミュレーション環境を**表 1** に示す.

4.2.1 評価プログラム

本ライブラリを動画像処理の面から評価するにあたり，サンプルプログラムとしてフレーム間差分プログラムを実装した．このプログラムは，時間軸上で隣り合う 2 枚のフレームの中で，RGB それぞれの値で，どれかひとつでも差分の絶対値が 10%より大きい場合にはその画素を白，それ以外の画素を黒とすることで移動体を検出する.

表 1 動画像処理シミュレーション環境
Table 1 Simulation environment for video processing.

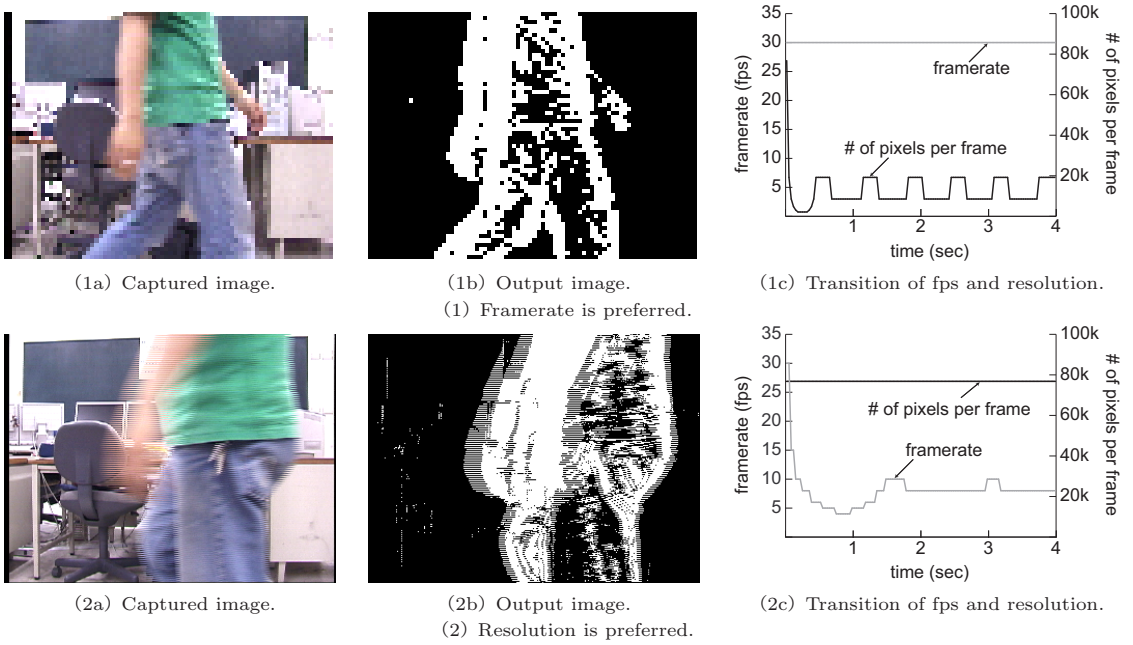
CPU	AMD Opteron Dual-Core (2GHz)
Memory	2GB
Camera	SONY DCR-TRV900
Capture board	I·O DATA GV-VCP2M
Format	NTSC
Interface	S-Terminal (S1)

4.2.2 出 力 結 果

カメラから 30fps で転送される解像度 320×240 のフレームをキャプチャし，前項で示したフレーム間差分プログラムを出力フレームレート優先と解像度優先でそれぞれ適用させることで，評価を行った．それぞれを優先した場合の処理画像，出力結果，および解像度と出力フレームレートの変化を**図 6** に示す.

出力フレームレートを優先した場合の処理画像（1a）は，演算量低減のために自動的に解像度が変更され，解像度優先の場合の処理画像（2a）に比べて画像が粗くなっているのが分かる。出力画像（1b）では（1a）の低い解像度に応じたフレーム差分が出力できており，

☆ http://www.vision.caltech.edu/



（1a）Captured image.　（1b）Output image.　（1c）Transition of fps and resolution.

（1）Framerate is preferred.

（2a）Captured image.　（2b）Output image.　（2c）Transition of fps and resolution.

（2）Resolution is preferred.

図 6　出力フレームレート優先時及び解像度優先時の処理結果
Fig. 6　Difference between framerate priority and resolution priority.

解像度が変更された場合でもプログラムの変更なしに期待した出力が得られることが確認できた．

（1c）および（2c）は，出力フレームレートおよび解像度のそれぞれを優先した場合の，処理開始から 4 秒後までの時間変化を表したグラフである．(1c）では出力フレームレートは常に最大値が維持されている一方で，解像度は処理開始直後より低下して行き，0.5 秒程度でほぼ一定の値に収束している様子が読みとれる．なお一度収束値よりも低い値まで落ち込んでいるのは，処理開始直後の過負荷により遅れてしまったフレーム処理をキャプチャフレームに追いつかせるために，一時的に定常状態よりも低い負荷にまで抑える必要があったためである．(2c）においても同様に，解像度は最大値を維持しつつ，出力フレームレートが適切な値にまで自動的に低減し，処理開始から 1.5 秒程度で定常状態に収束していることが分かる．

以上の結果から，本ライブラリがプログラマの指定する優先度に基づき正しく自動負荷調整を行えること，および解像度やフレームレートが変動した場合でもプログラムが正しく動作することが確認できた．

5. 関 連 研 究

並列処理の分野では，一般的に用いられる並列処理パターンをライブラリの形で抽象化して提供する並列スケルトンの考え方などが古くからあるが[2]，画像処理の分野でも，STL を基本とするテンプレートを用い抽象化したライブラリに VIGRA[3] がある．画像の反転や回転，エッジ処理などの基本的な処理から，ガウスやガボールに代表されるフィルタ処理，画像の分析処理などを抽象化して提供している．また OpenCV[4] は，画像処理の一般的なアルゴリズムを C の関数や C++のメソッドとして提供している．Linux 向け動画取込みライブラリを使用することにより，IEEE1394 カメラ経由のデータに対するリアルタイム処理も提供する．しかしこれらのライブラリを使用する場合，使用可能リソース量の動的な変化に応じた解像度変更は容易ではない．

トレードオフの関係にある処理精度および処理時間を動的に調整するアプローチとして，複数アルゴリズムの切換えがある．例えば，不完全な演算の結果を利用することで，与えた計算時間の長さに応じて精度が向上するモデル（Imprecise Computation Model）が提案されている[5]．またこのモデルに基づき，処理精度および処理時間に関して経験的に得た知識を利用することで，プログラマが予め記述した複数のアルゴリズムから，状況に応じて適したアルゴリズムを動的に選択する信頼度駆動アーキテクチャも提案されている[6]．しかしこの方法では，処理を計算負荷の異なる複数のアルゴリズムで実装する必要があり，依然プログラマに対する負担は大きい．

一方 Streaming VIOS[7] は，動的に処理解像度の変更を行い，プログラマから指定された画素座標を現解像度の対応座標に読みかえることのできるライブラリである．これは本ライブラリに近い考え方であるが，Streaming VIOS の提供する枠組ではほとんどの画像処理において解像度を透過的に扱うことはできない．このため，Streaming VIOS では画像の構成画素数等は依然プログラマから可視とされており，プログラマが逐一，現解像度に依存するパラメータを取得したうえでそのパラメータを用いた処理を行う必要がある．

これに対し本稿で提案した解像度非依存型動画像処理ライブラリは，動画像処理の抽象化と処理精度・処理時間の自動調整を両立することのできるものである．行いたい処理に対してプログラマは複数のアルゴリズムを記述する必要はなく，ライブラリ側が空間解像度（画像の構成画素数）および時間解像度（フレームレート）を状況に応じて自動的に変更することで処理精度・時間を調整する点で，既存手法とは完全に異なる．また解像度をプログラマから隠蔽することでより直観的な記述が可能となり，バグ混入の抑制や可搬性の向上といった効果も期待できる．

6. おわりに

本稿では，動的に使用可能リソースが変動するような環境においてリアルタイム動画像処理を保証するためには処理演算量の動的な変更が必要であることを示し，これを解像度またはフレームレートを変動させることにより実現する解像度非依存型動画像処理ライブラリについて述べた．また本ライブラリは，人間と計算機の画像の扱いの違いを吸収し，人間が画像に対して持つ処理イメージに近い形で記述できる新しいプログラミングパラダイムを提供するにあたり，解像度やフレームレートを隠蔽した動画像に対するさまざまな高階メソッドを実装することでこれを実現した．

まず顔認識プログラムを用いて画像処理の評価を行った．評価の結果，本ライブラリが十分な記述能力を持つこと，および解像度を自動調整した場合にもプログラムを一切変更することなく期待する動作が得られることを確認した．また動画像処理では，フレーム間差分の検出プログラムを用いてその評価を行った．評価の結果，解像度優先とフレームレート優先のそれぞれの場合において，自動的に演算負荷の軽減が行われること，やはりプログラムの変更なしにリアルタイム性を保証できることが確認できた．

今後の課題としてはまず，画像処理アルゴリズムは多数存在するため，現在実装済みの高階メソッドの仕様を吟味し改良すると同時に，より多くの画像処理をカバーできるよう新たなメソッドを実装していくことが挙げられる．また，省電力プロセシングに寄与する拡張も行っていきたい．防犯システム等の実用レベルで必要となるメディア処理においては，省電力化と認識率の高いアルゴリズムとの両立が要求される．現在でもソフトウェアにおいて，重要度の低い場面で処理の精度を粗くすることで負荷ひいては消費電力を低減させることは経験的に行われている．この，従来ヒューリスティックに行われてきた負荷調整をライブラリおよび実行環境内に取りこむことで，より抽象度の高いプログラミング環境を提供する．さらに，ストリーミング画像処理はパイプライン的に演算処理を行うことが可能な処理が多い．従って，比較的演算量の多い処理の場合を考慮した，複数の計算機を用いたパイプライン処理機能の実装もこのライブラリの大きな課題のひとつである．

謝辞 本研究の一部は，文部科学省科学研究費補助金（萌芽研究 18650020）による．

参 考 文 献

1) 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).
2) Campbell, D. K. G.: Towards the Classification of Algorithm Skeletons, Technical report, Dept. of Computer Science, Univ. of York (1996).
3) Köthe, U.: Generic Programming for Computer Vision: The VIGRA Computer Vision Library, http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/ (2006).
4) Intel Corp.: *Open Source Computer Vision Library* (2001).
5) Liu, J., Shih, W.-K., Lin, K.-J., Bettati, R. and Chung, J.-Y.: Imprecise Computations, *Proceedings of IEEE*, Vol.82, pp.83–94 (1994).
6) 吉本廣雅，有田大作，谷口倫一郎：実時間分散画像処理システムのための信頼度駆動アーキテクチャ，情処研報 2004-ARC-149 (HOKKE 2004), pp.85–90 (2004).
7) 奥村文洋，松尾啓志：疑似リアルタイム機能を備えた動画像処理系 Streaming VIOS の開発，第 2 回動画像処理実利用化ワークショップ，精密工学会，pp.62–65 (2001).