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

奥村文洋　　松尾啓志

名古屋工業大学電気情報工学科

近年、実時間動画像処理が要求されているが、汎用 OS 上では処理のリアルタイム性を保証することは不可能である。本研究では処理画像の解像度を動的に変更することにより、疑似的に実時間動画像処理を実現する手法を提案する。更に、解像度の変更を考慮することなくプログラム可能な動画像処理環境を構築すると共にその有効性、及び処理記述能力を確認した。

1. はじめに

近年、人物追跡など、リアルタイム動画像処理に関する研究が盛んに行われている[1],[2]。また、計算機の価格低下により、高い計算能力を持つ計算機を容易に手に入れることが可能となり、種々の動画像処理アルゴリズムが提案、実装されている。しかし、汎用OS 上での実装では厳密なリアルタイム性を保証することは困難である。

そこで本研究では、処理画像の解像度を動的に変化させることにより、汎用 OS 上で疑似的にリアルタイム動画像処理を実現可能とする手法を提案する。さらに、ユーザーは解像度の変更を考慮することなく、プログラムを記述可能な動画像処理環境として、動画像処理環境 Streaming VIOS を提案する。そして、実際に実装を行い、処理能力の検討を行う。

2. リアルタイム性の維持

一般に汎用 OS において、リアルタイム性の保証は困難である。主な要因として

- 画像の内容に依存する処理負荷の変動
- 他のプロセスとの同時実行により、目的の動画像処理に分配されるCPU リソースの減少
- I/O などによる待ち時間

などが挙げられる。

本研究では、汎用OS 上で疑似的にリアルタイム性を維持する手法として、処理画像の解像度を動的に変更する事により、処理負荷を変更する方法を提案する。これは、処理画像の解像度、すなわち画像のサイズを変更することで、動画像の処理負荷を変更する手法である。

本手法は、処理画像を直接の処理結果とする処理に対しては不適当である。しかし、動画像からの人物追跡など処理画像を直接結果としない処理に対して有効であると考える。

3. 動画像処理系 Streaming VIOS

汎用 OS 上で、疑似的なリアルタイム性の維持が可能な動画像処理プログラムを容易に記述するための環境として、動画像処理系 Streaming VIOS を提案する。機能として

- 解像度を意識することなくアルゴリズムの記述が可能な解像度非依存画像
- 変数に時刻の概念を導入したストリーミング処理機能
- 疑似リアルタイム性の維持を目的とした、処理負荷の自動的な変更機能

などを備える C++ ライブラリである。動作環境は、C++クラス及びC++テンプレートが利用可能な環境である。

4. Streaming VIOS の構成

Streaming VIOS は、Streaming VIOS 本体、及び、各種型から構成される。

Streaming VIOS 本体は

- 解像度非依存画像の解像度の管理機能
- ストリーミング処理に用いられるバッファの管理機能
- 負荷に応じた処理解像度の自動変更機能

などの機能を備える。

Streaming VIOS における型は、通常のC/C++言語で利用可能な型の他、Streaming VIOS 固有の型が定義される。次にStreaming VIOS 固有の型の概略について示す。

5. Streaming VIOS 固有の型

Streaming VIOS 固有の型の概略を以下に示す。

5.1 解像度非依存画像型

解像度非依存画像は解像度（サイズ）を意識することなく画素へのアクセスを可能とする。これは以下の２種類に大別される。

StrideImage　内部的に１つの画像を持ち、要求された画素の座標を現在の解像度に対応した座標への変換を行い、画素へのアクセスを行う。

用途として通常画像を解像度非依存画像に変換する場合などに用いられる。更に、様々なアクセスモードを持ち、アクセスモードを設定する事によって、利用可能な範囲を変更することが可能である。

ただし、解像度変更後の画像の内容は一般的に保証されない。

LayeredImage 内部的に各解像度に対応する画像を持つ。ある解像度における画像へのアクセスは、対応する解像度を持つ画像へのアクセスとなる。解像度の変更時には、必要に応じて画像間でコピーされる。また、StrideImage とは異なり、解像度を変更しても画像の内容は保持される。

この形式の画像用途としては、解像度に依存する処理間での画像の受け渡しを行う場合の一時変数などの他、汎用的な用途に用いることが可能である。

5.2 テンプレート型

C++テンプレートとして実装され、以下の２種類がある。これらの型は、既存の型を基に拡張された新しい型を提供する。

Stream　テンプレートクラス Stream<>は、変数が各時刻における値を蓄積可能とする。さらに、蓄積した値への相対時刻によるアクセスを可能とする。これにより、過去 i フレームなどの相対時刻による過去の値へのアクセスが可能となり、時系列処理の記述を容易にする。

Layered　テンプレートクラス Layered<>は、整数型などの変数が値を複数持つこと、すなわち、変数の多重化を可能とする。これは、変数が各解像度に対応する値を持つことを可能とする。

また、現在の解像度に応じて、実際にアクセスする変数を自動的に変更する機能を持つ。この機能により、変数に対しても解像度を考慮する必要の無い簡潔なアクセスを提供する。

6. ストリーミング処理

Streaming VIOS はストリーミング処理を可能とする。ストリーミング処理とは、変数に対して時系列情報を与え、簡潔な動画像の時系列処理の記述を提供する。ストリーミング処理は以下の手順で利用することが可能となる。

1. まず、Streaming VIOS の初期化時に、利用するストリームバッファの数を設定
2. Streaming VIOS が内部に持つバッファへデータを書き込む
3. ユーザは、Streaming VIOS の API を用いて過去 i フレームのデータを獲得する
4. ２～３を毎フレーム行う事で、ストリーミング処理が可能となる。

また、テンプレートクラス、Stream<>を用いることで任意の型に対しても、時系列情報を与えることが可能である。

ストリーミング処理において、データを格納、獲得する場合の記述例を以下に示す。

ストリーミング処理の記述例

```
const int  STREAM_LENGTH = 2;
  //ストリームの長さ
Stream<int> strm;
strm.Create(STREAM_LENGTH );
  //ストリームの初期化

int t=0;
while(1){  //処理を行うメインループ
  strm.Add( t++ ); //時刻 t のデータを格納
  strm.Get(0);  //時刻 t のデータを獲得
  strm.Get(1);  //時刻 t-1 のデータを獲得
}
```

7. 動画像処理の記述方式

ユーザーが解像度非依存画像を用いて動画像処理を記述する場合、処理の記述は以下の２種類に大別できる。

7.1 画素を基準とした記述

本記述は、解像度を完全に透過的に扱うことが不可能な部分で用いられる。プログラムは現在の画像から解像度に依存するパラメータを獲得し、その値を用いて画素に対する処理を記述する。ここで、解像度に依存するパラメータとは、画像中のピクセル数などである。

この記述はほぼ全ての画像処理を記述する場合に必要

となる。また、この方式によって記述された処理は解像度を変更することによって処理負荷を大幅に変化させることが可能となる。

また、画素へのアクセスは、operator によって実装されているため、関数呼び出しと同等のオーバーヘッドがあるが、コンパイラの一般的な最適化オプションの利用により、実際のオーバーヘッドは通常の画像へのアクセスの１～２割程度であることを確認済みである。

以下に、この記述方式を用いたプログラムの例を示す。処理として、2つの入力画像からの差分を行い、差分領域の重心を求める処理である。

画素基準の記述例

```
extern int PixelDiff( PIXEL i, PIXEL j );
extern int PIXEL_THRESHOLD;
IMAGE   in1, in2; //入力画像
int width = out.width(); //画像の縦画素数、
int height = out.height(); //横画素数
struct POSITION pos ={ 0, 0 };
 //差分領域の重心位置座標
int count =0;      //差分点の数

//この２重ループで画素基準の処理を記述
for( int h=0; h<height; h++ ){
 for( int w=0; w<width; w++ ){
 int tmp = PixelDiff( in1(w,h), in2(w,h) );
  if( tmp > PIXEL_THRESOLD ){
   pos.x += w;  pos.y += h;
   count++;
  }
 }
}
//重心位置を求める
//ここで求めた値は、画素基準の値のため
//解像度に依存している点に注意。
pos.x /= count;
pos.y /= count;
```

7. 2　画像を基準とした記述

本記述は、値に対して解像度に関する透過性を提供する。一般に画像処理において、画像の解像度に依存する値の存在は避けられない。よって、画像の解像度に依存する値を本来の解像度における値への変換、または、その逆の変換を行う必要がある。解像度に依存する値の例として、画像処理によって得られた画像中における人物位置座標、画像中の動領域の画素数などである。

以下に、この記述方式を用いたプログラムの例を示す。解像度に依存する値を変換関数PixelToScreen ()を用いることで解像度に依存しない値へと変換している。

9.　実験

9. 1　実験内容

Streaming VIOS の動作例として、カメラからの画像に対してフレーム間差分を行い、差分領域の重心の時系列情報を用いた簡単な人物追跡を実装した。さらに処理負荷を加え、自動的に解像度が調節され、フレームレートが維持されることを確認した。

実験環境としてCPU Pentium3 500MHz,Memory 512MbyteのＰＣを用い、Linux上に実装した。保証するフレームレートは 15frame/second、誤差 15%を許容範囲とし、フレームレートの調節間隔は1秒とした。

さらに、変化させる解像度は最大負荷を処理レベル１（画像サイズ３２０×２４０）、最小負荷を処理レベル8（画像サイズ40×15）とし、1～8までの 8 段階とする。各レベルでの処理画像のサイズを表１に示す。また、処理負荷は処理画素数にほぼ比例するため、処理レベル１における負荷を1とした場合、処理レベルi における負荷は

$$（処理負荷）= 2^{-(i-1)}$$

となる。

画像基準の記述例

```
IMAGE out;  //処理対象画像
struct POSITOIN posImage, posScreen;
  //座標を保存する構造体
/*
画像処理を行い、人物位置をposImage へ格納。
但し、画素基準の値であるため、
処理画像の解像度に依存している。
*/

posScreen = out.PixelToScreen( posImage );
  //画像基準の人物位置に変換する
  //これ以降、解像度を考慮すること無く
  //posScreen を利用可能となる。
```

表1： 各処理レベルに対応する画像サイズ

処理レベル	1	2	3
画像サイズ	320×240	320×120	160×120
処理レベル	4	5	6
画像サイズ	160×60	80×60	80×30
処理レベル	7	8	
画像サイズ	40×30	40×15	

9. 2 実験結果

処理レベル及びCPU 使用率のグラフを図1に示す。また、フレームレート及びCPU 使用率の変動のグラフを図2に示す。

図1より、負荷が高い場合に（T=17）、処理負荷が小さくなる方向へ処理レベルを変更され、負荷が低い場合には（T=32）、処理負荷が大きくなる方向へ処理レベルが自動的に変更されている。

さらに、図 2より、高負荷時において、フレームレートが一時的に目標フレームレートの下限を下回っているが、全体的に目標フレームレートを維持できている事を確認した。

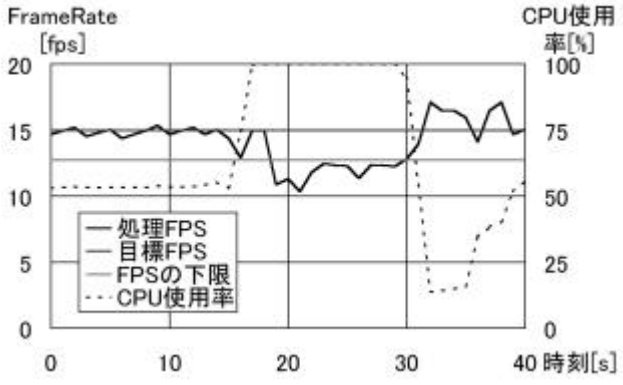


図1： 処理レベル及び、CPU 使用率の変動

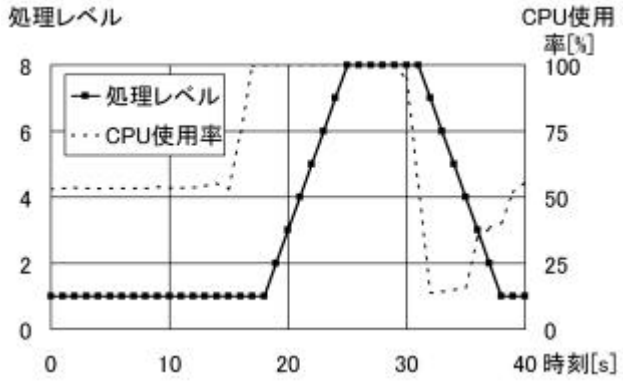
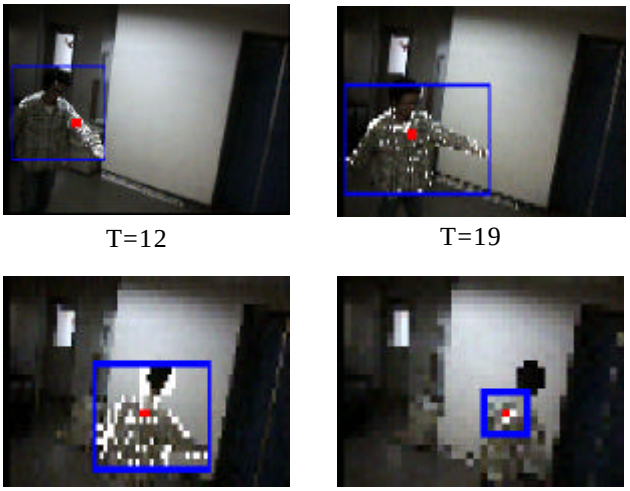


図2： フレームレート及びCPU 使用率の変 動

また、図3に各時刻における実行結果を示す。T=19 以降では処理解像度が徐々に落ちていく様が確認できる。さらに、低解像度においても、若干精度が落ちているが人物の追跡が可能であることを確認した。



T=12　　T=19

T=21　　T=23

図3： 各時刻における、実行結果

10. まとめ

汎用 OS を用いて、疑似リアルタイム性を維持しつつ動画像を処理可能な環境として、動画像処理系 Streaming VIOS を開発した。

この環境は処理負荷に応じて動的に画像の解像度を変更することにより疑似的にリアルタイム性の実現を可能とする。また、解像度の変更を考慮する必要のない動画像処理プログラムの簡潔な記述を提供可能である。

さらに、実際に動画像処理を実装し、処理負荷の変動に対して疑似リアルタイム性の維持が可能であることを確認した。

今後の予定として、解像度をより意識しない処理の記述を可能とする、言語の開発などを検討している。

11. 参考文献

[1] 和田 俊和 浮田 宗伯 松山 隆司 :視点固定型パンチルトズームカメラとその応用
電気情報通信学会論文誌 98/6 D-II Vol.J81 pp.1182-pp.1193

[2] 中澤 篤志 加籐 博一 井口征士 :分散カメラシステムによる人物の追跡
画像の認識・理解シンポジウム II(MIRU'98) '98/7 pp.1-pp.6