汎用動画像処理ソフトウェアライブラリ – MALib –
Generic Video Image Processing Library – MALib –

飯尾 洋* 谷田部 智之* 比屋根 一雄*
iiojuna@mri.co.jp tyatabe@mri.co.jp hiya@mri.co.jp

概要 PC の高性能化、デジタルビデオやデジタルカメラの普及、およびそれらの機器と PC とのインタフェースの進化に伴い、画像情報の PC を用いた加工や積層は現在とても身近な技術となりつつある。このような背景のもと、動画処理を行なうソフトウェアの開発は、ニーズの高い分野として注目を浴びている。しかし現在のところ手軽に利用できる汎用的なソフトウェアライブラリが存在していないため、個別に開発が重ねられているといった問題が残されている。そこで、それらの動画処理アプリケーション開発の促進を目的として、汎用に利用することができる動画処理ソフトウェアライブラリを構築し、LGPL ライセンスのもとでオープンソースソフトウェアとして公開した。

1 はじめに

近年の PC の性能向上は著しく、データ量の多い動画像を高速に処理することが可能になりつつある。またデジタルビデオやデジタルカメラの普及、およびそれらの機器と PC とのインタフェースの進化に伴い、画像情報の PC での加工や積層は既に身近な技術となっている。静止画像の処理ソフトウェアは現在様々なツールやライブラリが存在し、画像処理技術普及の牽引役を担っている。画像の認識処理についても、静止画像を対象にしたものは古くから開発用ライブラリ[1, 2]が存在しと広く利用されてきた。また imlib や gdl-pixbuf など、静止画像に関してはオープンソースの画像処理ライブラリも普及しており、自らのアプリケーションに簡単に組み込むことができるようになっている。

ところで映像処理、動画像処理のソフトウェアに関しては、ビデオキャプチャや IEEE1394 用いたデジタルビデオ接続を対象としたビデオ編集アプリケーションなど、個別のメカによる商用アプリケーションは存在するものの、手軽に利用できる映像利用アプリケーション開発ライブラリで汎用的なものはない。映像認識に特化したライブラリとしては、オーブンソースのライブラリであるが、本プロジェクトの目的は、画像だけでなく音声も含めたデータのハンドリングや映像データの出力力まで含めたより汎用的なライブラリの開発にある点で、これらのライブラリはややスコープが異なる。


本論文では、今回開発した汎用動画像処理ライブラリの概要、基本アーキテクチャと実装の概要、ライブラリの応用例として作成した移動物体認識アプリケーションの紹介について述べる。また今後の計画を述べるにあたり、企業におけるオープンソース開発事例のひとつとしての側面から捉えた本プロジェクトの立場について最後に補足する。

2 汎用動画像処理ライブラリ

汎用の動画像処理ライブラリ提供の目的から、とくにプラットフォームは限定せず移植性の高いオープンソースライブラリとして開発することに留意したが、まず開発のプラットフォームとして Linux を選択した。その理由としては、本開発の
入力
膨大な映像データベース
ネットワーク映像ソース (Web Cam等)　リアルタイム映像信号

低レベルインタフェース
映像編集処理
映像認識処理(共通)
映像認識処理(個別)

画像作成:ファイルI/O　ネットワークI/O　無意味画像削除　シーン分解　ソース合成　背景抽出　フレーム間差分量化等　画像認識　位置姿勢認識　事件検出

図 1: MAlib を利用した動画像処理アプロケーション

ベースとなったプロジェクトが Linux 用の画像処理ソフトウェア [7] であったこと、Linux を基盤とした場合、リアルタイムシステム、組込み機器から高性能ワークステーションまでのスケーラビリティが望めることなどが挙げられる。とくに組込み機器の展開については、現状では本成果を適用するまでは至っていないが、組込み用ハードウェアの更なる高性能化が期待されているところであり、今後も強力な分野のひとつと考えている。

さて本プロジェクトで構築し、公開したライブラリ MAlib は、動画像処理に関する汎用のデータ構造とプログラムを C 言語のライブラリ形式で提供する。その名称は、以下的目的を含む汎用的なライブラリの提供という、本プロジェクトが目指す最終的なゴールに由来する。MAlib は、以下の目的に合致するように汎用的に利用されるライブラリとして位置づけられる。

- 動画像認識、動画像を用いた信号処理等、動画像メディアを中心とした信号解析、認識処理の基礎を提供するライブラリとなる (Media Analysis)

- さらに動画像である特性を活かし、動画像中に含まれる移動物体の解析に特化した処理を提供する (Motion Analyzing)

- 認識処理、解析処理の基盤となる一方で、認識・解析処理を行なうための素材作成などに利用できるような、動画像 (映像) 処理基盤を提供する (Movie Architecture)

- さらに音声との同期などを考慮し、映像オーディオリンクツールを構成する要素としてのライブラリとしても利用可能とする (Media Authoring)

本プロジェクトの最終的なゴールは、あらゆる動画像処理に適用できるライブラリの構築である。まずその第一步として、MAlib は Linux プラットフォーム上の動画像処理の基盤を提供する。具体的には、Video4Linux を利用したリアルタイム動画像認識処理アプロケーション作成者、ファイルからの入力を想定した複数の動画像を扱うアプロケーション作成者による利用を想定し、これらのアプロケーションの作成を支援するソフトウェア部品としてのライブラリを提供する (図 1)。図1において網掛けの範囲で示している箇所が、現在の実装でライブラリが提供する機能を示している。

3 オブジェクト指向による実装

MAlib は、組込み機器への適用などの汎用性を考慮して C 言語で記述されているが、オブジェクト指向の概念に従って設計されている。C 言語でオブジェクト指向的記述を行うことは、汎用性と保守性的両立にある。すなわち、C 言語で実装することにより汎用性を高めるとともに、構造体と関数ポインタを利用してオブジェクト指向的に記述することにより、ライブラリ自身のメンテナンス性を向上させることができる。

MAlib におけるオブジェクト指向的な実装は、GTK+ [8] でのオブジェクト指向の記述に従い実現した。ただしシングナルなどの概念が不要であるため、若干簡略化された実装方法となっている。ここではまず、MAlib のクラス階層を示す (「*」印がついているものは今後拡張する予定)。

```
MalibObject (全てのルートクラス)
 ├── MalibHolder
 │    └── MalibGObject
 │        └── MalibFileOutput (*)
 │                    └── MalibNetworkOutput (*)
 │                        └── ...
 │                        └── MalibBuffer
 │                                └── MalibPlainBuf
```

2
フレームで遅って画像処理を行なう必要がある操作の場合に対してはこのリングバッファの利用が基本となるが、各時点でのフレームのみを対象しフレームデータの保持が必要な操作のために単フレームのバッファも用意している。

図 3: リングフレームアーキテクチャ

Malib の設計においては、前節のクラス階層の紹介したとおり、まずこのフレームワークにおける基本的な概念として Source と Holder、およびそれらのサブクラスとして定義される Filter と Buffer のクラスを導入した。

Source は映像データのソースとなるクラスであり、Holder は映像フレームのデータを保持するクラスである。Holder の基本的な機能は、Source を入力にとりデータを保持することである。Filter の機能は何らかの処理を行なって後段のバッファへ加工後のデータを渡すことであるから、Filter に接続された Holder からみると Filter が自身の映像データをソースとする。したがって Filter は Source のサ
クラスとして定義することが妥当である。Filterで拡張される重要な機能は、Filterへの入力がある、ということが挙げられる。

Sourceが入力の端点となる一方で、Holderは出力側の端点としてとらえることができる。Holderのサブクラスとして定義されるBufferは、Filterへ接続して後段へデータを渡すことができる点が拡張されている。

これらのクラスは次のように接続され、実際の処理が行なわれる。

Source → Buffer → Filter → (Buffer → Filter ...) → Holder

「(Buffer → Filter ...)」は処理が多段階になり得ることを示している。なお単純な一連の処理だけでなく、複数の入力を持つクラスや複数のバッファへの処理結果の配分なども可能としているため、一般には複雑な有向グラフを形成することになる（図4）。なお前述の例や図4における矢印は処理の流れを示すものであり、実際のポイント参照は図上の矢印の逆となることに注意。

なお現時点では、表1に示す11種類のフィルタが用意されている（記載順序は前述のクラス構成の順番に準じた）。

表1: MAlibが提供するフィルタ群

<table>
<thead>
<tr>
<th>フィルタ名</th>
<th>説明</th>
</tr>
</thead>
<tbody>
<tr>
<td>MalibDelay</td>
<td>時間平滑化フィルタ</td>
</tr>
<tr>
<td>MalibFrameDiff</td>
<td>時間差分フィルタ</td>
</tr>
<tr>
<td>MalibGenericFilter</td>
<td>沢用水素フィルタ</td>
</tr>
<tr>
<td>MalibGrey2Bx</td>
<td>グレーベクトルコンバータ</td>
</tr>
<tr>
<td>MalibNegative</td>
<td>色反転フィルタ</td>
</tr>
<tr>
<td>MalibRgb2Grey</td>
<td>RGB→ベクトルコンバータ</td>
</tr>
<tr>
<td>MalibRgba2Yuv</td>
<td>RGB→YUVコンバータ</td>
</tr>
<tr>
<td>MalibSempia</td>
<td>セピア系フィルタ</td>
</tr>
<tr>
<td>MalibMovingAvg</td>
<td>移動平均フィルタ</td>
</tr>
<tr>
<td>MalibSpatial3x3</td>
<td>沢用3×3空間フィルタ</td>
</tr>
<tr>
<td>Malib0verlap</td>
<td>重ね合わせフィルタ</td>
</tr>
</tbody>
</table>

MAlibの利用者が新たなフィルタを作成する方法はふたつある。ひとつは、これを継承して新たなフィルタクラスを実装する方法である。また、MAlibが提供するデータ構造に対応した画像処理アルゴリズムだけを実装した関数を記述し、沢用水素フィルタのパラメータとして関数ポインタを与える方法も利用されている。後者は利用することにより、新たなフィルタを容易にテストすることもできるようになっている。

5 フィルタの実装例

フィルタの実装の例として、色反転処理フィルタMalibNegativeを記述したソースファイルnegative.cを示す（図5）。実際のクラス構造の定義は、対応するヘッダファイルnegative.hで行われる。

```c
#include "negative.h"

static void malib_negative_write_frame_data (MalibNegative* filter, MalibFrame* frame)
{
    /* pointers to previous frame data and data area to store the result of calculation */
    input = ((MalibFilter*)filter)->buf;
    to = frame->data;
    max_val = filter->max_val;

    /* increment previous buffer data */
    frame->data = image->data + filter->max_val;
    input = (MalibFilter*)filter)->buf;

    /* pointers to rgb frame data and data area to store the result of calculation */
    input = (MalibFilter*)filter)->buf;
    to = frame->data;
    max_val = filter->max_val;

    /* calculate grayscale value of rgb data */
    image->data = frame->data;
    input = (MalibFilter*)filter)->buf;

    /* pointers to rgb frame data and data area to store the result of calculation */
    input = (MalibFilter*)filter)->buf;
    to = frame->data;
    max_val = filter->max_val;
}
```

図5: フィルタ実装の例(negative.c)

上記フィルタのソースコードは、プライベート関数のプロトタイプ宣言、パーサー関数テーブルの定義、バブリック関数定義（コンストラクタほか）、および実際のフィルタ処理を行う関数定義から構成される。

本クラスでは、実際のフィルタ処理はバーチャ
複雑な処理構造の例

図 4: Source - Filter - Buffer - Holder のリンク構造

ル関数として定義されるため、ファイルスコープに関したプライベート関数として定義されている。この関数のエントリポイントは関数ポインタとしてパーソナル関数テーブルに格納されており、実際には上位の抽象クラス MalibFilter で用意されるパーソナル関数の実体としてアクセスされる。

ここではフィルタ定義の概略を紹介するに留める。各種マクロ定義など本ソースの詳細については、公開されているソースコード一式を入手し、それらを参照していただきたい。

6 リファレンスカウンタの利用

MAlib における各オブジェクトは、リファレンスカウンタを持つ。オブジェクトのリファレンスカウンタは、現在のところ以下の目的で利用されている。

- リンクを辿ったオブジェクトの自動的消去
- 画像フレームの更新
- フレーム情報の共有管理

本節では、とくに重要なオブジェクトの消去処理および画像フレームの更新処理について詳しく述べる。フレーム情報の共有管理は、各フィルタが持つ対象フレーム情報の変更を行う際、共有されている参照数を管理するためリファレンスカウンタを利用するものである。

不要になったリンク構造を成す画像処理オブジェクトの消去は、関連するリンクを全て辿って再帰的に行われる。例えば次に示す連結が存在する場合、終端オブジェクトである Holder D を消去するだけで全てのオブジェクトおよび関連するデータ領域がメモリから解放される。

Source A → Buffer B → Filter C → Buffer D

ただし、オブジェクトのリンクは一般には直列とは限らないので、自動的な消去に関してリファレンスカウンタを利用して、オブジェクトのデストラクタはリファレンスカウンタを減算しているが、カウンタが 0 になった時点で実際の消去を実施する。順次処理の性質上循環構造を成すリンクはあらかじめ、リファレンスカウンタによる消去処理は問題なく動作する。

次に、画像フレームの更新処理における同期の確保に関しリファレンスカウンタの利用について説明する。

画像処理はリンク構造の最後尾のオブジェクトに対して maltib_holder_increment_frame() を呼び出すことで開始される。同関数では前段の処理オブジェクトへ参照を辿り再帰的に処理関数を呼ぶ。リンク構造の先頭、すなわちソースオブジェクトまで関数呼び出しを完了した時点で、そのソースでは新規フレームを生成する処理が行なわれる。そして、呼び出しを戻りつつ新規に生成されたフレームのデータに各フィルタ処理を加えることにより、前節で示したとおりの一連の画像処理が行なわれる。

このような、動画像処理は各時刻において逐次画像フレームを更新していくことで進行する。その際、正確な解析を実現するには各フィルタの処理対象が異なっていてはならない。比較処理を行ながら場合、比較対象が異なっていては意味がないので、これらは顕著な問題となる。

先に示した Source A から Holder D までの一連の処理に類した直列処理の場合は、端点から端点
点までの参照関係は全て一対一の対応となっており、問題はない。ところが、図4に示すような複雑な構造の場合、終端のHolder Kから始点のSource Aまでのパスは複数存在する。ファレレンスカウンティングなしの単純な関数呼び出しで処理を行うと、次に示す呼び出し順序で処理が行なわされることになる。

\[
K \to J \\
\quad \to D \to C \to B \to A \\
\quad \to H \to G \to F \to E \to B \to A \\
\quad \to I \to E \to B \to A
\]

ここでオブジェクトに付加されたアンダーラインは、一回のフレーム更新処理に関して複数回呼び出されていることを示す。上記の例では、Holder Kに対する一回の更新処理についてSource Aが3回呼ばれているため、Buffer Dからの系列、Buffer Hからの系列、Buffer Iからの系列の全てが異なるフレームデータに基づいた処理を行なうことになり問題である。

そこで、このような複雑なリンク構造においても、元データの同一性の確保すなわち処理の同期を保持するために、通常のファレレンスカウンタに加え一時的に値が増減するテンボラリファレレンスカウンタを導入した。さらに次に述べる規則の導入により問題解決を図ることができ、なおこれらのカウンタはMalibObjectにおいて、メンバ変数refcountおよびrefvarとして実装されている。

テンボラリカウンタrefvarの取扱いに関する規則は以下のとおりである。

1. refvarの初期値はrefcountに等しい。すなわち通常のファレレンスカウンティングを行う場合は、refcountの増減に合わせてrefvarの値の増減も行なう。

2. 終端のオブジェクトからリンク構造を辿りつつ画像処理を進める際には、refvarとrefcountの値が等しい場合のみ、前段へ処理を進める。始点オブジェクトの場合は、refvarとrefcountの値が等しい場合のみ、新規の画像を生成する。refvarとrefcountの値が等しくない場合は、キャッシュされている画像を利用し後段へ処理を進める。

3. 一回の呼び出しにつき、refcountの値をひとつ減算する。refcountの値が0になった時点でrefcountの値に初期化する。

上記の規則を適用すると、先ほどの呼び出し順序は次のように修正される。

\[
K \to J \\
\quad \to D \to C \to B \to A \\
\quad \to H \to G \to F \to E \to (B) \\
\quad \to I \to (E)
\]

括弧で示した箇所は、前段に対する処理の呼び出しを実施せずに既にあるデータに対して処理を行なっているものである。この呼び出し順序によると、Source Aの新規画像作成処理は一度しか行なわれない。また例えばFilter Eにおけるフィルタ処理の適用画像は、既にFilter CからBuffer Bの関数が呼び出された結果として生成され、Buffer Bにキャッシュされているものと同じデータである。このように全てに対して同じソースからのデータが、フレームがずれることなく順番全ての処理オブジェクトに対して渡されるようになっていることがわかる。

7 アプリケーション例

MALibを利用したアプリケーションの例題として、道路を通る車両の認識を行なうアプリケーションの構築と実験を行なった。アプリケーションobjtraqは、グレースケールの入力動画像から移動物体を抽出する。

objtraqに入力する動画像のスナップショットを図6に示す。

図6: オリジナル画像

この瞬間は、画面右上に駐車しているトラックと、画面中央を左から右へ通行している小型車、および画面下側を右から左へ通行しているタクシーの3台の車両が画像に表示されている。objtraqは移動物体のみを認識するため、中央左の小型車と左下のタクシーが認識の対象となる。objtraqを実行すると、上記の入力画像の他、
認識途中のパターンを入力画像に重ねて表示するマスク画像（図7）と認識結果を表示するトラッキング画像（図8）の合計3枚のウィンドウが表示される。

図7: マスク画像

図8: トラッキング画像

図8には、移動している2台の車両が認識されていることがわかる。実際の表示においては、リアルタイム認識として連続的に認識処理が行われていることが示される。

上記の処理を実現するコードの主要部を図9に抜粋する。ライブラリ MAlib が高度にモジュール化されているため、MAlib の各機能を接続することで実現する画像処理の主要部のコード量は約70行と、非常にコンパクトな記述となっていることに注目される。

またobjtraqの認識処理を構成する画像処理オブジェクトのリンク構造を図10に示す。画像処理のプロセスが、Source → Filter → Buffer → … → Holder（表示）といったリンク構造として示されている。

なお認識処理の中心は図10におけるfilter1、filter2、filter3、およびfilter5の各フィルタであり、順に空間平滑化、フレーム間差分による移動物の検出、ブロックセグメンテーション、移動物体のトラッキングの各処理を実現するものである。
8 今後の計画

最後にプロジェクトの今後の計画について説明する。プロジェクトの将来計画を説明するにあたり、本プロジェクトの背景とビジネスモデル、企業活動におけるオープンソース開発のあり方について補足する。また今後解決すべき技術的な課題を挙げる。

本プロジェクトは情報処理振興事業協会による平成12年度末期ソフトウェア創造事業の一環として実施された。未踏ソフトウェア創造事業は当協会その他の一般的な助成事業と異なり、その支援対象が組織に対してではなく個人に対するものであるという重要な特徴を有している。

企業の業務においては、その開発成果物の著作権は顧客あるいは所属組織に属する場合が一般的である。しかし本件に関しては当該事業の目的に応じ、知的所有権については開発者個人に属するものとなっている。したがって、オープンソースソフトウェアとして公開する点に関する業務上の制約は存在しなかった。

一方、現状の様々な事例に散見されるように、オープンソースソフトウェアに基づいたビジネス展開については、多くの事業者がビジネスモデルを模索中の段階である。筆者の所属する組織でも、オープンソース活動を支援することで利益を上げる様々な方法の検討が行われている。

そのなかで筆者らは、オープンソースプロジェクトとして本プロジェクトの開発を進めるとともに、本プロジェクトの成果物である MAlib をベースとした個別のアプリケーション開発あるいは MAlib を利用したサービスの提供で収益を得る事業構想を想定している。なおライブラリ利用者の便宜を図るためといった理由に加え、このような柔軟な運用にも合致させるためにも、本プロジェクトにおいてはライセンス形態として LGPL を採用する必要があった。

MAlib はまだ基本的な枠組みができあがったばかりの段階であり、まだ実装が不十分な箇所もある。重要な検討項目として、音声と画像を同期させたデータ構造の定義、様々なパラメータや処理結果データを伝搬する方法の検討、画像データの差異（サイズや色深度など）を吸収し統一的に扱う手法の確立などが解決すべき項目として残されており、今後の課題である。

9 まとめ

本プロジェクトでは汎用動画処理ソフトウェアライブラリ MAlib を、C 言語を用いたオブジェクト指向設計によって汎用性と保守性を両立させつつ実現した。またその利用例として、移動物体認識アプリケーションが簡単に実装できることを示す。さらに本プロジェクトの位置づけを示し将来展望を説明するにあたり、補足として企業におけるオープンソース開発について述べた。

MAlib は現在、次の URL で公開している。

http://www.malib.net/

先に述べたとおり本プロジェクトは引き続き開発を行なっており、共同開発者やユーザも鉄意募集中である。興味のある方はぜひもう上記 URL を参照していただき、コメントを頂ければ幸いである。

参考文献