DirectShow のフィルタの開発技法

要約

この文書は、DirectShow によるフィルタを開発するための有用な知識を書き残したものです。
基本的には、C または C++での Windows プログラミングを習得している人を対象に書いていますが、最悪 C 言語をなんとか習得している人でも遅れないように若干の注釈を付けています。(多分この程度の注釈ではどうにもならないでしょうが…)
正直な話、ある程度、複数のプラットフォームでの開発経験の無い人には DirectShow のプログラミングは難しいものだと思いますので、腕に自信の無い方は、覚悟して始めてください。

開発環境の説明

今回は、開発環境に VisualC++6.0、ライブラリに、DirectX SDK9c を使用しています。

今回 Web カメラと動画の制御に使用している DirectShow は、そもそもマイクロソフトの開発している DirectX のシステムの一部である、DirectX は通常の Windows 環境でのプログラミングでは実現できない、高速な画面描画や周辺機器の制御等を実現するために開発された API(アプリケーション・プログラム・インターフェース)群である。

DirectX はいくつかのコンポーネント(部品群)に分かれており、3D のレンダリングや 2D 画面の高速な描画等を行なうための DirectGraphics、ゲームコントローラーの対応を行う DirectInput 等があることから、特に Windows 環境でのゲーム開発等で使われているが、今回使う DirectShow は、動画の圧縮／展開、カメラ等のデバイスの制御の機能等、音声や動画の再生に関する機能を中心に預かっており、これを使うことで動画像処理に必要なものを一通りそろえることができる。

DirectShow プログラミングの必要知識

DirectShow は DirectX のその他の API と同じく、マイクロソフトの提唱している COM(コンポーネント・オブジェクト・モデル)に基づいて開発されている。そのため、DirectShow でプログラミングを行なう際には通常の Windows でのプログラミングに加えて COM プログラミングの作法を知っておく必要がある。

COM プログラミングに関する基礎知識がない場合には、文書最後の「COM プログラミングの基礎」の項を参照すること。

また、それ以上に DirectShow に関する資料は書籍、Web を見ても非常に少なく、プログラミング初学者が DirectShow を扱おうとすると、必ず苦労させられる事を付け加えておきます。

(実際自分が Web 上で資料を集めているときも、その資料の少なさを嘆く人たちを多く見かけています)

フィルタとフィルタグラフの解説

ここからは実際にDirectShow内部の解説を行なう。DirectShowは、カメラ、ディスプレイ、動画の圧縮／展開等を行なうプログラムの集合である。DirectShowではこれらの事を「フィルタ」と呼び、プログラム中で、これらのフィルタ同士を接続していくことで、カメラ映像のキャプチャや、動画の再生などを行なう。

呼び出したフィルタは「フィルタグラフ」と呼ばれるプログラムに登録し、その内部で管理されて、フィルタ同士の接続／切断などを行なわれる。

つまりは、Directshowで何か作業をさせようとすると、次の知識が必要となる。

1. DirectshowのAPIを、WINDWS標準のAPI群に組み込んで利用する知識
2. Directshowのフィルタを呼び出し、正しく接続する知識
3. 欲しい機能が、準備されていない場合、フィルタを開発するための知識

等である。

Directshowに付いているフィルタを探してみると、画像を変換、または解析するフィルタは非常に少なく、これらの機能は、自力で開発する必要があると思われる。

この文書では、このDirectshowでのフィルタ開発に関して、分かっていることをまとめておく。

DirectShow フィルタの製作法

DirectShow でのフィルタを１から開発するのは正直な話、現実的ではない。

そこで、DirectX SDK 付属のクラスを継承して、サンプルを手本に作成するのが良い。

DirectShow 画像変換フィルタの基本

DirectShow は前に解説した様に、COM に対応したプログラム群である。そのため、COM プログラミングに対応した、作法を知る必要があるが、DirectShow のフィルタはこれに次の作法を継ぎ足した dll であると言い換えることができる

- フィルタグラフに登録するときの、フィルタの基本情報
- フィルタ同士を接続するための、ピンの作成
- 実際の画像変換の関数

これらの基本的にフィルタの機能から開発していくのは大変な労力がかかる。

そのため、Directshow 開発者側が、SDK 内部に基本的な機能を備えた基底クラスや、それらを利用したサンプルフィルタを作成している。

そのため、これらを利用して基本的な機能を実装してしまい。欲しい機能の開発のみに注力するのが正しいフィルタ開発の作法といえる。

DirectShow 基本クラス

（このドキュメントは DirectShowSDK 日本語ドキュメント内の、「DirectShow の使い方 ＞ DirectShow フィルタの開発 ＞ 変換フィルタの作成 ＞ 手順 1. 基底クラスの選択」を基準に取っています。）

DirectShow のフィルタを開発する際には、SDK 付属の基本クラスを継承して、最低限の機能を実装して、それを発展させながらフィルタを開発していく。

SDK の基本クラスは次のものがある。

- CBaseFilter　：　すべてのフィルタの基本クラス。汎用性が高い代わりに書くべきコードが多くなる。
- CTransformFilter　：　画像変換フィルタ。画像Ａを画像Ｂに、コピーした後に変換する。
- CTransInPlaceFilter ： 画像置換フィルタ。送られてきた画像を書き換えるフィルタである。
- CVideoTransformFilter　：　主にビデオデコーダー用に設計されており。使用する機会は無い。

この中で、画像解析で使うのフィルタを作る場合、実際に基底クラスとして選択するのは CTransformFilter か CTransInPlaceFilter になる。

どちらを使っても良いが、どちらの方が適しているかは、使う画像変換アルゴリズムによる。

お勧めのサンプルソース

DirectShow フィルタのサンプルフィルタから、お勧めのものは次の 2 つである。

- CtransformFilter を基底クラスとするなら EZRgb24 フィルタ
- CTransInPlaceFilter を基底クラスとするなら、NullInPlace フィルタ

「DXSDK¥DirectShow¥C++¥Filter」フォルダ内部にサンプルコードがあるので、いずれかを利用して開いてみる。

このとき、オリジナルのコードをどこかに保存しておくのを忘れないこと。

実際にフィルタを登録、利用するためには SDK 付属の、GraphEdit.exe を利用することになる。使い方は後の項に載せておくので参照してほしい。

- EZRgb24 フィルタは簡単な画像変換フィルタである。色の反転や画像のぼかし等の基本的な変換を最初から実装している。CTransInPlaceFilter でのフィルタ開発を行う場合にも、画像変換部分のコードはのぞいておいて損はない。
- NullInPlace は何の画像変換の機能も持っていない単純なコピー変換フィルタである。CTransInPlaceFilter で開発を場合には、無駄なコードがいっさい入っていないため、これにコードを継ぎ足していけば行けばフィルタの開発が簡単に出来る。

GraphEdit の説明

DirectShow は、フィルタと言われるプログラムを、相互に接続していくことで、欲しい機能を実現していく。

これらのフィルタの接続状態を、グラフィカルに表示しながら変更できるツールが GraphEedit である。

普通、動画は配布する際に圧縮されているので、動画を再生するには

1. 動画読み込みフィルタ
2. 動画の展開フィルタ
3. ディスプレイへの画像の転送フィルタ

と繋がることになる。

実際には音声がついているので、フィルタの接続はもっと複雑になるが、画像の変換フィルタをテストするには、上の２と３の間に画像の変換フィルタを挟んでやれば、変換された動画を見る事ができる。

これを理解したうえで GraphEdit の使い方を説明する。

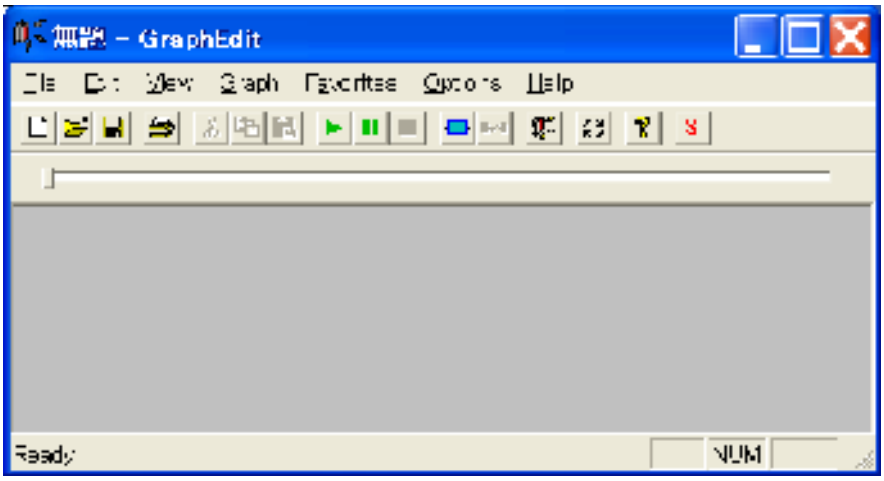
GraphEdit の使い方

GraphEdit は

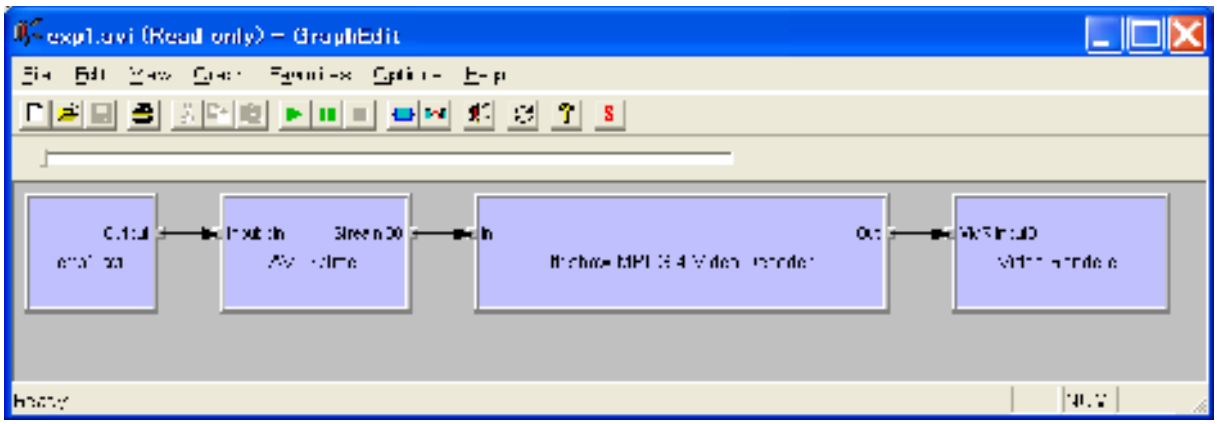
「スタートメニュー ＞ プログラム ＞ DirectX 9.0 SDK ＞ DirectX Utilities ＞ GraphEdit」

で呼び出せます。結構使う機会が多いのでショートカットを作っておくと良いかもしれません。

（ちなみに、実行ファイルは「DXSDK¥Bin¥DXUtils¥graphedt.exe」に入っています。）



GraphEdit を開くと、メニューから「File ＞ Render Media File」で何でも良いので動画ファイルを開いてみましょう。

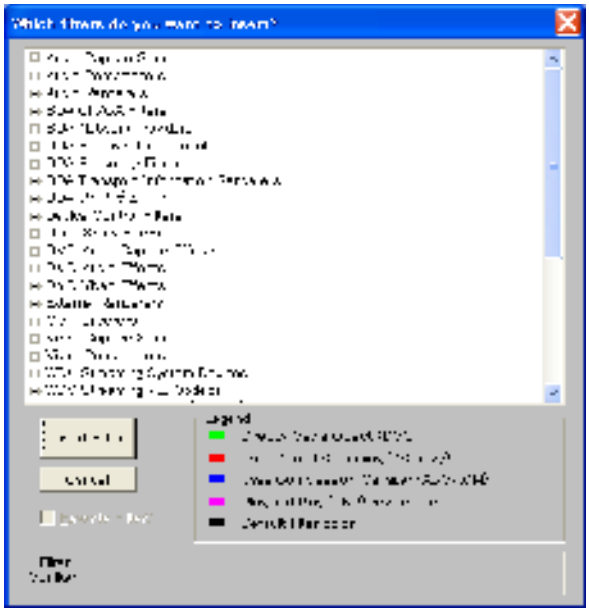


このように、ファイルを再生できるようにフィルタを勝手に繋いでくれます。

あとは、自分の繋げたいフィルタを呼び出す方法です。

フィルタを指定して呼び出す

メニューの「Graph ＞ Insert Filters」を呼び出してみましょう。



このダイアログから好きなフィルタを指定して呼び出します。

画像変換フィルタをOSに登録すると、普通は「DirectShow Filters」のカテゴリに入っています。

フィルタ接続の接続

Directshowのフィルタ同士は、必ず接続出来るというわけでは無い。

当然だが、音声を展開、変換するフィルタと画像を扱うフィルタ同士を接続出来はしない。

なので、フィルタ同士は接続するときにそれぞれが何を扱うフィルタなのかと確認して、

出来る場合にのみ接続を行う。

フィルタ接続の Tips

幾つか使っている間に上手くいかなかったこと等に関する経験論を書いておきます

フィルタは全部切断してつなげなおそう

既に繋がっているフィルタに新しくフィルタを挟み込むとき、データのタイプはおかしくないのにフィルタが繋がらない場合がある。

フィルタ同士は接続するときに、毎秒何フレーム処理をするか等の細かい設定を決める。

そのため。新しくフィルタを挟み込む場合、詳細の設定を詰めるときに矛盾がおきて接続が出来ない場合がある。

これを回避するためには、いったん全てのフィルタの接続を切って、もう一度最初の方のフィルタから繋ぎ直すと上手くいく場合がある。

ColorSpaceConverter を入れると良いよ。

これは、何でか分かりませんが、開発した画像変換フィルタをきちんと繋いで再生をすると、強制終了する場合がある、一番考えられる原因は、プログラムにバグを仕込んでしまった可能性だが、バグが無いのに繋がらない場合には、画像変換フィルタの両端に ColorSpaceConverter フィルタを挟んでやると良い。

どうも色空間が適切だとバグが起きない様で、上手く行く場合があります。

便利なフィルタの紹介

DirectShow で標準で付いているフィルタでも、特に画像変換フィルタの開発に便利なフィルタを紹介します。

ColorSpaceConverter

色座標系を適切なものに変換します。

光の３原色から、RGB 座標系が有名ですが、他にも色座標系は数多くあり、印刷関係では CYMK 座標系が有名です。

動画の場合は YUV 座標系が一般的ですので、画像変換フィルタと繋ぐ場合には

YUV 座標系から RGB 座標系に変換、またはその逆を行います。

SmartTee

画像の入り口が一つしかないのに、出口が２つあるフィルタです。

元々が送られてきた画像を、ビデオを再生するための接続と、ビデオを保存するための接続に分割するためのフィルタですが、それ以外でもビデオのライン分割をしたい場合には大体使えます。

画像変換後の映像と、変換していない映像を同時に表示したい場合に便利だと思われます。

VideoRenderer

標準のビデオ再生フィルタ。

送られてきた映像を、ディスプレイに転送します。他にもビデオ再生フィルタはありますが、これが標準で、一番機能がシンプルなものです。

実際のフィルタ開発

さて、ここまででフィルタの開発に必要な基礎の開発はすべて終わったので実際にコードを覗きながらフィルタの開発を行なってゆく。

フィルタを開発する際には、プログラムをDirectShowに対応させて、動くための最低限のコードの量が多く、その開発途中でミスがあってプログラムが動かない場合、何処が原因であるか分かりにくい。そのためSDK付属のサンプルを改造してしまうのが良い方法である。

開発には先程説明したように、EZrgb24、またはNullInPlaceを基礎に開発を行なうのが良いと思われるが、今回は Ezrgb24 フィルタをベースに開発を行い、それを解説していく。

(この文書は、基本的にDirectX SDK付属のヘルプの要約となっています。DirectShowの詳しい説明を行なっている書籍やWebページは大変少なく、SDK付属のヘルプは常に最重要資料となります。ほとんどの内容はDirectShow＞DirectShow の使い方＞DirectShow フィルタの開発にありますので、そちらを参照してください)

EZrgb24フィルタの改造

サンプルのフィルタで書き換えるべき場所は

- フィルタグラフに登録するときの、フィルタの基本情報
- フィルタ同士を接続するための、ピンの作成
- 実際の画像変換の関数

の3箇所であるが、今回サンプルで選んだEZrgb24の場合、フルカラーの画像の書き換えを行なって吐き出すだけのフィルタなので、ピンの部分の書き換えは必要ない。

なので、基本情報の書き換えと画像変換の方法を解説する。

基本情報の書き換え

ここで最低限書き換えなければいけない基本情報は、OS側でフィルタの内容を識別するための

GUIDだけである。

「Ezuid.h」を開いて

```
// Special effects filter CLSID
// { 8B498501-1218-11cf-ADC4-00A0D100041B }
DEFINE_GUID(CLSID_EZrgb24,
0x8b498501, 0x1218, 0x11cf, 0xad, 0xc4, 0x0, 0xa0, 0xd1, 0x0, 0x4, 0x1b);
```

となっているところをDEFINE_GUID内部を書き換える。

GUIDの説明

Global Unique ID の略で、OSがCOMに対応したDLL等を管理するために必要な128ビットのID。

GUIDの生成方法は多くあるが、マイクロソフトの配布しているguidgen.exeを使用するのが良いと思われる。

「スタートメニュー＞ファイル名を指定して実行」で「guidgen.exe」と実行すると呼び出せる。インターフェースが非常に単純なので、使い方は見れば分かります。

レジストリへの登録

最後に出来たフィルタをOSに登録してやる必要がある。

EZrgb24のプロジェクトをビルドして出来る「Debug_Unicode」フォルダの中にある「ezrgb24.ax」が実際のフィルタであるので

これも「スタートメニュー＞ファイル名を指定して実行」を呼び出して

「regsvr32.exe 作成したフィルタのファイル名」

を実行してやる。成功すると、それを示すダイアログが表示されるので、これで基本的な作業は終わりである。

実際の画像変換箇所

画像変換フィルタ内部は、多くのメソッドがあり、迷う事もあるかも知れないが、普通は書き換えるべき箇所はたった一カ所、 Transformメソッドだけである。

ここで実際の画像の変換を行っている。

（これは、CTransformFilterでもCTransInPlaceFilterを基底に選んでも同じである。）

その他のメソッドは、フィルタ同士の接続を行うための手続きを自動的に行ったり、OSにフィルタの情報を登録したりするためのものが殆どで、最初は触る必要がない。

Transformメソッド内部の書き換え

Transformメソッドは、IMediaSampleという形式で、画像のデータを受け取る。

なので、画像の変換を行う場合には

1. IMediaSample形式で受け取ったデータから、画像データ部分へのアクセスを準備する。
2. 画像を順番に捜査していって、画像処理を行う。

CtransformFilterを継承して作ったフィルタでは、さらに、出力側に画像を代入する必要があるが、おおむね同じ流れで上手くいく。

Ezrgb24フィルタ内部では

```
HRESULT CEZrgb24::Transform(IMediaSample *pIn, IMediaSample *pOut)
HRESULT CEZrgb24::Transform(IMediaSample *pMediaSample)
```

の２つのTransfomメソッドがあるが、今回は引数が２つある「Transform(IMediaSample

*pIn, IMediaSample *pOut)」の方を書き換える。

実際の中身だが、画像処理の実際のアルゴリズム以外は、手順が決まっているので、コピーして利用してしまうのがよい。

なので Transform メソッド内部は次のコードをコピーして必要な箇所のみ書き換えるのがよい。

```
HRESULT CEZrgb24::Transform(IMediaSample *pIn, IMediaSample *pOut){
	AM_MEDIA_TYPE* pType = &m_pInput->CurrentMediaType();
	VIDEOINFOHEADER *pvi = (VIDEOINFOHEADER *) pType->pbFormat;
	ASSERT(pvi);

	BYTE *pD, *pD2; // もらったメディアサンプルの画像への参照を取得
	CheckPointer(pIn,E_POINTER);    // もらったデータが正しいかチェック
	pIn->GetPointer(&pD);   // ここで IMediaSample 内部の画像へのポインタを取得
	CheckPointer(pOut,E_POINTER);
	pOut->GetPointer(&pD2);

	// 画像処理に便利なように RGBTRIPLE 型に変換。これで赤、緑、青などの色成分を簡単に取り出せます
	RGBTRIPLE *irgb=(RGBTRIPLE*) pD, *orgb=(RGBTRIPLE*) pD2;
	// 画像の幅と高さをもらう
	int h=pvi->bmiHeader.biHeight, w=pvi->bmiHeader.biWidth;
	// 画像を上から順番に走査する
	for( y=0; y < h; y++){
		for( x=0; x < w; x++, irgb++, orgb++) {
			// ここに、画像変換のコードを書き込みます
			// 今回は入力側から、出力側へ画像を完全コピー
			irgb->rgbtRed = orgb->rgbtRed; // 赤成分をコピー
			irgb->rgbtGreen = orgb->rgbtGreen;      // 緑成分をコピー
			irgb->rgbtBlue = orgb->rgbtBlue;        // 青成分をコピー
		}
	}
	return NOERROR;
}
```

何でコピー変換するのか？

ちょっと関係無いかも知れませんが、解説。

画像変換フィルタは何故、画像のコピーを必ず行うのかと言う理由が、最初分かっていなくて失敗したので。

フィルタの間を画像を通る場合には、IMediaSample のデータで毎度通ることによって動

いているが、データはフィルタの間をポインタを送ることで処理しているので、フィルタのデータというのは、全部同じものなのです。

入力側の画像をいったんコピーしないと、他のラインも変換してしまった画像がフィルタグラフを通ってしまうのです。

ポインタを利用した画像処理

Directshow 内部で画像処理を行い場合には、フィルタに送られて来た画像の処理は、高速化のために、ポインタを利用している。

Windows プログラミングを修得して DirectX を学びだしている場合、C 言語のポインタが分からない。なんて事は通常は無いと思いわれるが、一応、ポインタを利用した画像処理プログラミングの方法を復習しておく。

（何でかと言いますと、理由を正直に言います。私は一瞬困りました。）

画像はピクセルと呼ばれる、小さな点の集合で出来ていますが、番号を付けていくと、コンピュータ内部では、次の様にメモリに格納されている。

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

図 1: メモリ内部の画像の配置

そしてポインタとは、データの「メモリ上の位置」を持っているものです。

なので、画像データの最初の置き場所から、順繰りに移動して画像処理を行いたい場合には、次のピクセルの置き場まで移動すればよい。

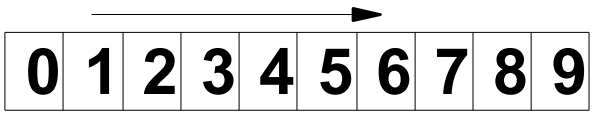


図 2: ポインタの移動イメージ

このときは、画像のポインタをひとつインクリメントしてやれば良い。

```
irgb++;          // 次のピクセルに移動
```

これで画像の次のピクセルへと移動できる。

なので、一つ右や一つ左のピクセルは

```
irgb++;// 次のピクセルに移動
irgb--;// 前のピクセルに移動
```

で移動できる。では上下のピクセルに移動するにはどうすれば良いのかというと

```
// int w=pvi->bmiHeader.biWidth;        // これで画像の幅を取得
irgb += w;      // 一つ下のピクセルに移動
irgb -= w;      // 一つ上のピクセルに移動
```

という風に、画像の幅の分だけポインタを移動してやれば、良い。

これはDirectshow内部だけでなく、すべての画像処理プログラミングに共通して言える事である。

(送られてくる画像形式の環境によっては、上下が逆になっている場合がありますので注意してください。Directshowでの画像変換の場合は画像が上下逆に送られてきている…、はずです。経験上)

便利なコード集

ここでは、開発途中に発見した、あると便利なコードを書き残しておく。

デバイスコンテキストに画像データを送る。

ポインタを辿りながら、画像処理を行うが、場合によってはテキストの描画や、円や線などの図形を描画する必要が出てくる。

これを行うには、Windows の標準 API が使えると大変便利である。

そのためには、受け取った画像データを、Windows の DC(デバイスコンテキスト)形式に変換する必要がある。

```
HRESULT CEZrgb24::Transform(IMediaSample *pIn, IMediaSample *pOut){
    BYTE *pD, *pD2;

    AM_MEDIA_TYPE* pType = &m_pInput->CurrentMediaType();
    VIDEOINFOHEADER *pvi = (VIDEOINFOHEADER *) pType->pbFormat;
    ASSERT(pvi);

        CheckPointer(pIn,E_POINTER);
        pIn->GetPointer(&pD);
        CheckPointer(pOut,E_POINTER);
        pOut->GetPointer(&pD2);

    HRESULT hr = Copy(pIn, pOut);       // 画像を入力から出力にコピー

        int h=pvi->bmiHeader.biHeight, w=pvi->bmiHeader.biWidth;

        BITMAPINFO biBMP;
        ZeroMemory(&biBMP, sizeof(biBMP));
        biBMP.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
        biBMP.bmiHeader.biBitCount = 32;
        biBMP.bmiHeader.biPlanes = 1;
        biBMP.bmiHeader.biWidth = w;
        biBMP.bmiHeader.biHeight = -1*h;

        HDC hdc=NULL;   // 画像を代入するハードウェアデバイスコンテキスト
        // これで画像データを Windows の DIB 形式 Bitmap に変換
        HBITMAP hbmpBMP = CreateDIBSection(NULL, &biBMP, DIB_RGB_COLORS, (LPVOID*)(&pD2), NULL, NULL);
        if( NULL==hbmpBMP )     return NULL;
```

```
        // DIBSectionのHBITMAPをメモリデバイスコンテキストに選択
        HBITMAP hbmpOld = (HBITMAP)SelectObject(hdc, hbmpBMP);
        // デバイスコンテキストに画像を転送
        BitBlt( hdc, 0, 0, w, h, (HDC)hbmpOld, 0, 0, SRCCOPY );

        // ここの箇所に画像編集関連のコードを挟みます
        TextOut( hdc, 10, 10, L"Hello", 5 );    // テキストの書き込み
        // ここで編集終了

        hbmpOld=CreateCompatibleBitmap( hdc, w, h );
        GetDIBits(hdc, hbmpOld, 0, h, (LPVOID*)(&pD2), (LPBITMAPINFO)&biBMP, DIB_RGB_COLORS);

        SelectObject(hdc, hbmpOld);     // DIBSectionをメモリデバイスコンテキストの選択から外す
        DeleteObject(hbmpBMP);  // DIBSectionを削除
        DeleteDC(hdc);  // メモリデバイスコンテキストを削除*/

        return NOERROR;
}
```

となる。

（コード自体は２ちゃんねるのDirectshowスレッドから見つけてきたものですので、必要とあらばそちらの方も参照してください）

http://pc11.2ch.net/test/read.cgi/tech/1162913156/571-572

送られてきたIMediaSampleデータの保存。

フィルタに送られてくるIMediaSampleのデータは、入力データと出力データは、新しくデータが送られてくると上書きされて消えてしまう。

なので、動画データの場合、ある時点での画像データをメモリに保存しておきたい場合は、IMediaSampleでメモリ上にコピーを保存しておけると大変有用である。

手順は２つに分かれていて

1. IMediaSample形式のデータを保存するための領域を準備する
2. 準備した領域にデータをコピー

と言う流れになる。

バッファの確保は次の２つのメソッドを追加すると良い。

```
/**
 * IMemAllocatorを使って、送られて来たIMediaSample型のデータをメモリ上に保存するための領域を準備します
```

```
 */
HRESULT CEZrgb24::CreateCopySamples(){
        HRESULT hr=NULL;
        IMemAllocator *pAllocator=NULL;
        m_pInput->GetAllocator( &pAllocator );
        ALLOCATOR_PROPERTIES Props, Actual;

        hr=CreateMemoryAllocator( &m_TmpSmpAlloc );
        //hr=CoCreateInstance(CLSID_MemoryAllocator, 0, CLSCTX_INPROC_SERVER, IID_IMemAllocator,
(LPVOID*)ppAllocator);

        if(SUCCEEDED(hr))
            hr = pAllocator->GetProperties(&Props);

        Props.cBuffers=3;

    if (SUCCEEDED(hr))
        hr = m_TmpSmpAlloc->SetProperties(&Props, &Actual);
        if(SUCCEEDED(hr))
                hr=m_TmpSmpAlloc->Commit();
        return hr;
}
```

```
/**
 * 第２引数でもらったサンプルを第１引数のサンプルと同じプロパティで作成します
 */
HRESULT CEZrgb24::CreateCopySample(IMediaSample *pSample, IMediaSample **ppOutSample){
        HRESULT hr=NULL;
        IMediaSample *pOutSample=NULL;
        AM_SAMPLE2_PROPERTIES * const pProps = m_pInput->SampleProps();
        DWORD dwFlags = m_bSampleSkipped ? AM_GBF_PREVFRAMESKIPPED : 0;

        hr = m_TmpSmpAlloc->GetBuffer(
                &pOutSample,
                pProps->dwSampleFlags & AM_SAMPLE_TIMEVALID ?&pProps->tStart : NULL,
                pProps->dwSampleFlags & AM_SAMPLE_STOPVALID ?&pProps->tStop : NULL,
                dwFlags
        );
        *ppOutSample = pOutSample;
        if (FAILED(hr))
```

```
            return hr;

    ASSERT(pOutSample);
    IMediaSample2 *pOutSample2=NULL;
    hr=pOutSample->QueryInterface( IID_IMediaSample2, (LPVOID*)&pOutSample2 );
    if( SUCCEEDED(hr) ){//  Modify it
            AM_SAMPLE2_PROPERTIES OutProps;
            EXECUTE_ASSERT(SUCCEEDED(pOutSample2->GetProperties(
        FIELD_OFFSET(AM_SAMPLE2_PROPERTIES, tStart), (PBYTE)&OutProps)
    ));
    OutProps.dwTypeSpecificFlags = pProps->dwTypeSpecificFlags;
    OutProps.dwSampleFlags =
        (OutProps.dwSampleFlags & AM_SAMPLE_TYPECHANGED) |
        (pProps->dwSampleFlags & ~AM_SAMPLE_TYPECHANGED);

    OutProps.tStart = pProps->tStart;
    OutProps.tStop  = pProps->tStop;
    OutProps.cbData = FIELD_OFFSET(AM_SAMPLE2_PROPERTIES, dwStreamId);
    hr = pOutSample2->SetProperties(
        FIELD_OFFSET(AM_SAMPLE2_PROPERTIES, dwStreamId), (PBYTE)&OutProps
    );
    if (pProps->dwSampleFlags & AM_SAMPLE_DATADISCONTINUITY)
        m_bSampleSkipped = FALSE;

    pOutSample2->Release();
} else {
            if (pProps->dwSampleFlags & AM_SAMPLE_TIMEVALID)
        pOutSample->SetTime(&pProps->tStart, &pProps->tStop);

    if (pProps->dwSampleFlags & AM_SAMPLE_SPLICEPOINT)
        pOutSample->SetSyncPoint(TRUE);
    if (pProps->dwSampleFlags & AM_SAMPLE_DATADISCONTINUITY) {
        pOutSample->SetDiscontinuity(TRUE);
        m_bSampleSkipped = FALSE;
    }
    // Copy the media times

    LONGLONG MediaStart, MediaEnd;
    if (pSample->GetMediaTime(&MediaStart,&MediaEnd) == NOERROR)
        pOutSample->SetMediaTime(&MediaStart,&MediaEnd);
}
```

```
    return S_OK;
}
```

コピーを行う場合には、Copy メソッドが EZrgb24 フィルタ内部に既にあるので。これを使えばよい。

（コード自体は DirectX SDK のコードから、CTransform フィルタ内部のコードを探って見つけたものを下敷きにしています。）

Directshow フィルタのプロジェクトファイルの改造

Directshow フィルタの開発途中に、ezrgb24 を下敷きにしていたのですが、もう全然違う作業をさせているというのに、「ezrgb24.ax」という名前で DLL を生成しているのが不自然だったので、これを直す方法を探してみました。

基本的には、新しいプロジェクトファイルを作成して(やる必要は特になかった)、そこで新しい DLL の名前を指定してやればよいだけです。

下敷きは、下のアドレスにある「オリジナルフィルタの作成法」ですので、こちらの方を良く参照して下さい。

http://www.geocities.co.jp/SiliconValley/7406/tips/dshow/dshow5.html

まずは、プロジェクトファイルを上の文書に従って、作成してください

その後に、以前作成していたときのプロジェクトから、ヘッダファイル、cpp ファイル、モジュール定義ファイル(def ファイル)をすべて移動して、新しいプロジェクトにとろくし直します。

その後にビルドしてしまえば、それだけで良いです。

ビルドに成功しない場合は、なんらかの作業ミスが考えられます。

ミスの例

fatal error C1083: ｲﾝｸﾙｰﾄﾞ ﾌｧｲﾙがｵｰﾌﾟﾝできません。'streams.h': No such file or directory

おそらくインクルードファイルの設定が適切に行われていません。

「インクルードファイルのパス 」か「追加ライブラリのパス 」を正しく設定してください。

ビルドに成功したフィルタが regsvr32 を使っても登録できない

これの原因は、モジュール定義フィル(def ファイル)の設定が間違っているためです。

「LIBRARY」で設定されている DLL の内部名を、プロジェクトファイルで設定しているモノにあわせてください。

プロジェクトファイルの変更時に設定しているはずですが、一応「設定　＞　'リンク'タブ ＞ '出力ファイル名'テキストボックス」の設定です。

```
;============================================================================
;  Copyright (c) 1992-2002  Microsoft Corporation.  All Rights Reserved.
;============================================================================

LIBRARY      EZrgb24.ax

EXPORTS
```

```
        DllMain                 PRIVATE
        DllGetClassObject       PRIVATE
        DllCanUnloadNow         PRIVATE
        DllRegisterServer       PRIVATE
        DllUnregisterServer     PRIVATE
```