## １．ＶＦＷの復習

Windows で以前より使われている動画関係の API である、VideoForWindowsAPI を使った動画ファイルを扱う基本プログラムを紹介します。

○ AVI ファイルと VideoForWindowsAPI

Windows では数多くの動画ファイル（メディア・ファイルとも呼ばれる）を扱うことができます。この中でいわゆる「計測」に適したものはデジタルビデオカメラのフォーマットに対応した DV-AVI ファイルです。

DV-AVI ファイルには Type1 と Type2 が存在し、VideoForWindowsAPI で扱えるものは Type2 の DV-AVI ファイルです。ですからデジタルビデオカメラから DV-AVI ファイルにキャプチャを行う際にはファイル型式を Type2 の DV-AVI ファイルを選ぶ必要があります。Type2 の DV-AVI ファイルのプロパティの例を図２に示します

この AVI ファイルからファイル情報や１フレーム毎の画像、音声を切り出すには VideoForWindowsAPI を利用しますが、動画計測で利用する１フレーム分の画像を得るには図４に示すいくつかの API 関数を利用します。AVI ファイルは論理的にはストリームと呼ばれる時系列データとして動画や音声が扱われます。VideoForWindowsAPI ではこのストリームをオープンして個々のデータをアクセスするというスタイルになります。


音声の取り出しなど操作範囲を広げた場合の VideoForWindowsAPI とハンドルとの関係を図４に示します。VideoForWindowsAPI の解説やリファレンスは MSDN の PlatformSDK/Graphics and Multimedia/Video for Windows にあります。

○実際のプログラミング

VideoForWindowsAPI 自体はメッセージ処理や MFC との関係など、いわゆるウィンドウプログラミングを意識せずに利用できます。ですが、処理結果をモニタする表示部分等が必要になることが多いので、結果的にウィンドウプログラミングが必要になります。手始めに DV-AVI ファイルの動画から１フレームづつ画像を取り出し描画ウィンドウに連続して表示するプログラムを紹介します。

リスト１ではウィンドウ周りの処理を専用のスレッドに任せ、またその部分を別リストにまとめることにより、動画の処理部分を通常のコンソールプログラムと同じ使い勝手でプログラミングすることを可能にしています。

リスト１では、動画の切り出しを行うメインスレッドとウィンドウプロシージャを含む表示部分のスレッドの両方からアクセスする表示データの構造体 IMG0 img00 を通じて全体の動作がまとめられています。

VFWAPI 関係の処理はストリームのオープン、フレーム切り出しの準備、フレーム切り出しの順で行い、動画中の任意の１フレームを DIB 形式のデータとして取り出します。この切り出しのあとは従来からの DIB 形式ですから各ピクセルについての処理が可能になります。

リスト１は表示だけなのでこの DIB を表示データの構造体 IMG0 img00 に設定して再描画要求を行います。再描画要求を行うことで、スレッド側で動画から取り出された DIB データの表示ウインドウへの描画動作が行われます。

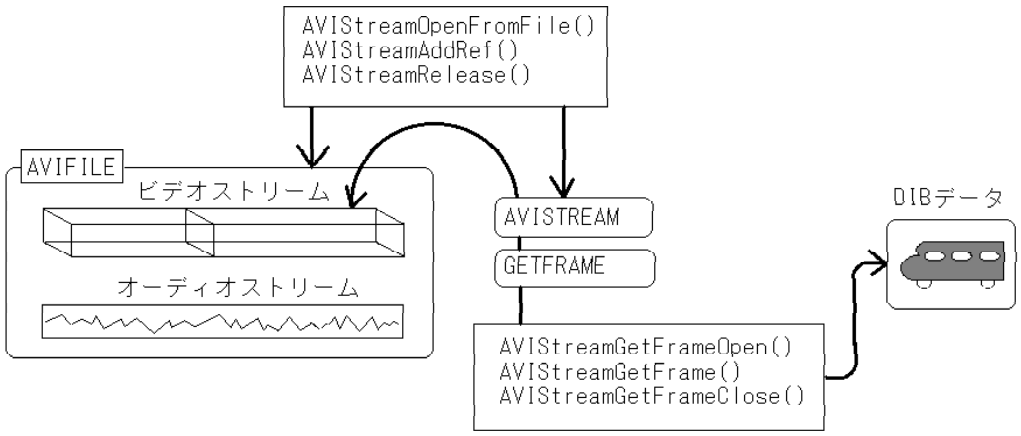


図３　AVI ファイルと VFW-API 関数との関係

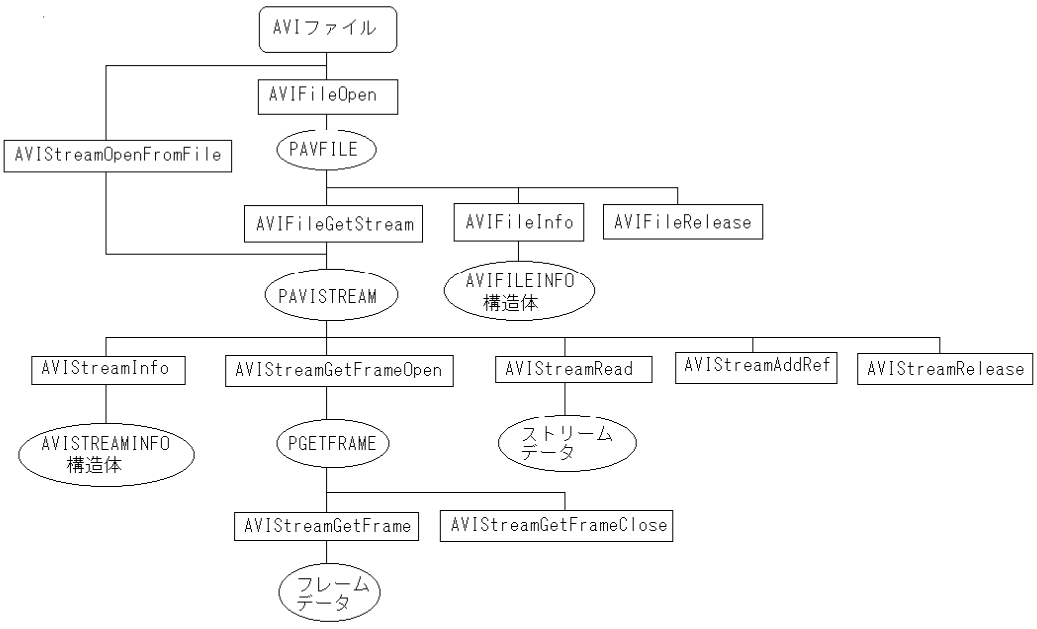


図４　VFW-API 関数の系統図

リスト１　list1a.cpp、list1b.cpp、list1.h

```
//  list1.h

#include <Windows.h>
#include <vfw.h>          // Video for Windows include file

// グラフィック用ウィンドウのデータをまとめた構造体
typedef struct{
    HINSTANCE         hi;
    int                   x;            // 表示開始位置
    int                   y;
    HWND              hwnd;         // 自分のウィンドウハンドル
    BYTE              *lpBmpData; // BMPのデータ部分
    BITMAPINFOHEADER  bih;
} IMGO;

int gr_reg( void );
void gr_init(IMGO *);
DWORD th_Proc( void *);
LRESULT CALLBACK grProc( HWND, UINT, WPARAM, LPARAM );
```

```
// list1_a.cpp
//
// 動画の連続処理例                  list1a.cpp
//   動作：AVIファイルから１フレームを連続して表示する
//         処理部分を書きやすくするため
//         コンソールアプリケーションの形態をとり
//         ウィンドウ関係の処理はスレッドに任せる
//   注意：VFWを使うのでVFWのコーディックが必要
//          コンパイルのC/C++の設定をマルチスレッドに変更
//          リンクのvfw32.libの追加が必要
//         動作マシンは24bitカラーであること
//
#include <stdio.h>
#include <conio.h>         // getch()の宣言

#include "list1.h"

char * file_open(void );

void main()
{
    IMGO img00;
    char * pfn;                          // 入力AVIファイルの名前
    int  ssw = 0;                   // ステップ処理のスイッチ

    AVISTREAMINFO sivi;
    PAVISTREAM pvis;
    HRESULT rc;
    char cbuf[100];

    pfn = file_open();

    rc = AVIStreamOpenFromFile( &pvis, pfn, streamtypeVIDEO, 0,
                                OF_READ, NULL);       // Videoストリームをオープン
    if( rc ) { printf( "v open err¥n" );
        return;
    }

    AVIStreamAddRef( pvis );            // ストリームの参照カウントを１増やす
    AVIStreamInfo( pvis, &sivi, sizeof(AVISTREAMINFO) );
    printf( "vi_Rate %7d : vi_Scale %6d : vi_Length %d ¥n",
            sivi.dwRate, sivi.dwScale,  sivi.dwLength);
    printf( "処理方法              コマ送り - 's'キー ¥n");
    printf( "                      連続処理 - その他のキー ?¥n" );
    printf("(連続処理中でも、どれかキーを押すと処理が中断されます)¥n");
    if(getch() == 's')
        ssw = 1;

    memset( &(img00.bih), 0, sizeof(BITMAPINFOHEADER) );
```

```
// ビットマップ情報をストリーム情報からコピー
img00.bih.biWidth  = sivi.rcFrame.right;
img00.bih.biHeight = sivi.rcFrame.bottom;
// ストリーム情報からは色、画像サイズは得られないが
// DV形式の24bitカラー使用を前提に画像サイズを計算する
// より柔軟性を持たせるのであるなら
// AVIStreamReadFormatで完全なビットマップ情報を得るのがベスト
img00.bih.biSizeImage = img00.bih.biWidth * img00.bih.biHeight * 3;
img00.bih.biSize      = sizeof(BITMAPINFOHEADER);
img00.bih.biPlanes    =  1;
img00.bih.biBitCount  = 24;

img00.hi = (HINSTANCE)GetWindowLong( HWND_DESKTOP, GWL_HINSTANCE );
img00.x = 100; img00.y = 100;

gr_reg();
gr_init(&img00);                // グラフィックウィンドウを生成

BYTE * pp;
BYTE * pp0;
pp0 = (BYTE *) malloc( img00.bih.biSizeImage );       // DIBのバッファを確保
PGETFRAME pvf;
LPBITMAPINFOHEADER pbmpih;
int r;
unsigned int  jj, i;
for( jj = 0; jj< sivi.dwLength; jj++){       // フレームのループ

    pvf = AVIStreamGetFrameOpen( pvis,       // 解凍の準備
          (LPBITMAPINFOHEADER)AVIGETFRAMEF_BESTDISPLAYFMT);
    pbmpih = (LPBITMAPINFOHEADER)AVIStreamGetFrame(pvf,jj);
                                             // jjフレームを取り出す
    pp = (BYTE *)pbmpih  + sizeof(BITMAPINFOHEADER);
    for( i = 0; i <  img00.bih.biSizeImage ; i++){
        pp0[i] = pp[i];                      // ユーザ定義の配列にコピー
    }
    AVIStreamGetFrameClose( pvf );           // VFWからのデータはリリース

    wsprintf(cbuf,"[%s] frame %d/%d ",pfn, jj, sivi.dwLength);
    SetWindowText( img00.hwnd, cbuf );       // キャプションのセット

    img00.lpBmpData = pp0 ;
    InvalidateRect( img00.hwnd, NULL, FALSE);         // 再描画を要求
                                            // 全面書換、背景は書き換えない
    if( ssw ){
        r = MessageBox(NULL,
                "次のフレーム-はい 前のフレーム-いいえ 連続表示-キャンセル",
                "...",
                MB_YESNOCANCEL);
        if( r == IDNO )
            jj = jj - 2;
        if( r == IDCANCEL )
            ssw = 0;
    }
    else{
        if( kbhit() ){
            getch();
            r = MessageBox(NULL,
                "終了-はい 続行-いいえ ステップ表示-キャンセル",
                "処理を強制終了しますか？",
                MB_YESNOCANCEL);
            if( r == IDYES )
                break;                       // forループを抜けだす
            if( r == IDCANCEL )
                ssw = 1;
        }
    }
}
AVIStreamRelease( pvis );
free( pp0 );

printf( "quit? push any key" );
getch();
SendMessage( img00.hwnd, WM_CLOSE, 0, 0 );        // グラフィック画面の終了
// PostMessage( hwnd, WM_QUIT, 0, 0 ) でもスレッドのメッセージループは終了するが
// PostQuitMessage()ではスレッドのメッセージループは終了しない。
// PostMessage( hwnd, WM_QUIT, 0, 0 ) の場合、
// ループ終了時点ではウィンドウは存在するが、スレッド終了とともに
// ウィンドウは強制終了する。
// これらの処理をしなくとも、main関数の終了に伴いスレッド、
```

```
    // 及びウィンドウも強制終了する

    return ;
}

//////////
char * file_open(void )
{
    OPENFILENAME fname;
    static char fn[256];
    char filename[64];
    memset(&fname, 0, sizeof(OPENFILENAME));
    fname.lStructSize    = sizeof(OPENFILENAME);
    fname.hwndOwner      = HWND_DESKTOP;
    fname.lpstrFilter    = "*.*¥0*.*¥0¥0";
    fname.nFilterIndex   = 1;           // １番目のファイルフィルタを使う
    fname.lpstrFile      = fn;          // パス付きファイル名が格納されるアドレス
    fname.nMaxFile       = sizeof(fn);
    fname.lpstrFileTitle = filename;
    fname.nMaxFileTitle  = sizeof(filename)-1;
    fname.lpstrTitle     = "AVIファイルを指定します";
    fname.Flags          = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
    if( ! GetOpenFileName( &fname ) )   return NULL;
    return fn;
}



// list1b.cpp
//         処理部分を書きやすくするため
//         コンソールアプリケーションの形態をとり
//         ウィンドウ関係の処理はスレッドに任せる
//          そのためのスレッド側、ウィンドウ側リスト
//   注意：VFWを使うのでVFWのコーディックが必要
//          コンパイルのC/C++の設定をマルチスレッドに変更
//          リンクのvfw32.libの追加が必要
//         動作マシンは24bitカラーであること
//
#include "list1.h"

// グラフィック用のウィンドウクラスの登録
int gr_reg( void )
{
    WNDCLASSEX    wc;                           // 新しくつくるウインドクラス

    memset( &wc, 0, sizeof(WNDCLASSEX) );
    wc.cbSize        = sizeof(WNDCLASSEX);
    wc.lpfnWndProc   = grProc;              // このクラスの持つウインドプロシージャ
    wc.hInstance     = (HINSTANCE)GetWindowLong( HWND_DESKTOP, GWL_HINSTANCE );
    wc.lpszClassName = "GRC0";              // このクラスの名前
    wc.cbWndExtra    = 10;                  // 関連する構造体のポインタ用のエリアを確保
    return RegisterClassEx( &wc );          // ウィンドクラスの登録
}

//  グラフィックウィンドウの生成, メッセージループ用スレッドの起動
void gr_init( IMG0 * pimg00)
{
    DWORD tid;

    pimg00 -> hwnd = NULL;
    CreateThread( NULL, 0,
                  (LPTHREAD_START_ROUTINE)th_Proc,
                  (void *)pimg00, 0, &tid );
                                                    // メッセージループのスレッドを起動
    while( !(pimg00 -> hwnd) );                 // ウィンドウが表示されるのを待つ
}

//  メッセージループのためのスレッド
DWORD th_Proc( void * pp)
{
    MSG msg;

    int sm0 = GetSystemMetrics( SM_CYCAPTION );
    int sm1 = GetSystemMetrics( SM_CXDLGFRAME ); // WS_OVRELAPPの場合、枠の太さは
    int sm2 = GetSystemMetrics( SM_CYDLGFRAME ); // SM_C?DLGFRAMEになる
    // 必ずスレッドの中でウィンドウを作る
    ((IMG0 *)pp) -> hwnd = CreateWindow( "GRC0", // クラスの名前
                  "...",
```

```
                    WS_OVERLAPPED | WS_VISIBLE,          // ウィンドウの属性
                    ((IMGO *)pp) -> x, ((IMGO *)pp) -> y,         // 表示位置
                    ((IMGO *)pp) -> bih.biWidth  + sm1 * 2,  // 描画サイズから大きさを計算
                    ((IMGO *)pp) -> bih.biHeight + sm0 + sm2 * 2,
                    HWND_DESKTOP,                        // 親はディスクトップ
                    NULL, ((IMGO *)pp)->hi, NULL         );
     SetWindowLong(((IMGO *)pp) -> hwnd, 0, (LONG)pp);
     while( GetMessage( &msg, NULL, 0, 0 ) ){
          TranslateMessage( &msg );
          DispatchMessage( &msg );
     }

     return 0;
}

//  ウィンドウプロシージャ，再描画のみを行う
LRESULT CALLBACK grProc( HWND hwnd, UINT msg, WPARAM wp, LPARAM lp )
{
     IMGO * pimg;
     PAINTSTRUCT  ps;

     pimg = (IMGO * )GetWindowLong( hwnd , 0 );    // 拡張ウィンドウメモリより
                                                   // このウィンドウ用データへの
                                                   // ポインタを取り出す
    switch (msg) {

     case WM_PAINT:                                // ビットマップの描画
          BeginPaint( hwnd, &ps );
          SetDIBitsToDevice( ps.hdc, 0, 0,         // コピー先x,y座標
                        pimg -> bih.biWidth,       // DIBの幅
                        pimg -> bih.biHeight,      // DIBの高さ
                        0, 0,                           // DIBの座標
                        0,                              // 走査線
                        pimg -> bih.biHeight,      // 走査線数
                        pimg -> lpBmpData,
                        (BITMAPINFO *)&( pimg -> bih),  // BITMAPINFOにキャスト
                        DIB_RGB_COLORS );
         EndPaint( hwnd, &ps );
          return 0;

     case WM_DESTROY:                              // スレッドの
          PostQuitMessage( 0 );                    // メッセージループを終了させる
          break;

     default:
          return DefWindowProc( hwnd, msg, wp, lp );
     }
     return 0;

}
```


図５　リスト１の実行画面