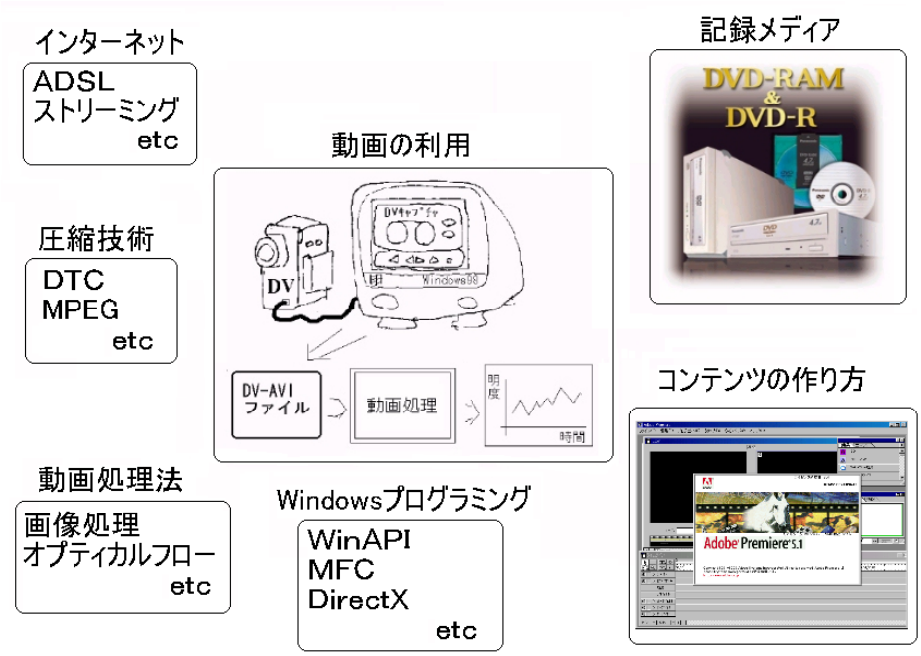
パソコンによる動画計測とその周辺

2002.03.08

目次

１．動画処理のための準備

1.1　動画をとりまく状況



動画処理は単に動画を扱うＡＰＩの知識以外のそのデータの大きさ、扱いなどに考慮した総合的な技術が要求されます。

1.2　WinAPIの準備

○ＢＭＰファイルを表示する～コンソールからウィンドウを表示する

計算処理や計測などの用途ではコンソール（ＤＯＳ窓）のインターフェースの方がプログラムが簡単で使い勝手がいい場合があります。コンソールでウィンドウを使う場合、一番やっかいなのがメッセージループをどうするかです。メッセージループを別スレッドで記述することによりこの問題は解決できます。

```
//   list88951.cpp  スレッドの応用  コンソール用グラフィックウィンドウ
//   コンソール側リスト

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>          // getch()の宣言
#include <math.h>
#include <windows.h>
#include <process.h>

#define      BFTYPE            'M'*256+'B'

typedef struct{
    BITMAPINFOHEADER  bih;
    BYTE              *lpBmpData; //BMPのデータ部分
} IMGO;

IMGO * p;

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

HINSTANCEhi;
HWND     hwnd;
int          x_size, y_size;                          // グラフの大きさ
int          sw = 0;

void main()
{
    IMGO img;

    p = &img;

    OPENFILENAME  fname;
    static char       fn[256];
    char          filename[64];
    static char       filefilter[] = "BMPファイル(*.bmp)¥0*.bmp¥0"
                                     "すべてのファイル(*.*)¥0*.*¥0¥0";

    memset( &fname, 0, sizeof(OPENFILENAME) );
    fname.lStructSize    = sizeof(OPENFILENAME);
    fname.lpstrFilter    = filefilter;
    fname.nFilterIndex   = 1;
    fname.lpstrFile      = fn;
    fname.nMaxFile       = sizeof(fn);
    fname.lpstrFileTitle = filename;
    fname.nMaxFileTitle  = sizeof(filename)-1;
    fname.Flags          = OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;

    if( !GetOpenFileName(&fname) )  return ;
    printf("file name [%s]¥n", fn );

    FILE  * fp;
    fp = fopen( fn, "rb" );

    BITMAPFILEHEADER bfh;

    if( fread( &bfh, sizeof(BITMAPFILEHEADER) , 1, fp ) !=  1 ) {
        printf("ファイル読み込みのエラー¥n");
        return ;
    }
    // ファイルヘッダ読み込み

    if ( bfh.bfType != BFTYPE ){
        printf( "%s はDIBフォーマットではありません", fn );
        return ;
    }
    printf( "------ BITMAPFILEHEADER ------¥n");
    printf( "WORD  ¥t bfType      ¥t %04X ('%c%c') ¥n",
        bfh.bfType, bfh.bfType % 256, bfh.bfType / 256);
    printf( "DWORD ¥t bfSize      ¥t %d(byte) ¥n", bfh.bfSize);
    printf( "WORD  ¥t bfReserved1 ¥t %d ¥n",       bfh.bfReserved1);
    printf( "WORD  ¥t bfReserved2 ¥t %d ¥n",       bfh.bfReserved2);
    printf( "DWORD ¥t bfOffBits   ¥t %d ¥n",       bfh.bfOffBits);

    if( fread( &(p -> bih), sizeof(BITMAPINFOHEADER) , 1, fp ) !=  1 ) {
```

```
        printf("ファイル読み込みのエラー¥n");
        return ;
    }

    printf( "------ BITMAPINFOHEADER ------¥n");
    printf( "DWORD ¥t biSize          ¥t %d ¥n", p -> bih.biSize);
    printf( "LONG  ¥t biWidth         ¥t %d ¥n", p -> bih.biWidth);
    printf( "LONG  ¥t biHeight        ¥t %d ¥n", p -> bih.biHeight );
    printf( "WORD  ¥t biPlanes        ¥t %d ¥n", p -> bih.biPlanes );
    printf( "WORD  ¥t biBitCount      ¥t %d ¥n", p -> bih.biBitCount );
    printf( "DWORD ¥t biCompression   ¥t %d ¥n", p -> bih.biCompression);
    printf( "DWORD ¥t biSizeImage     ¥t %d ¥n", p -> bih.biSizeImage);
    printf( "LONG  ¥t biXPelsPerMeter ¥t %d ¥n", p -> bih.biYPelsPerMeter);
    printf( "LONG  ¥t biYPelsPerMeter ¥t %d ¥n", p -> bih.biYPelsPerMeter);
    printf( "DWORD ¥t biClrUsed       ¥t %d ¥n", p -> bih.biClrImportant );
    printf( "DWORD ¥t biClrImportant  ¥t %d ¥n", p -> bih.biClrImportant );

    if( p->bih.biBitCount != 24 ){
        printf("本プログラムは24bitカラーのBMPファイルのみ扱えます¥n");
        return;
    }

    p -> lpBmpData = (BYTE *)malloc( p -> bih.biSizeImage );
    // BMPデータのDIB部分のバッファの確保

    if( fread( p->lpBmpData, 1, p -> bih.biSizeImage,  fp ) !=  p -> bih.biSizeImage ) {
        printf("ファイル(DIB部)読み込みのエラー¥n");
        free(p ->lpBmpData);
        return ;
    }

    gr_init( p -> bih.biWidth,
             p -> bih.biHeight  );                  // グラフィックウィンドウを生成

    printf("¥n push any key");
    getch();
    SendMessage( hwnd, WM_CLOSE, 0, 0 );// グラフィックウィンドウの終了
}

//   スレッドの応用/コンソールアプリ用グラフィックウィンドウ
//   ウィンドウ側リスト
//

//   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(int x, int y)
{
    hi = (HINSTANCE)GetWindowLong( HWND_DESKTOP, GWL_HINSTANCE );
    x_size = x; y_size = y;

    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     = hi;
    wc.lpszClassName = "GRCO";                  //このクラスの名前

    if(! RegisterClassEx( &wc )) return ;       // ウィンドクラスの登録
    _beginthread( th_Proc, 0, 0 );              // メッセージループのスレッドを起動
    while( !sw );                               // 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
void th_Proc( void *)
{
    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
    hwnd = CreateWindow( "GRCO",                    // クラスの名前
                "BMPファイルの表示",
                WS_OVERLAPPED | WS_VISIBLE,         // ウィンドウの属性
                CW_USEDEFAULT, CW_USEDEFAULT,       // 位置は指定しない
                x_size + sm1 * 2,                   // 描画サイズからウィンドウの大きさを計算
                y_size + sm0 + sm2 * 2,
                HWND_DESKTOP,                       // 親はディスクトップ
                NULL, hi, NULL         );

    sw = 1;                                             // 表示されたことを知らせる
```

```
    while( GetMessage( &msg, NULL, 0, 0 ) ){
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
    return ;
}

//  ウィンドウプロシージャ, 描画オブジェクトの準備と再描画を行う
LRESULT CALLBACK grProc( HWND hwnd, UINT msg,
                         WPARAM wParam, LPARAM lParam )
{

    switch(msg){

    case WM_PAINT:{
        PAINTSTRUCT ps;
        BeginPaint( hwnd, &ps );
        SetDIBitsToDevice( ps.hdc, 0, 0,
                           p -> bih.biWidth,
                           p -> bih.biHeight,
                           0, 0, 0,
                           p -> bih.biHeight,
                           p -> lpBmpData,
                           (BITMAPINFO *)(&(p -> bih)), DIB_RGB_COLORS );
        //  BMPデータからのコピー
        EndPaint( hwnd, &ps );
        break;
    }

    default:
        return DefWindowProc( hwnd, msg, wParam, lParam );
    }
    return 0;
}
```