〔非公開〕

ＴＲ－Ｍ－００１９

映像の動き抽出に関する研究

辻　川　　知　伸　　井　上　　誠　喜
Tomonobu　TSUJIKAWA　　Seiki　INOUE

１９９７．３．１９

ＡＴＲ知能映像通信研究所

実務訓練レポート

映像の動き抽出に関する研究

平成 9 年 3 月 19 日

ATR 知能映像通信研究所　第三研究室
法政大学　実務訓練生

辻川 知伸

1 プログラムの概要

本実習では顔と手の領域を追跡するプログラムの一貫として、Sirius Video からの映像の出力を得るプログラムと Pronto Video の映像の入出力を行うプログラムを作成した。作成したプログラムの概要を表 1 に示す。

また、ソースプログラムと実行ファイルは miris37:/usr3/tujikawa/program の中に使用機器や目的別ごとにディレクトリを分けて保存してある。使い方の説明は 4 章を参照してもらいたい。下表のプログラムの他に FileManager を用いたファイル形式変換プログラムが convert というディレクトリの中にあるので参照されたい。

表 1: プログラム概要

ソースプログラム名	概要
get_center.c	全オブジェクトの平均位置を連続出力する
get_finger.c	人物の指先の位置を連続出力する
get_head.c	人物の首の位置を連続出力する
svCapture.c	ProntoVideo から指定フレームを ppm に保存する
svWriter.c	ProntoVideo の指定フレームに ppm を書き込む
get_vl.c	SiriusVideo から画像を ppm に保存する
get_y_sirius.c	SiriusVideo から 1/4 画像を pgm に保存する

2 SiriusVideo と ProntoVideo

2.1 SiriusVideo

SiriusVideo のプログラムを書く上で必要となる事柄について簡単に説明する。詳しくは参考文献 [1] を参考のこと。ただし、この文献は Sirius Video 用のものではないので、足りないところや使えない機能などがあるので注意する必要がある。

- 他のユーザーが SiriusVideo を使用していないことを確認する。（videod が 3 つ立ち上がっていなければ良いと思われる）
- vlOpenVideo() で videod と接続する。
- source と drain のノードを作成し、両者を結ぶパスを作成する。
- 出力フォーマットやタイミングなどの設定を行う。
- 映像のデータを格納するリングバッファを作成・登録する。
- 実際にデータ転送を開始する。
- データ転送を終了したら、必ず vlCloseVideo() で videod との接続を切る。（これを忘れるとシステムをリブートしなければならなくなる）

今回作成したプログラムのうち、 *get_vl* はディジタル 1 からデータを取り込み、*get_center*, *get_finger*, *get_head* はアナログからデータを取り込み処理するようになっている。

SiriusVideo のライブラリ関数である VL を使用すると、静止しているシステム上で 1 秒間に 60 回のイベントを取得することができる。1 回のイベントで 1 フィールド（720 × 243）のデータが送られてくるので、最良の状態では 1 秒間に 30 フレームを取得することが可能となる。この時のフィールドデータは RGB もしくは YUV 形式を指定できる。

今回はイベント処理関数を登録することで、プログラムはイベントが発生したときにだけ処理を行うようにしたが、この時に発生するイベントの内、実際に使っているのは VLTransferComplete だけである。その他のイベントのほとんどはどのように処理すれば良いのか分からなかったので使用していない。ここをしっかりとプログラムすれば他のユーザが SiriusVideo を使用していてもプログラムを動かすことができるかもしれない。

2.2 ProntoVideo

ProntoVideo のプログラムを書く上で必要となる事柄について簡単に説明する。詳しくは参考文献 [2] を参考のこと。

- sv_open() で ProntoVideo と接続する。
- ビデオモードなどの設定を行う。
- sv_sv2host() や sv_host2sv() でデータ転送を行う。
- sv_close() で ProntoVideo との接続を終了する。

SiriusVideo の場合と異なり、他のユーザが使用していてもプログラムを走らせることは可能だが、フレーム取り出し時に画面がちらつくのと、 ProntoVideo の設定が変更されてしまうので、なるべくなら同時使用は避けたほうが良いと思われる。今回は使用前のフレーム番号と同期を元に戻すようにしたが、それでもいくつかの設定が変更されたままのようである。

ProntoVideo から取り込んだり、 ProntoVideo へ書き込んだりするデータは、現在のところ YUV 形式のみの対応となっている。この時のデータは一度に 1 フレーム分送受信するが、最初にフィールド 0 のデータ、その次にフィールド 1 のデータが格納されている形式となっている。

3　テクニカルドキュメント

この章では、*get_center*, *get_finger*, *get_head* のプログラムでどのようにオブジェクトを判断しているかについて説明する。まず、おおまかな流れを次に示す。

- 顔と手の部分を抽出した縮小グレースケール画像を得る。
- 2 値化・縮退し、ノイズを取り除いた白黒画像を得る。
- 2 値画像からオブジェクトを取り出す。
- 対象オブジェクトの判定を行う。
- 対象オブジェクトの希望位置を出力する。

8mm カメラから取り込んだ映像を図 1 に示す。

図 1: オリジナル画像

このような画像データを ChromaScan によってリアルタイムに顔と手の部分の色を抽出する。この出力を SiriusVideo のアナログ入力とし、プログラムから 1 フィールド分取り込む。この時のデータの様子を図 2 に示す。

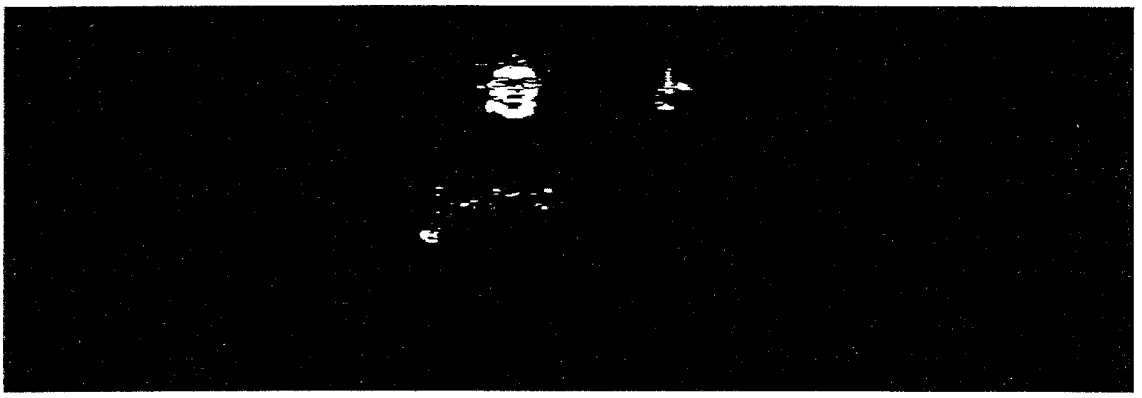

図 2: ChromaScan の出力

次に、図 2 の YUV 画像から輝度データのみを 1 つおきに抜き出し、1 フレームを 1/4 に縮小したグレースケール画像を作成する。この時の画像サイズは 360 × 243 となり、その画像を図 3 に示す。

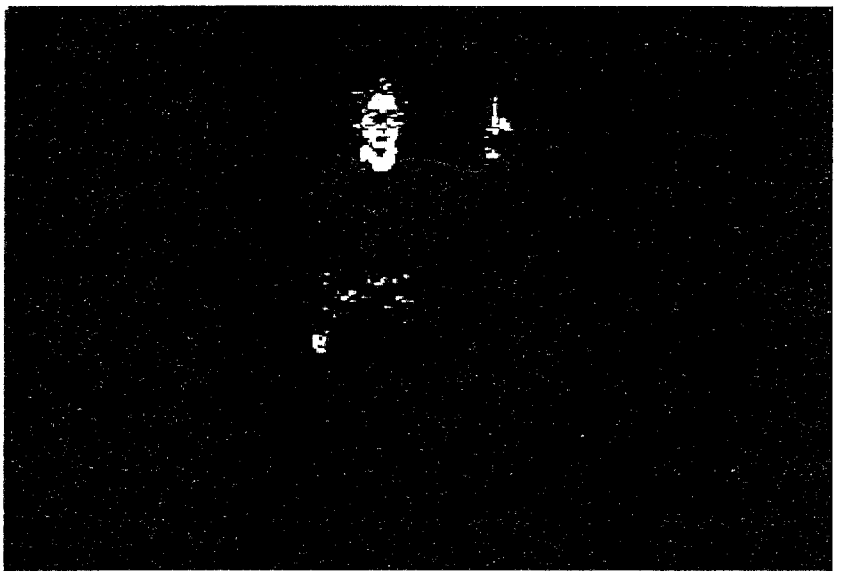

図 3: 縮小グレー画像

次に縮小グレー画像を固定閾値で 2 値化し、ノイズを減らすために縮退する。プログラムの中では、少しでも処理を速くするために縮退を少し簡略化し、孤立した白画素のみ消去するようにした。閾値を 180 として、簡略縮退を行った後のサンプル画像を図 4 に示す。

図 4: 2 値化と簡略縮退の結果

その後、ある程度まとまった領域を 1 つのオブジェクトとして認識する作業を行う。普通にラベリングするのでは適切にオブジェクトを認識できないばかりか、速度の面からも好ましくないと思われたので、今回は下に示すような流れでオブジェクトの探索と判定を行った。

- 左上から右下へ横方向に走査し、最初に見つかった白画素の座標を (x_1, y_1) とする。
- (x_1, y_1) から右方向へ n 個の連続した黒画素が続くまでサーチし、(x_2, y_1) で最初に連続黒画素が現れたとする。
- 一つ下のラインを同様にサーチする。つまり、$(x_1, y_1 + 1)$ からサーチを始め、n 個の連続黒画素が現れるまでサーチする。ただし、最初の白画素を見つけるまでは $n + (x_2 - x_1)$ 個の連続黒画素の出現まで許容する。
- 同様のサーチを繰り返し行い、(x_1, y_m) で連続黒画素が現れ、y_{m+n} までの n 回連続して x_1 で連続黒画素が現れたら、(x_1, y_m) が下方向のオブジェクトの終りだと判断する。
- 同様の走査を (x_1, y_1) から左方向にも行い、下方向により多く進んだ方をオブジェクトの下限とする。
- 左方向・右方向へそれぞれ多くサーチしたものをオブジェクトの左端と右端とする。

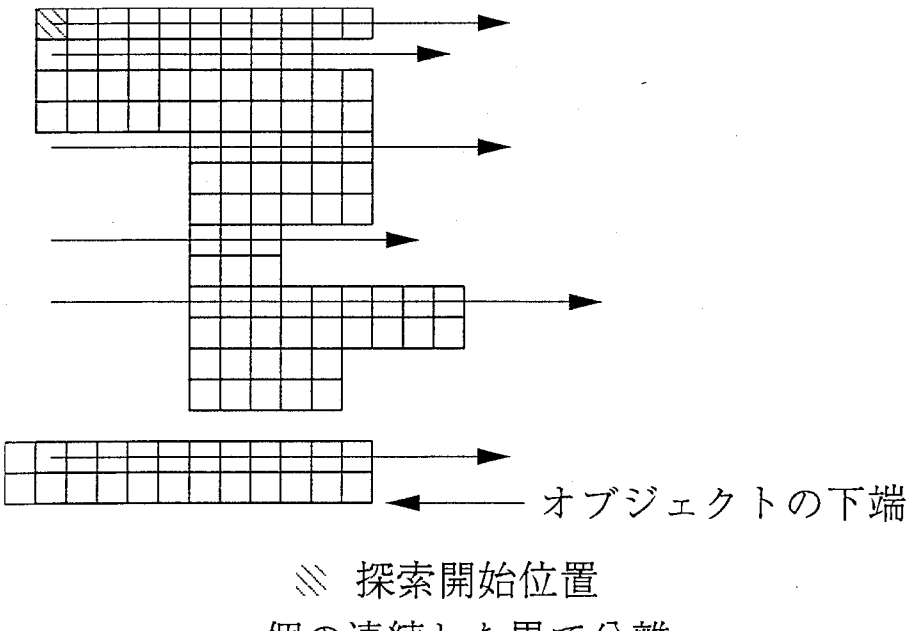

図 5: 右方向への探索の様子

サーチするにつれて最初の白画素が見つかるまでの連続黒画素数を増やしていくのは、時としてオブジェクトの輪郭しか捉えられず、1 つのオブジェクトを複数のオブジェクトとして判定してしまうためである。

このようにして得られたオブジェクトの内、ある程度以上の面積と面積あたりの白画素を含むものをオブジェクトとして登録する。オブジェクトの切目とする連続黒画素の数を 8、オブジェクトの最小面積を 20、白画素の最小割合を 0.1 として得られた結果を図 6 に示す。

図 6 から分かるように、顔部分の面積が一番大きく、次に指し手部分の面積が大きくなっている。よって、顔部分を追跡するためには 1 番大きいオブジェクトを、指し手を追跡するためには 2 番目に大きいオブジェクトを対象とすることにした。

しかし、顔部分のオブジェクトと指し手部分のオブジェクトの面積が逆転することも考えられるので、一度対象オブジェクトを捕獲したら、次回のフレームからは検索対象を対象オブジェクトの周辺のみとした。

また、1 フレーム前と比べて急激に面積が変化したオブジェクトはノイズであると思われるので、このようなオブジェクトは対象から外すようにした。

最後に、指し手の指先の出力位置の Y 座標はオブジェクトの上端とし、X 座標はオブジェクトの左上から右方向へ検索していき、一番最初に見つかった白画素の位置とした。また、首の出力位置の Y 座標はオブジェクトの下端とし、X 座標はオブジェクトの中心位置とした。

図 6: オブジェクト探索の結果

4 プログラムの使用方法

この章では表 1 に示した各プログラムの使用方法について説明する。

4.1 get_center, get_finger, get_head

人物の動きを追跡するプログラムのオプション一覧を表 2 に示す。オプションにはそれぞれ既存値があり、既存値を変える場合にのみオプションを指定する。また、オプションはそれぞれ整数値を指定するが、s オプションのみ実数値を指定することができる。

キャプチャ画像のバッファ数とは、 VL 関数が提供するリングバッファの数を言う。リングバッファからデータが溢れると、 "Sequence Lost" と表示して異常終了してしまうので、このような時にバッファ数を大きくすると良い。デフォルト値は 20 である。

連続黒画素、最小面積、移動画素数、面積比率、白画素割合といったオプションはプログラム内部のアルゴリズムと大きく関わっているので、前章を参照してもらいたい。デフォルト値はそれぞれ 8, 20, 10, 2.0, 0.1 である。

オブジェクトの探索数はいくつのオブジェクトが見つかった時点で探索を終了するかを指定するオプションである。ノイズが全く無い良好な状態であれば 3 を指定すれば十分であるが、ノイズが乗ることを考慮してデフォルト値には大きめの 10 を指定してある。

閾値は 2 値化時に使用される。照明や服の色、 ChromaScan の設定によって大き

表 2: オプション一覧

オプション	意味
$-b < n >$	キャプチャ画像のバッファ数を指定する
$-c < n >$	オブジェクトの切目となる連続黒画素の数を指定する
$-m < n >$	オブジェクトの最小面積を指定する
$-o < n >$	オブジェクト探索数を指定する
$-r < n >$	1 フレームでオブジェクトが移動する画素数を指定する
$-s < f >$	1 フレーム間の対象オブジェクトの面積比率を指定する
$-t < n >$	閾値を指定する（ 1 ～ 255 ）
$-w < f >$	オブジェクト領域中の白画素の最小割合を指定する

く変わってくるので、デフォルト値は 80 とやや低めに設定してある。経験からは、大体 150 から 200 程度で調節するのが良いと思われる。

4.2 svCapture, svWriter

この 2 つのプログラムには 2 つの引数を指定しなければならない。第一引数にはファイル名を、第二引数にはフレーム番号を指定する。次にそれぞれの使用例を示す。

>> svCapture test.ppm 5000
ProntoVideo の 5000 フレーム目を test.ppm という名前で保存する

>> svWriter test.ppm 5000
ProntoVideo の 5000 フレーム目に test.ppm という画像を書き込む

4.3 get_vl, get_y_sirius

これらのプログラムは 実行時の映像を SiriusVideo から取り込むプログラムであるが、 *get_vl* は ディジタル 1 入力の、 *get_y_sirius* はアナログ入力の画像をキャプチャするようにプログラムされている。引数にはファイル名を指定する。使用例を次に示す。

>> get_vl test.ppm
SiriusVideo の Digital 1 入力の画像を test.ppm という名前で保存する

>> get_y_sirius test.ppm
SiriusVideo の Analog 入力の画像を test.ppm という名前で保存する

5 プログラムの関数の説明

この章ではプログラム内の関数について簡単に説明する。ソースプログラムにもコメントがあるので、詳細な説明が必要な場合はそちらを参照して頂きたい。

5.1 get_center, get_finger, get_head, get_vl, get_y_sirius について

表 3: オプション一覧

関数名	内容
main	設定が終了後、無限ループする。
Initialize	変数の初期化と引数解析、シグナルのフックを行う。
CommunicateDaemon	SiriusVideo の設定を行い、データ転送を始める。
CheckSiriusVideo	他ユーザが SiriusVideo を使用しているか調べる。
ProcessEvent	VL イベントを処理する。
GetRegion	対象オブジェクトの位置を求め、出力する。
MakeBWPicture	2 値化、縮退して縮小画像を作成する。
MakeObject	ある領域内からオブジェクトを抽出する。
SearchRegionEdge	オブジェクト領域の端を探索する。
GetWhiteNum	オブジェクト領域内の白画素の数を数える。
DrowRect	オブジェクト領域内のラベル付けを行う。
EraseRect	小面積のオブジェクトを消去する。
GetLength	連続黒画素が現れるまでの画素数を調べる。
GetBigSizeObject	1,2 番目に大きいオブジェクトを探す。
GetSameObject	前回のオブジェクトと同じものを探す。
docleanup	終了手続きを行う。
SignalHandler	シグナルを受け取り、終了する。
Save	2 値画像を保存する関数。デバッグ用。
DumpImage	画像を保存する関数。 get_vl と get_y_sirius で使用。

5.2 svCapture について

表 4: オプション一覧

関数名	内容
main	ProntoVideo への接続と設定、切断を行う。
$sv_transfer$	ProntoVideo との転送を行う。
yuv2rgb	YUV 形式から IRIS RGB 形式へ変換する。
rgb2yuv	IRIS RGB 形式から YUV 形式へ変換する。

5.3 svWriter について

表 5: オプション一覧

関数名	内容
InitApplication	引数解析を行う。
ReadFileWithConvert	FM を用いて ppm を読み込み、YUV へ変換する。
InitProntoVideo	ProntoVideo への接続と設定を行う。
SVTransfer	ProntoVideo に書き込みを行う。
EndApplication	ProntoVideo の設定を戻し、接続を切る。
FMErrorExit	FM のエラー時における終了手続きを行う。
SVErrorExit	ProntoVideo エラー時の終了手続きを行う。

6 get_finger と MIDI, DVS-6000 との共同使用

get_finger を用いることによって、指の動きを追跡することができるので、指の位置によって MIDI 音源を操作したり、DVS-6000 を操作することができる。この章ではそれぞれについて説明する。

6.1 get_finger と MIDI

get_finger プログラムは SiriusVideo を用いるので、miris37 でしか動かすことができない。また、MIDI 音源を操作する send_midi プログラムは miris61 でしか動かす意味がない。そこで、*rsh* を用いて miris61 から get_finger プログラムの結果を得ることにする。今回使うプログラムは /home/tujikawa/program の中に入っている。使用方法は次の通り。

```
>> cd /home/tujikawa/program
>> rsh miris37 ~tujikawa/program/search/get_finger | ./convert | \
    ./send_midi
```

convert というプログラムは get_finger の出力を send_midi が受け取れる範囲に調整するためのものである。また、 *send_midi* は元々は EOF の判定がなかったものを付け加えたので、必ず指定ディレクトリの中にある send_midi を使うようにする必要がある。

エラーが無ければ、カメラの前に立って手を左右に動かすと音程が、手を上下に動かすと音量が変化する。

6.2 get_finger と DVS-6000

DVS-6000 の操作プログラムは miris37:/home/mic3_pub/bin/devctl である。今回はたくさんある機能のうち、円を動かす機能を使うことにした。スイッチャーコントロールパネルと ONYXtty4 を接続し、 devctl にパスが通っていることを確認した上で次のように入力する。

```
>> cd /usr3/tujikawa/program/search
>> ./get_finger | ./convert | dvsctl
```

convert は先と同様に、 devctl の出力に合う形に変換するだけのものである。

エラーがなければ、カメラの前に立って手を動かすと、動きにしたがって円が移動する。

6.3 get_finger と DVS-6000, MIDI の同時使用

get_finger と DVS6000 による円の制御と get_finger と MIDI による音程音量の制御を同時に行うことができる。ソケットを使って通信を行いつつ、それぞれの機器用に出力を変換するプログラムがあるので、それを使用する。まず miris61 のソケットサーバーを先に立ち上げてから、 miris37 のソケットクライアントを立ち上げる必要がある。具体的な使用方法は次の通り。

```
miris61%> cd /home/tujikawa/program
miris61%> ./socketconvert | ./send_midi

miris37%> cd /usr3/tujikawa/program/search
miris37%> ./get_finger | ./socketconvert miris61 | devctl
```

miris37 側の socketconvert でサーバーホスト名を指定しないとエラーとなり終了するが、 get_finger が標準出力に何かを出力するまでパイプが途切れたという事に気付かないようなので、注意する必要がある。

A FileManager

ここではファイル形式を様々に変換することのできる FileManager について簡単に説明する。FileManager が対応しているフォーマットは Raw YUV422, BMP, DVS ISP YUV422, DVS ISP Mono, DVS ISP RGB, JPEG, PPM, IRIS RGB, Softimage, SVJ, Targa, TIFF の 12 形式である。

次にプログラムの流れを簡単に説明する。

- fm_initialize() で FileManager の初期化を行う。
- fm_fileformat_allocate() で変換元と変換先のフォーマットを指定する。
- fm_open() で変換元ファイルを開き、fm_read() でデータを読み込む。
- fm_create() で変換先ファイルを作成する。
- fm_convert() で変換を行い、fm_write() でファイルに書き込む。
- fm_close() や fm_fileformat_free(), fm_deinitialize() で終了手続きをする。

なお、*fm_config* という一見するとどこからも呼ばれていない関数がソースプログラムの中に存在するが、これは fm_initialize() 関数が初期化のために使用する関数で除外することはできない。

また、あるファイル形式のデータを YUV 形式に fm_convert で変換すると、メモリ内では、1 画像分の Y データが最初に全て格納され、次いで Cb データが 1 画面分、最後に Cr データが 1 画面分という風に格納されている。svWriter プログラムのようにメモリから直接データを読み出す場合には注意が必要である。

B 参考文献

参考文献

[1] "IRIS Digital Media Programming Guide (日本語版)", 日本シリコングラフィックス株式会社, pp.217-316.

[2] "MMS C library Manual Revision 3.2".

[3] "色抽出装置 CHROMASCAN ユーザーズマニュアル", 株式会社 応用計測研究所.

[4] "C 言語で学ぶ 実践画像処理", オーム社

```
/*
 *                          searching.c
 *              ある領域を追跡してそのデータを出力するプログラム
 *                  Usage : searching <threshold>
 *
 *          アナログ入力だけを取り込み、リアルタイムに重心位置を
 *                      出力するプログラム
 *
 *
 *                          <warning>
 *                    Don't Add Optimizing Flag!!
 *
 *                  To exit application is 'Ctrl-C'.
 *                This program hook the 'Ctrl-C' Vector.
 */

#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <gl/gl.h>
#include <gl/image.h>
#include <dmedia/vl.h>
#include <dmedia/vl_sirius.h>


#define USAGE       "%s: <threshold>\n"
#define BUFNUM      10                  /* 作成するバッファの数 */
#define DEBUG


/********************/
/* グローバルデータ */
/********************/
char *_progName;                        /* プログラムの名前 */
char *dataBuffer;
int now_is_f1;                         /* 取り込んだ画像の数 */
int fields;
long fieldSize;                         /* 1フレームあたりのバッファサイズ */
int xsize, ysize;                       /* フィールドあたりの画面の大きさ */
int half_x;
int dec_y;
long imageSize;
int threshold;

VLServer svr;
VLBuffer buffer;                        /* ディジタルのキャプチャバッファ */
VLPath path;                            /* ディジタルのビデオパス */
VLNode src, drn;                        /* ディジタルのノード */



/********************/
/* プロトタイプ宣言 */
/********************/
void Initialize (int, char**);
void CommunicateDaemon (void);
void ProcessEvent (VLServer, VLEvent*, void*);
void DumpImage (unsigned char*);
void CalcCenter (unsigned char*);
```



```
void docleanup (int);
void SignalHandler (int);



void main (int argc, char *argv[])
{
    Initialize (argc, argv);
    CommunicateDaemon ();

    /* ユーザが終了を指示するまでループする */
    vlMainLoop ();
}



/*
 * アプリケーションの初期化を行う
 */
void Initialize (int argc, char *argv[])
{
    int i;

    _progName = argv[0];
    if (argc!=2) {
        fprintf (stderr, USAGE, _progName);
        exit (1);
    }
    threshold = atoi (argv[1]);
    if (threshold<=0 || threshold>256) {
        fprintf (stderr, "threshold value is 1 to 255.\n");
        exit (1);
    }
#ifdef DEBUG
    fprintf (stderr, "threshold is %d.\n", threshold);
#endif

    fields = 0;

    /* シグナルのフック */
    for(i=1;i<=32;i++)  signal(i,SignalHandler);
}



/*
 * Sirius Video への接続と初期化
 */
void CommunicateDaemon (void)
{
    VLDevList             devlist;
    VLControlValue        val;
    VLTransferDescriptor  xferDesc;
    int                   i, timing, devicenum;

    /* デーモンへ接続 */
    if ((svr=vlOpenVideo(""))==NULL) {
        fprintf (stderr, "%s: couldn't open video\n", _progName);
        exit (1);
    }
    else  fprintf (stderr, "Connected Server Daemon.\n");

    /* ノードの設定 */
    src = vlGetNode (svr, VL_SRC, VL_VIDEO, SIR_SRC_ANALOG_VIDEO);
    drn = vlGetNode (svr, VL_DRN, VL_MEM, VL_ANY);
```

```
    /* ハードの設定とパス仕様の定義 */
    if ((path=vlCreatePath(svr,VL_ANY,src,drn)) < 0) {
        vlPerror ("create path failed.\n");
        vlCloseVideo (svr);
        exit (1);
    }

    if (vlSetupPaths(svr, (VLPathList)&path, 1, VL_SHARE, VL_SHARE)!=0) {
        vlPerror ("setup path failed.\n");
        vlCloseVideo (svr);
        exit (1);
    }

    /* 受信したいイベントを設定 */
    vlSelectEvents (svr, path,
                    VLTransferCompleteMask | VLStreamPreemptedMask |
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLTransferFailedMask | VLSequenceLostMask);

    /* タイミングの設定(src) */
    val.intVal = VL_TIMING_525_CCIR601;
    if ((vlSetControl(svr,path,src,VL_TIMING,&val)) != 0) {
        vlPerror ("set timing of digital failed.\n");
        vlCloseVideo (svr);
        exit (1);
    }

    /* ビデオ出力フォーマットの指定(drn) */
    val.intVal = VL_FORMAT_SMPTE_YUV;
    vlSetControl (svr, path, drn, VL_FORMAT, &val);

    /* タイミングの設定(drn) */
    if (vlGetControl(svr,path,src,VL_TIMING,&val) == 0) {
        if ((vlSetControl(svr,path,drn,VL_TIMING,&val)!=0)
            && (vlErrno!=VLBadControl)) {
            vlPerror ("SetControl Failed.\n");
            vlCloseVideo (svr);
            exit (1);
        }
    }

    /* 取り込むフレームのタイプを指定(drn) */
    val.intVal = VL_CAPTURE_INTERLEAVED;
    vlSetControl (svr, path, drn, VL_CAP_TYPE, &val);

    /* ピクセルをYUVで保存するための設定(drn) */
    val.intVal = VL_PACKING_YVYU_422_8;
    vlSetControl (svr, path, drn, VL_PACKING, &val);

    /* 最初に取り込まれるフィールドを知る(src) */
    if (vlGetControl(svr,path,src,VL_SIR_FIELD_DOMINANCE,&val) == 0) {
        timing = ((val.intVal == VL_TIMING_525_SQ_PIX) ||
                  (val.intVal == VL_TIMING_525_CCIR601));
        switch (val.intVal) {
          case SIR_F1_IS_DOMINANT:
            if(timing)  now_is_f1 = 0;
            else        now_is_f1 = 0xFFFF;
            break;
          case SIR_F2_IS_DOMINANT:
            if (timing)  now_is_f1 = 0xFFFF;
            else         now_is_f1 = 0;
            break;
        }
    }

    /* ビデオサイズとそのバイト数を得る */
```



```
    vlGetControl (svr, path, drn, VL_SIZE, &val);
    xsize= val.xyVal.x;
    ysize = val.xyVal.y;
    fieldSize = vlGetTransferSize (svr, path);

    half_x = xsize/2;
    dec_y = ysize-1;
    imageSize = half_x*ysize;

    /* 転送のためのコールバックルーチンを設定 */
    vlAddCallback (svr, path,
                   VLTransferCompleteMask | VLStreamPreemptedMask |
                   VLStreamStartedMask | VLStreamStoppedMask |
                   VLSequenceLostMask | VLTransferFailedMask,
                   ProcessEvent, NULL);

    /* BUFNUM分のフレームバッファの作成と登録 */
    if ((buffer=vlCreateBuffer(svr,path,drn,BUFNUM))==NULL) {
        vlPerror ("create buffer is failed.\n");
        vlCloseVideo (svr);
        exit (1);
    }

    if (vlRegisterBuffer(svr, path, drn, buffer)!=0) {
        vlPerror ("to regist buffer is failed.\n");
        vlDestroyBuffer (svr, buffer);
        vlCloseVideo (svr);
        exit (1);
    }

    /* 縮退用のバッファを確保 */
    if ((dataBuffer=malloc(imageSize*sizeof(char)))==NULL) {
        vlPerror ("malloc error.\n");
        docleanup (1);
    }

    xferDesc.mode = VL_TRANSFER_MODE_CONTINUOUS;
    xferDesc.count = 10000;          /* CONTINUOUS なら無視される */
    xferDesc.delay = 0;              /* VLTriggerImmediate なら無視される */
    xferDesc.trigger = VLTriggerImmediate;

    /* アナログのデータ転送の開始 */
    if (vlBeginTransfer(svr,path,1,&xferDesc)) {
        vlPerror ("transfer is failed.\n");
        vlDestroyBuffer (svr, buffer);
        vlCloseVideo (svr);
        exit (1);
    }
    fprintf (stderr, "data transfer is started.\n");
}



/*
 *  ビデオライブラリイベントを処理する
 * <input>
 *  svr : VLServerへのポインタ
 *  ev : 何のイベントで呼ばれたか？
 *  data : ユーザーが使用すると宣言したデータへのポインタ
 */
void ProcessEvent (VLServer svr, VLEvent *ev, void *data)
{
    VLInfoPtr info;
    char      *dataPtr;

    switch (ev->reason) {
```

```
      case VLStreamStarted:
        break;
      case VLStreamStopped:
      case VLTransferComplete:
        /* バッファ内のフィールドを読み出す */
        while (info=vlGetNextValid(svr,buffer)) {
            dataPtr = vlGetActiveRegion (svr, buffer, info);
            if (now_is_fl!=0)           /* if now_is_fl==TRUE, then ... */
                CalcCenter (dataPtr);
            now_is_fl ^= 0xFFFF;        /* FALSE->TRUE, TRUE->FALSE */
            vlPutFree (svr, buffer);
            fields++;
        }
        break;
      case VLStreamPreempted:
        fprintf (stderr, "%s: Path for this stream preempted.\n", _progName);
        docleanup (1);
        break;
      case VLSequenceLost:
        fprintf (stderr, "%s: Sequence Lost.\n", _progName);
        docleanup (1);
        break;
      case VLTransferFailed:
        fprintf (stderr, "%s: Transfer failed.\n", _progName);
        docleanup (1);
        break;
      default:
        break;
    }
}



/*
 *  重心を求める
 * <input>
 *  data : YUVフォーマットのデータへのポインタ
 */
void CalcCenter (unsigned char *data)
{
    unsigned char *tmpBuffer = dataBuffer;
/*    unsigned char *originalData = data; */
    long i, j, x =0L, y=0L;
    int center_x, center_y, count=0;

    /* Yデータの間引き抜き出しと２値化 */
    data++;
    for (i=0L;i<imageSize;i++) {
        if (*data < threshold)  *(tmpBuffer++) = 1;
        else                    *(tmpBuffer++) = 0;
        data+=4;
    }
/*    data = originalData; */

    /* 縮退して重心を求める(端のピクセルは無視する) */
    tmpBuffer = dataBuffer + half_x-1;
    for (i=1;i<dec_y;i++) {
        tmpBuffer+=2;
        for (j=2;j<half_x;j++) {
            if (*tmpBuffer==0) {
                if (*(tmpBuffer-1)==0        && *(tmpBuffer+1)==0        &&
                    *(tmpBuffer-half_x)==0   && *(tmpBuffer+half_x)==0   &&
                    *(tmpBuffer-half_x-1)==0 && *(tmpBuffer-half_x+1)==0 &&
                    *(tmpBuffer+half_x-1)==0 && *(tmpBuffer+half_x+1)==0) {
                    count++;
                    x += j-1;
```



```
                    y += i;
                }
            }
            tmpBuffer++;
        }
    }

    /* 重心位置のデータの出力 */
    if (count==0) {
        fprintf (stderr, "No Count.\n");
    }
    else {
        center_x = x/count;
/*      center_x <<= 1; */
        center_y = y/count;
/*      data += center_y*xsize + center_x;
        printf ("%d : %d %d : %x %x %x %x\n", fields, center_x, center_y,
                *data, *(data+1), *(data+2), *(data+3));
*/
        printf ("%d %d\n", center_x, center_y);
        fflush (stdout);
    }
}



/*
 *  終了前のクリーンアップを行う。
 *  <input>
 *    ret : 終了コード
 */
void docleanup (int ret)
{
    vlEndTransfer (svr, path);
    vlDeregisterBuffer (svr, path, drn, buffer);
    vlDestroyBuffer (svr, buffer);
    vlDestroyPath (svr, path);
    vlCloseVideo (svr);
    free (dataBuffer);
    exit (ret);
}



/*
 * シグナルを受け取るフックルーチン
 * <input>
 *  sig : 受信したシグナル番号
 */
void SignalHandler (int sig)
{
    fprintf (stderr, "%s: catch the signal <%d>.\n", _progName, sig);
    fprintf (stderr, "%d fields transferred.\n", fields);
    docleanup(sig);
}
```

```
/*
 *                              searching.c
 *        顔/手の領域を判断し、手の領域の左上座標を出力するプログラム
 *                Usage : searching <threshold> <minimum S>
 *
 *
 *                                <warning>
 *                           Don't Add Optimizing Flag!!
 *
 *                        To exit application is 'Ctrl-C'.
 *                     This program hook the 'Ctrl-C' Vector.
 */

#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <sys/types.h>
#include <dmedia/vl.h>
#include <dmedia/vl_sirius.h>



#define USAGE "\nusage: %s %s\n"\
    "where option arguments :\n"\
    "         -b<n>  use <n> ring buffers                    (default is 20).\n"\
    "         -c<n>  <n> continuous black separate object    (default is 8).\n"\
    "         -m<n>  minimum object size is <n>              (default is 20).\n"\
    "         -o<n>  maximal search object is <n>            (default is 10).\n"\
    "         -r<n>  object move <n> pixels in 1 frame       (default is 10).\n"\
    "         -s<f>  target object's size ratio in 1 frame (default is 2.0).\n"\
    "         -t<n>  threshold value is <n>                  (default is 80).\n"\
    "         -w<f>  minimum object white ratio is <f>       (default is 0.1).\n"\
    "\n"\
    "                <n> is integer, <f> is float.\n\n"
#define JUSAGE "\nusage: %s %s\n"\
    "オプション一覧 :\n"\
    "         -b<n>  <n>個のリングバッファを使用する         (default is 20).\n"\
    "         -c<n>  連続<n>の黒画素がオブジェクトの切目     (default is 8).\n"\
    "         -m<n>  オブジェクトの最小面積を<n>とする       (default is 20).\n"\
    "         -o<n>  最大オブジェクト探索数を<n>とする       (default is 10).\n"\
    "         -r<n>  1フレームで対象物は最大<n>画素移動      (default is 10).\n"\
    "         -s<f>  1フレームで対象物は最大<f>面積が変化 (default is 2.0).\n"\
    "         -t<n>  閾値を<n>とする                         (default is 80).\n"\
    "         -w<f>  領域中の白画素の割合の最小は<f>とする (default is 0.1).\n"\
    "\n"\
    "                <n> は整数, <f> は実数.\n\n"

#define CMDARGS       "b:c:m:o:r:s:t:w:"
#define WHITE         0
#define BLACK         1
#define XSIZE         720       /* 1フィールドあたりの画面の大きさ */
#define YSIZE         243
#define HALF_X        XSIZE/2         /* プログラム内で使用 */
#define DEC_Y         YSIZE-1
#define IMAGE_SIZE    HALF_X*YSIZE    /* 縮小画像の大きさ */

#define DEBUG
#undef  DEBUG
#define DEBUG_WITH_PICTURE          /* 2値画像付きデバッグ */
#undef  DEBUG_WITH_PICTURE


/**************/
/* 構造体定義 */
```



```
/**************/
typedef struct {
    int left, right, top, bottom;
} Rect;



/********************/
/* グローバルデータ */
/********************/
char   *gProgName;                  /* プログラムの名前 */
char   *gDataBuffer;                /* 2値画像用のバッファ */
short  gField1Flag;                 /* 今どちらのフィールドであるか */
int    gFields;                     /* 取り込んだ画像の数 */
Rect   *gObject;                    /* オブジェクトの記憶領域（顔と手） */
long   *gObjectSize;                /* オブジェクトの面積を記憶 */
Rect   gOldRect;                    /* 前の状態の四角形を保持 */
long   gOldObjectSize;              /* 前のオブジェクトの大きさを保持 */
int    gBufNum = 20;                /* 作成するVLバッファの数 */
int    gRange = 10;                 /* 1フレームの間に物が移動する最大距離 */
double gSizeRatio = 2.5;            /* 1フレーム前の探索物の大きさの比率 */
double gWhiteRatio = 0.1;           /* 領域中の白画素の割合の最小値 */
int    gContCol = 8;                /* 物の切目とする黒画素の数 */
int    gObjectNum = 10;             /* 探索するオブジェクトの最大数 */
int    gThreshold = 80;             /* しきい値 */
int    gMinSize = 20;               /* オブジェクトの最小面積 */


VLServer  gServer;                  /* vl関係のグローバル変数 */
VLBuffer  gBuffer;
VLPath    gPath;
VLNode    gSrc, gDrn;



/********************/
/* プロトタイプ宣言 */
/********************/
void Initialize (int, char**);
void CommunicateDaemon (void);
void CheckSiriusVideo (void);
void ProcessEvent (VLServer, VLEvent*, void*);
void GetRegion (unsigned char*);
void MakeBWPicture (char*, char*);
int MakeObject (void);
void SearchRegionEdge (char*, int, int, int, int*, int*);
long GetWhiteNum (int);
void DrowRect (Rect*, int);
void EraseRect (Rect*);
int GetLength (char *, int, int, int);
void GetBigSizeObject (int, int*, int*);
int GetSameObject (int);
void docleanup (int);
void SignalHandler (int);
void Save (unsigned char*, char*);



void main (int argc, char *argv[])
{
    Initialize (argc, argv);
    CommunicateDaemon ();

    /* ユーザが終了を指示するまでループする */
    vlMainLoop ();
}
```

```
/*
 * アプリケーションの初期化を行う
 */
void Initialize (int argc, char *argv[])
{
    char *lang;
    int i, c;

    gFields = 0;

    gProgName = argv[0];

    while ((c=getopt(argc,argv,CMDARGS)) != EOF) {
        switch (c) {
          case 'b' :
              gBufNum = atoi(optarg);
              if (gBufNum<0) {
                  fprintf (stderr, "ring buffer num is big to 0.\n");
                  exit (1);
              }
              break;
          case 'c' :
              gContCol = atoi(optarg);
              if (gContCol<0) {
                  fprintf (stderr, "black continuous is big to 0.\n");
                  exit (1);
              }
              break;
          case 'm' :
              gMinSize = atoi(optarg);
              if (gMinSize<0) {
                  fprintf (stderr, "object's minimum size is big to 0.\n");
                  exit (1);
              }
              break;
          case 'o' :
              gObjectNum = atoi(optarg);
              if (gObjectNum<0) {
                  fprintf (stderr,"number of searching object is big to 0.\n");
                  exit (1);
              }
              break;
          case 'r' :
              gRange = atoi(optarg);
              if (gRange<0) {
                  fprintf (stderr, "finger movement range is big to 0.\n");
                  exit (1);
              }
              break;
          case 's' :
              gSizeRatio = atof(optarg);
              if (gSizeRatio<1.0) {
                  fprintf (stderr, "size ratio is big to 1.\n");
                  exit (1);
              }
              break;
          case 't' :
              gThreshold = atoi(optarg);
              if (gThreshold<=0 || gThreshold>256) {
                  fprintf (stderr, "threshold value is 1 to 255.\n");
                  exit (1);
              }
              break;
```



```
          case 'w' :
              gWhiteRatio = atof(optarg);
              if (gWhiteRatio>1.0) {
                  fprintf (stderr, "white ratio is small to 1.\n");
                  exit (1);
              }
              break;
          default  :
              lang = getenv ("LANG");
              if (lang!=NULL && *lang=='j' && *(lang+1)=='a')
                  fprintf (stderr, JUSAGE, gProgName, CMDARGS);
              else
                  fprintf (stderr, USAGE, gProgName, CMDARGS);
              exit(1);
              break;
        }
    }
#ifdef DEBUG
    fprintf (stderr, "gBufNum is %d\n", gBufNum);
    fprintf (stderr, "gContCol is %d\n", gContCol);
    fprintf (stderr, "gMinSize is %d\n", gMinSize);
    fprintf (stderr, "gObjectNum is %d\n", gObjectNum);
    fprintf (stderr, "gRange is %d\n", gRange);
    fprintf (stderr, "gSizeRatio is %lf\n", gSizeRatio);
    fprintf (stderr, "gThreshold is %d\n", gThreshold);
#endif

    /* gObjectとgObjectSizeの領域作成 */
    if ((gObject=malloc(gObjectNum*sizeof(Rect)))==NULL) {
        fprintf (stderr, "malloc error.\n");
        exit (1);
    }
    if ((gObjectSize=malloc(gObjectNum*sizeof(long)))==NULL) {
        fprintf (stderr, "malloc error.\n");
        exit (1);
    }

    /* オブジェクト検索領域の設定 */
    gOldRect.left   = 0;
    gOldRect.top    = 0;
    gOldRect.right  = HALF_X-1;
    gOldRect.bottom = DEC_Y;

    /* シグナルのフック */
    for(i=1;i<=32;i++)  signal(i,SignalHandler);
}



/*
 * Sirius Video への接続と初期化
 */
void CommunicateDaemon (void)
{
    VLDevList             devlist;
    VLControlValue        val;
    VLTransferDescriptor  xferDesc;
    int                   i, timing, devicenum;
    long                  gFieldsize;

    /* videodが使用中かどうかチェックする */
    CheckSiriusVideo ();

    /* デーモンへ接続 */
    if ((gServer=vlOpenVideo(""))==NULL) {
        fprintf (stderr, "%s: couldn't open video\n", gProgName);
```

```
        exit (1);
    }
    else ; /*fprintf (stderr, "Connected Server Daemon.\n");*/

    /* ノードの設定 */
    gSrc = vlGetNode (gServer, VL_SRC, VL_VIDEO, SIR_SRC_ANALOG_VIDEO);
    gDrn = vlGetNode (gServer, VL_DRN, VL_MEM, VL_ANY);

    /* ハードの設定とパス仕様の定義 */
    if ((gPath=vlCreatePath(gServer,VL_ANY,gSrc,gDrn)) < 0) {
        vlPerror ("create path failed.\n");
        vlCloseVideo (gServer);
        exit (1);
    }

    if (vlSetupPaths(gServer,(VLPathList)&gPath,1,VL_SHARE,VL_SHARE)!=0) {
        vlPerror ("setup path failed.\n");
        vlCloseVideo (gServer);
        exit (1);
    }

    /* 受信したいイベントを設定 */
    vlSelectEvents (gServer, gPath,
                    VLTransferCompleteMask | VLStreamPreemptedMask |
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLTransferFailedMask | VLSequenceLostMask);

    /* タイミングの設定(gSrc) */
    val.intVal = VL_TIMING_525_CCIR601;
    if ((vlSetControl(gServer,gPath,gSrc,VL_TIMING,&val)) != 0) {
        vlPerror ("set timing of digital failed.\n");
        vlCloseVideo (gServer);
        exit (1);
    }

    /* ビデオ出力フォーマットの指定(gDrn) */
    val.intVal = VL_FORMAT_SMPTE_YUV;
    vlSetControl (gServer, gPath, gDrn, VL_FORMAT, &val);

    /* タイミングの設定(gDrn) */
    if (vlGetControl(gServer,gPath,gSrc,VL_TIMING,&val) == 0) {
        if ((vlSetControl(gServer,gPath,gDrn,VL_TIMING,&val)!=0)
            && (vlErrno!=VLBadControl)) {
            vlPerror ("SetControl Failed.\n");
            vlCloseVideo (gServer);
            exit (1);
        }
    }

    /* 取り込むフレームのタイプを指定(gDrn) */
    val.intVal = VL_CAPTURE_INTERLEAVED;
    vlSetControl (gServer, gPath, gDrn, VL_CAP_TYPE, &val);

    /* ピクセルをYUVで保存するための設定(gDrn) */
    val.intVal = VL_PACKING_YVYU_422_8;
    vlSetControl (gServer, gPath, gDrn, VL_PACKING, &val);

    /* 最初に取り込まれるフィールドを知る(gSrc) */
    if (vlGetControl(gServer,gPath,gSrc,VL_SIR_FIELD_DOMINANCE,&val) == 0) {
        timing = ((val.intVal == VL_TIMING_525_SQ_PIX) ||
                  (val.intVal == VL_TIMING_525_CCIR601));
        switch (val.intVal) {
          case SIR_F1_IS_DOMINANT:
            if(timing)  gField1Flag = 0;
            else        gField1Flag = 0xFFFF;
            break;
```



```
          case SIR_F2_IS_DOMINANT:
            if (timing)  gField1Flag = 0xFFFF;
            else         gField1Flag = 0;
            break;
        }
    }

    /* ビデオサイズとそのバイト数を得る */
    vlGetControl (gServer, gPath, gDrn, VL_SIZE, &val);
    if (val.xyVal.x!=XSIZE || val.xyVal.y!=YSIZE) {
        vlPerror ("picture size is strange.\n");
        vlCloseVideo (gServer);
        exit (1);
    }
    gFieldsize = vlGetTransferSize (gServer, gPath);          /* no use */

    /* 転送のためのコールバックルーチンを設定 */
    vlAddCallback (gServer, gPath,
                   VLTransferCompleteMask | VLStreamPreemptedMask |
                   VLStreamStartedMask | VLStreamStoppedMask |
                   VLSequenceLostMask | VLTransferFailedMask,
                   ProcessEvent, NULL);

    /* gBufNum分のフレームバッファの作成と登録 */
    if ((gBuffer=vlCreateBuffer(gServer,gPath,gDrn,gBufNum))==NULL) {
        vlPerror ("create buffer is failed.\n");
        vlCloseVideo (gServer);
        exit (1);
    }

    if (vlRegisterBuffer(gServer, gPath, gDrn, gBuffer)!=0) {
        vlPerror ("to regist buffer is failed.\n");
        vlDestroyBuffer (gServer, gBuffer);
        vlCloseVideo (gServer);
        exit (1);
    }

    /* 縮退用のバッファを確保 */
    if ((gDataBuffer=malloc(IMAGE_SIZE*sizeof(char)))==NULL) {
        vlPerror ("malloc error.\n");
        docleanup (1);
    }

    xferDesc.mode = VL_TRANSFER_MODE_CONTINUOUS;
    xferDesc.count = 10000;          /* CONTINUOUS なら無視される */
    xferDesc.delay = 0;              /* VLTriggerImmediate なら無視される */
    xferDesc.trigger = VLTriggerImmediate;

    /* アナログのデータ転送の開始 */
    if (vlBeginTransfer(gServer,gPath,1,&xferDesc)) {
        vlPerror ("transfer is failed.\n");
        vlDestroyBuffer (gServer, gBuffer);
        vlCloseVideo (gServer);
        exit (1);
    }
   /* fprintf (stderr, "data transfer is started.\n");*/
}



/*
 *  Sirius Videoが使用中かどうか調べる
 */
void CheckSiriusVideo (void)
{
    char cmd[128], filename[128], buffer[128];
```

```
    int num;
    FILE *fp;

    /* videodプロセスを取り出す */
    sprintf (filename, "/tmp/tmp%05d", getpid());
    sprintf (cmd, "ps -e|grep videod > %s", filename);
    system (cmd);

    /* videodの数を調査 */
    num = 0;
    if ((fp=fopen(filename,"rt"))==NULL) {
        fprintf (stderr, "tmp-file open error.\n");
        exit(1);
    }
    while((fgets(buffer,128,fp))!=NULL)  num++;
    fclose (fp);
    unlink (filename);

    if (num==0) {
        fprintf (stderr, "videod is not started.\n");
        exit (1);
    }
    if (num>2) {
        fprintf (stderr, "videod is already used by another program.\n");
        exit (1);
    }
}



/*
 *  ビデオライブラリイベントを処理する
 * <input>
 *  gServer : VLServerへのポインタ
 *  ev : 何のイベントで呼ばれたか？
 *  data : ユーザーが使用すると宣言したデータへのポインタ
 */
void ProcessEvent (VLServer gServer, VLEvent *ev, void *data)
{
    VLInfoPtr info;
    char      *dataPtr;

    switch (ev->reason) {
      case VLStreamStarted:
        break;
      case VLStreamStopped:
      case VLTransferComplete:
        /* バッファ内のフィールドを読み出す */
        while (info=vlGetNextValid(gServer,gBuffer)) {
            dataPtr = vlGetActiveRegion (gServer, gBuffer, info);
            if (gFieldlFlag!=0)          /* if gFieldlFlag==TRUE, then ... */
                GetRegion (dataPtr);
            gFieldlFlag ^= 0xFFFF;       /* FALSE->TRUE, TRUE->FALSE */
            vlPutFree (gServer, gBuffer);
            gFields++;
        }
        break;
      case VLStreamPreempted:
        fprintf (stderr, "%s: Path for this stream preempted.\n", gProgName);
        docleanup (1);
        break;
      case VLSequenceLost:
        fprintf (stderr, "%s: Sequence Lost.\n", gProgName);
        docleanup (1);
        break;
      case VLTransferFailed:
```



```
        fprintf (stderr, "%s: Transfer failed.\n", gProgName);
        docleanup (1);
        break;
      default:
        break;
    }
}



/*
 *  オブジェクトデータの領域を求める関数
 * <input>
 *  data : YUVフォーマットのデータへのポインタ
 *
 * <attetion>
 *  画像にはノイズも乗るので、gContCol以上切目（黒）が続いた
 *  時にのみ、オブジェクトが終わったと判断する。そのため、
 *  あまり近づいた物体は判別できない。
 *  また、領域がgMinSizeよりも小さいときには、この領域は
 *  ノイズであるとして保持しない。
 */
void GetRegion (unsigned char *data)
{
    unsigned char *tmpBuffer;
    int i;
    int objnum;
    int first, second;

    /* Yデータの間引き抜き出しと２値化と孤立した白画素の除去 */
    MakeBWPicture (data, gDataBuffer);

#ifdef DEBUG_WITH_PICTURE
    if (gFields<100) {
        char name[50];
        sprintf (name, "%05d-binary.pbm", gFields);
        Save (name, gDataBuffer);
    }
#endif

    /* gOldRect内でオブジェクトデータを含む四角形を求める */
    objnum = MakeObject ();


#ifdef DEBUG_WITH_PICTURE
    /* オブジェクト以外を消去した後の画像を出力するルーチン */
    /*                  (HDD容量に注意！！)                  */
    if (gFields<100) {
        char name[50];
        sprintf (name, "%05d-object.pbm", gFields);
        Save (name, gDataBuffer);
    }
#endif
#ifdef DEBUG
    {
    int j;
    fprintf (stderr, " obj = %2d, ", objnum);
    for (j=0;j<objnum;j++)  fprintf (stderr, "%5d ", gObjectSize[j]);
    fprintf (stderr, "\n");
    }
#endif

    /* 複数のオブジェクトの内、１・２番目に大きい領域を得る */
    if (gFields==0)
        GetBigSizeObject (objnum, &first, &second);
    else {
```

```
        second = GetSameObject (objnum);
        /* 最適なオブジェクトが見つからなかった場合 */
        if (second<0)
            GetBigSizeObject (objnum, &first, &second);
    }

    /* 適切なオブジェクトが得られたかどうか？ */
    if (second<0) {
/*      fprintf (stderr, "No Hand-Object <objnum is %d>.\n", objnum);*/
        gOldObjectSize  = 0L;
        gOldRect.left   = 0;
        gOldRect.top    = 0;
        gOldRect.right  = HALF_X-1;
        gOldRect.bottom = DEC_Y;
    }
    else {
        gOldObjectSize = (gObject[second].right - gObject[second].left + 1) *
            (gObject[second].bottom - gObject[second].top + 1);

        /* 次の検索範囲を設定 */
        gOldRect.left   = gObject[second].left - gRange;
        gOldRect.top    = gObject[second].top - gRange;
        gOldRect.right  = gObject[second].right + gRange;
        gOldRect.bottom = gObject[second].bottom + gRange;

#ifdef DEBUG
fprintf (stderr, "Next Search Range is %d %d %d %d\n", gOldRect.left,
         gOldRect.top, gOldRect.right, gOldRect.bottom);
#endif

        if(gOldRect.left  <  0)       gOldRect.left   = 0;
        if(gOldRect.top   <  0)       gOldRect.top    = 0;
        if(gOldRect.right >= HALF_X)  gOldRect.right  = HALF_X-1;
        if(gOldRect.bottom > DEC_Y)   gOldRect.bottom = DEC_Y;

        /* オブジェクト領域の出力(指先位置の表示) */
        tmpBuffer = gDataBuffer +
            gObject[second].top * HALF_X + gObject[second].left;
        for (i=gObject[second].left;i<=gObject[second].right;i++) {
            if (*(tmpBuffer++)==WHITE+(second+1)*2) {
                printf ("%d %d\n", i, gObject[second].top);
                break;
            }
        }
    }
    fflush (stdout);
}



/*
 *  YUV画像から２値画像を作成するルーチン
 * <input>
 *  yuv_data : YUV形式の変換元データ領域へのポインタ
 *  bw_data  : B/Wの変換先データ領域へのポインタ（ただし１画素に１バイト）
 */
void MakeBWPicture (unsigned char *yuv_data, unsigned char *bw_data)
{
    long i;

    yuv_data++;
    for (i=0L;i<IMAGE_SIZE;i++) {
        if (*yuv_data<gThreshold)
            *(bw_data++) = BLACK;
        else if (*(bw_data-HALF_X-1)==WHITE         ||
```



```
                 *(bw_data-HALF_X)==WHITE           ||
                 *(bw_data-HALF_X+1)==WHITE         ||
                 *(bw_data-1)==WHITE                ||
                 *(yuv_data+4)>=gThreshold          ||
                 *(yuv_data+HALF_X*4-4)>=gThreshold ||
                 *(yuv_data+HALF_X*4)>=gThreshold   ||
                 *(yuv_data+HALF_X*4+4)>=gThreshold )
            *(bw_data++) = WHITE;
        else
            *(bw_data++) = BLACK;
        yuv_data+=4;
    }
}



/*
 *  gOldRect領域内の２値画像からオブジェクトを抽出する関数
 * <output>
 *  obj : 得られたオブジェクトの数(最大でgObjectNum個)
 *
 * <warning>
 *  オブジェクトの片鱗がgOldRect内に入っていれば抽出します
 */
int MakeObject (void)
{
    unsigned char *tmpBuffer;
    int obj = 0;
    int bottom, left, right, distance;
    int i, j, tmp;
    int h, v;           /* オブジェクトの左右と上下の広さを得るために使用 */
    long whiteCounter;  /* オブジェクト領域内の白画素の数 */

    /* レジスタの数によっては少し速度が上がるかも */
    bottom = gOldRect.bottom;
    left   = gOldRect.left;
    right  = gOldRect.right;
    tmpBuffer = gDataBuffer + gOldRect.top*HALF_X + left;
    distance = HALF_X - (right - left) - 1;

    for (j=gOldRect.top;j<=bottom;j++) {
        for (i=left;i<=right;i++) {
            if (*tmpBuffer==WHITE) {
                gObject[obj].top = j;               /* topは決定 */

                /* 右方向への調査 */
                SearchRegionEdge (tmpBuffer, YSIZE-j, 1, i, &h, &v);
                gObject[obj].right = i + h;
                gObject[obj].bottom = j + v;        /* 仮決定 */

                /* 左方向への調査 */
                SearchRegionEdge (tmpBuffer, YSIZE-j, -1, i, &h, &v);
                gObject[obj].left = i - h;
                tmp = j + v;
                if (gObject[obj].bottom < tmp)          /* 本決定 */
                    gObject[obj].bottom = tmp;

                /* 領域補正（必要か？） */
                if(gObject[obj].bottom>DEC_Y)   gObject[obj].bottom = DEC_Y;
                if(gObject[obj].left<0)         gObject[obj].left   = 0;
                if(gObject[obj].right>=HALF_X)  gObject[obj].right  = HALF_X-1;

                /* 領域が小さすぎたり、白画素が少ない時には捨てる */
                gObjectSize[obj] = (gObject[obj].bottom-gObject[obj].top+1)*
                    (gObject[obj].right - gObject[obj].left + 1);
```

```
                whiteCounter = GetWhiteNum (obj);
                if (gObjectSize[obj] > gMinSize &&
                    (double)whiteCounter/gObjectSize[obj] > gWhiteRatio) {
                    DrowRect (&gObject[obj], obj*2);
                    obj++;
                    /* 速度を得るためgObjectNum回しか調べないことにした */
                    if (obj==gObjectNum)  goto object_search_end;
                }
                else
                    EraseRect (&gObject[obj]);
            }
            tmpBuffer++;
        }
        tmpBuffer += distance;
    }
  object_search_end:
    return obj;
}



/*
 *  オブジェクト領域の端を探索するルーチン
 * <input>
 *  buffer : 探索を開始するバッファの位置
 *  limit  : Y方向の最大探索数
 *  num    : 1(右方向へ探索)か -1(左方向へ探索)
 *  x      : 現在のX座標位置
 *
 *  horizontal : 右左端までの距離
 *  vertical   : 下端までの距離
 */
void SearchRegionEdge (unsigned char *buffer, int ylimit, int direction, int x,
                       int *horizontal, int *vertical)
{
    int i, num, count;
    int h, v;
    int xlimit, first_limit;

    if (direction>0)  xlimit = HALF_X -x;
    else              xlimit = x;
    /* そのラインの最初の白画素検出までどれだけ耐えるか？ */
    first_limit = 1;
    h = v = 0;
#ifdef DEBUG
    fprintf (stderr, "\n%d : %d : ", gFields, YSIZE-ylimit);
#endif
    for (i=0;i<ylimit;i++) {
        num = GetLength (buffer, first_limit, xlimit, direction);
#ifdef DEBUG
        fprintf (stderr, "%d ",num);
#endif
        if (h<=num) {                           /* 最大値を求める */
            h = num;
            count = 0;
            if (gContCol+h < xlimit)  first_limit = gContCol + h;
            else                      first_limit = xlimit;
        }
        else if (num==0) {
            if (++count==gContCol) {
                v = i - gContCol;
                goto object_search_end;       /* equal break */
            }
        }
        else
```



```
            count = 0;
        buffer += HALF_X;
    }
    v = ylimit -1 - count;

  object_search_end:
    *horizontal = h-1;
    *vertical = v;
}



/*
 *  領域中の白画素の数をカウントする
 * <input>
 *  obj : オブジェクトの番号
 */
long GetWhiteNum (int obj)
{
    char *tmpBuffer;
    int i, j, distance;
    long counter = 0L;

    tmpBuffer = gDataBuffer + gObject[obj].top*HALF_X + gObject[obj].left;
    distance = HALF_X - (gObject[obj].right - gObject[obj].left) - 1;

    for (j=gObject[obj].top; j<=gObject[obj].bottom;j++) {
        for (i=gObject[obj].left; i<=gObject[obj].right;i++) {
            if (*(tmpBuffer++)==WHITE)  counter++;
        }
        tmpBuffer+=distance;
    }
    return counter;
}



/*
 *  領域中画素値をプラスする
 * <input>
 *  rect : 領域
 *  num : 値
 */
void DrowRect (Rect *rect, int num)
{
    char *tmpBuffer;
    int i, j, bottom, left, right, distance;

    num+=2;                 /* 仕様 */
    bottom = rect->bottom;
    right = rect->right;
    left = rect->left;
    distance = HALF_X - (right - left) - 2;
    tmpBuffer = gDataBuffer + (rect->top)*HALF_X + left-1;

    for (i=(rect->top);i<=bottom;i++) {
        *(tmpBuffer++) = 250;           /* 選択領域には縦白線を引く */
        for (j=left;j<=right;j++) {
            *(tmpBuffer++) += num;
        }
        *tmpBuffer = 250;               /* 縦白線を引く */
        tmpBuffer += distance;
    }
}
```

```
/* 領域のピクセル値を消去する */
void EraseRect (Rect *rect)
{
    char *tmpBuffer;
    int i, j, top, bottom, left, right, distance;

    bottom = rect->bottom;
    right = rect->right;
    left = rect->left;
    distance = HALF_X - (right - left) - 1;
    tmpBuffer = gDataBuffer + (rect->top)*HALF_X + left;

    for (i=(rect->top);i<=bottom;i++) {
        for (j=left;j<=right;j++) {
            *(tmpBuffer++) = BLACK;
        }
        tmpBuffer += distance;
    }
}



/*
 *  gContCol分の黒がなく、白がいくつ連続して続いているかを調べる
 * <input>
 *  buffer : 初期位置を示すバッファ
 *  first  : 最初に白画素を見つけるまでの画素数
 *  limit  : 検索するピクセル数
 *  addnum : 1回の検査につき、どれだけバッファを移動するか？
 * <output>
 *   (擬似的に) 連続した白画素の個数
 */
int GetLength (char *buffer, int first, int limit, int addnum)
{
    int count = 0, ret = 0;

    /* buffer ～ buffer+firstまでの間に白画素があるか調べる */
    if ((memchr(buffer,WHITE,first))==NULL)  return 0;

    while (limit>0) {
        if (*buffer==WHITE) {
            ret += count+1;
            count = 0;
        }
        else {
            if((++count)==gContCol)  return ret;
        }
        buffer += addnum;
        limit--;
    }
    return ret;
}



/*
 *  1・2番目に大きいオブジェクトを得るルーチン
 * <input>
 *  num : 得られたオブジェクトの数
 *
 *  first : 1番に大きいオブジェクトの番号
```



```
 *  second : 2番目に大きいオブジェクトの番号
 */
void GetBigSizeObject (int num, int *first, int *second)
{
    int i;
    int f, s;

    f = 0;
    if(num>=2)  s = 1;

    for (i=1;i<num;i++)
        if(gObjectSize[f]<gObjectSize[i])          f = i;      /* 顔？ */
    for (i=0;i<num;i++)
        if(f!=i && gObjectSize[s]<gObjectSize[i])  s = i;      /* 指し手？ */
    if (num==0) {
        *first = -1;
        *second = -1;
    }
    else if (num==1) {
        *first = f;
        *second = -1;
    }
    else {
        *first = f;
        *second = s;
    }
}



/*
 *  オブジェクトの面積から追跡オブジェクトと同じらしいものの番号を得る
 * <input>
 *  objnum : 得られたオブジェクトの数
 * <output>
 *  num : オブジェクトの番号。
 *        ただし、負の数のものは無効
 *
 * <reaad me>
 *  oldObjectSizeに一番近いオブジェクトを探します。
 *  ただし、gSizeRatio比を越えた場合には無効とします。
 */
int GetSameObject (int objnum)
{
    int i, num;
    double ratio, tmp_ratio;

    if (gOldObjectSize==0L) {          /* 前はオブジェクトがなかったか？ */
        return -1;
    }
    else {
        num = -1;
        ratio = gSizeRatio;
        for (i=0;i<objnum;i++) {
            /* tmp_ratioを常に1以上にするための比較 */
            if (gObjectSize[i]>gOldObjectSize)
                tmp_ratio = gObjectSize[i] / gOldObjectSize;
            else
                tmp_ratio = gOldObjectSize / gObjectSize[i];
            /* gSizeRatio内に収まってないと無効 */
            if (tmp_ratio < gSizeRatio) {
                if (tmp_ratio-ratio<0) {                /* より1に近い */
                    ratio = tmp_ratio;
                    num = i;
                }
            }
```

```
        }
    }
    return num;
}



/*
 *  終了前のクリーンアップを行う。
 *  <input>
 *    ret : 終了コード
 */
void docleanup (int ret)
{
    vlEndTransfer (gServer, gPath);
    vlDeregisterBuffer (gServer, gPath, gDrn, gBuffer);
    vlDestroyBuffer (gServer, gBuffer);
    vlDestroyPath (gServer, gPath);
    vlCloseVideo (gServer);
    free (gDataBuffer);
    free (gObject);
    free (gObjectSize);
    exit (ret);
}



/*
 * シグナルを受け取るフックルーチン
 * <input>
 *  sig : 受信したシグナル番号
 */
void SignalHandler (int sig)
{
    fprintf (stderr, "%s: catch the signal <%d>.\n", gProgName, sig);
    fprintf (stderr, "%d fields transferred.\n", gFields);
    docleanup(sig);
}



#ifdef DEBUG_WITH_PICTURE
/*
 *  2値画像をセーブする関数
 * <input>
 *  name : 出力ファイル名
 *  picture : 2値画像へのポインタ
 *
 *
 * <warning>
 *  画素値 % 2 == 1 なら黒画素
 *  画素値 % 2 == 0 なら白画素
 *  と判定します。
 */
void Save (unsigned char *name, char *picture)
{
    FILE *out_fp;
    int i, shift;
    unsigned char data, tmp;

    if ((out_fp=fopen(name,"wb"))==NULL) {
        fprintf (stderr, "%s: output file don't create.\n");
        exit (1);
    }
```



```
    fprintf (out_fp, "P4\n");
    fprintf (out_fp, "%d %d\n", HALF_X, YSIZE);

    for (i=0;i<HALF_X*YSIZE/8;i++) {
        data = 0;
        for (shift=7;shift>=0;shift--) {
            tmp = *(picture++) % 2;
            data += tmp << shift;              /* |= でも一緒だよね？*/
        }
        fwrite (&data, 1, 1, out_fp);
    }
    fclose (out_fp);
}
#endif
```

```
/*
 *                          searching.c
 *        顔/手の領域を判断し、頭の領域の中心下座標を出力するプログラム
 *                  Usage : get_head "b:c:m:o:r:s:t:"
 *
 *
 *                              <warning>
 *                          Don't Add Optimizing Flag!!
 *
 *                      To exit application is 'Ctrl-C'.
 *                  This program hook the 'Ctrl-C' Vector.
 */

#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>
#include <sys/types.h>
#include <dmedia/vl.h>
#include <dmedia/vl_sirius.h>



#define USAGE "\nusage: %s %s\n"\
    "where option arguments :\n"\
    "         -b<n>  use <n> ring buffers                    (default is 20).\n"\
    "         -c<n>  <n> continuous black separate object    (default is 8).\n"\
    "         -m<n>  minimum object size is <n>              (default is 20).\n"\
    "         -o<n>  maximal search object is <n>            (default is 10).\n"\
    "         -r<n>  object move <n> pixels in 1 frame       (default is 10).\n"\
    "         -s<f>  target object's size ratio in 1 frame (default is 2.0).\n"\
    "         -t<n>  threshold value is <n>                  (default is 80).\n"\
    "\n"\
    "                 <n> is integer, <f> is float.\n\n"
#define JUSAGE "\nusage: %s %s\n"\
    "オプション一覧 :\n"\
    "         -b<n>  <n>個のリングバッファを使用する         (default is 20).\n"\
    "         -c<n>  連続<n>の黒画素がオブジェクトの切目     (default is 8).\n"\
    "         -m<n>  オブジェクトの最小面積を<n>とする       (default is 20).\n"\
    "         -o<n>  最大オブジェクト探索数を<n>とする       (default is 10).\n"\
    "         -r<n>  1フレームで対象物は最大<n>画素移動      (default is 10).\n"\
    "         -s<f>  1フレームで対象物は最大<f>面積が変化 (default is 2.0).\n"\
    "         -t<n>  閾値を<n>とする                         (default is 80).\n"\
    "\n"\
    "                 <n> は整数, <f> は実数.\n\n"

#define CMDARGS         "b:c:m:o:r:s:t:"
#define WHITE           0
#define BLACK           1
#define XSIZE           720       /* 1フィールドあたりの画面の大きさ */
#define YSIZE           243
#define HALF_X          XSIZE/2         /* プログラム内で使用 */
#define DEC_Y           YSIZE-1
#define IMAGE_SIZE      HALF_X*YSIZE    /* 縮小画像の大きさ */

#define DEBUG
#undef  DEBUG
#define DEBUG_WITH_PICTURE          /* 2値画像付きデバッグ */
#undef  DEBUG_WITH_PICTURE

/**************/
/* 構造体定義 */
/**************/
typedef struct {
    int left, right, top, bottom;
} Rect;
```



```
/********************/
/* グローバルデータ */
/********************/
char   *gProgName;                  /* プログラムの名前 */
char   *gDataBuffer;                /* 2値画像用のバッファ */
short  gField1Flag;                 /* 今どちらのフィールドであるか */
int    gFields;                     /* 取り込んだ画像の数 */
Rect   *gObject;                    /* オブジェクトの記憶領域（顔と手） */
long   *gObjectSize;                /* オブジェクトの面積を記憶 */
Rect   gOldRect;                    /* 前の状態の四角形を保持 */
long   gOldObjectSize;              /* 前のオブジェクトの大きさを保持 */
int    gBufNum = 20;                /* 作成するVLバッファの数 */
int    gRange = 10;                 /* 1フレームの間に物が移動する最大距離 */
double gSizeRatio = 2.0;            /* 1フレーム前の探索物の大きさの比率 */
int    gContCol = 8;                /* 物の切目とする黒画素の数 */
int    gObjectNum = 10;             /* 探索するオブジェクトの最大数 */
int    gThreshold = 80;             /* しきい値 */
int    gMinSize = 20;               /* オブジェクトの最小面積 */


VLServer  gServer;                  /* vl関係のグローバル変数 */
VLBuffer  gBuffer;
VLPath    gPath;
VLNode    gSrc, gDrn;



/********************/
/* プロトタイプ宣言 */
/********************/
void Initialize (int, char**);
void CommunicateDaemon (void);
void ProcessEvent (VLServer, VLEvent*, void*);
void GetRegion (unsigned char*);
void MakeBWPicture (char*, char*);
int MakeObject (void);
void SearchRegionEdge (char*, int, int, int, int*, int*);
void DrowRect (Rect*, int);
void EraseRect (Rect*);
int GetLength (char *, int, int);
void GetBigSizeObject (int, int*, int*);
int GetSameObject (int);
void docleanup (int);
void SignalHandler (int);
void Save (unsigned char*, char*);



void main (int argc, char *argv[])
{
    Initialize (argc, argv);
    CommunicateDaemon ();

    /* ユーザが終了を指示するまでループする */
    vlMainLoop ();
}



/*
 * アプリケーションの初期化を行う
 */
void Initialize (int argc, char *argv[])
```

```
{
    char *lang;
    int i, c;

    gFields = 0;
    gProgName = argv[0];

    while ((c=getopt(argc,argv,CMDARGS)) != EOF) {
        switch (c) {
          case 'b' :
              gBufNum = atoi(optarg);
              if (gBufNum<0) {
                  fprintf (stderr, "ring buffer num is big to 0.\n");
                  exit (1);
              }
              break;
          case 'c' :
              gContCol = atoi(optarg);
              if (gContCol<0) {
                  fprintf (stderr, "black continuous is big to 0.\n");
                  exit (1);
              }
              break;
          case 'm' :
              gMinSize = atoi(optarg);
              if (gMinSize<0) {
                  fprintf (stderr, "object's minimum size is big to 0.\n");
                  exit (1);
              }
              break;
          case 'o' :
              gObjectNum = atoi(optarg);
              if (gObjectNum<0) {
                  fprintf (stderr,"number of searching object is big to 0.\n");
                  exit (1);
              }
              break;
          case 'r' :
              gRange = atoi(optarg);
              if (gRange<0) {
                  fprintf (stderr, "finger movement range is big to 0.\n");
                  exit (1);
              }
              break;
          case 's' :
              gSizeRatio = atof(optarg);
              if (gSizeRatio<1.0) {
                  fprintf (stderr, "size ratio is big to 1.\n");
                  exit (1);
              }
              break;
          case 't' :
              gThreshold = atoi(optarg);
              if (gThreshold<=0 || gThreshold>256) {
                  fprintf (stderr, "threshold value is 1 to 255.\n");
                  exit (1);
              }
              break;
          default  :
              lang = getenv ("LANG");
              if (lang!=NULL && *lang=='j' && *(lang+1)=='a')
                  fprintf (stderr, JUSAGE, gProgName, CMDARGS);
              else
                  fprintf (stderr, USAGE, gProgName, CMDARGS);
              exit(1);
              break;
```



```
        }
    }
#ifdef DEBUG
    fprintf (stderr, "gBufNum is %d\n", gBufNum);
    fprintf (stderr, "gContCol is %d\n", gContCol);
    fprintf (stderr, "gMinSize is %d\n", gMinSize);
    fprintf (stderr, "gObjectNum is %d\n", gObjectNum);
    fprintf (stderr, "gRange is %d\n", gRange);
    fprintf (stderr, "gSizeRatio is %lf\n", gSizeRatio);
    fprintf (stderr, "gThreshold is %d\n", gThreshold);
#endif

    /* gObjectとgObjectSizeの領域作成 */
    if ((gObject=malloc(gObjectNum*sizeof(Rect)))==NULL) {
        fprintf (stderr, "malloc error.\n");
        exit (1);
    }
    if ((gObjectSize=malloc(gObjectNum*sizeof(long)))==NULL) {
        fprintf (stderr, "malloc error.\n");
        exit (1);
    }

    /* オブジェクト検索領域の設定 */
    gOldRect.left   = 0;
    gOldRect.top    = 0;
    gOldRect.right  = HALF_X-1;
    gOldRect.bottom = DEC_Y;

    /* シグナルのフック */
    for(i=1;i<=32;i++)  signal(i,SignalHandler);
}



/*
 * Sirius Video への接続と初期化
 */
void CommunicateDaemon (void)
{
    VLDevList             devlist;
    VLControlValue        val;
    VLTransferDescriptor  xferDesc;
    int                   i, timing, devicenum;
    long                  gFieldsize;

    /* デーモンへ接続 */
    if ((gServer=vlOpenVideo(""))==NULL) {
        fprintf (stderr, "%s: couldn't open video\n", gProgName);
        exit (1);
    }
    else  fprintf (stderr, "Connected Server Daemon.\n");

    /* ノードの設定 */
    gSrc = vlGetNode (gServer, VL_SRC, VL_VIDEO, SIR_SRC_ANALOG_VIDEO);
    gDrn = vlGetNode (gServer, VL_DRN, VL_MEM, VL_ANY);

    /* パードの設定とパス仕様の定義 */
    if ((gPath=vlCreatePath(gServer,VL_ANY,gSrc,gDrn)) < 0) {
        vlPerror ("create path failed.\n");
        vlCloseVideo (gServer);
        exit (1);
    }

    if (vlSetupPaths(gServer,(VLPathList)&gPath,1,VL_SHARE,VL_SHARE)!=0) {
        vlPerror ("setup path failed.\n");
        vlCloseVideo (gServer);
```

```
        exit (1);
    }

    /* 受信したいイベントを設定 */
    vlSelectEvents (gServer, gPath,
                    VLTransferCompleteMask | VLStreamPreemptedMask |
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLTransferFailedMask | VLSequenceLostMask);

    /* タイミングの設定(gSrc) */
    val.intVal = VL_TIMING_525_CCIR601;
    if ((vlSetControl(gServer,gPath,gSrc,VL_TIMING,&val)) != 0) {
        vlPerror ("set timing of digital failed.\n");
        vlCloseVideo (gServer);
        exit (1);
    }

    /* ビデオ出力フォーマットの指定(gDrn) */
    val.intVal = VL_FORMAT_SMPTE_YUV;
    vlSetControl (gServer, gPath, gDrn, VL_FORMAT, &val);

    /* タイミングの設定(gDrn) */
    if (vlGetControl(gServer,gPath,gSrc,VL_TIMING,&val) == 0) {
        if ((vlSetControl(gServer,gPath,gDrn,VL_TIMING,&val)!=0)
            && (vlErrno!=VLBadControl)) {
            vlPerror ("SetControl Failed.\n");
            vlCloseVideo (gServer);
            exit (1);
        }
    }

    /* 取り込むフレームのタイプを指定(gDrn) */
    val.intVal = VL_CAPTURE_INTERLEAVED;
    vlSetControl (gServer, gPath, gDrn, VL_CAP_TYPE, &val);

    /* ピクセルをYUVで保存するための設定(gDrn) */
    val.intVal = VL_PACKING_YVYU_422_8;
    vlSetControl (gServer, gPath, gDrn, VL_PACKING, &val);

    /* 最初に取り込まれるフィールドを知る(gSrc) */
    if (vlGetControl(gServer,gPath,gSrc,VL_SIR_FIELD_DOMINANCE,&val) == 0) {
        timing = ((val.intVal == VL_TIMING_525_SQ_PIX) ||
                  (val.intVal == VL_TIMING_525_CCIR601));
        switch (val.intVal) {
          case SIR_F1_IS_DOMINANT:
            if(timing)  gField1Flag = 0;
            else        gField1Flag = 0xFFFF;
            break;
          case SIR_F2_IS_DOMINANT:
            if (timing)  gField1Flag = 0xFFFF;
            else         gField1Flag = 0;
            break;
        }
    }

    /* ビデオサイズとそのバイト数を得る */
    vlGetControl (gServer, gPath, gDrn, VL_SIZE, &val);
    if (val.xyVal.x!=XSIZE || val.xyVal.y!=YSIZE) {
        vlPerror ("picture size is strange.\n");
        vlCloseVideo (gServer);
        exit (1);
    }
    gFieldsize = vlGetTransferSize (gServer, gPath);          /* no use */

    /* 転送のためのコールバックルーチンを設定 */
    vlAddCallback (gServer, gPath,
```



```
                    VLTransferCompleteMask | VLStreamPreemptedMask |
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLSequenceLostMask | VLTransferFailedMask,
                    ProcessEvent, NULL);

    /* gBufNum分のフレームバッファの作成と登録 */
    if ((gBuffer=vlCreateBuffer(gServer,gPath,gDrn,gBufNum))==NULL) {
        vlPerror ("create buffer is failed.\n");
        vlCloseVideo (gServer);
        exit (1);
    }

    if (vlRegisterBuffer(gServer, gPath, gDrn, gBuffer)!=0) {
        vlPerror ("to regist buffer is failed.\n");
        vlDestroyBuffer (gServer, gBuffer);
        vlCloseVideo (gServer);
        exit (1);
    }

    /* 縮退用のバッファを確保 */
    if ((gDataBuffer=malloc(IMAGE_SIZE*sizeof(char)))==NULL) {
        vlPerror ("malloc error.\n");
        docleanup (1);
    }

    xferDesc.mode = VL_TRANSFER_MODE_CONTINUOUS;
    xferDesc.count = 10000;          /* CONTINUOUS なら無視される */
    xferDesc.delay = 0;              /* VLTriggerImmediate なら無視される */
    xferDesc.trigger = VLTriggerImmediate;

    /* アナログのデータ転送の開始 */
    if (vlBeginTransfer(gServer,gPath,1,&xferDesc)) {
        vlPerror ("transfer is failed.\n");
        vlDestroyBuffer (gServer, gBuffer);
        vlCloseVideo (gServer);
        exit (1);
    }
    fprintf (stderr, "data transfer is started.\n");
}



/*
 *  ビデオライブラリイベントを処理する
 * <input>
 *  gServer : VLServerへのポインタ
 *  ev : 何のイベントで呼ばれたか？
 *  data : ユーザーが使用すると宣言したデータへのポインタ
 */
void ProcessEvent (VLServer gServer, VLEvent *ev, void *data)
{
    VLInfoPtr info;
    char      *dataPtr;

    switch (ev->reason) {
      case VLStreamStarted:
        break;
      case VLStreamStopped:
      case VLTransferComplete:
        /* バッファ内のフィールドを読み出す */
        while (info=vlGetNextValid(gServer,gBuffer)) {
            dataPtr = vlGetActiveRegion (gServer, gBuffer, info);
            if (gField1Flag!=0)          /* if gField1Flag==TRUE, then ... */
                GetRegion (dataPtr);
            gField1Flag ^= 0xFFFF;       /* FALSE->TRUE, TRUE->FALSE */
            vlPutFree (gServer, gBuffer);
```

```
            gFields++;
          }
          break;
        case VLStreamPreempted:
          fprintf (stderr, "%s: Path for this stream preempted.\n", gProgName);
          docleanup (1);
          break;
        case VLSequenceLost:
          fprintf (stderr, "%s: Sequence Lost.\n", gProgName);
          docleanup (1);
          break;
        case VLTransferFailed:
          fprintf (stderr, "%s: Transfer failed.\n", gProgName);
          docleanup (1);
          break;
        default:
          break;
      }
}



/*
 *  オブジェクトデータの領域を求める関数
 * <input>
 *  data : YUVフォーマットのデータへのポインタ
 *
 * <attetion>
 *  画像にはノイズも乗るので、gContCol以上切目（黒）が続いた
 *  時にのみ、オブジェクトが終わったと判断する。そのため、
 *  あまり近づいた物体は判別できない。
 *  また、領域がgMinSizeよりも小さいときには、この領域は
 *  ノイズであるとして保持しない。
 */
void GetRegion (unsigned char *data)
{
    unsigned char *tmpBuffer;
    int i;
    int objnum;
    int first, second;

    /* Yデータの間引き抜き出しと２値化と孤立した白画素の除去 */
    MakeBWPicture (data, gDataBuffer);

#ifdef DEBUG_WITH_PICTURE
    if (gFields<50) {
        char name[50];
        sprintf (name, "%05d-binary.pbm", gFields);
        Save (name, gDataBuffer);
    }
#endif

    /* gOldRect内でオブジェクトデータを含む四角形を求める */
    objnum = MakeObject ();


#ifdef DEBUG_WITH_PICTURE
    /* オブジェクト以外を消去した後の画像を出力するルーチン */
    /*                  (HDD容量に注意！！)                  */
    if (gFields<50) {
        char name[50];
        sprintf (name, "%05d-object.pbm", gFields);
        Save (name, gDataBuffer);
    }
#endif
#ifdef DEBUG
```



```
    {
    int j;
    fprintf (stderr, " obj = %2d, ", objnum);
    for (j=0;j<objnum;j++)  fprintf (stderr, "%5d ", gObjectSize[j]);
    fprintf (stderr, "\n");
    }
#endif

    /* 複数のオブジェクトの内、１・２番目に大きい領域を得る */
    if (gFields==0)
        GetBigSizeObject (objnum, &first, &second);
    else {
        first = GetSameObject (objnum);
        /* 最適なオブジェクトが見つからなかった場合 */
        if (first<0)
            GetBigSizeObject (objnum, &first, &second);
    }

    /* 適切なオブジェクトが得られたかどうか？ */
    if (first<0) {
        fprintf (stderr, "No Hand-Object <objnum is %d>.\n", objnum);
        gOldObjectSize  = 0L;
        gOldRect.left   = 0;
        gOldRect.top    = 0;
        gOldRect.right  = HALF_X-1;
        gOldRect.bottom = DEC_Y;
    }
    else {
        gOldObjectSize = (gObject[first].right - gObject[first].left + 1) *
            (gObject[first].bottom - gObject[first].top + 1);

        /* 次の検索範囲を設定 */
        gOldRect.left   = gObject[first].left - gRange;
        gOldRect.top    = gObject[first].top - gRange;
        gOldRect.right  = gObject[first].right + gRange;
        gOldRect.bottom = gObject[first].bottom + gRange;

#ifdef DEBUG
fprintf (stderr, "Next Search Range is %d %d %d %d\n", gOldRect.left,
     gOldRect.top, gOldRect.right, gOldRect.bottom);
#endif

        if(gOldRect.left  <  0)       gOldRect.left   = 0;
        if(gOldRect.top   <  0)       gOldRect.top    = 0;
        if(gOldRect.right >= HALF_X)  gOldRect.right  = HALF_X-1;
        if(gOldRect.bottom > DEC_Y)   gOldRect.bottom = DEC_Y;

        /* オブジェクト領域の出力(指先位置の表示) */
        printf ("%d %d\n", (int)((gObject[first].left+gObject[first].right)/2),
                gObject[first].bottom);
    }
    fflush (stdout);
}



/*
 *  YUV画像から２値画像を作成するルーチン
 * <input>
 *  yuv_data : YUV形式の変換元データ領域へのポインタ
 *  bw_data  : B/Wの変換先データ領域へのポインタ（ただし１画素に１バイト）
 */
void MakeBWPicture (unsigned char *yuv_data, unsigned char *bw_data)
{
    long i;
```

```
    yuv_data++;
    for (i=0L;i<IMAGE_SIZE;i++) {
        if (*yuv_data<gThreshold)
            *(bw_data++) = BLACK;
        else if (*(bw_data-HALF_X-1)==WHITE          ||
                 *(bw_data-HALF_X)==WHITE            ||
                 *(bw_data-HALF_X+1)==WHITE          ||
                 *(bw_data-1)==WHITE                 ||
                 *(yuv_data+4)>=gThreshold           ||
                 *(yuv_data+HALF_X*4-4)>=gThreshold  ||
                 *(yuv_data+HALF_X*4)>=gThreshold    ||
                 *(yuv_data+HALF_X*4+4)>=gThreshold )
            *(bw_data++) = WHITE;
        else
            *(bw_data++) = BLACK;
        yuv_data+=4;
    }
}



/*
 *  gOldRect領域内の２値画像からオブジェクトを抽出する関数
 * <output>
 *  obj : 得られたオブジェクトの数(最大でgObjectNum個)
 *
 * <warning>
 *  オブジェクトの片鱗がgOldRect内に入っていれば抽出します
 */
int MakeObject (void)
{
    unsigned char *tmpBuffer;
    int obj = 0;
    int bottom, left, right, distance;
    int i, j, tmp;
    int h, v;           /* オブジェクトの左右と上下の広さを得るために使用 */

    /* レジスタの数によっては少し速度が上がるかも */
    bottom = gOldRect.bottom;
    left   = gOldRect.left;
    right  = gOldRect.right;
    tmpBuffer = gDataBuffer + gOldRect.top*HALF_X + left;
    distance = HALF_X - (right - left) - 1;

    for (j=gOldRect.top;j<=bottom;j++) {
        for (i=left;i<=right;i++) {
            if (*tmpBuffer==WHITE) {
                gObject[obj].top = j;               /* topは決定 */

                /* 右方向への調査 */
                SearchRegionEdge (tmpBuffer, YSIZE-j, 1, i, &h, &v);
                gObject[obj].right = i + h;
                gObject[obj].bottom = j + v;        /* 仮決定 */

                /* 左方向への調査 */
                SearchRegionEdge (tmpBuffer, YSIZE-j, -1, i, &h, &v);
                gObject[obj].left = i - h;
                tmp = j + v;
                if (gObject[obj].bottom < tmp)          /* 本決定 */
                    gObject[obj].bottom = tmp;

                /* 領域補正 */
                if(gObject[obj].bottom>DEC_Y)   gObject[obj].bottom = DEC_Y;
                if(gObject[obj].left<0)         gObject[obj].left   = 0;
```



```
                if(gObject[obj].right>=HALF_X)  gObject[obj].right  = HALF_X-1;

                /* あまりに領域が小さいときには捨てる */
                gObjectSize[obj] = (gObject[obj].bottom-gObject[obj].top+1)*
                    (gObject[obj].right - gObject[obj].left + 1);
                if (gObjectSize[obj] > gMinSize) {
                    DrowRect (&gObject[obj], obj*2);
                    obj++;
                    /* 速度を得るためgObjectNum回しか調べないことにした */
                    if (obj==gObjectNum)  goto object_search_end;
                }
                else
                    EraseRect (&gObject[obj]);
            }
            tmpBuffer++;
        }
        tmpBuffer += distance;
    }
  object_search_end:
    return obj;
}



/*
 *  オブジェクト領域の端を探索するルーチン
 * <input>
 *  buffer : 探索を開始するバッファの位置
 *  limit  : Y方向の最大探索数
 *  num    : 1(右方向へ探索)か -1(左方向へ探索)
 *  x      : 現在のx座標位置
 *
 *  horizontal : 右左端までの距離
 *  vertical   : 下端までの距離
 */
void SearchRegionEdge (unsigned char *buffer, int ylimit, int direction, int x,
                       int *horizontal, int *vertical)
{
    int i, num, count;
    int h, v;
    int search_range, xlimit;

    xlimit = HALF_X-x;
    if (xlimit<gContCol)  search_range = xlimit;
    else                  search_range = gContCol;
    h = v = 0;
#ifdef DEBUG
    printf ("\n%d : %d : ", gFields, YSIZE-ylimit);
#endif
    for (i=0;i<ylimit;i++) {
        num = GetLength (buffer, search_range, direction);
#ifdef DEBUG
        printf ("%d ",num);
#endif
        if (h<=num) {                           /* 最大値を求める */
            h = num;
            count = 0;
            if (gContCol+h < xlimit)  search_range = gContCol + h;
        }
        else if (num==0) {
            if (++count==gContCol) {
                v = i - gContCol;
                goto object_search_end;       /* equal break */
            }
        }
```

```
        buffer += HALF_X;
    }
    v = ylimit -1 - count;

  object_search_end:
    *horizontal = h-1;
    *vertical = v;
}




/*
 *  領域中画素値をプラスする
 * <input>
 *  rect : 領域
 *  num : 値
 */
void DrowRect (Rect *rect, int num)
{
    char *tmpBuffer;
    int i, j, bottom, left, right, distance;

    num+=2;                 /* 仕様 */
    bottom = rect->bottom;
    right = rect->right;
    left = (rect->left) + 1;
    distance = HALF_X - (right - left) - 1;
    tmpBuffer = gDataBuffer + (rect->top)*HALF_X + (left - 1);

    for (i=(rect->top);i<=bottom;i++) {
        *(tmpBuffer++) = WHITE+num;       /* 選択領域には縦白線を引く */
        for (j=left;j<right;j++) {
            *tmpBuffer += num;
            tmpBuffer++;
        }
        *tmpBuffer = WHITE+num;       /* 縦白線を引く */
        tmpBuffer += distance;
    }
}



/* 領域のピクセル値を消去する */
void EraseRect (Rect *rect)
{
    char *tmpBuffer;
    int i, j, top, bottom, left, right, distance;

    bottom = rect->bottom;
    right = rect->right;
    left = rect->left;
    distance = HALF_X - (right - left) - 1;
    tmpBuffer = gDataBuffer + (rect->top)*HALF_X + left;

    for (i=(rect->top);i<=bottom;i++) {
        for (j=left;j<=right;j++) {
            *(tmpBuffer++) = BLACK;
        }
        tmpBuffer += distance;
    }
}
```



```
/*
 *  gContCol分の黒がなく、白がいくつ連続して続いているかを調べる
 * <input>
 *  buffer : 初期位置を示すバッファ
 *  limit  : 検索するピクセル数
 *  addnum : 1回の検査につき、どれだけバッファを移動するか？
 * <output>
 *  （擬似的に）連続した白画素の個数
 */
int GetLength (char *buffer, int limit, int addnum)
{
    int count = 0, ret = 0;

    while (limit>0) {
        if (*buffer==WHITE) {
            ret += count+1;
            count = 0;
        }
        else {
            count++;
        }
        buffer += addnum;
        limit--;
    }
    return ret;
}



/*
 *  1・2番目に大きいオブジェクトを得るルーチン
 * <input>
 *  num : 得られたオブジェクトの数
 *
 *  first : 1番に大きいオブジェクトの番号
 *  second : 2番目に大きいオブジェクトの番号
 */
void GetBigSizeObject (int num, int *first, int *second)
{
    int i;
    int f, s;

    f = s = 0;
    for (i=0;i<num;i++)
        if(gObjectSize[f]<gObjectSize[i])          f = i;      /* 顔？ */
    for (i=0;i<num;i++)
        if(f!=i && gObjectSize[s]<gObjectSize[i])  s = i;      /* 指し手？ */
    if (num==0) {
        *first = -1;
        *second = -1;
    }
    else if (num==1) {
        *first = f;
        *second = -1;
    }
    else {
        *first = f;
        *second = s;
    }
}



/*
```

```
 *  オブジェクトの面積から追跡オブジェクトと同じらしいものの番号を得る
 * <input>
 *  objnum : 得られたオブジェクトの数
 * <output>
 *  num : オブジェクトの番号。
 *        ただし、負の数のものは無効
 *
 * <reaad me>
 *  oldObjectSizeに一番近いオブジェクトを探します。
 *  ただし、gSizeRatio比を越えた場合には無効とします。
 */
int GetSameObject (int objnum)
{
    int i, num;
    double ratio, tmp_ratio;

    if (gOldObjectSize==0L) {          /* 前はオブジェクトがなかったか？ */
        return -1;
    }
    else {
        num = -1;
        ratio = gSizeRatio;
        for (i=0;i<objnum;i++) {
            /* tmp_ratioを常に１以上にするための比較 */
            if (gObjectSize[i]>gOldObjectSize)
                tmp_ratio = gObjectSize[i] / gOldObjectSize;
            else
                tmp_ratio = gOldObjectSize / gObjectSize[i];
            /* gSizeRatio内に収まってないと無効 */
            if (tmp_ratio < gSizeRatio) {
                if (tmp_ratio-ratio<0) {                /* より１に近い */
                    ratio = tmp_ratio;
                    num = i;
                }
            }
        }
    }
    return num;
}



/*
 *  終了前のクリーンアップを行う。
 *  <input>
 *    ret : 終了コード
 */
void docleanup (int ret)
{
    vlEndTransfer (gServer, gPath);
    vlDeregisterBuffer (gServer, gPath, gDrn, gBuffer);
    vlDestroyBuffer (gServer, gBuffer);
    vlDestroyPath (gServer, gPath);
    vlCloseVideo (gServer);
    free (gDataBuffer);
    free (gObject);
    free (gObjectSize);
    exit (ret);
}



/*
 * シグナルを受け取るフックルーチン
 * <input>
```



```
 *  sig : 受信したシグナル番号
 */
void SignalHandler (int sig)
{
    fprintf (stderr, "%s: catch the signal <%d>.\n", gProgName, sig);
    fprintf (stderr, "%d fields transferred.\n", gFields);
    docleanup(sig);
}



#ifdef DEBUG_WITH_PICTURE
/*
 *  ２値画像をセーブする関数
 * <input>
 *  name : 出力ファイル名
 *  picture : ２値画像へのポインタ
 *
 *
 * <warning>
 *  画素値 % 2 == 1 なら黒画素
 *  画素値 % 2 == 0 なら白画素
 *  と判定します。
 */
void Save (unsigned char *name, char *picture)
{
    FILE *out_fp;
    int i, shift;
    unsigned char data, tmp;

    if ((out_fp=fopen(name,"wb"))==NULL) {
        fprintf (stderr, "%s: output file don't create.\n");
        exit (1);
    }
    fprintf (out_fp, "P4\n");
    fprintf (out_fp, "%d %d\n", HALF_X, YSIZE);

    for (i=0;i<HALF_X*YSIZE/8;i++) {
        data = 0;
        for (shift=7;shift>=0;shift--) {
            tmp = *(picture++) % 2;
            data += tmp << shift;              /* |= でも一緒だよね？*/
        }
        fwrite (&data, 1, 1, out_fp);
    }
    fclose (out_fp);
}
#endif
```

```
#include <stdio.h>
#include <signal.h>


#define FALSE 0
#define TRUE !FALSE


void SignalHandler (int);



/*
 * 座標値の変換を行う (スイッチャー用)
 * <input>
 *  x : 0 to 359 (but range is 2 to 359 why?)
 *  y : 0 to 242 (but range is 1 to 241)
 * <output>
 *  x : 15000 to 55000
 *  y : 15000 to 55000
 */
void main (void)
{
    int in_x, in_y;
    int i, count;
    long out_x, out_y;

    /* Ctrl-Cベクタのフック */
    for(i=1;i<=32;i++)  signal (i, SignalHandler);

    printf ("Dvs6000 Mel Xpt PgmBus 1\n");
    printf ("Dvs6000 Mel Xpt PstBus 3\n");
    printf ("Dvs6000 Mel TransitionType Wipe\n");
    printf ("Dvs6000 Mel FaderPositionerAnalog Fader 5000\n");
    fflush (stdout);

    while(TRUE) {
        count = scanf ("%d %d", &in_x, &in_y);
        if (count==2) {
            if (in_x<2 || in_x>359 || in_y<1 || in_y>241)
                fprintf (stderr, "<x=%d><y=%d>Range Error.\n", in_x, in_y);
            else {
                out_x = in_x;      /* out_x is 0 to 357 */
                out_y = in_y;      /* out_y is 0 to 240 */
                out_x = (out_x*40000)/360;
                out_y = (out_y*40000)/243;
                out_x += 15000;
                out_y += 15000;

                printf ("Dvs6000 Mel FaderPositionerAnalog Positioner %ld %ld\n", out
_x, out_y);
                fflush (stdout);
            }
        }
    }
}


void SignalHandler (int sig)
{
    fprintf (stderr, "catch the signal <%d>.\n", sig);
    printf ("Quit\n");
    fflush (stdout);
    exit (sig);
}
```

```
#include <stdio.h>
#include <signal.h>
#include "socket.h"


#define FALSE 0
#define TRUE !FALSE


void SignalHandler (int);


int gClient;


/*
 * 座標値の変換を行い、ソケット通信する（スイッチャー用）
 * <input>
 *  x : 0 to 359 (but range is 2 to 359 why?)
 *  y : 0 to 242 (but range is 1 to 241)
 * <output>
 *  x : 15000 to 55000
 *  y : 15000 to 55000
 */
void main (int argc, char *argv[])
{
    char hostname[64];
    char send_packet[PKTSIZE];
    char receive_packet[PKTSIZE];
    int  i, count, in_x, in_y;
    long out_x, out_y;

    memset (hostname, NULL, PKTSIZE);
    memset (send_packet, NULL, PKTSIZE);

    /* Ctrl-Cベクタのフック */
    for(i=1;i<=32;i++)  signal (i, SignalHandler);

    /* ホスト名の */
    if (argc!=2) {
        fprintf (stderr, "No Socket Hostname!!\n");
        exit (1);
    }
    else
        strcpy (hostname, argv[1]);

    if ((gClient=OpenSocketClient(hostname,PORTNUM)) == -1) {
        fprintf (stderr, "socket open error.\n");
        exit (-1);
    }

/*    printf ("Dvs6000 Me1 Xpt PgmBus 3\n");
    printf ("Dvs6000 Me1 Xpt PstBus 5\n");
*/    printf ("Dvs6000 Me1 TransitionType Wipe\n");
    printf ("Dvs6000 Me1 FaderPositionerAnalog Fader 5000\n");
    fflush (stdout);

    while(TRUE) {
        count = scanf ("%d %d", &in_x, &in_y);
        if (count==2) {
            if (in_x<2 || in_x>359 || in_y<1 || in_y>241)
                fprintf (stderr, "<x=%d><y=%d>Range Error.\n", in_x, in_y);
            else {
                sprintf (send_packet, "%d %d\n", in_x, in_y);
                if (SendSocket(gClient,send_packet,PKTSIZE) == -1) {
                    perror ("send");
```



```
                    exit (-1);
                }
                out_x = in_x;      /* out_x is 0 to 357 */
                out_y = in_y;      /* out_y is 0 to 240 */
                out_x = (out_x*40000)/360;
                out_y = (out_y*40000)/243;
                out_x += 15000;
                out_y += 15000;

                printf ("Dvs6000 Me1 FaderPositionerAnalog Positioner %ld %ld\n", out
_x, out_y);
                fflush (stdout);

                /* 返答が来るまで待つ */
                if (ReceiveSocket(gClient,receive_packet,PKTSIZE) == -1) {
                    perror ("ReceiveSocket");
                    exit (-1);
                }
            }
        }
    }
}


void SignalHandler (int sig)
{
    fprintf (stderr, "socket convert : catch the signal <%d>.\n", sig);
    printf ("Quit\n");
    fflush (stdout);
    CloseSocket (gClient);
    exit (sig);
}
```

```
#include <stdlib.h>
#include <string.h>
#include <ulocks.h>
#include <fm.h>
#include <scsivideo.h>


#define XSIZE   720
#define YSIZE   486
#define FIELD   XSIZE*YSIZE
#define START   0
#define USAGE "%s: <output filename> <flame number>\n"
#define DEBUG
#undef DEBUG


/*******************/
/* グローバルデータ*/
/*******************/
char *_progName;                        /* 起動プログラム名 */
char *outfilename;                      /* 画像の出力ファイル名*/

/*********************/
/* プロトタイプ宣言 */
/*********************/
int sv_transfer (sv_handle*, char*, int, int);
void yuv2rgb (unsigned char*);
unsigned char *rgb2yuv (unsigned char*);



void main (int argc, char *argv[])
{
    int res;
    sv_handle *sv;
    sv_info info;
    unsigned char buffer[2*XSIZE*YSIZE];
    int number;                         /* キャプチャするフレーム番号 */
    int old_frame;
    int old_sync_mode;

    _progName = argv[0];

    if (argc!=3) {
        fprintf (stderr, USAGE, _progName);
        exit (1);
    }

    outfilename = argv[1];
    number = atoi (argv[2]);

    /* 入手するフレーム番号を尋ねる */
/*    printf ("capture flame number : ");
    scanf ("%d", &number);
*/
    /* ProntoVideoへの接続 */
    if ((sv=sv_open (""))==NULL) {
        fprintf (stderr, "Failure to connect to video device.\n");
        exit (1);
    }
#ifdef DEBUG
    printf ("Device Open correctly.\n");
#endif

    /* 現在のフレーム番号を入手 */
```



```
    if ((res=sv_status(sv,&info))!=SV_OK) {
        fprintf (stderr, "Error  getting present setting.\n");
        sv_errorprint (sv,res);
    }
    old_frame = info.video.position;
    old_sync_mode = info.sync;

    /* ビデオモードの設定 */
    if((res=sv_videomode(sv,SV_MODE_NTSC)) != SV_OK) {
        fprintf (stderr, "Error setting 29.97MHz operation.\n");
        sv_errorprint (sv, res);
    }

    /* 1フレームを取り出す */
    sv_transfer (sv, buffer, sizeof(buffer), number);

    /* 元の sync モードへ戻す (why change sync mode ?) */
    if ((res=sv_sync(sv,old_sync_mode))!=SV_OK) {
        fprintf (stderr, "Error setting genlock_analog sync mode.\n");
        sv_errorprint (sv,res);
    }

    /* 元のフレーム位置へ戻す */
    if ((res=sv_goto(sv,old_frame)) != SV_OK) {
        fprintf (stderr, "Error sv doesn't go old frame position.\n");
        sv_errorprint (sv, res);
    }

    /* 接続を終了する */
    if ((res=sv_close(sv)) != SV_OK) {
        fprintf (stderr, "Error closing video device./n");
        sv_errorprint (sv, res);
    }

    /* YUVからRGBへの変換 */
    printf ("capture finished.\nyub->rgb transfer start...\n");
    printf ("output filename : %s\n", outfilename);
    yuv2rgb(buffer);
}



/*
 * 画像の取り込み
 * <input>
 *  sv : sv_handle へのポインタ
 *  buffer : 取り込むバッファ
 *  size : 取り込む画像の大きさ
 *  number : 取り込むフレーム番号
 */
int sv_transfer (sv_handle *sv, char *buffer, int size, int number)
{
    int res;

    res = sv_goto (sv, number);
#ifdef DEBUG
    printf ("sv_goto res : %d\n", res);
#endif
    res = sv_sv2host (sv, buffer ,size, XSIZE, YSIZE, number, 1,
                      SV_TYPE_YUV422 || SV_DATASIZE_8BIT);
    if (res!=SV_OK) {
        printf ("data transfer failed.\n");
        sv_errorprint (sv, res);
    }
}
```

```
/*
 * YUVフォーマットからRGBフォーマットへの変換
 * <input>
 *  buffer : 変換するデータへのポインタ
 *
 *      R  = 1.164 * (Y - 16) + 1.596 * (Cr - 128)
 *      G  = 1.164 * (Y - 16) - 0.813 * (Cr - 128) - 0.391 * (Cb - 128)
 *      B  = 1.164 * (Y - 16)                      + 2.018 * (Cb - 128)
 *
 *  coded by Mr.Matusita (matsu@mmip.tutics.tut.ac.jp)
 */
void yuv2rgb (unsigned char buffer[2*XSIZE*YSIZE])
{
    FILE *fp;
    int i, j;
    long k;
    int calc;
    unsigned char rgbBuffer[YSIZE][XSIZE][3];
    unsigned char r, g, b;

    fp = fopen (outfilename, "wb");
    fprintf (fp, "P6\n");
    fprintf (fp, "%d %d\n", XSIZE, YSIZE);
    fprintf (fp, "255\n");

    for (i=0;i<YSIZE;i++) {
        for (j=0;j<XSIZE;j+=2) {
            k = i*2*XSIZE+j*2;

            calc = 1.164*(buffer[k+1]-16) + 1.596*(buffer[k+2]-128);
            if(calc>255)  rgbBuffer[i][j][0] = 255;
            else if(calc<0)    rgbBuffer[i][j][0] = 0;
            else               rgbBuffer[i][j][0] = calc;
            calc = 1.164*(buffer[k+1]-16) - 0.813*(buffer[k+2]-128)
                - 0.391*(buffer[k]-128);
            if(calc>255)  rgbBuffer[i][j][1] = 255;
            else if(calc<0)      rgbBuffer[i][j][1] = 0;
            else               rgbBuffer[i][j][1] = calc;
            calc = 1.164*(buffer[k+1]-16) + 2.018*(buffer[k]-128);
            if(calc>255)  rgbBuffer[i][j][2] = 255;
            else if(calc<0)    rgbBuffer[i][j][2] = 0;
            else               rgbBuffer[i][j][2] = calc;

            calc = 1.164*(buffer[k+3]-16) + 1.596*(buffer[k+2]-128);
            if(calc>255)  rgbBuffer[i][j+1][0] = 255;
            else if(calc<0)    rgbBuffer[i][j+1][0] = 0;
            else               rgbBuffer[i][j+1][0] = calc;
            calc = 1.164*(buffer[k+3]-16) - 0.813*(buffer[k+2]-128)
                - 0.391*(buffer[k]-128);
            if(calc>255)  rgbBuffer[i][j+1][1] = 255;
            else if(calc<0)    rgbBuffer[i][j+1][1] = 0;
            else               rgbBuffer[i][j+1][1] = calc;
            calc = 1.164*(buffer[k+3]-16) + 2.018*(buffer[k]-128);
            if(calc>255)  rgbBuffer[i][j+1][2] = 255;
            else if(calc<0)    rgbBuffer[i][j+1][2] = 0;
            else               rgbBuffer[i][j+1][2] = calc;
        }
    }

    for (i=0;i<YSIZE/2;i++) {
        fwrite (rgbBuffer[i+YSIZE/2], 3*XSIZE, 1, fp);
        fwrite (rgbBuffer[i], 3*XSIZE, 1, fp);
    }
    fclose (fp);
```



```
}



/*
 * RGBフォーマットからYUVフォーマットへの変換
 * <input>
 *  buffer : 変換するデータへのポインタ
 *
 *      Y  = 0.3R + 0.59G + 0.11B
 *      Cr = 0.5R - 0.42G - 0.08B
 *      Cb =-0.17R- 0.33G + 0.5B
 */
unsigned char *rgb2yuv (unsigned char buffer[3*XSIZE*YSIZE])
{
    long i, j, k;
    unsigned char *yuvBuffer;

    if ((yuvBuffer=malloc(2*XSIZE*YSIZE*sizeof(char)))==NULL) {
        fprintf; (stderr, "yuvBuffer malloc error.\n");
        exit (1);
    }
    for (i=0L;i<XSIZE*YSIZE;i+=2L) {
        j = 2*i;
        k = 3*i;
        *(yuvBuffer+j+1) =
            0.299*buffer[k] + 0.587*buffer[k+1] + 0.114*buffer[k+2];
        *(yuvBuffer+j+2) = buffer[k] - *(yuvBuffer+j+1);
        *(yuvBuffer+j) = buffer[k+2] - *(yuvBuffer+j+1);
        *(yuvBuffer+j+3) =
            0.299*buffer[k+3] + 0.587*buffer[k+4] + 0.114*buffer[k+5];
    }
    return yuvBuffer;
}
```

```
/*
 *                              svWriter.c
 *              ProntoVideoにppmファイルを書き込むプログラム
 *          Usage : svWriter <input filename> <output FrameNumber>
 */

#include <stdlib.h>
#include <unistd.h>
#include <ulocks.h>
#include <fm.h>
#include <scsivideo.h>


#define XSIZE  720
#define YSIZE  486
#define USAGE  "%s: <filename> <output frame number>\n"
#define DEBUG



/*******************/
/* グローバルデータ*/
/*******************/
char        *gProgName;                     /* 起動プログラム名 */
char        *gBuffer;                       /* YUV形式の画像バッファ */
char        *gInputFileName;                /* 画像の出力ファイル名 */
int         gFrameNumber;                   /* 書き込むフレーム番号 */
sv_handle   *gSVHandle;                     /* sv関係の変数 */
sv_info     gSVInfo;



/*********************/
/* プロトタイプ宣言 */
/*********************/
void InitApplication (int, char**);
void ReadFileWithConvert (void);
void InitProntoVideo (void);
void SVTransfer (void);
void EndApplication (void);
void FMErrorExit (int, char*, ff_rec*, ff_rec*);
void SVErrorExit (int, int, char*);



void main (int argc, char *argv[])
{
    InitApplication (argc, argv);
    InitProntoVideo ();

    SVTransfer ();      /* 1フレームを出力する */

    EndApplication ();
}



/*
 * 引数の解析とYUVデータの作成
 *<input>
 *  argc : 引数の数+1
 *  argv : 引数の値
 *
 */
void InitApplication (int argc, char *argv[])
```



```
{
    gProgName = argv[0];
    if (argc!=3) {
        fprintf (stderr, USAGE, gProgName);
        exit (1);
    }
    gInputFileName = argv[1];
    gFrameNumber = atoi (argv[2]);

    ReadFileWithConvert ();   /* PPMファイルを読み込み、YUVに変換する */
}



/*
 *  PPMの読み込みとYUVへの変換 (with FM)
 */
void ReadFileWithConvert (void)
{
    int i;
    char *tmpBuffer, *yBuffer, *crBuffer, *cbBuffer;
    ff_rec *in_ff, *out_ff;

    /* FileManager使用の準備 */
    if (fm_initialize() != FM_OK)
        FMErrorExit (0, "FM initialize error.\n", in_ff, out_ff);
    if ((in_ff=fm_fileformat_allocate("ppm"))==NULL)
        FMErrorExit (0, "PPM initialzie error.\n", in_ff, out_ff);
    if ((out_ff=fm_fileformat_allocate("dvsyuv"))==NULL)
        FMErrorExit (1, "YUV initialize error.\n", in_ff, out_ff);

    /* 変換元ファイルからデータを読み出し、変換する */
    if(fm_open(in_ff,gInputFileName) != FM_OK)
        FMErrorExit (2, "fm_open error.\n", in_ff, out_ff);
    if(fm_create(out_ff, "/tmp/tmp.lum", in_ff) != FM_OK)
        FMErrorExit (3, "fm_create error.\n", in_ff, out_ff);
    if(fm_read(in_ff,0) != FM_OK)
        FMErrorExit (4, "fm_read error.\n", in_ff, out_ff);
    if(fm_convert(in_ff,out_ff) != FM_OK)
        FMErrorExit (4, "fm_convert error.\n", in_ff, out_ff);

    /* 読み出したデータを保存し、FileManagerのメモリを解放する */
    if ((gBuffer=malloc(4*XSIZE*YSIZE))==NULL)
        FMErrorExit (4, "malloc error.\n", in_ff, out_ff);
    tmpBuffer = gBuffer + 2*XSIZE*YSIZE;

    /* Y,Cr, Cbデータの調整 */
    yBuffer   = (out_ff->buffer);
    cbBuffer  = (out_ff->buffer) + XSIZE*YSIZE;
    crBuffer  = (out_ff->buffer) + XSIZE*YSIZE + XSIZE*YSIZE/2;
    for (i=0;i<XSIZE*YSIZE/2;i++) {
        *(tmpBuffer++) = *(cbBuffer++);
        *(tmpBuffer++) = *(yBuffer++);
        *(tmpBuffer++) = *(crBuffer++);
        *(tmpBuffer++) = *(yBuffer++);
    }
    tmpBuffer -= 2*XSIZE*YSIZE;

    /* ProntoVideo形式に変換 (フィールドデータ毎にまとめる)  */
    for (i=0;i<YSIZE/2;i++) {
        memcpy (gBuffer+XSIZE*YSIZE, tmpBuffer, 2*XSIZE);
        tmpBuffer += 2*XSIZE;
        memcpy (gBuffer, tmpBuffer, 2*XSIZE);
        tmpBuffer += 2*XSIZE;
        gBuffer += 2*XSIZE;
    }
```

```
    gBuffer -= XSIZE*YSIZE;

    fm_close (in_ff);
    fm_close (out_ff);
    fm_fileformat_free (in_ff);
    fm_fileformat_free (out_ff);
    fm_deinitialize ();
    unlink ("/tmp/tmp.bmy");
    unlink ("/tmp/tmp.lum");
    unlink ("/tmp/tmp.rmy");
}



/*
 *  ProntoVideoの初期化
 */
void InitProntoVideo (void)
{
    int res;

    /* ProntoVideoへの接続 */
    if ((gSVHandle=sv_open (""))==NULL)
        SVErrorExit (0, res, "Failure to connect to Pronto Video device.\n");
    else
        fprintf (stderr, "Device Open correctly.\n");

    /* 現在のフレーム番号を入手 */
    if ((res=sv_status(gSVHandle,&gSVInfo))!=SV_OK)
        SVErrorExit (1, res, "Error getting present setting.\n");

    /* ビデオモードの設定 */
    if((res=sv_videomode(gSVHandle,SV_MODE_NTSC)) != SV_OK)
        SVErrorExit (1, res, "Error setting 29.97MHz operation.\n");
}



/*
 *  ProntoVideoへの出力
 */
void SVTransfer (void)
{
    int res;

    res = sv_host2sv (gSVHandle, gBuffer ,2*XSIZE*YSIZE, XSIZE, YSIZE,
                      gFrameNumber, 1, SV_TYPE_YUV422 || SV_DATASIZE_8BIT);
    if (res!=SV_OK)
        SVErrorExit (2, res, "data transfer failed.\n");
}



/*
 *  ProntoVideoを元の状態に戻し、バッファを解放する
 */
void EndApplication (void)
{
    int res;

    /* 元の sync モードへ戻す (why change sync mode ?) */
    if ((res=sv_sync(gSVHandle,gSVInfo.sync))!=SV_OK)
        SVErrorExit (3, res, "Error settin original sync mode.\n");

    /* 元のフレーム位置へ戻す */
    if ((res=sv_goto(gSVHandle,gSVInfo.video.position)) != SV_OK)
```



```
        SVErrorExit (1, res, "Error sv doesn't go old frame position.\n");

    /* 接続を終了する */
    if ((res=sv_close(gSVHandle)) != SV_OK)
        SVErrorExit (0, res, "Error closing video device.\n");

    free (gBuffer);
}



/*
 *  使用する画像フォーマットの宣言 (fm_initialize時に呼ばれる)
 */
void fm_config (void)
{
    fm_ppm_register ();             /*  Input側 : PPM */
    fm_dvs_register ();             /* Output側 : YUV */
}



/*
 *  FMによるエラー終了
 * <input>
 *  num     : どこまでFMが進んだか？
 *  message : 出力エラーメッセージへのポインタ
 *  in      : Input側のFM-format
 *  out     : Output側のFM-format
 */
void FMErrorExit (int num, char *message, ff_rec *in, ff_rec *out)
{
    fprintf (stderr, message);
    if(num>=4)  fm_close (out);
    if(num>=3)  fm_close (in);
    if(num>=2)  fm_fileformat_free (out);
    if(num>=1)  fm_fileformat_free (in);
    fm_deinitialize ();
    exit (1);
}



/*
 *  SVによるエラー終了
 * <input>
 *  num     : どこまでSVが進んだか？
 *  res     : SVのエラーメッセージ番号
 *  message : 出力エラーメッセージへのポインタ
 *
 */
void SVErrorExit (int num, int res, char *message)
{
    fprintf (stderr, message);
    sv_errorprint (gSVHandle, res);

    if (num>=3)  sv_goto (gSVHandle, gSVInfo.video.position);
    if (num==2)  EndApplication ();
    if (num>=1)  sv_close (gSVHandle);

    free (gBuffer);
    exit (1);
}
```

```
/*
 *                          get_line.c
 *             Sirius Video から 1ラインを取り出すプログラム
 *               Usage : get_line <line number> <line offset>
 *                                <line number> is 0 to 484
 *                                <line offset> is 0 to 719
 *
 *                   Warning : Don't Add Optimize Flag!!
 *
 *                     To exit application is 'Ctrl-C'.
 *                  This program hook the 'Ctrl-C' Vector.
 */

#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <gl/gl.h>
#include <gl/image.h>
#include <dmedia/vl.h>
#include <dmedia/vl_sirius.h>


#define MAXNAMLEN  1024
#define USAGE      "%s: <line number> <line offset>\n"
#define IMAGECOUNT 10000                /* 取り込むフィールドの数 */
#define BUFNUM     10                   /* 作成するバッファの数 */
#define DEBUG


/********************/
/* グローバルデータ */
/********************/
char *deviceName;                       /* 使用するデバイスの名前(Sirius) */
char *_progName;                        /* プログラムの名前 */
char *oldBuffer;                        /* 1フィールド前のデータバッファ */
int fields;                             /* 取り込んだ画像の数 */
VLServer svr;                           /* ビデオサーバー */
VLBuffer buffer;                        /* キャプチャバッファ */
VLPath path;                            /* ビデオパス */
VLNode src, drn;                        /* ノード */
long fieldSize;                         /* 1フレームあたりのバッファサイズ */
int f1_is_first;                        /* どちらのフィールドを先に取り込むか */
int line_is_f1;                         /* 希望するデータを含むのはどちらか？ */
int xsize, ysize;                       /* フィールドあたりの画面の大きさ */
int get_line;                           /* 取り込むライン */
int get_pixel;                          /* 取り込む画素 */
long data_ofs;                          /* 取り込むラインのオフセット */


/********************/
/* プロトタイプ宣言 */
/********************/
void Initialize (int, char**);
void CommunicateDaemon (void);
void ProcessEvent (VLServer, VLEvent*, void*);
void DumpImage (unsigned char*);
void docleanup (int);
void SignalHandler (int);
```



```
void main (int argc, char *argv[])
{
    Initialize (argc, argv);
    CommunicateDaemon ();

    /* ユーザが終了を指示するまでループする(今は回数制限あり) */
    vlMainLoop ();
}



/*
 * アプリケーションの初期化を行う
 * <input>
 *  argc : 引数の数
 *  argv : 引数
 */
void Initialize (int argc, char *argv[])
{
    int i;

    fields = 0;
    _progName = argv[0];

    if (argc!=3) {
        fprintf (stderr, USAGE, _progName);
        exit (1);
    }

    /* 取り込むライン番号の変換( > -> >= ??) */
    get_line = atoi (argv[1]);
    if (get_line<0 || get_line>485) {
        fprintf (stderr, "%s: <get_line> is 0 to 485.\n", _progName);
        exit (1);
    }
    /* ラインのどの画素を抽出するか？ */
    get_pixel = atoi (argv[2]);
    if (get_pixel<0 || get_pixel>=720) {
        fprintf (stderr, "%s: <line offset> is 0 to 719.\n", _progName);
        exit (1);
    }

    /* 前フィールド用のバッファの確保 */
    if ((oldBuffer=malloc(2*243*720))==NULL) {
        fprintf (stderr, "%s: malloc Error.\n", _progName);
        exit (1);
    }

    /* シグナルのフック */
    for(i=1;i<=32;i++)  signal(i,SignalHandler);
}



void CommunicateDaemon (void)
{
    VLDevList             devlist;
    VLControlValue        val;
    VLTransferDescriptor  xferDesc;
    int                   i, timing, devicenum;

    /* デーモンへ接続 */
    if ((svr=vlOpenVideo(""))==NULL) {
        fprintf (stderr, "%s: couldn't open video\n", _progName);
```

```
        exit (1);
    }
    else
        printf ("Connected Server Daemon.\n");

    /* ノードの設定 */
    src = vlGetNode (svr, VL_SRC, VL_VIDEO, SIR_SRC_DIGITAL_VIDEO_1);
    drn = vlGetNode (svr, VL_DRN, VL_MEM, VL_ANY);

    if ((path=vlCreatePath(svr,VL_ANY,src,drn)) < 0) {
        vlPerror (_progName);
        vlCloseVideo (svr);
        exit (1);
    }

    /* ハードの設定とパス仕様の定義 */
    if (vlSetupPaths(svr, (VLPathList)&path, 1, VL_SHARE, VL_SHARE)!=0) {
        vlPerror (_progName);
        vlCloseVideo (svr);
        exit (1);
    }

    /* 受信したいイベントを設定 */
    vlSelectEvents (svr, path,
                    VLTransferCompleteMask | VLStreamPreemptedMask |
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLTransferFailedMask | VLSequenceLostMask);

    /* タイミングをディジタルNTSCに設定(src) */
    val.intVal = VL_TIMING_525_CCIR601;
    if ((vlSetControl(svr,path,src,VL_TIMING,&val)) != 0) {
        vlPerror ("set timing failed");
        vlCloseVideo (svr);
        exit (1);
    }

    /* ビデオ出力フォーマットの指定(drn) */
    val.intVal = VL_FORMAT_SMPTE_YUV;
    vlSetControl (svr, path, drn, VL_FORMAT, &val);

    /* タイミングの設定(drn) */
    if (vlGetControl(svr,path,src,VL_TIMING,&val) == 0) {
        if ((vlSetControl(svr,path,drn,VL_TIMING,&val)!=0)
            && (vlErrno!=VLBadControl)) {
            vlPerror ("SetControl Failed");
            vlCloseVideo (svr);
            exit (1);
        }
    }
    else {
        vlPerror ("get timing failed");
        vlCloseVideo (svr);
        exit (1);
    }

    /* 取り込むフレームのタイプを指定(drn) */
    val.intVal = VL_CAPTURE_INTERLEAVED;
    vlSetControl (svr, path, drn, VL_CAP_TYPE, &val);

    /* ピクセルをYUVで保存するための設定(drn) */
    val.intVal = VL_PACKING_YVYU_422_8;
    vlSetControl (svr, path, drn, VL_PACKING, &val);

    /* 最初に取り込まれるフィールドを知る(src) */
    if (vlGetControl(svr,path,src,VL_SIR_FIELD_DOMINANCE,&val) == 0) {
        timing = ((val.intVal == VL_TIMING_525_SQ_PIX) ||
```



```
                  (val.intVal == VL_TIMING_525_CCIR601));
        switch (val.intVal) {
          case SIR_F1_IS_DOMINANT:
            if(timing)  f1_is_first = FALSE;
            else        f1_is_first = TRUE;
            break;
          case SIR_F2_IS_DOMINANT:
            if (timing)  f1_is_first = TRUE;
            else         f1_is_first = FALSE;
            break;
        }
    }

    /* ビデオサイズとそのバイト数を得る */
    vlGetControl (svr, path, drn, VL_SIZE, &val);
    xsize= val.xyVal.x;
    ysize = val.xyVal.y;
    fieldSize = vlGetTransferSize (svr, path);
#ifdef DEBUG
    printf ("for debug: fieldSize is %ld\n", fieldSize);
#endif

    /* データ位置と どちらのフィールドにデータがあるかを特定する */
    line_is_f1 = (get_line%2==0);
    data_ofs = 2*xsize*(get_line/2) + get_pixel*2;
#ifdef DEBUG
    printf ("get_line is %d get_pixel is %d\n", get_line, get_pixel);
    printf ("line data is %d %d\n", line_is_f1, data_ofs);
    printf ("x,y is %d %d\n", xsize, ysize);
#endif

    /* 転送のためのコールバックルーチンを設定 */
    vlAddCallback (svr, path,
                   VLTransferCompleteMask | VLStreamPreemptedMask |
                   VLStreamStartedMask | VLStreamStoppedMask |
                   VLSequenceLostMask | VLTransferFailedMask,
                   ProcessEvent, NULL);

    /* BUFNUM分のフレームバッファの作成と登録 */
    buffer = vlCreateBuffer (svr, path, drn, BUFNUM);
    if (buffer==NULL) {
        vlPerror (_progName);
        vlCloseVideo (svr);
        exit (1);
    }

    if (vlRegisterBuffer(svr, path, drn, buffer)!=0) {
        vlPerror (_progName);
        vlDestroyBuffer (svr, buffer);
        vlCloseVideo (svr);
        exit (1);
    }

    xferDesc.mode = VL_TRANSFER_MODE_CONTINUOUS;
    xferDesc.count = IMAGECOUNT;
    xferDesc.delay = 0;             /* VLTriggerImmediate なら無視される */
    xferDesc.trigger = VLTriggerImmediate;

    /* データ転送の開始 */
    if (vlBeginTransfer(svr,path,1,&xferDesc)) {
        vlPerror (_progName);
        vlDestroyBuffer (svr, buffer);
        vlCloseVideo (svr);
        exit (1);
    }
}
```

```
/*
 *  ビデオライブラリイベントを処理する
 * <input>
 *  svr : VLServerへのポインタ
 *  ev : 何のイベントで呼ばれたか？
 *  data : ユーザーが使用すると宣言したデータへのポインタ
 */
void ProcessEvent (VLServer svr, VLEvent *ev, void *data)
{
    VLInfoPtr info;
    char      *dataPtr;

    switch (ev->reason) {
      case VLStreamStarted:
        break;
      case VLStreamStopped:
      case VLTransferComplete:
        /* バッファ内のフィールドを読み出す */
        while (info=vlGetNextValid(svr,buffer)) {
            dataPtr = vlGetActiveRegion (svr, buffer, info);
            if (fields < IMAGECOUNT) {
                /* フレームデータを書き出す */
                DumpImage (dataPtr);
                fields++;
            }
            vlPutFree (svr, buffer);
        }
        if (fields == IMAGECOUNT)
            docleanup (0);
        break;
      case VLStreamPreempted:
        fprintf (stderr, "%s: Path for this stream preempted.\n", _progName);
        docleanup (1);
        break;
      case VLSequenceLost:
        fprintf (stderr, "%s: Sequence Lost.\n", _progName);
        docleanup (1);
        break;
      case VLTransferFailed:
        fprintf (stderr, "%s: Transfer failed.\n", _progName);
        docleanup (1);
        break;
      default:
        break;
    }
}



/*
 *  データ構造の変換と書き出し
 * <input>
 *  data : YUVフォーマットのデータへのポインタ
 * <warning>
 *  フィールド２が画像の１ライン目でフィールド１が２ライン目です。
 */
void DumpImage (unsigned char *data)
{
    int i;

    if (f1_is_first==TRUE) { /* fields=0,2,4,6...  line=0,2,4,6 */
        if (line_is_f1==TRUE) { /*get line = 0,2,4,6... */
            if (fields%2==0)
                for (i=0;i<4;i++)
                    printf ("%x ", *(data+data_ofs+i));
        }
        else { /* get line = 0,2,4,6... */
            if (fields%2==1)
                for (i=0;i<4;i++)
                    printf ("%x ", *(data+data_ofs+i));
        }
    }
    else {
        if (line_is_f1==TRUE) {
            if (fields%2==1)
                for (i=0;i<4;i++)
                    printf ("%x ", *(data+data_ofs+i));
        }
        else {
            if (fields%2==0)
                for (i=0;i<4;i++)
                    printf ("%x ", *(data+data_ofs+i));
        }
    }
    printf ("\n");
}
```



```
/*
 *  終了前のクリーンアップを行う。
 *  <input>
 *    ret : 終了コード
 */
void docleanup (int ret)
{
    vlEndTransfer (svr, path);
    vlDeregisterBuffer (svr, path, drn, buffer);
    vlDestroyBuffer (svr, buffer);
    vlDestroyPath (svr, path);
    vlCloseVideo (svr);
    if(oldBuffer!=NULL)  free (oldBuffer);
    exit (ret);
}



/*
 * シグナルを受け取るフックルーチン
 * <input>
 *  sig : 受信したシグナル番号
 */
void SignalHandler (int sig)
{
    fprintf (stderr, "%s: catch the signal <%d>.\n", _progName, sig);
    fprintf (stderr, "%d fields transferred.\n", fields);
    docleanup(sig);
}
```

```
/*
 *                              get_vl.c
 *         Sirius Videoから画像をキャプチャし、ppmに保存するプログラム
 *                      Usage : get_vl <base filename>
 *          Created filename will be <basefilename>-<number>.ppm
 *
 */

#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <gl/gl.h>
#include <gl/image.h>
#include <dmedia/vl.h>
#include <dmedia/vl_sirius.h>


#define MAXNAMLEN 1024
#define GRABFILE  "out"
#define USAGE     "%s: <output filename>\n"
#define DEBUG


/********************/
/* グローバルデータ */
/********************/
char *deviceName;                       /* 使用するデバイスの名前(Sirius) */
char *_progName;                        /* プログラムの名前 */
char *oldBuffer;                        /* 一つ前のキャプチャ画像 */
char iname[MAXNAMLEN];                  /* 出力ファイル名 */
char iname_tmp[MAXNAMLEN];              /* テンポラリファイル名 */
char *outfilename;                      /* ベース出力ファイル名 */
int imageCount = 2;                     /* キャプチャする画像の数 */
int writeFrame = TRUE;                  /* ファイルに出力するか？(yes) */
int displayFrame = FALSE;               /* 画面に出力するか？(no) */
VLServer svr;                           /* ビデオサーバー */
VLBuffer buffer;                        /* キャプチャバッファ */
int fields = 0;                         /* 今までに取り込まれた画像の数 */
VLPath path;                            /* ビデオパス */
VLNode src, drn, dev;                   /* ノード */
long fieldSize;                         /* 1フレームあたりのバッファサイズ */
int fl_is_first;                        /* どちらのフィールドを先に取り込むか */
int xsize, ysize;                       /* フィールドあたりの画面の大きさ */


/********************/
/* プロトタイプ宣言 */
/********************/
void ProcessEvent (VLServer, VLEvent*, void*);
void DumpImage (char*);
void docleanup (int);
void Compare (int, int, unsigned char*, unsigned char*);
void SignalHandler (int);


void main (int argc, char *argv[])
{
    VLDevList           devlist;
    VLControlValue      val;
```



```
    VLTransferDescriptor  xferDesc;
    int i, timing, devicenum;

    _progName = argv[0];

    if (argc!=2) {
        fprintf (stderr, USAGE, _progName);
        exit (1);
    }
    outfilename = argv[1];

    /* シグナルのフック */
    for(i=1;i<=32;i++)  signal(i,SignalHandler);

    /* デーモンへ接続 */
    if ((svr=vlOpenVideo(""))==NULL) {
        fprintf (stderr, "%s: couldn't open video\n", _progName);
        exit (1);
    }
    else
        printf ("Connected Server.\n");

    /* デーモンがサポートするデバイスリストを得る(no use) */
    if (vlGetDeviceList(svr,&devlist)!=VLSuccess) {
        fprintf (stderr, "%s: getting device list: %s\n",
                 _progName, vlStrError(vlErrno));
        exit (1);
    }
#ifdef DEBUG
    printf ("for debug: devicenum is %d\n", devlist.numDevices);
#endif

    /* ノードの設定 */
    src = vlGetNode (svr, VL_SRC, VL_VIDEO, SIR_SRC_DIGITAL_VIDEO_1);
    drn = vlGetNode (svr, VL_DRN, VL_MEM, VL_ANY);
    dev = vlGetNode (svr, VL_DEVICE, 0, 0);

    if ((path=vlCreatePath(svr,VL_ANY,src,drn)) < 0) {
        vlPerror (_progName);
        exit (1);
    }
    vlAddNode (svr, path, dev);

#ifdef DEBUG
    /* devlist.devices[devicenum]... は厳密には違う。
       しかし、miris37では、使用できるデバイスが１つだけなので、
       これでも大丈夫。また、Siriusのプログラムが一度にmallocを行い、
       連続リストになっているのなら、これも大丈夫。 */
    devicenum = vlGetDevice (svr,path);
    deviceName = devlist.devices[devicenum].name;
    printf("for debug: devicenum is %d, deviceName is %s\n",
           devicenum, deviceName);
    printf ("VIDEO->MEMORY NODE IDs:\n");
    printf ("video source = %d\n", src);
    printf ("memory drain = %d\n", drn);
    printf ("VIDEO TO MEMORY PATH ID = %d\n", path);
#endif

    /* ハードの設定とパス仕様の定義 */
    if (vlSetupPaths(svr, (VLPathList)&path, 1, VL_SHARE, VL_SHARE)!=0) {
        vlPerror (_progName);
        exit (1);
    }

    /* 受信したいイベントを設定 */
    vlSelectEvents (svr, path,
```

```
                    VLTransferCompleteMask | VLStreamPreemptedMask |
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLTransferFailedMask | VLSequenceLostMask);

    /* タイミングをディジタルNTSCに設定(src) */
    val.intVal = VL_TIMING_525_CCIR601;
    if ((vlSetControl(svr,path,src,VL_TIMING,&val)) != 0) {
        vlPerror ("set timing failed");
        vlCloseVideo (svr);
        exit (1);
    }

    /* ビデオ出力フォーマットの指定(drn) */
    val.intVal = VL_FORMAT_RGB;
    vlSetControl (svr, path, drn, VL_FORMAT, &val);

    /* タイミングの設定(drn) */
    if (vlGetControl(svr,path,src,VL_TIMING,&val) == 0) {
        if ((vlSetControl(svr,path,drn,VL_TIMING,&val)!=0)
            && (vlErrno!=VLBadControl)) {
            vlPerror ("SetControl Failed");
            exit (1);
        }
    }
    else {
        vlPerror ("get timing failed");
        vlCloseVideo (svr);
        exit (1);
    }

    /* 取り込むフレームのタイプを指定(drn) */
    val.intVal = VL_CAPTURE_INTERLEAVED;
    vlSetControl (svr, path, drn, VL_CAP_TYPE, &val);

    /* ピクセルをRGBで保存するための設定(drn) */
    val.intVal = VL_PACKING_RGB_8;
    vlSetControl (svr, path, drn, VL_PACKING, &val);

    /* 最初に取り込まれるフィールドを知る(src) */
    if (vlGetControl(svr,path,src,VL_SIR_FIELD_DOMINANCE,&val) == 0) {
        timing = ((val.intVal == VL_TIMING_525_SQ_PIX) ||
                  (val.intVal == VL_TIMING_525_CCIR601));
        switch (val.intVal) {
          case SIR_F1_IS_DOMINANT:
            if(timing)  f1_is_first = FALSE;
            else        f1_is_first = TRUE;
            break;
          case SIR_F2_IS_DOMINANT:
            if (timing)  f1_is_first = TRUE;
            else         f1_is_first = FALSE;
            break;
        }
    }

    /* ビデオサイズとそのバイト数を得る */
    vlGetControl (svr, path, drn, VL_SIZE, &val);
    xsize= val.xyVal.x;
    ysize = val.xyVal.y;
    fieldSize = vlGetTransferSize (svr, path);
#ifdef DEBUG
    printf ("for debug: fieldSize is %ld\n", fieldSize);
#endif

    /* 転送のためのコールバックルーチンを設定 */
    vlAddCallback (svr, path,
                   VLTransferCompleteMask | VLStreamPreemptedMask |
```



```
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLSequenceLostMask | VLTransferFailedMask,
                    ProcessEvent, NULL);

    /* imageCount分のフレームバッファの作成と登録 */
    buffer = vlCreateBuffer (svr, path, drn, imageCount);
    if (buffer==NULL) {
        vlPerror (_progName);
        exit (1);
    }
    oldBuffer = malloc (fieldSize*sizeof(unsigned char));
    if (oldBuffer==NULL) {
        fprintf (stderr, "%s: malloc Error.\n", _progName);
        exit (1);
    }

    if (vlRegisterBuffer(svr, path, drn, buffer)!=0) {
        vlPerror (_progName);
        exit (1);
    }

    xferDesc.mode = VL_TRANSFER_MODE_DISCRETE;
    xferDesc.count = imageCount;
    xferDesc.delay = 0;             /* VLTriggerImmediate なら無視される */
    xferDesc.trigger = VLTriggerImmediate;

    /* データ転送の開始 */
    if (vlBeginTransfer(svr,path,1,&xferDesc)) {
        vlPerror (_progName);
        exit (1);
    }

    /* フレーム転送が終了するまでループする */
    vlMainLoop ();
}



/*
 *  ビデオライブラリイベントを処理する
 * <input>
 *  svr : VLServerへのポインタ
 *  ev : 何のイベントで呼ばれたか？
 *  data : ユーザーが使用すると宣言したデータへのポインタ
 */
void ProcessEvent (VLServer svr, VLEvent *ev, void *data)
{
    VLInfoPtr info;
    char      *dataPtr;

    switch (ev->reason) {
      case VLStreamStarted:
        break;
      case VLStreamStopped:
      case VLTransferComplete:
        /* バッファ内のフィールドを読み出す */
        while (info=vlGetNextValid(svr,buffer)) {
            dataPtr = vlGetActiveRegion (svr, buffer, info);
            if (fields < imageCount) {
/*              if (fields!=1)
                    Compare (xsize, ysize*2, dataPtr, oldBuffer);
                memcpy (oldBuffer, dataPtr, fieldSize);
*/
                /* フレームデータを書き出す */
                if (writeFrame==TRUE)
                    DumpImage (dataPtr);
```

```
                fields++;
                /* フレームをディスプレイに表示 */
                if (displayFrame==TRUE) {
                    char cmd[100];
                    sprintf (cmd, "ipaste %s", iname);
                    system (cmd);
                    if (writeFrame==FALSE)  unlink(iname);
                }
            }
            vlPutFree (svr, buffer);
        }
        if (fields == imageCount)
            docleanup (0);
        break;
      case VLStreamPreempted:
        fprintf (stderr, "%s: Path for this stream preempted.\n", _progName);
        docleanup (1);
        break;
      case VLSequenceLost:
        fprintf (stderr, "%s: Sequence Lost.\n", _progName);
        docleanup (1);
        break;
      case VLTransferFailed:
        fprintf (stderr, "%s: Transfer failed.\n", _progName);
        docleanup (1);
        break;
      default:
        break;
    }
}



/*
 *  データ構造の変換と書き出し
 * <input>
 *  data : VL_PACKING_RGB_8フォーマットのデータへのポインタ
 */
void DumpImage (char *data)
{
    FILE *image;
    unsigned char *rgbBuffer;
    unsigned char *beforefield = oldBuffer;
    int i, x, y;
    long k;

    if (fields%2==0) {
        memcpy (oldBuffer, data, fieldSize);
    }
    else {
        sprintf (iname, "%s-%05d.ppm", outfilename, (int)(fields/2));
        if (writeFrame==TRUE) {
            sprintf (iname_tmp, "%s.%d", iname, getpid());
            image = fopen (iname_tmp, "wb");
            if (image==NULL) {
                fprintf (stderr, "%s: can't create image file.\n",_progName);
                return;
            }
            fprintf (image, "P6\n");
            fprintf (image, "%d %d\n", xsize, ysize*2);
            fprintf (image, "255\n");
        }

        /* RGBデータとしてバッファに書き出す */
        rgbBuffer = malloc (2*3*xsize*ysize);
        k = 0;
```



```
        for (y=0;y<ysize;y++) {
            for (x=0;x<xsize;x++) {
                data++;
                *(rgbBuffer+k+2) = *(data++);
                *(rgbBuffer+k+1) = *(data++);
                *(rgbBuffer+k+0) = *(data++);
                k+=3;
            }
        }
        for (y=0;y<ysize;y++) {
            for (x=0;x<xsize;x++) {
                beforefield++;
                *(rgbBuffer+k+2) = *(beforefield++);
                *(rgbBuffer+k+1) = *(beforefield++);
                *(rgbBuffer+k+0) = *(beforefield++);
                k+=3;
            }
        }

        if (writeFrame==TRUE) {
            if (fl_is_first==TRUE) {
                for (i=0;i<ysize;i++) {
                    k = 3*i*xsize;
                    fwrite (rgbBuffer+k, 3*xsize, 1, image);
                    fwrite (rgbBuffer+k+3*xsize*ysize, 3*xsize, 1, image);
                }
            }
            else {
                for (i=0;i<ysize;i++) {
                    k = 3*i*xsize;
                    fwrite (rgbBuffer+k+3*xsize*ysize, 3*xsize, 1, image);
                    fwrite (rgbBuffer+k, 3*xsize, 1, image);
                }
            }
            fclose (image);
            if (rename(iname_tmp, iname) != 0) {
                fprintf (stderr, "Cannot rename tmp file to %s, errno=%d\n",
                         iname, errno);
                unlink (iname_tmp);
            }
        }
        fprintf (stderr, "%s: saved image to file %s\n",
                 _progName, iname);
    }
}



/*
 *  終了前のクリーンアップを行う。
 *  <input>
 *    ret : 終了コード
 */
void docleanup (int ret)
{
    vlEndTransfer (svr, path);
    vlDeregisterBuffer (svr, path, drn, buffer);
    vlDestroyBuffer (svr, buffer);
    vlDestroyPath (svr, path);
    vlCloseVideo (svr);
    if(oldBuffer!=NULL)  free(oldBuffer);
    exit (ret);
}
```

```
/*
 * VL_PACKING_RGB_8フォーマット上で比較を行う
 * <input>
 *   x,y : 画像のx,y方向の大きさ
 *   a,b : 比較する画像データへのポインタ
 */
void Compare (int x, int y, unsigned char *a, unsigned char *b)
{
    int i, j;

    printf ("( x , y ) :  R G B :  R G B\n");
    a++; b++;

    for (j=y-1;j>=0;j--) {
        for (i=0;i<x;i++) {
            if ((strncmp (a,b,3))!=0) {
/*              printf ("(%3d,%3d) : %2x%2x%2x : %2x%2x%2x\n",
                        i, j, *(a+2), *(a+1), *a, *(b+2), *(b+1), *b);
*/          }
            a+=4; b+=4;
        }
    }
}



/*
 * シグナルを受け取るフックルーチン
 * <input>
 *  sig : 受信したシグナル番号
 */
void SignalHandler (int sig)
{
    fprintf (stderr, "%s: catch the signal <%d>.\n", _progName, sig);
    fprintf (stderr, "%d fields transferred.\n", fields);
    docleanup(sig);
}
```

```
/*
 *                              get_vl.c
 *          Sirius Videoから画像をキャプチャし、ppmに保存するプログラム
 *                      Usage : get_vl <base filename>
 *           Created filename will be <basefilename>-<number>.ppm
 *
 */

#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <gl/gl.h>
#include <gl/image.h>
#include <dmedia/vl.h>
#include <dmedia/vl_sirius.h>


#define MAXNAMLEN 1024
#define GRABFILE  "out"
#define USAGE     "%s: <output filename>\n"
#define DEBUG


/********************/
/* グローバルデータ */
/********************/
char *deviceName;                       /* 使用するデバイスの名前(Sirius) */
char *_progName;                        /* プログラムの名前 */
char *oldBuffer;                        /* 一つ前のキャプチャ画像 */
char iname[MAXNAMLEN];                  /* 出力ファイル名 */
char iname_tmp[MAXNAMLEN];              /* テンポラリファイル名 */
char *outfilename;                      /* ベース出力ファイル名 */
int imageCount = 2;                     /* キャプチャする画像の数 */
int writeFrame = TRUE;                  /* ファイルに出力するか？(yes) */
int displayFrame = FALSE;               /* 画面に出力するか？(no) */
VLServer svr;                           /* ビデオサーバー */
VLBuffer buffer;                        /* キャプチャバッファ */
int fields = 0;                         /* 今までに取り込まれた画像の数 */
VLPath path;                            /* ビデオパス */
VLNode src, drn, dev;                   /* ノード */
long fieldSize;                         /* 1フレームあたりのバッファサイズ */
int f1_is_first;                        /* どちらのフィールドを先に取り込むか */
int xsize, ysize;                       /* フィールドあたりの画面の大きさ */


/********************/
/* プロトタイプ宣言 */
/********************/
void ProcessEvent (VLServer, VLEvent*, void*);
void DumpImage (char*);
void docleanup (int);
void Compare (int, int, unsigned char*, unsigned char*);
void SignalHandler (int);


void main (int argc, char *argv[])
{
    VLDevList           devlist;
    VLControlValue      val;
```



```
    VLTransferDescriptor  xferDesc;
    int i, timing, devicenum;

    _progName = argv[0];

    if (argc!=2) {
        fprintf (stderr, USAGE, _progName);
        exit (1);
    }
    outfilename = argv[1];

    /* シグナルのフック */
    for(i=1;i<=32;i++)  signal(i,SignalHandler);

    /* デーモンへ接続 */
    if ((svr=vlOpenVideo(""))==NULL) {
        fprintf (stderr, "%s: couldn't open video\n", _progName);
        exit (1);
    }
    else
        printf ("Connected Server.\n");

    /* デーモンがサポートするデバイスリストを得る(no use) */
    if (vlGetDeviceList(svr,&devlist)!=VLSuccess) {
        fprintf (stderr, "%s: getting device list: %s\n",
                 _progName, vlStrError(vlErrno));
        exit (1);
    }
#ifdef DEBUG
    printf ("for debug: devicenum is %d\n", devlist.numDevices);
#endif

    /* ノードの設定 */
    src = vlGetNode (svr, VL_SRC, VL_VIDEO, SIR_SRC_ANALOG_VIDEO);
    drn = vlGetNode (svr, VL_DRN, VL_MEM, VL_ANY);
    dev = vlGetNode (svr, VL_DEVICE, 0, 0);

    if ((path=vlCreatePath(svr,VL_ANY,src,drn)) < 0) {
        vlPerror (_progName);
        exit (1);
    }
    vlAddNode (svr, path, dev);

#ifdef DEBUG
    /* devlist.devices[devicenum]... は厳密には違う。
       しかし、miris37では、使用できるデバイスが1つだけなので、
       これでも大丈夫。また、Siriusのプログラムが一度にmallocを行い、
       連続リストになっているのなら、これも大丈夫。 */
    devicenum = vlGetDevice (svr,path);
    deviceName = devlist.devices[devicenum].name;
    printf("for debug: devicenum is %d, deviceName is %s\n",
           devicenum, deviceName);
    printf ("VIDEO->MEMORY NODE IDs:\n");
    printf ("video source = %d\n", src);
    printf ("memory drain = %d\n", drn);
    printf ("VIDEO TO MEMORY PATH ID = %d\n", path);
#endif

    /* ハードの設定とパス仕様の定義 */
    if (vlSetupPaths(svr, (VLPathList)&path, 1, VL_SHARE, VL_SHARE)!=0) {
        vlPerror (_progName);
        exit (1);
    }

    /* 受信したいイベントを設定 */
    vlSelectEvents (svr, path,
```

```
                    VLTransferCompleteMask | VLStreamPreemptedMask |
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLTransferFailedMask | VLSequenceLostMask);

    /* タイミングをディジタルNTSCに設定(src) */
    val.intVal = VL_TIMING_525_CCIR601;
    if ((vlSetControl(svr,path,src,VL_TIMING,&val)) != 0) {
       vlPerror ("set timing failed");
       vlCloseVideo (svr);
       exit (1);
    }

    /* ビデオ出力フォーマットの指定(drn) */
    val.intVal = VL_FORMAT_RGB;
    vlSetControl (svr, path, drn, VL_FORMAT, &val);

    /* タイミングの設定(drn) */
    if (vlGetControl(svr,path,src,VL_TIMING,&val) == 0) {
        if ((vlSetControl(svr,path,drn,VL_TIMING,&val)!=0)
            && (vlErrno!=VLBadControl)) {
            vlPerror ("SetControl Failed");
            exit (1);
        }
    }
    else {
        vlPerror ("get timing failed");
        vlCloseVideo (svr);
        exit (1);
    }

    /* 取り込むフレームのタイプを指定(drn) */
    val.intVal = VL_CAPTURE_INTERLEAVED;
    vlSetControl (svr, path, drn, VL_CAP_TYPE, &val);

    /* ピクセルをRGBで保存するための設定(drn) */
    val.intVal = VL_PACKING_RGB_8;
    vlSetControl (svr, path, drn, VL_PACKING, &val);

    /* 最初に取り込まれるフィールドを知る(src) */
    if (vlGetControl(svr,path,src,VL_SIR_FIELD_DOMINANCE,&val) == 0) {
        timing = ((val.intVal == VL_TIMING_525_SQ_PIX) ||
                  (val.intVal == VL_TIMING_525_CCIR601));
        switch (val.intVal) {
          case SIR_F1_IS_DOMINANT:
            if(timing)  f1_is_first = FALSE;
            else        f1_is_first = TRUE;
            break;
          case SIR_F2_IS_DOMINANT:
            if (timing)  f1_is_first = TRUE;
            else         f1_is_first = FALSE;
            break;
        }
    }

    /* ビデオサイズとそのバイト数を得る */
    vlGetControl (svr, path, drn, VL_SIZE, &val);
    xsize= val.xyVal.x;
    ysize = val.xyVal.y;
    fieldSize = vlGetTransferSize (svr, path);
#ifdef DEBUG
    printf ("for debug: fieldSize is %ld\n", fieldSize);
#endif

    /* 転送のためのコールバックルーチンを設定 */
    vlAddCallback (svr, path,
                   VLTransferCompleteMask | VLStreamPreemptedMask |
```



```
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLSequenceLostMask | VLTransferFailedMask,
                    ProcessEvent, NULL);

    /* imageCount分のフレームバッファの作成と登録 */
    buffer = vlCreateBuffer (svr, path, drn, imageCount);
    if (buffer==NULL) {
        vlPerror (_progName);
        exit (1);
    }
    oldBuffer = malloc (fieldSize*sizeof(unsigned char));
    if (oldBuffer==NULL) {
        fprintf (stderr, "%s: malloc Error.\n", _progName);
        exit (1);
    }

    if (vlRegisterBuffer(svr, path, drn, buffer)!=0) {
        vlPerror (_progName);
        exit (1);
    }

    xferDesc.mode = VL_TRANSFER_MODE_DISCRETE;
    xferDesc.count = imageCount;
    xferDesc.delay = 0;              /* VLTriggerImmediate なら無視される */
    xferDesc.trigger = VLTriggerImmediate;

    /* データ転送の開始 */
    if (vlBeginTransfer(svr,path,1,&xferDesc)) {
        vlPerror (_progName);
        exit (1);
    }

    /* フレーム転送が終了するまでループする */
    vlMainLoop ();
}



/*
 *  ビデオライブラリイベントを処理する
 * <input>
 *  svr : VLServerへのポインタ
 *  ev : 何のイベントで呼ばれたか？
 *  data : ユーザーが使用すると宣言したデータへのポインタ
 */
void ProcessEvent (VLServer svr, VLEvent *ev, void *data)
{
    VLInfoPtr info;
    char      *dataPtr;

    switch (ev->reason) {
      case VLStreamStarted:
        break;
      case VLStreamStopped:
      case VLTransferComplete:
        /* バッファ内のフィールドを読み出す */
        while (info=vlGetNextValid(svr,buffer)) {
            dataPtr = vlGetActiveRegion (svr, buffer, info);
            if (fields < imageCount) {
/*              if (fields!=1)
                    Compare (xsize, ysize*2, dataPtr, oldBuffer);
                memcpy (oldBuffer, dataPtr, fieldSize);
*/
                /* フレームデータを書き出す */
                if (writeFrame==TRUE)
                    DumpImage (dataPtr);
```

```
                fields++;
                /* フレームをディスプレイに表示 */
                if (displayFrame==TRUE) {
                    char cmd[100];
                    sprintf (cmd, "ipaste %s", iname);
                    system (cmd);
                    if (writeFrame==FALSE)  unlink(iname);
                }
            }
            vlPutFree (svr, buffer);
        }
        if (fields == imageCount)
            docleanup (0);
        break;
      case VLStreamPreempted:
        fprintf (stderr, "%s: Path for this stream preempted.\n", _progName);
        docleanup (1);
        break;
      case VLSequenceLost:
        fprintf (stderr, "%s: Sequence Lost.\n", _progName);
        docleanup (1);
        break;
      case VLTransferFailed:
        fprintf (stderr, "%s: Transfer failed.\n", _progName);
        docleanup (1);
        break;
      default:
        break;
    }
}



/*
 *  データ構造の変換と書き出し
 * <input>
 *  data : VL_PACKING_RGB_8フォーマットのデータへのポインタ
 */
void DumpImage (char *data)
{
    FILE *image;
    unsigned char *rgbBuffer;
    unsigned char *beforefield = oldBuffer;
    int i, x, y;
    long k;

    if (fields%2==0) {
        memcpy (oldBuffer, data, fieldSize);
    }
    else {
        sprintf (iname, "%s-%05d.ppm", outfilename, (int)(fields/2));
        if (writeFrame==TRUE) {
            sprintf (iname_tmp, "%s.%d", iname, getpid());
            image = fopen (iname_tmp, "wb");
            if (image==NULL) {
                fprintf (stderr, "%s: can't create image file.\n",_progName);
                return;
            }
            fprintf (image, "P6\n");
            fprintf (image, "%d %d\n", xsize, ysize*2);
            fprintf (image, "255\n");
        }

        /* RGBデータとしてバッファに書き出す */
        rgbBuffer = malloc (2*3*xsize*ysize);
        k = 0;
```



```
        for (y=0;y<ysize;y++) {
            for (x=0;x<xsize;x++) {
                data++;
                *(rgbBuffer+k+2) = *(data++);
                *(rgbBuffer+k+1) = *(data++);
                *(rgbBuffer+k+0) = *(data++);
                k+=3;
            }
        }
        for (y=0;y<ysize;y++) {
            for (x=0;x<xsize;x++) {
                beforefield++;
                *(rgbBuffer+k+2) = *(beforefield++);
                *(rgbBuffer+k+1) = *(beforefield++);
                *(rgbBuffer+k+0) = *(beforefield++);
                k+=3;
            }
        }

        if (writeFrame==TRUE) {
            if (fl_is_first==TRUE) {
                for (i=0;i<ysize;i++) {
                    k = 3*i*xsize;
                    fwrite (rgbBuffer+k, 3*xsize, 1, image);
                    fwrite (rgbBuffer+k+3*xsize*ysize, 3*xsize, 1, image);
                }
            }
            else {
                for (i=0;i<ysize;i++) {
                    k = 3*i*xsize;
                    fwrite (rgbBuffer+k+3*xsize*ysize, 3*xsize, 1, image);
                    fwrite (rgbBuffer+k, 3*xsize, 1, image);
                }
            }
            fclose (image);
            if (rename(iname_tmp, iname) != 0) {
                fprintf (stderr, "Cannot rename tmp file to %s, errno=%d\n",
                         iname, errno);
                unlink (iname_tmp);
            }
        }
        fprintf (stderr, "%s: saved image to file %s\n",
                 _progName, iname);
    }
}



/*
 *  終了前のクリーンアップを行う。
 *  <input>
 *    ret : 終了コード
 */
void docleanup (int ret)
{
    vlEndTransfer (svr, path);
    vlDeregisterBuffer (svr, path, drn, buffer);
    vlDestroyBuffer (svr, buffer);
    vlDestroyPath (svr, path);
    vlCloseVideo (svr);
    if(oldBuffer!=NULL)  free(oldBuffer);
    exit (ret);
}
```

```
/*
 * VL_PACKING_RGB_8フォーマット上で比較を行う
 * <input>
 *   x,y : 画像のx,y方向の大きさ
 *   a,b : 比較する画像データへのポインタ
 */
void Compare (int x, int y, unsigned char *a, unsigned char *b)
{
    int i, j;

    printf ("( x , y ) :  R G B :  R G B\n");
    a++; b++;

    for (j=y-1;j>=0;j--) {
        for (i=0;i<x;i++) {
            if ((strncmp (a,b,3))!=0) {
/*              printf ("(%3d,%3d) : %2x%2x%2x : %2x%2x%2x\n",
                        i, j, *(a+2), *(a+1), *a, *(b+2), *(b+1), *b);
*/          }
            a+=4; b+=4;
        }
    }
}



/*
 * シグナルを受け取るフックルーチン
 * <input>
 *  sig : 受信したシグナル番号
 */
void SignalHandler (int sig)
{
    fprintf (stderr, "%s: catch the signal <%d>.\n", _progName, sig);
    fprintf (stderr, "%d fields transferred.\n", fields);
    docleanup(sig);
}
```

```
/*
 *                              get_y_sirius.c
 *          Sirius Video から 縮小されたY画像をを取り出すプログラム
 *                  Usage : get_y_sirius <filename>
 *                Created filename will be <filename>.pgm
 */

#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <stdlib.h>
#include <malloc.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <gl/gl.h>
#include <gl/image.h>
#include <dmedia/vl.h>
#include <dmedia/vl_sirius.h>


#define MAXNAMLEN  1024
#define USAGE      "%s: <output filename>\n"
#define IMAGECOUNT 1                    /* 取り込むフィールドの数 */
#define BUFNUM     1                    /* 作成するバッファの数 */
#define DEBUG


/********************/
/* グローバルデータ */
/********************/
char *deviceName;                       /* 使用するデバイスの名前(Sirius) */
char *_progName;                        /* プログラムの名前 */
char *outfilename;                      /* 出力ファイル名 */
char iname[MAXNAMLEN];                  /* 出力ファイル名 */
char iname_tmp[MAXNAMLEN];              /* テンポラリファイル名 */
int fields;                             /* 取り込んだ画像の数 */
VLServer svr;                           /* ビデオサーバー */
VLBuffer buffer;                        /* キャプチャバッファ */
VLPath path;                            /* ビデオパス */
VLNode src, drn;                        /* ノード */
long fieldSize;                         /* 1フレームあたりのバッファサイズ */
int fl_is_first;                        /* どちらのフィールドを先に取り込むか */
int xsize, ysize;                       /* フィールドあたりの画面の大きさ */
int get_line;                           /* 取り込むライン */
int get_pixel;                          /* 取り込む画素 */
long data_ofs;                          /* 取り込むラインのオフセット */


/********************/
/* プロトタイプ宣言 */
/********************/
void Initialize (int, char**);
void CommunicateDaemon (void);
void ProcessEvent (VLServer, VLEvent*, void*);
void DumpImage (unsigned char*);
void docleanup (int);
void SignalHandler (int);



void main (int argc, char *argv[])
{
    Initialize (argc, argv);
```



```
    CommunicateDaemon ();

    /* ユーザが終了を指示するまでループする(今は回数制限あり) */
    vlMainLoop ();
}



/*
 * アプリケーションの初期化を行う
 * <input>
 *  argc : 引数の数
 *  argv : 引数
 */
void Initialize (int argc, char *argv[])
{
    int i;

    fields = 0;
    _progName = argv[0];

    if (argc!=2) {
        fprintf (stderr, USAGE, _progName);
        exit (1);
    }
    outfilename = argv[1];

    /* シグナルのフック */
    for(i=1;i<=32;i++)  signal(i,SignalHandler);
}



void CommunicateDaemon (void)
{
    VLDevList             devlist;
    VLControlValue        val;
    VLTransferDescriptor  xferDesc;
    int                   i, timing, devicenum;

    /* デーモンへ接続 */
    if ((svr=vlOpenVideo(""))==NULL) {
        fprintf (stderr, "%s: couldn't open video\n", _progName);
        exit (1);
    }
    else
        printf ("Connected Server Daemon.\n");

    /* ノードの設定 */
    src = vlGetNode (svr, VL_SRC, VL_VIDEO, SIR_SRC_ANALOG_VIDEO);
    drn = vlGetNode (svr, VL_DRN, VL_MEM, VL_ANY);

    if ((path=vlCreatePath(svr,VL_ANY,src,drn)) < 0) {
        vlPerror (_progName);
        vlCloseVideo (svr);
        exit (1);
    }

    /* ハードの設定とパス仕様の定義 */
    if (vlSetupPaths(svr, (VLPathList)&path, 1, VL_SHARE, VL_SHARE)!=0) {
        vlPerror (_progName);
        vlCloseVideo (svr);
        exit (1);
    }

    /* 受信したいイベントを設定 */
```

```
    vlSelectEvents (svr, path,
                    VLTransferCompleteMask | VLStreamPreemptedMask |
                    VLStreamStartedMask | VLStreamStoppedMask |
                    VLTransferFailedMask | VLSequenceLostMask);

    /* タイミングをディジタルNTSCに設定(src) */
    val.intVal = VL_TIMING_525_CCIR601;
    if ((vlSetControl(svr,path,src,VL_TIMING,&val)) != 0) {
        vlPerror ("set timing failed");
        vlCloseVideo (svr);
        exit (1);
    }

    /* ビデオ出力フォーマットの指定(drn) */
    val.intVal = VL_FORMAT_SMPTE_YUV;
    vlSetControl (svr, path, drn, VL_FORMAT, &val);

    /* タイミングの設定(drn) */
    if (vlGetControl(svr,path,src,VL_TIMING,&val) == 0) {
        if ((vlSetControl(svr,path,drn,VL_TIMING,&val)!=0)
            && (vlErrno!=VLBadControl)) {
            vlPerror ("SetControl Failed");
            vlCloseVideo (svr);
            exit (1);
        }
    }
    else {
        vlPerror ("get timing failed");
        vlCloseVideo (svr);
        exit (1);
    }

    /* 取り込むフレームのタイプを指定(drn) */
    val.intVal = VL_CAPTURE_INTERLEAVED;
    vlSetControl (svr, path, drn, VL_CAP_TYPE, &val);

    /* ピクセルをYUVで保存するための設定(drn) */
    val.intVal = VL_PACKING_YVYU_422_8;
    vlSetControl (svr, path, drn, VL_PACKING, &val);

    /* 最初に取り込まれるフィールドを知る(src) */
    if (vlGetControl(svr,path,src,VL_SIR_FIELD_DOMINANCE,&val) == 0) {
        timing = ((val.intVal == VL_TIMING_525_SQ_PIX) ||
                  (val.intVal == VL_TIMING_525_CCIR601));
        switch (val.intVal) {
          case SIR_F1_IS_DOMINANT:
            if(timing)  f1_is_first = FALSE;
            else        f1_is_first = TRUE;
            break;
          case SIR_F2_IS_DOMINANT:
            if (timing)  f1_is_first = TRUE;
            else         f1_is_first = FALSE;
            break;
        }
    }

    /* ビデオサイズとそのバイト数を得る */
    vlGetControl (svr, path, drn, VL_SIZE, &val);
    xsize= val.xyVal.x;
    ysize = val.xyVal.y;
    fieldSize = vlGetTransferSize (svr, path);
#ifdef DEBUG
    printf ("for debug: fieldSize is %ld\n", fieldSize);
#endif

    /* 転送のためのコールバックルーチンを設定 */
```



```
    vlAddCallback (svr, path,
                   VLTransferCompleteMask | VLStreamPreemptedMask |
                   VLStreamStartedMask | VLStreamStoppedMask |
                   VLSequenceLostMask | VLTransferFailedMask,
                   ProcessEvent, NULL);

    /* BUFNUM分のフレームバッファの作成と登録 */
    buffer = vlCreateBuffer (svr, path, drn, BUFNUM);
    if (buffer==NULL) {
        vlPerror (_progName);
        vlCloseVideo (svr);
        exit (1);
    }

    if (vlRegisterBuffer(svr, path, drn, buffer)!=0) {
        vlPerror (_progName);
        vlDestroyBuffer (svr, buffer);
        vlCloseVideo (svr);
        exit (1);
    }

    xferDesc.mode = VL_TRANSFER_MODE_DISCRETE;
    xferDesc.count = IMAGECOUNT;
    xferDesc.delay = 0;             /* VLTriggerImmediate なら無視される */
    xferDesc.trigger = VLTriggerImmediate;

    /* データ転送の開始 */
    if (vlBeginTransfer(svr,path,1,&xferDesc)) {
        vlPerror (_progName);
        vlDestroyBuffer (svr, buffer);
        vlCloseVideo (svr);
        exit (1);
    }
}



/*
 *  ビデオライブラリイベントを処理する
 * <input>
 *  svr : VLServerへのポインタ
 *  ev : 何のイベントで呼ばれたか？
 *  data : ユーザーが使用すると宣言したデータへのポインタ
 */
void ProcessEvent (VLServer svr, VLEvent *ev, void *data)
{
    VLInfoPtr info;
    char      *dataPtr;

    switch (ev->reason) {
      case VLStreamStarted:
        break;
      case VLStreamStopped:
      case VLTransferComplete:
        /* バッファ内のフィールドを読み出す */
        while (info=vlGetNextValid(svr,buffer)) {
            dataPtr = vlGetActiveRegion (svr, buffer, info);
            if (fields < IMAGECOUNT) {
                /* フィールド1のデータのみを書き出す */
                if (f1_is_first==TRUE && fields%2==0)
                    DumpImage (dataPtr);
                else if (f1_is_first==FALSE && fields%2==1)
                    DumpImage (dataPtr);
                fields++;
            }
            vlPutFree (svr, buffer);
```

```
        }
        if (fields == IMAGECOUNT)
            docleanup (0);
        break;
      case VLStreamPreempted:
        fprintf (stderr, "%s: Path for this stream preempted.\n", _progName);
        docleanup (1);
        break;
      case VLSequenceLost:
        fprintf (stderr, "%s: Sequence Lost.\n", _progName);
        docleanup (1);
        break;
      case VLTransferFailed:
        fprintf (stderr, "%s: Transfer failed.\n", _progName);
        docleanup (1);
        break;
      default:
        break;
    }
}



/*
 *  フィールド1の偶数列のデータをファイルに出力する
 * <input>
 *  data : YUVフォーマットのデータへのポインタ
 */
void DumpImage (unsigned char *data)
{
    long i;
    unsigned char *tmpbuf, *tmporiginal;
    FILE *fp;

    sprintf (iname_tmp, "%s.%d", outfilename, getpid());
    sprintf (iname, "%s-%05d.pgm", outfilename, (int)(fields/2));

    /* ファイルの作成 */
    if ((fp=fopen(iname_tmp,"wb"))==NULL) {
        fprintf (stderr, "%s: can't create image file.\n", _progName);
        return;
    }
    fprintf (fp, "P5\n");
    fprintf (fp, "%d %d\n", xsize/2, ysize);
    fprintf (fp, "255\n");

    /* テンポラリバッファの作成 */
    if ((tmpbuf=malloc(xsize*ysize*sizeof(unsigned char)/2))==NULL) {
        fprintf (stderr, "%s:can't create image file.\n", _progName);
        return;
    }

    /* Yデータの抽出と書き込み */
    tmporiginal = tmpbuf;
    data++;
    for (i=0;i<xsize*ysize/2;i++) {
        *tmpbuf = *data;
        tmpbuf++;
        data+=4;
    }
    fwrite (tmporiginal, xsize*ysize/2, 1, fp);
    fclose (fp);

    /* ファイルのリネーム */
    if ((rename(iname_tmp,iname)) != 0) {
        fprintf (stderr, "can't rename tmp file to %s", iname);
```



```
        unlink (iname_tmp);
    }

    free (tmporiginal);
}



/*
 *  終了前のクリーンアップを行う。
 *  <input>
 *    ret : 終了コード
 */
void docleanup (int ret)
{
    vlEndTransfer (svr, path);
    vlDeregisterBuffer (svr, path, drn, buffer);
    vlDestroyBuffer (svr, buffer);
    vlDestroyPath (svr, path);
    vlCloseVideo (svr);
    exit (ret);
}



/*
 * シグナルを受け取るフックルーチン
 * <input>
 *  sig : 受信したシグナル番号
 */
void SignalHandler (int sig)
{
    fprintf (stderr, "%s: catch the signal <%d>.\n", _progName, sig);
    fprintf (stderr, "%d fields transferred.\n", fields);
    docleanup(sig);
}
```

```
/*
 *                              calccenter.c
 *              2値化・ノイズ除去の後、重心を求めるプログラム
 *        Usage : calccenter <in filename> <out filename> <threshold>
 *                           <in filename>  is input pgm file.
 *                           <out filename> is output pbm file.
 *                           <threshold> is threshold value.
 */

#include <stdlib.h>
#include <malloc.h>
#include <memory.h>


#define USAGE      "%s: <in filename> <out filename> <threshold>\n"
#define DEBUG


/********************/
/* グローバルデータ */
/********************/
char *_progName;                        /* プログラムの名前 */
char *in_filename;                      /* 入力ファイル名 */
char *out_filename;                     /* 出力ファイル名 */
char *buffer;                           /* 画像データのバッファ */
char *afterBuffer;                      /* 縮小後の画像バッファ */
int xsize, ysize;                       /* フィールドあたりの画面の大きさ */
int threshold;                          /* 閾値 */


/********************/
/* プロトタイプ宣言 */
/********************/
void Initialize (int, char**);          /* アプリケーションの初期化 */
void Binary (void);                     /* 画像の2値化 */
void Contraction (void);                /* 収縮処理を行う */
void CalcCenter (void);                 /* 重心を求める */
void Save (void);                       /* ファイルにセーブ */



void main (int argc, char *argv[])
{
    Initialize (argc, argv);
    Binary ();
    Contraction ();
    CalcCenter ();
    Save ();

    free (buffer);
}



/*
 * アプリケーションの初期化を行う
 * <input>
 *  argc : 引数の数
 *  argv : 引数
 */
void Initialize (int argc, char *argv[])
{
#define MAXLEN 256

    FILE *in_fp;
    int i;
```



```
    unsigned char info[MAXLEN];

    _progName = argv[0];

    if (argc!=4) {
        fprintf (stderr, USAGE, _progName);
        exit (1);
    }
    in_filename  = argv[1];
    out_filename = argv[2];
    threshold = atoi (argv[3]);

    if ((in_fp=fopen(in_filename,"rb"))==NULL) {
        fprintf (stderr, "%s: in file couldn't open.\n", _progName);
        exit (1);
    }

    /* PGMファイルであるか確認する */
    fgets (info, MAXLEN, in_fp);
    if (info[0]!='P' || info[1]!='5') {
        fprintf (stderr, "%s: This file isn't PGM file.\n");
        fclose (in_fp);
        exit (1);
    }

    /* コメントを飛ばして、サイズ情報を得る */
    do {
        fgets (info, MAXLEN, in_fp);
    } while(info[0]=='#');
    sscanf (info, "%d %d", &xsize, &ysize);

    /* 階調情報はスルーする */
    fgets (info, MAXLEN, in_fp);

    /* バッファの確保とデータの読み込み */
    if ((buffer=malloc(xsize*ysize*2))==NULL) {
        fprintf (stderr, "%s: malloc Error.\n", _progName);
        exit (1);
    }
    afterBuffer = buffer+xsize*ysize;
    fread (buffer, xsize*ysize, 1, in_fp);
}



/*
 *  threshold で2値化を行う関数
 */
void Binary (void)
{
    long i;
    unsigned char *tmp_buffer = buffer;

    for (i=0L;i<xsize*ysize;i++) {
        if (*tmp_buffer<threshold)  *(tmp_buffer++) = 1;
        else                        *(tmp_buffer++) = 0;
    }
}



/*
 * 縮退を行う関数
 */
void Contraction (void)
{
```

```
    int x, y, act_y = ysize-1;
    unsigned char *tmp_buffer = buffer;
    unsigned char *tmp_after_buffer = afterBuffer;

    /* ライン０はデータをまるまるコピーする */
    memcpy (tmp_after_buffer, tmp_buffer, xsize-1);
    tmp_after_buffer += xsize-1;
    tmp_buffer += xsize-1;

    /* データが０の時のみ縮退を行う */
    for (y=1;y<act_y;y++) {
        *(tmp_after_buffer++) = *(tmp_buffer++);
        *(tmp_after_buffer++) = *(tmp_buffer++);
        for (x=2;x<xsize;x++) {
            if (*tmp_buffer==0) {
                if (*(tmp_buffer-1)==1        || *(tmp_buffer+1)==1        ||
                    *(tmp_buffer-xsize)==1    || *(tmp_buffer+xsize)==1    ||
                    *(tmp_buffer-xsize-1)==1  || *(tmp_buffer-xsize+1)==1  ||
                    *(tmp_buffer+xsize-1)==1  || *(tmp_buffer+xsize+1)==1)
                    *tmp_after_buffer = 1;
                else
                    *tmp_after_buffer = 0;
            }
            else {
                *tmp_after_buffer = *tmp_buffer;
            }
            tmp_after_buffer++;
            tmp_buffer++;
        }
    }

    /* 最後のラインもまるまるコピーする */
    memcpy (tmp_after_buffer, tmp_buffer, xsize+1);
}



/*
 * １ドット以外の連結成分の重心を求める関数
 * <warning>
 *  ここでは、連結成分が１つのみであると仮定しています
 */
void CalcCenter (void)
{
    unsigned char *tmp_buffer = afterBuffer;
    int x, y, act_y, count = 0;
    long center_x=0L, center_y=0L;

    act_y = ysize-1;
    tmp_buffer += xsize;

    for (y=1;y<act_y;y++) {
        for (x=0;x<xsize;x++) {
            if (*tmp_buffer==0) {
                if (*(tmp_buffer-1)==0        || *(tmp_buffer+1)==0        ||
                    *(tmp_buffer-xsize)==0    || *(tmp_buffer+xsize)==0    ||
                    *(tmp_buffer-xsize-1)==0  || *(tmp_buffer-xsize+1)==0  ||
                    *(tmp_buffer+xsize-1)==0  || *(tmp_buffer+xsize+1)==0) {
                    center_x += x;
                    center_y += y;
                    count++;
                }
            }
            tmp_buffer++;
        }
    }
```



```
    if (count!=0)
        printf ("%d %d\n", (int)(center_x/count), (int)(center_y/count));
    else
        fprintf (stderr, "No Count.\n");
}



/*
 * pbmファイルに保存する関数
 */
void Save (void)
{
    FILE *out_fp;
    int i, shift;
    unsigned char *tmp_buffer=afterBuffer;
    unsigned char data;

    if ((out_fp=fopen(out_filename,"wb"))==NULL) {
        fprintf (stderr, "%s: output file don't create.\n");
        exit (1);
    }
    fprintf (out_fp, "P4\n");
    fprintf (out_fp, "%d %d\n", xsize, ysize);

    for (i=0;i<xsize*ysize/8;i++) {
        data = 0;
        for (shift=7;shift>=0;shift--) {
            data += (*(tmp_buffer++)) << shift;
        }
        fwrite (&data, 1, 1, out_fp);
    }
    fclose (out_fp);
}
```

```
/*
 *                              pgm2pbm.c
 *                PGMファイルをPBMファイルに変換するプログラム
 *            Usage : pgm2pbm <in filename> <out filename> <threshold>
 *                            <in filename>  is pgm file.
 *                            <out filename> is pbm file.
 *                            <threshold> is threshold value.
 */

#include <stdlib.h>
#include <malloc.h>


#define USAGE      "%s: <in filename> <out filename> <threshold>\n"
#define DEBUG


/********************/
/* グローバルデータ */
/********************/
char *_progName;                        /* プログラムの名前 */
char *in_filename;                      /* 入力ファイル名 */
char *out_filename;                     /* 出力ファイル名 */
char *buffer;                           /* 画像データのバッファ */
int xsize, ysize;                       /* フィールドあたりの画面の大きさ */
int threshold;                          /* 閾値 */


/********************/
/* プロトタイプ宣言 */
/********************/
void Initialize (int, char**);          /* アプリケーションの初期化 */
void Binary (void);                     /* 画像の２値化 */
void Save (void);                       /* ファイルにセーブ */



void main (int argc, char *argv[])
{
    Initialize (argc, argv);
    Binary ();
    Save ();

    free (buffer);
}



/*
 * アプリケーションの初期化を行う
 * <input>
 *  argc : 引数の数
 *  argv : 引数
 */
void Initialize (int argc, char *argv[])
{
#define MAXLEN 256

    FILE *in_fp;
    int i;
    unsigned char info[MAXLEN];

    _progName = argv[0];

    if (argc!=4) {
        fprintf (stderr, USAGE, _progName);
```



```
        exit (1);
    }
    in_filename  = argv[1];
    out_filename = argv[2];
    threshold = atoi (argv[3]);
#ifdef DEBUG
    printf ("for debug: threshold is %d\n", threshold);
#endif

    if ((in_fp=fopen(in_filename,"rb"))==NULL) {
        fprintf (stderr, "%s: in file couldn't open.\n", _progName);
        exit (1);
    }

    /* PGMファイルであるか確認する */
    fgets (info, MAXLEN, in_fp);
    if (info[0]!='P' || info[1]!='5') {
        fprintf (stderr, "%s: This file isn't PGM file.\n");
        fclose (in_fp);
        exit (1);
    }

    /* コメントを飛ばして、サイズ情報を得る */
    do {
        fgets (info, MAXLEN, in_fp);
    } while(info[0]=='#');
    sscanf (info, "%d %d", &xsize, &ysize);

    /* 階調情報はスルーする */
    fgets (info, MAXLEN, in_fp);

    /* バッファの確保とデータの読み込み */
    if ((buffer=malloc(xsize*ysize))==NULL) {
        fprintf (stderr, "%s: malloc Error.\n", _progName);
        exit (1);
    }
    fread (buffer, xsize*ysize, 1, in_fp);
}



void Binary (void)
{
    long i;
    unsigned char *tmp_buffer = buffer;

    for (i=0L;i<xsize*ysize;i++) {
        if (*tmp_buffer<threshold)  *(tmp_buffer++) = 1;
        else                        *(tmp_buffer++) = 0;
    }
}



void Save (void)
{
    FILE *out_fp;
    int i, shift;
    unsigned char *tmp_buffer=buffer;
    unsigned char data;

    if ((out_fp=fopen(out_filename,"wb"))==NULL) {
        fprintf (stderr, "%s: output file don't create.\n");
        exit (1);
    }
    fprintf (out_fp, "P4\n");
```

```
    fprintf (out_fp, "%d %d\n", xsize, ysize);

    for (i=0;i<xsize*ysize/8;i++) {
        data = 0;
        for (shift=7;shift>=0;shift--) {
            data += (*(tmp_buffer++)) << shift;
        }
        fwrite (&data, 1, 1, out_fp);
    }
    fclose (out_fp);

    free (tmp_buffer);
}
```

```
/*
 *                              rmnoise.c
 *              2 値化したあと、ノイズを除去するプログラム
 *          Usage : rmnoise <in filename> <out filename> <threshold>
 *                          <in filename>  is pgm file.
 *                          <out filename> is pbm file.
 *                          <threshold> is threshold value.
 */

#include <stdlib.h>
#include <malloc.h>
#include <memory.h>


#define USAGE       "%s: <in filename> <out filename> <threshold>\n"
#define DEBUG


/********************/
/* グローバルデータ */
/********************/
char *_progName;                        /* プログラムの名前 */
char *in_filename;                      /* 入力ファイル名 */
char *out_filename;                     /* 出力ファイル名 */
char *buffer;                           /* 画像データのバッファ */
char *afterBuffer;                      /* 縮小後の画像バッファ */
int xsize, ysize;                       /* フィールドあたりの画面の大きさ */
int threshold;                          /* 閾値 */


/********************/
/* プロトタイプ宣言 */
/********************/
void Initialize (int, char**);          /* アプリケーションの初期化 */
void Binary (void);                     /* 画像の２値化 */
void Contraction (void);                /* 収縮処理を行う */
void Save (void);                       /* ファイルにセーブ */



void main (int argc, char *argv[])
{
    Initialize (argc, argv);
    Binary ();
    Contraction ();
    Save ();

    free (buffer);
}



/*
 * アプリケーションの初期化を行う
 * <input>
 *  argc : 引数の数
 *  argv : 引数
 */
void Initialize (int argc, char *argv[])
{
#define MAXLEN 256

    FILE *in_fp;
    int i;
    unsigned char info[MAXLEN];
```



```
    _progName = argv[0];

    if (argc!=4) {
        fprintf (stderr, USAGE, _progName);
        exit (1);
    }
    in_filename  = argv[1];
    out_filename = argv[2];
    threshold = atoi (argv[3]);
#ifdef DEBUG
    printf ("for debug: threshold is %d\n", threshold);
#endif

    if ((in_fp=fopen(in_filename,"rb"))==NULL) {
        fprintf (stderr, "%s: in file couldn't open.\n", _progName);
        exit (1);
    }

    /* PGMファイルであるか確認する */
    fgets (info, MAXLEN, in_fp);
    if (info[0]!='P' || info[1]!='5') {
        fprintf (stderr, "%s: This file isn't PGM file.\n");
        fclose (in_fp);
        exit (1);
    }

    /* コメントを飛ばして、サイズ情報を得る */
    do {
        fgets (info, MAXLEN, in_fp);
    } while(info[0]=='#');
    sscanf (info, "%d %d", &xsize, &ysize);

    /* 階調情報はスルーする */
    fgets (info, MAXLEN, in_fp);

    /* バッファの確保とデータの読み込み */
    if ((buffer=malloc(xsize*ysize*2))==NULL) {
        fprintf (stderr, "%s: malloc Error.\n", _progName);
        exit (1);
    }
    afterBuffer = buffer+xsize*ysize;
    fread (buffer, xsize*ysize, 1, in_fp);
}



void Binary (void)
{
    long i;
    unsigned char *tmp_buffer = buffer;

    for (i=0L;i<xsize*ysize;i++) {
        if (*tmp_buffer<threshold)  *(tmp_buffer++) = 1;
        else                        *(tmp_buffer++) = 0;
    }
}



void Contraction (void)
{
    int x, y, act_y = ysize-1;
    unsigned char *tmp_buffer = buffer;
    unsigned char *tmp_after_buffer = afterBuffer;

    /* ライン０はデータをまるまるコピーする */
```

```
    memcpy (tmp_after_buffer, tmp_buffer, xsize-1);
    tmp_after_buffer += xsize-1;
    tmp_buffer += xsize-1;

    /* データが０の時のみ縮退を行う */
    for (y=1;y<act_y;y++) {
        *(tmp_after_buffer++) = *(tmp_buffer++);
        *(tmp_after_buffer++) = *(tmp_buffer++);
        for (x=2;x<xsize;x++) {
            if (*tmp_buffer==0) {
                if (*(tmp_buffer-1)==1       || *(tmp_buffer+1)==1       ||
                    *(tmp_buffer-xsize)==1   || *(tmp_buffer+xsize)==1   ||
                    *(tmp_buffer-xsize-1)==1 || *(tmp_buffer-xsize+1)==1 ||
                    *(tmp_buffer+xsize-1)==1 || *(tmp_buffer+xsize+1)==1)
                    *tmp_after_buffer = 1;
                else
                    *tmp_after_buffer = 0;
            }
            else {
                *tmp_after_buffer = *tmp_buffer;
            }
            tmp_after_buffer++;
            tmp_buffer++;
        }
    }

    /* 最後のラインもまるまるコピーする */
    memcpy (tmp_after_buffer, tmp_buffer, xsize+1);
}



void Save (void)
{
    FILE *out_fp;
    int i, shift;
    unsigned char *tmp_buffer=afterBuffer;
    unsigned char data;

    if ((out_fp=fopen(out_filename,"wb"))==NULL) {
        fprintf (stderr, "%s: output file don't create.\n");
        exit (1);
    }
    fprintf (out_fp, "P4\n");
    fprintf (out_fp, "%d %d\n", xsize, ysize);

    for (i=0;i<xsize*ysize/8;i++) {
        data = 0;
        for (shift=7;shift>=0;shift--) {
            data += (*(tmp_buffer++)) << shift;
        }
        fwrite (&data, 1, 1, out_fp);
    }
    fclose (out_fp);
}
```

```
/*
 *
 *                  Picture Format Convert Program using FM
 *                        programmed by sheloon..
 *
 *                             <compiled option>
 *            gcc rgb2yuv.c -o rgb2yuv -I./include -L./lib -lfmdvs
 *
 */

#include <stdlib.h>
#include <ulocks.h>
#include <fm.h>


#define USAGE "%s: <from filename> <to filename>\n"
#define DEBUG
#undef  DEBUG



/*******************/
/* グローバル変数 */
/*******************/
char   *gInputFileName, *gOutputFileName;
ff_rec *gInFF, *gOutFF;



/*********************/
/* プロトタイプ宣言 */
/*********************/
void InitApplication (int, char**);
void InitFileManager (void);
void Convert (void);
void EndFileManager (void);
void ErrorExit (int, char*);



void main (int argc, char *argv[])
{
    InitApplication (argc, argv);
    InitFileManager ();

    Convert ();

    EndFileManager ();
}



/*
 *  アプリケーションの初期化
 * <input>
 *  argc : 引数の数
 *  argv : 引数の値
 */
void InitApplication (int argc, char *argv[])
{
    if (argc!=3) {
        fprintf (stderr, USAGE, argv[0]);
        exit (1);
    }
    gInputFileName  = argv[1];
    gOutputFileName = argv[2];
```



```
}



/*
 * FileManager の初期化
 *
 */
void InitFileManager (void)
{
    /* fm_configを呼び出し、fmの初期化を行う */
    if(fm_initialize() != FM_OK)
        ErrorExit (0, "FM initialize error.\n");

    /* 変換元の画像フォーマットの指定 */
    gInFF =  fm_fileformat_allocate ("sgi");
    if (gInFF==NULL)
        ErrorExit (0, "RGB initialize Error.\n");

    /* 変換先の画像フォーマットの指定 */
    gOutFF =  fm_fileformat_allocate ("dvsyuv");
    if (gOutFF==NULL)
        ErrorExit (1, "YUV initialize Error.\n");
}



/*
 * 使用する画像フォーマットの宣言 (fm_initialize時に呼ばれます)
 */
void fm_config (void)
{
    fm_sgi_register ();             /* SGI(RGB format)を使用 */
    fm_dvs_register ();             /* YUV(YUV format)を使用*/
}



/*
 *  変換を行い、結果をファイルに書き込む
 */
void Convert (void)
{
    /* 変換元データをオープンする */
    if(fm_open(gInFF,gInputFileName) != FM_OK)
        ErrorExit (2, "fm_open Error.\n");
    /* 変換先データを作成する */
    if (fm_create(gOutFF,gOutputFileName,gInFF) != FM_OK)
        ErrorExit (3, "fm_create Error.\n");


    /* 変換元ファイルからデータを読み出す */
    if (fm_read(gInFF,0) != FM_OK)
        ErrorExit (3, "fm_read Error.\n");
    /* データ形式の変換を行う */
    if (fm_convert(gInFF,gOutFF) != FM_OK)
        ErrorExit (3, "fm_convert Error.\n");
    /* 変換先ファイルへ変換したデータを書き込む */
    if (fm_write(gOutFF,0) != FM_OK)
        ErrorExit (3, "fm_write Error.\n");


    /* ファイルをクローズする */
    fm_close (gInFF);
    fm_close (gOutFF);
```

```
    /* 画像フォーマット指定用のメモリを解放する */
    fm_fileformat_free (gInFF);
    fm_fileformat_free (gOutFF);
}



/*
 *  FMの終了
 */
void EndFileManager (void)
{
    /* FMの使用を終了する */
    fm_deinitialize ();
}


/*
 * エラー終了
 * <input>
 *  num     : どこまで進んだか？ (deallocate や close を行うため)
 *  message : 出力エラーメッセージへのポインタ
 */
void ErrorExit (int num, char *message)
{
    fprintf (stderr, message);
    if(num>=3)  fm_close (gInFF);
    if(num>=2)  fm_fileformat_free (gOutFF);
    if(num>=1)  fm_fileformat_free (gInFF);
    fm_deinitialize ();
    exit (1);
}
```

```
/*
 *
 *                  Picture Format Convert Program using FM
 *                        programmed by sheloon..
 *
 *                             <compiled option>
 *              gcc rgb2yuv.c -o rgb2yuv -I./include -L./lib -lfmdvs
 *
 */

#include <stdlib.h>
#include <ulocks.h>
#include <fm.h>


#define USAGE "%s: <from filename> <to filename>\n"
#define DEBUG
#undef  DEBUG



/******************/
/* グローバル変数 */
/******************/
char   *gInputFileName, *gOutputFileName;
ff_rec *gInFF, *gOutFF;



/********************/
/* プロトタイプ宣言 */
/********************/
void InitApplication (int, char**);
void InitFileManager (void);
void Convert (void);
void EndFileManager (void);
void ErrorExit (int, char*);



void main (int argc, char *argv[])
{
    InitApplication (argc, argv);
    InitFileManager ();

    Convert ();

    EndFileManager ();
}



/*
 *  アプリケーションの初期化
 * <input>
 *  argc : 引数の数
 *  argv : 引数の値
 */
void InitApplication (int argc, char *argv[])
{
    if (argc!=3) {
        fprintf (stderr, USAGE, argv[0]);
        exit (1);
    }
    gInputFileName  = argv[1];
    gOutputFileName = argv[2];
```



```
}



/*
 * FileManager の初期化
 *
 */
void InitFileManager (void)
{
    /* fm_configを呼び出し、fmの初期化を行う */
    if(fm_initialize() != FM_OK)
        ErrorExit (0, "FM initialize error.\n");

    /* 変換元の画像フォーマットの指定 */
    gInFF =  fm_fileformat_allocate ("dvsyuv");
    if (gInFF==NULL)
        ErrorExit (0, "RGB initialize Error.\n");

    /* 変換先の画像フォーマットの指定 */
    gOutFF =  fm_fileformat_allocate ("sgi");
    if (gOutFF==NULL)
        ErrorExit (1, "YUV initialize Error.\n");
}



/*
 * 使用する画像フォーマットの宣言 (fm_initialize時に呼ばれます)
 */
void fm_config (void)
{
    fm_sgi_register ();              /* SGI(RGB format)を使用 */
    fm_dvs_register ();              /* YUV(YUV format)を使用*/
}



/*
 *  変換を行い、結果をファイルに書き込む
 */
void Convert (void)
{
    /* 変換元データをオープンする */
    if(fm_open(gInFF,gInputFileName) != FM_OK)
        ErrorExit (2, "fm_open Error.\n");
    /* 変換先データを作成する */
    if (fm_create(gOutFF,gOutputFileName,gInFF) != FM_OK)
        ErrorExit (3, "fm_create Error.\n");


    /* 変換元ファイルからデータを読み出す */
    if (fm_read(gInFF,0) != FM_OK)
        ErrorExit (3, "fm_read Error.\n");
    /* データ形式の変換を行う */
    if (fm_convert(gInFF,gOutFF) != FM_OK)
        ErrorExit (3, "fm_convert Error.\n");
    /* 変換先ファイルへ変換したデータを書き込む */
    if (fm_write(gOutFF,0) != FM_OK)
        ErrorExit (3, "fm_write Error.\n");


    /* ファイルをクローズする */
    fm_close (gInFF);
    fm_close (gOutFF);
```

```
    /* 画像フォーマット指定用のメモリを解放する */
    fm_fileformat_free (gInFF);
    fm_fileformat_free (gOutFF);
}



/*
 *  FMの終了
 */
void EndFileManager (void)
{
    /* FMの使用を終了する */
    fm_deinitialize ();
}


/*
 * エラー終了
 * <input>
 *  num     : どこまで進んだか？ (deallocate や close を行うため)
 *  message : 出力エラーメッセージへのポインタ
 */
void ErrorExit (int num, char *message)
{
    fprintf (stderr, message);
    if(num>=3)  fm_close (gInFF);
    if(num>=2)  fm_fileformat_free (gOutFF);
    if(num>=1)  fm_fileformat_free (gInFF);
    fm_deinitialize ();
    exit (1);
}
```