# Volume Extractor 開発編
# (画像フィルタ作成マニュアル)

(株)アイプランツ・システムズ

# 1. はじめに

Volume Extractor Ver. 3.0（VE）では，ユーザが独自に作成した画像フィルタや，公開されている画像抽出ツール（例えば，OXFORD 大学の Brain Extraction Tool (BET)[1]）を，VE の画像フィルタダイアログから起動させ，その結果を VE に取り込むことが可能になりました．この機能は，一部，制約はありますが，ユーザが研究レベルの画像処理アルゴリズムを実装して，プラグイン形式で，VE 上からより柔軟に利用することが可能となります．図 1 は，読み込んだ脳の MRI 画像から，外部実行モジュールである画像抽出ツール BET を VE 上で起動し，出力された画像結果を VE 内部に取り込んで，その結果を表示した例です．

この機能を使うためには，最初に FilterSettings.xml ファイルに，使用したい画像モジュール（画像フィルタや画像抽出ツール）のモジュール名と引数のパラメータを記述します．VE は，この FilterSettings.xml を読み込み，その外部モジュール名に対応したメニュー名を画像処理フィルタダイアログ（VEFilter ダイアログ）に表示します．ユーザは，このフィルタダイアログに登録されたメニューをクリックするだけで，フォーカスの当っている（現在、使用中の）3 次元画像（画面に表示している 3 次元画像）を外部ファイルとして出力し，指定した画像フィルタに引き渡して，画像フィルタが処理を実行します．その処理が終了した時点で，処理された 3 次元画像を自動的に読み込んで，ウィンドウに結果を表示します．本機能は，単純に外部モジュールを実行するだけではないため，外部モジュールが出力した画像ファイルを対話的に読み込む必要もありません（図 2）．

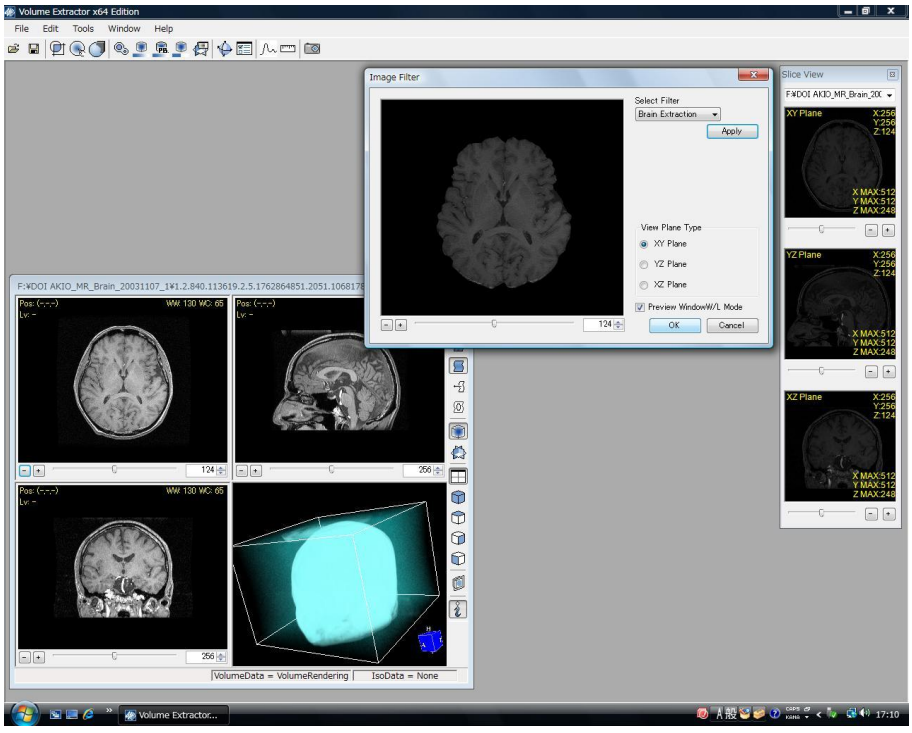



図 1　画像抽出ツール BET による脳領域の抽出

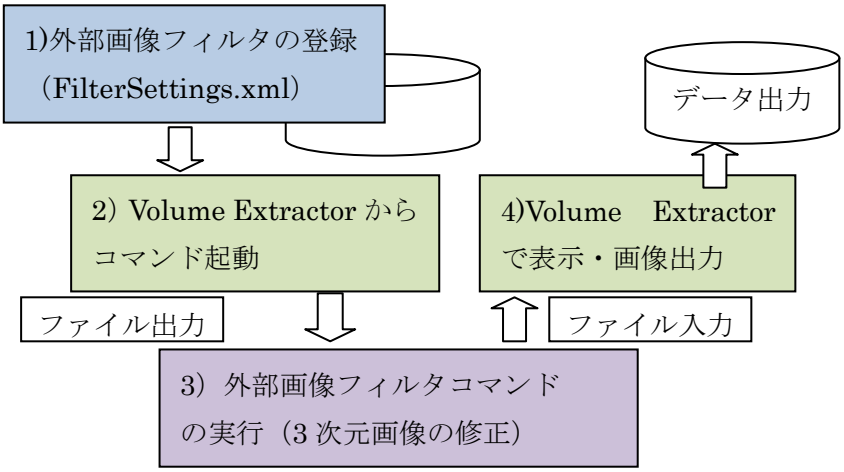



図 2　外部実行形式の実行手順

## ２．外部 3 次元画像処理フィルタの利用

### 2.1 外部画像フィルタの配置

外部画像フィルタ（以下画像フィルタ）を使用するためには，使用する画像フィルタの実行ファイルを，[filter]ディレクトリに移動する必要があります.

１．　マイコンピュータなどで，[マイドキュメント]-[i-Plants]-[Volume Extractor 3.0]-[filter]フォルダを開く

２．　使用する画像フィルタの実行ファイルを[filter]フォルダへコピーする

以下にコピー後のフォルダ構造の例を示します.

```
i-Plants
└─Volume Extractor 3.0
        FilterSettings.xml
        ├─cache
        ├─editcache
        ├─filter
        │       UserFilter1.exe
        │       UserFilter2.exe
        └─SampleData
```

## 2.2　画像処理フィルタの作成方法

一般に 3 次元画像（ボリュームデータ）には，ノイズなどが含まれています．そのため，ノイズ領域を，あらかじめ，平滑化フィルタなどを用いて，取り除いておくことがあります．VE にも平滑化フィルタが用意されていますが，同様なことを外部の平滑化処理を行う画像処理フィルタプログラムを用いて，実現できます．VE でサポートされた画像フィルタは，その処理をすべてメモリ間で行っていますが，外部の画像処理フィルタでの実装では，画像ファイルの入出力を介して，行っています．これらの処理を VE 上で行っている機能を VFilter（Volume　extractor Filter function）と呼びます．

### 外部画像フィルタの入出力

作成する画像フィルタは，VE が出力するボリュームデータと同じ形式でのファイルの読み込み・書き込みが出来る必要があります．現在 VEFilter が対応しているファイル形式は以下の三種類です( VEFilter は出力した形式と同じ形式で結果を読み込みます).

- Analyze75 フォーマット　　(*.hdr ファイルおよび*.img ファイル)
- vdf フォーマット　　(*.vdf ファイル)
- vif フォーマット　　(*.vif ファイル及び*.vol ファイル)

### コマンドライン引数

VEFilter は，読み込んだボリュームデータをファイルとして書き出し，外部フィルタで処理した結果を再びファイルから読み込むことで実現されています．
従って，外部フィルタは FilterSettings.xml で<in>要素に指定したファイル名のファイルを読み込み，<out>要素に指定したファイル名で結果を出力する必要があります．

この実現方法の例として，次のような引数をとるフィルタが考えられます．

```
>FilterName <SrcFileName> <DstFileName> [OPTIONS]
```

・SrcFileName

フィルタ処理の対象となる元データ名を，拡張子を除いて指定します．

・DstFileName

フィルタを適用した後に出力される処理結果データ名を，拡張子を除いて指定します．

・OPTIONS

フィルタへ渡すオプションです．オプションをとる必要がない場合には省略が可能です．

例: Src.hdr，Src.img ファイルに–a オプションを付加して Filter.exe を適用する

```
>Filter.exe Src Dst –a
```

Src.img データにフィルタ処理を行い，その結果を Dst.hdr，Dst.img というファイルとし

て出力します.

### 2.3 設定ファイル(FilterSettings.xml)の編集

[filter]フォルダに入っている画像フィルタを VE で読み込むため, 設定ファイルを編集する必要があります. [マイドキュメント]-[i-Plants]-[Volume Extractor 3.0]フォルダ内の「FilterSettings.xml」ファイルをメモ帳などで開き，以下の例を参考に画像フィルタタイトル，実行ファイル名，オプション，ファイルタイプを設定してください.

設定ファイルの例(FilterSettings.xml)

```
<?xml version="1.0" encoding="Shift_JIS"?> ←―― ①
<Filters>      ←―― ②
  <Filter>     ←―― ③
    <title>UserFilter1</title> ←―― ④
    <run>UserFilter1.exe</run> ←―― ⑤
    <option></option> ←―― ⑥
    <type>Analyze75</type> ←―― ⑦
  </Filter> ←―― ⑧
  <Filter>
    <title>MyFilter</title>
    <run>UserFilter2.exe</run>
    <option>-s</option>
    <type>vif</type>
  </Filter> ←―― ⑨
</Filters>
```

① 先頭行

以下のような XML 宣言を記述します.

```
<?xml version="1.0" encoding="Shift_JIS"?>
```

② <Filters>タグ

これ以降の記述は全て<Filter>と</Filter>の間に記述します.

③ <Filter>タグ

一つの画像フィルタの設定(④～⑦)は<Filter>と</Filter>の間にまとめて記述します.

④ 画像フィルタタイトル

書式：

```
<title>画像フィルタの表示名</title>
```

意味：

VEFilter の画像フィルター一覧に表示さる画像フィルタ名．空白の場合，VEFilter の画像フィルター一覧には「画像フィルタの実行ファイル名+”オプション”」の形式（デフォルト名）で表示されます．

⑤ 画像フィルタの実行ファイル名

書式：

```
<run>実行ファイル名</run>
```

意味：

[filter]ディレクトリにコピーしたフィルタの実行ファイル名を記述します．

⑥ 画像フィルタのオプション

書式：

```
<option>オプション</option>
```

意味：

画像フィルタへ渡すオプションを記述する．詳しくは**「4. 画像処理フィルタプログラムの作成」**を参照して下さい．

⑦ 画像フィルタが扱うファイルタイプ

書式：

```
<type>ファイルタイプ</type>
```

意味：

画像フィルタが扱うファイル形式を指定します．現在，VEFilter で扱えるファイル形式は以下の三種類となります． .hdr，.img フォーマットは，画像処理パッケージである MAYO Clinic で開発されている ANALYZE が使用している画像フォーマットで，広く使用されています．.vif ファイルは，ANALYZE75[2]の.hdr ファイルに相当するファイルで，画像サイズや画素のデータタイプの情報を持っています．.vol フォーマットは，.img ファイルに相当し，画像の RAW データになります．.vdf フォーマットは，.vif フォーマットと.vol フォーマットを同一のファイルにしたデータフォーマットです[注 A)]．

詳しくは「**4. 画像処理フィルタプログラムの作成**」を参照して下さい．．

| 設定ファイル上の表記 | ファイルの拡張子 | |
|---|---|---|
| Analyze75 | .hdr | .img |
| Vif | .vif | .vol |
| Vdf | .vdf | |

⑧ ③に対応した終了タグ．
⑨ ②に対応した終了タグ．

## 2.4 DICOM 画像の読み込み

画像フィルタの実行モジュールを所定のディレクトリに置き，設定ファイル（FilterSetting.xml）を編集したら，VE を稼働させて下さい．次にインポート機能により，DICOM 画像を読み込み，3 次元画像（ボリュームデータ）を作成します．
次にインポート終了後，File－Save の 3 次元画像保存機能を用いて，VDF フォーマットか VOL フォーマット[注 A)]で，3 次元画像を保存しておくと便利です．次回からは，インポート機能を用いずに，File-Open の 3 次元画像読み込み機能で，その 3 次元画像ファイルを直接，読み込めます．複数の DICOM 画像を読み込んだ場合は，一番上にある（アクティブになっている）ワークフォームの画像が処理対象となります．

画像フィルタの実行は，Edit->ImageFilter ダイアログから行います．

注 A）VDF，VOL ファイルフォーマット
VDF，VOL フォーマットは，Volume Extractor 専用の 3 次元画像フォーマットです．VDF フォーマットは，ファイルの中に画像サイズと原画像が含まれています．VOL ファイルは，原画像のみのデータ（ファイルタイプが.vol）と，同じファイル名でファイルタイプが.vif の 2 種類のファイルが生成されます．ファイルタイプが.vif のファイルには画像サイズや画素サイズが書かれています．他のアプリケーションで原画像を読み込む際は，VOL ファイルフォーマットが便利です．

## 2.5. 3 次元画像ファイルフォーマット

Volume Extractor で扱う 3 次元画像のファイルフォーマット（VDF および VOL フォーマット）について、説明します。

### VDF フォーマット

Volume Extractor 用のファイルフォーマットです。

| ファイル情報　［256byte］ |
|---|
| 画像データ　（Raw 形式）<br>（縦 x 横 x 高 x 単位データサイズ） |

### ファイル情報

先頭の 256byte にはファイル情報として下記の情報が ASCII で格納されます。
各項目は、スペース（0x20）で区切られ格納されます。

| | 項目 | 内容 |
|---|---|---|
| ファイル情報<br>［256byte］ | ファイルヘッダ | “VDF_1.0_VE12.8” |
| | StartPoint（描画開始位置）タグ | “sp” |
| | 開始位置 X 座標 | 数値（実数） |
| | 開始位置 Y 座標 | 数値（実数） |
| | 開始位置 Z 座標 | 数値（実数） |
| | グリッドサイズタグ | “n” |
| | X 軸サイズ | 数値（整数） |
| | Y 軸サイズ | 数値（整数） |
| | Z 軸サイズ | 数値（整数） |
| | ピッチサイズタグ | “pitch” |
| | X 方向 Pitch | 数値（実数） |
| | Y 方向 Pitch | 数値（実数） |
| | Z 方向 Pitch | 数値（実数） |
| | データタイプタグ | “dt” |
| | データタイプ(1～4) | 数値（1～4） |
| | 情報終了識別子 | ¥n(0x0A) |
| | （予備） | 0x00 |

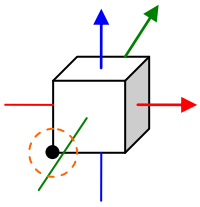
・　ファイルヘッダ

文字列“VDF_1.0_VE12.8”固定

・ **StartPoint**

3D 空間上での配置位置の指定

右図のようにモデルの 3D 空間上での開始点を指定します。



・ **グリッドサイズ**

X 軸、Y 軸、Z 軸、各方向のピクセル（ボクセル）数のサイズ

・ **データタイプ**

１ピクセル（ボクセル）のデータで使用されるデータの型

1 ： Byte 型（1byte）

2 ： Unsigned Short 型（2byte）

3 ： Short 型（2byte）

4 ： int 型（4byte）

・ **情報終了識別子**

ファイル情報の最後に“¥n”(0x0A)を格納

・ **（予備）**

この部分は、今後の拡張等のための箇所として用意されています。0x00 で埋めます。

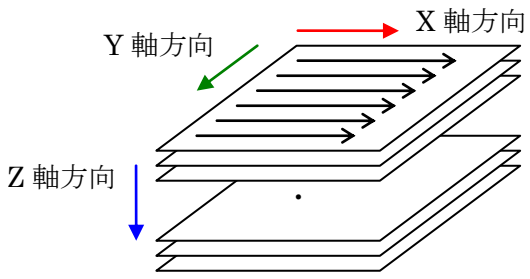
**画像データ**

画像データが RAW 形式で格納されます。

各データ要素は「データタイプ」で指定されたサイズになり、データ全体としては下記のサイズになります。

**（グリッド X 軸サイズ）×（グリッド Y 軸サイズ）×（グリッド Z 軸サイズ）×（データタイプの指定 byte 数）**

格納順序は、X 軸方向の要素から格納開始され、Y 軸方向、Z 軸方向へと格納されています。（右図イメージ参照）




※VE での表示は、Z 軸方向を上向きにしていますのでご注意下さい。

### VOL フォーマット

拡張子、“vol” と “vif” の 2 つのファイルにより構成されたボリュームデータファイルです。vif がファイル情報部分、vol が画像データ部分となります。

#### vif

vif ファイルにはファイル情報として下記の内容が ASCII として 5 行構成で格納されます。（改行コードは ¥r¥n（0x0D0A））

| 行数 | 項目 | 記述内容 |
|---|---|---|
| 1 | ファイルヘッダ文字列 | “VIF 1.0 VE12.8” |
| 2 | StartPoint | “start_pt [X 座標] [Y 座標] [Z 座標]” |
| 3 | 画像（グリッド）サイズ | “size [X 方向] [Y 方向] [Z 方向]” |
| 4 | 各方向のピッチ | “pitch [X 方向] [Y 方向] [Z 方向]” |
| 5 | データタイプ | “data_type [1～4]” |

2 行目以降の各項目は、内容を示す文字列（”start_pt”等）が記述され、続けて値（数値）が格納されます。

文字列と値の間は半角スペース（0x20）2 文字分で区切り、各値（数値）の間は半角スペース 1 文字分で区切ります。

【例】

```
VIF 1.0 VE12.8
start_pt  -0.5 -0.5 -0.5
size  512 512 469
pitch  0.1693333 0.1693333 0.64
data_type  2
```

#### vol

画像データ要素が RAW 形式で格納されます。

格納されるデータのサイズ、順序等は VDF のデータ部分と同じです(VDF フォーマット参照)。

出力サンプルコードは、**「付録 4. 3 次元画像ファイルの出力サンプル」**を参照してください。

## 3. 既存の外部画像フィルタ（ツール） 利用

本項では，Oxford 大学の Brain Extraction Tool（BET）[1]を Volume Extractor から使用する例を紹介します.

①Brain Extraction Tool の目的

BET は，人間の頭部の 3 次元 MRI 画像から，脳領域を自動的に抽出するツールで，大脳のセグメンテーションや解析などに使用されます.

②BET の配置

bet.exe を，[マイドキュメント] – [i-Plants] – [Volume Extractor 3.0] – [Filter]フォルダへコピーします.

③設定ファイルの編集

[マイドキュメント] – [i-Plants] – [Volume Extractor 3.0]ディレクトリ内の[FilterSettings.xml]ファイルをテキストエディタで開き，以下の設定を追加します.

```
<Filter>
  <title>BrainExtraction</title>
  <command>bet.exe ___tmp___ o___tmp___</command>
  <in>___tmp___</in>
  <out>o___tmp___</out>
  <type>Analyze75</type>
</Filter>
```

④外部画像フィルタの実行

・Volume Extractor を起動し，頭部の 3 次元画像を読み込みます（図 3-A）.

・[Edit]メニューから，[ImageFilter]を起動します.

・[Select Filter]ドロップダウンリストボックスから[BrainExtraction]を選択します(図 3-B). FilterSettings.xml ファイルに追加していない場合，この[BrainExtraction]が表示されません.

・[Apply]ボタンをクリックし，bet ツールを起動します(図 3-C).

・ツールが正常に動作したことを確認します(図 3-D).

・[OK]ボタンをクリックすることで Volume Extractor 内部の画像データにその結果が反映されます(図 3-E).

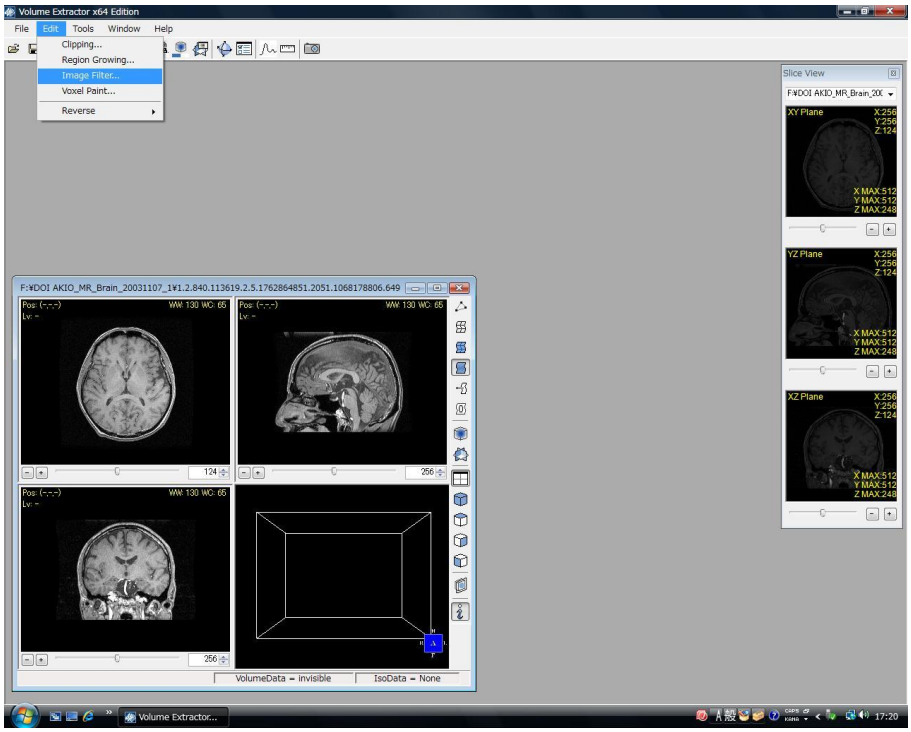



図３-Ａ　画像フィルタ機能の選択（Edit->ImageFilter... ）

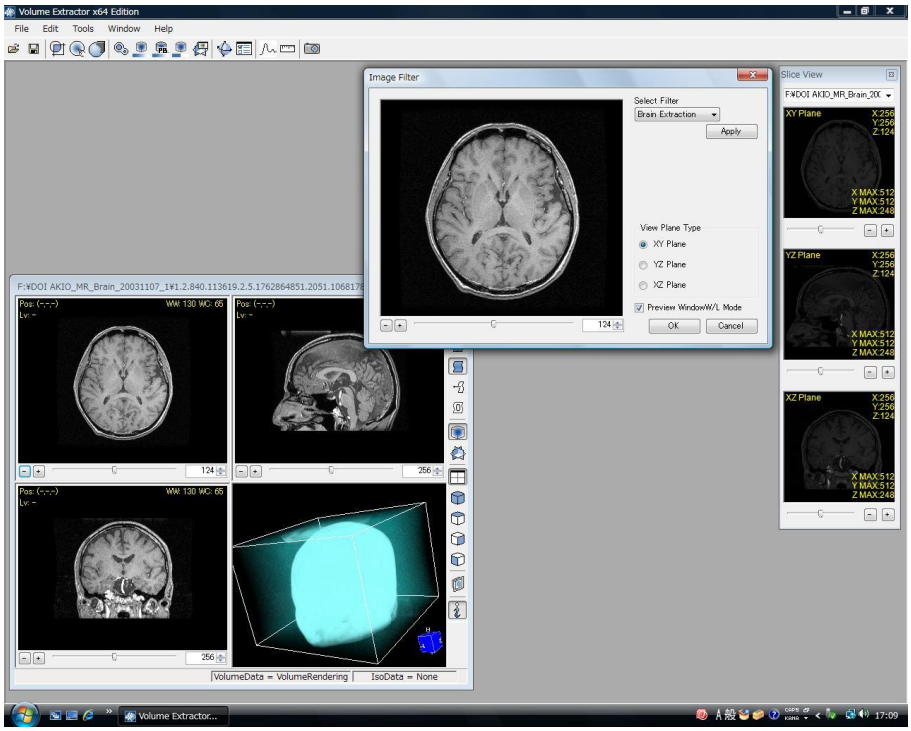



図３-Ｂ　３次元画像フィルタダイアログの表示

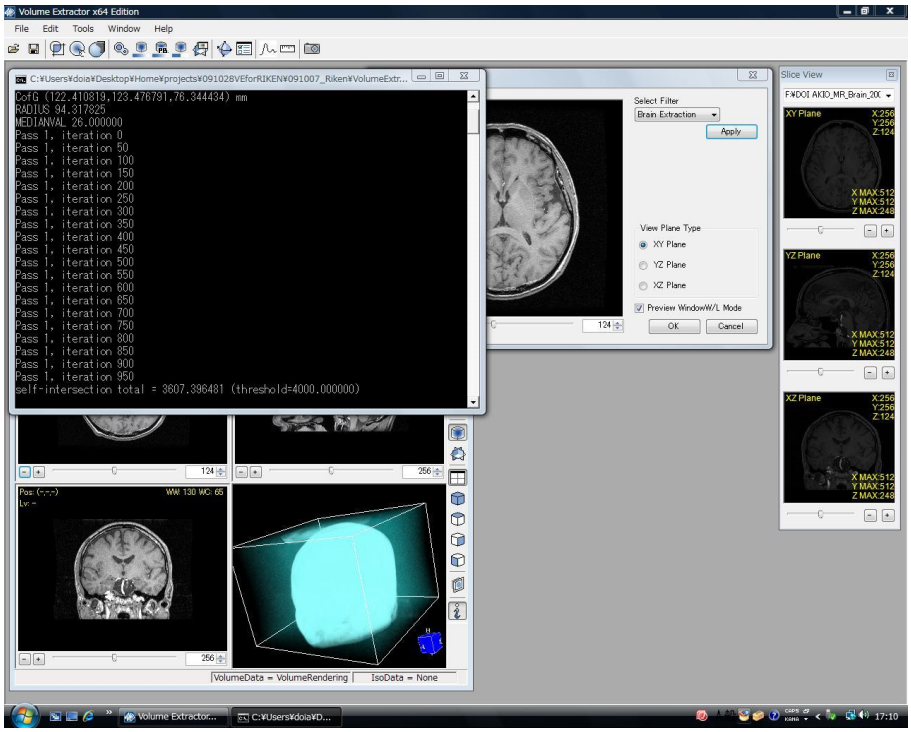



図 3 - C　3 次元画像フィルタ機能から，BET を実行

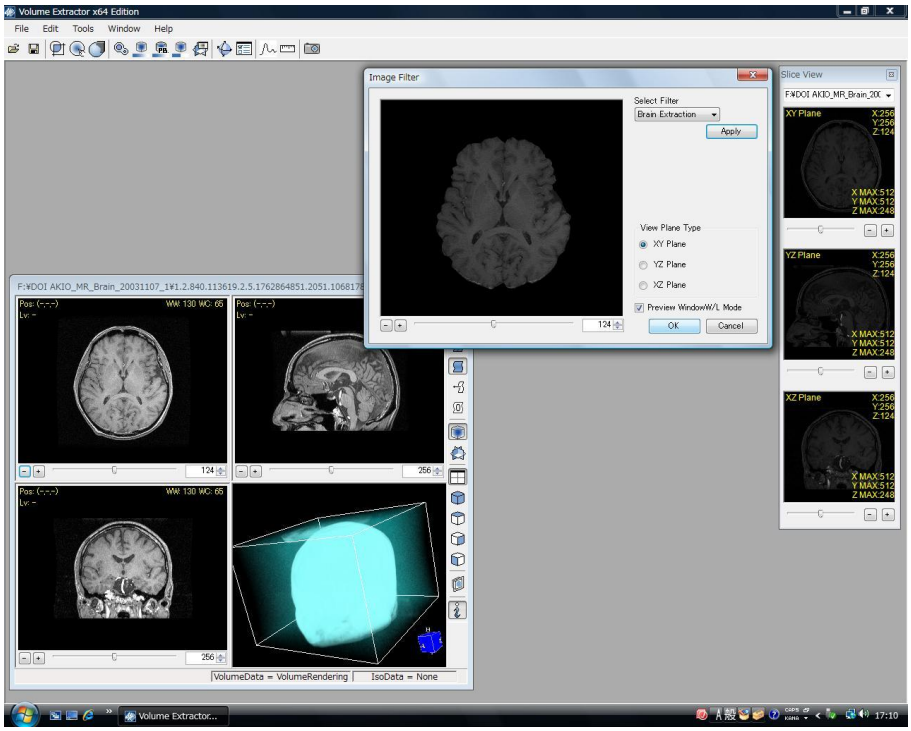



図 3 - D　BET による脳領域の抽出と確認

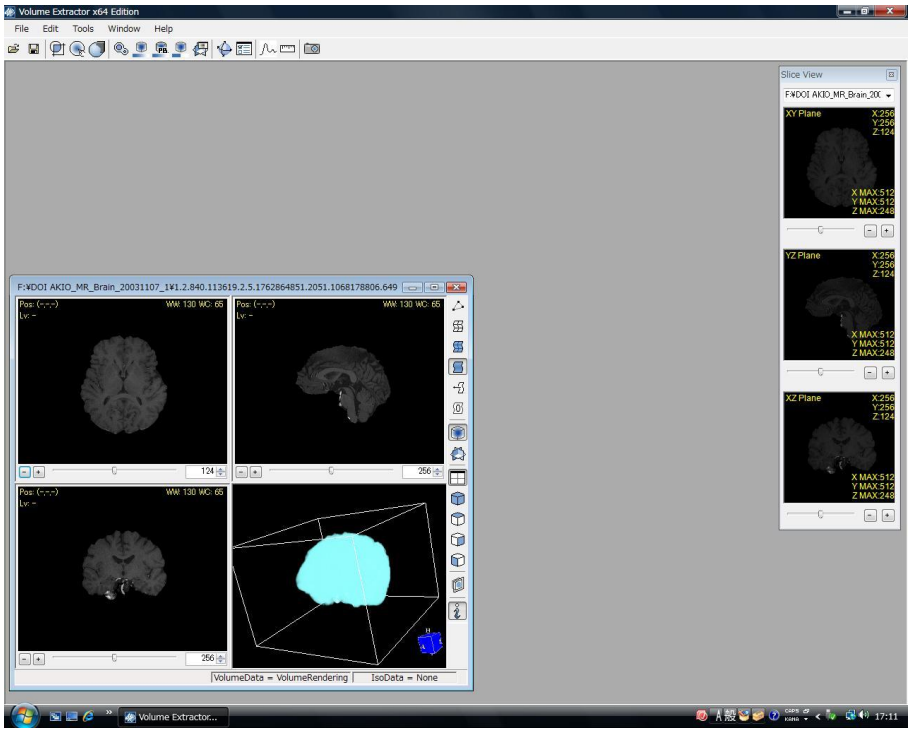



図 3 - E　BET による脳領域の抽出終了

# 4. 画像処理フィルタプログラムの作成

本章では，Sobel フィルタを例に画像フィルタの作成と使用について解説します.

## 4.1.サンプルフィルタ「Sobel_sample」の概要

①機能

外部画像フィルタのサンプルとして，Sobel フィルタを示します(図 4，付録 3).このフィルタプログラムは，入力，出力共に type 3 (signed short)の vif/vol フォーマットファイルを対象として，記述されています.

$$fx:\begin{bmatrix}0&0&0\\-1&-2&-1\\0&0&0\end{bmatrix}\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}\begin{bmatrix}0&0&0\\1&2&1\\0&0&0\end{bmatrix}\quad fy:\begin{bmatrix}0&-1&0\\0&-2&0\\0&-1&0\end{bmatrix}\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}\begin{bmatrix}0&1&0\\0&2&0\\0&1&0\end{bmatrix}$$

図 4　　4-A サンプル中の Sobel フィルタ

②入出力形式の決定

作成する外部画像フィルタは，VEFilter が出力するボリュームデータと同じ形式でのファイルの読み込み・書き込みが出来る必要があります(2 章参照)．本章のサンプルプログラムは vif/vol フォーマットに対応します．対応している画像フォーマットは，vif/vol フォーマット，vdf フォーマット，ANALYZE75 フォーマットの 3 種類です.

③コマンドライン引数

VEFilter は外部フィルタに対し，引数を用いて処理に必要な情報を提供します.本章のサンプルフィルタにオプションはないため，以下の形式での呼び出しを想定しています.

```
>Sobel_sample.exe <SrcFileName> <DstFileName>
```

・SrcFileName

フィルタ処理の対象となる元データ名を，拡張子を除いて指定します.

・DstFileName

フィルタを適用した後に出力される処理結果データ名を，拡張子を除いて指定します.

④サンプルフィルタの作成

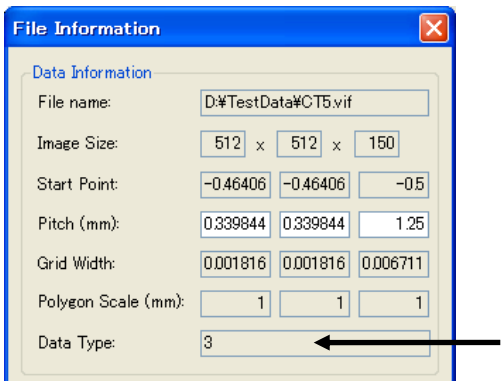
付録 3 のソースコードを参考にプログラムを記述し，「Sobel_sample.exe」というファイル名でコンパイルします.

⑤Volume Extractor の設定

出力された Sobel_sample.exe を[マイドキュメント]-[i-Plants]-[Volume Extractor 3.0]-[filter]ディレクトリの中に配置し，FilterSettings.xml へ以下の設定を追加します.

```
<Filter>
  <title>Sobel</title>
  <command>Sobel_sample.exe ___tmp___ o___tmp___</command>
  <in>___tmp___</in>
  <out>o___tmp___</out>
  <type>vif</type>
</Filter>
```

⑥Volume Extractor を起動し，Data Type が 3 である 3 次元画像を開きます. DataType は[Tools] - [File Information]から確認できます.




⑦[Image Filter]を起動し，[Select Filter]から[Sobel]を選択します.

⑧[Apply]ボタンをクリックします.

## 5. おわりに

本マニュアルでは，既に使用されているユーザの外部画像フィルタや画像抽出ツールを VE から使用する方法を説明しました．また，サンプルプログラムの具体例を挙げながら，その操作方法も説明しました．本機能を使用することにより，ユーザは，独自の画像処理アルゴリズムを，VE に組み込むことが出来ます．

本ソフトウェアの試行版（Volume Extractor Ver.3.0 Light 32/64 ビット版）は，株式会社 i-Plants Systems のホームページ[3]より，ダウンロード可能です．VE の詳細な操作方法は，操作マニュアル[4]を参照して下さい．

## 参考文献


1) University of Oxford, Brain Extraction Tool (BET), FMRIB Centre, Department of Clinical Neurology, "http://www.fmrib.ox.ac.uk/analysis/research/bet/"

2) Analyze75 data format, http://eeg.sourceforge.net/ANALYZE75.pdf, 2010.

3) 株式会社 i-Plants Systems, ”VE Ver.3.0”, http://www.i-plants.co.jp/hp/download, (2008)

4) 株式会社 i-Plants Systems, Volume Extractor Ver.3.0 操作マニュアル

# 付録

## 付録１．Analyze75 フォーマット作成プログラム

```
//
//  引用先：http://eeg.sourceforge.net/ANALYZE75.pdf  (VC++, UNIX)
//
// analyze75_make_header.cpp : コンソールアプリケーション用のエントリポイントの定義
//  VC++ 2008

#include "stdafx.h"
/*****************************************************************************
make_header heart.hdr 128 128 97 3 CHAR 255 0
Makes the header file heart.hdr with the following dimensions:
x dimension = 128
y dimension = 128
slices/volume = 97
volumes in file = 3
bits/pixel = 8
global max = 255
global min = 0
*****************************************************************************
/

/* This program creates an ANALYZETM database header */
/*
* (c) Copyright, 1986-1995
* Biomedical Imaging Resource
* Mayo Foundation
*
* to compile:
*
* cc -o make_hdr make_hdr.c
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>

#include "dbh.h"

void usage()
{
        printf("usage: make_hdr name.hdr x y z t datatype max min ¥n¥n");
        printf(" name.hdr = the name of the header file¥n");
        printf(" x = width, y = height, z = depth, t = number of volumes¥n");
        printf(" acceptable datatype values are: BINARY, CHAR, SHORT,¥n");
        printf(" INT, FLOAT, COMPLEX, DOUBLE, and RGB¥n");
        printf(" max = maximum voxel value, min = minimum voxel value¥n");

}

main(int argc, char **argv) /* file x y z t datatype max min */
{
        int i;
        struct dsr hdr;
        FILE *fp;
        static char DataTypes[9][12] = {"UNKNOWN", "BINARY",
                "CHAR", "SHORT", "INT","FLOAT", "COMPLEX",
                "DOUBLE", "RGB"};
        static int DataTypeSizes[9] = {0,1,8,16,32,32,64,64,24};
        if(argc != 9)
        {
                usage();
```

```
                //exit(0);
                return(0);
        }
        memset(&hdr,0, sizeof(struct dsr));
        for(i=0;i<8;i++)
                hdr.dime.pixdim[i] = 0.0;
        hdr.dime.vox_offset = 0.0;
        hdr.dime.funused1 = 0.0;
        hdr.dime.funused2 = 0.0;
        hdr.dime.funused3 = 0.0;
        hdr.dime.cal_max = 0.0;
        hdr.dime.cal_min = 0.0;
        hdr.dime.datatype = -1;
        for(i=1;i<=8;i++)
                if(!strcmp(argv[6],DataTypes[i]))
                {
                        hdr.dime.datatype = (1<<(i-1));
                        hdr.dime.bitpix = DataTypeSizes[i];
                        break;
                }
                if(hdr.dime.datatype <= 0)
                {
                        printf("<%s> is an unacceptable datatype ¥n¥n", argv[6]);
                        usage();
                        exit(0);
                }
                if((fp=fopen(argv[1],"w"))==0)
                {
                        printf("unable to create: %s¥n",argv[1]);
                        exit(0);
                }
                hdr.dime.dim[0] = 4; /* all Analyze images are taken as 4 dimensional
*/
                hdr.hk.regular = 'r';
                hdr.hk.sizeof_hdr = sizeof(struct dsr);
                hdr.dime.dim[1] = atoi(argv[2]); /* slice width in pixels */
                hdr.dime.dim[2] = atoi(argv[3]); /* slice height in pixels */
                hdr.dime.dim[3] = atoi(argv[4]); /* volume depth in slices */
                hdr.dime.dim[4] = atoi(argv[5]); /* number of volumes per file */
                hdr.dime.glmax = atoi(argv[7]); /* maximum voxel value */
                hdr.dime.glmin = atoi(argv[8]); /* minimum voxel value */

        /* Set the voxel dimension fields:
                                                                                A
value of 0.0 for these fields implies

        that the value is unknown.

        Change these values to what is appropriate

        for your data or pass additional command

        line arguments */
                hdr.dime.pixdim[1] = 0.0; /* voxel x dimension */
                hdr.dime.pixdim[2] = 0.0; /* voxel y dimension */
                hdr.dime.pixdim[3] = 0.0; /* pixel z dimension, slice thickness */
                                                                /* Assume zero
offset in .img file, byte at which pixel
                data starts in the image file */
                hdr.dime.vox_offset = 0.0;
                /* Planar Orientation; */
                /* Movie flag OFF: 0 = transverse, 1 = coronal, 2 = sagittal
                Movie flag ON: 3 = transverse, 4 = coronal, 5 = sagittal */
                hdr.hist.orient = 0;

                /* up to 3 characters for the voxels units label; i.e. mm., um., cm. */
```

```
                    strcpy(hdr.dime.vox_units," ");
                    /* up to 7 characters for the calibration units label; i.e. HU */
                    strcpy(hdr.dime.cal_units," ");
                    /* Calibration maximum and minimum values;
                    values of 0.0 for both fields imply that no
                    calibration max and min values are used */

                    hdr.dime.cal_max = 0.0;
                    hdr.dime.cal_min = 0.0;
                    fwrite(&hdr,sizeof(struct dsr),1,fp);
                    fclose(fp);

                    return (0);
}
```

## 付録2. 3次元画像ファイルの読み込みプログラム例

```
// vif/volフォーマットの3次元画像を読み込み，書き込みします（VC++, UNIX）.
//
#include "stdafx.h"
#define MAXWORD 100
#define MAXDATA  100
// 構造体
// 座標用
typedef struct {
        char data_name[MAXWORD];
        double x, y, z;
} Coord;

// 画像サイズ用
typedef struct {
        char data_name[MAXWORD];
        int width, height, depth;
} Imgsize;

// データタイプ用
typedef struct {
        char data_name[MAXWORD];
        int type;
} Datatype;

// vifファイル用
typedef struct {
        char str_header[MAXWORD];// ファイルヘッダ文字列
        Coord start_pt;                                 // データ開始位置
        Imgsize img;                                    // 画像サイズ
        Coord pitch;                                    // 各方向のピッチ
        Datatype data_tp;                               // データタイプ
} Vifdata;

// volデータ用
typedef struct {
        int numofdata;                                  // 総データ数
        char *type_c;                                   // データ格納ポインタ
        unsigned short int *type_usi;
        short *type_si;
        int *type_i;
} Voldata;


// 実験
#define voxel1( VIF, VOL, X, Y, Z ) (VOL)->type_c[(X) + (Y) * (VIF)->img.width + (Z) *
(VIF)->img.width * (VIF)->img.height]
#define voxel2( VIF, VOL, X, Y, Z ) (VOL)->type_usi[(X) + (Y) * (VIF)->img.width    +
(Z) * (VIF)->img.width * (VIF)->img.height]
#define voxel3( VIF, VOL, X, Y, Z ) (VOL)->type_si[(X) + (Y) * (VIF)->img.width + (Z) *
(VIF)->img.width * (VIF)->img.height]
#define voxel4( VIF, VOL, X, Y, Z ) (VOL)->type_i[(X) + (Y) * (VIF)->img.width + (Z) *
(VIF)->img.width * (VIF)->img.height]


// 関数プロトタイプ宣言
int loadVif( Vifdata * );
int loadVol( Vifdata *, Voldata * );
int saveVif( Vifdata * );
int saveVol( Vifdata *, Voldata * );
void debugProg( Vifdata *, Voldata * );


/**
/* main関数
```

```
**/
int _tmain(int argc, _TCHAR* argv[])
{
	static Vifdata vif;			// vifデータ用
	static Voldata vol;			// volデータ用

	int ret;

	// vifファイル読み込み
	ret = loadVif( &vif );
	if( ret == -1 ) {
		printf( "It fails to read a vif file¥n" );
		return -1;
	}

	// volファイル読み込み
	ret = loadVol( &vif, &vol );
	if( ret == -1 ) {
		printf( "It fails to read a vol file¥n" );
		return -1;
	} else if( ret == -2 ) {
		printf( "Memory allocation error¥n" );
		return -2;
	}

	// vifファイル書き出し
	ret = saveVif( &vif );
	if( ret == -1 ) {
		printf( "It fails to write a vif file¥n" );
		return -1;
	}

	// volファイル書き出し
	ret = saveVol( &vif, &vol );
	if( ret == -1 ) {
		printf( "It fails to write a vol file¥n" );
		return -1;
	}

	return 0;
}


/**
/* vifファイル読み込み
/* @param vif構造体
/* @return 0 or -1
**/
int loadVif( Vifdata *vif ) {
	FILE *inp;
		// ファイルポインタ
	char filename[MAXWORD];			// vifファイル名

	//
	//
	// 読み込むファイル名入力
	printf( "Road viffile name?->" );
	scanf( "%s",filename );

	// データファイルオープン
	if( ( inp = fopen( filename, "r" ) )  == NULL ) return -1;

	// ファイルヘッダ文字列読み込み
	fgets( vif->str_header, MAXDATA, inp );
	printf( "%s¥n", vif->str_header );
```

```
	// データ開始位置読み込み
	fscanf( inp, "%s %lf %lf %lf", vif->start_pt.data_name,
		&vif->start_pt.x, &vif->start_pt.y, &vif->start_pt.z );
	printf( "%lf %lf %lf¥n", vif->start_pt.x, vif->start_pt.y, vif->start_pt.z );

	// 画像サイズ読み込み
	fscanf( inp, "%s %d %d %d", vif->img.data_name,
		&vif->img.width, &vif->img.height, &vif->img.depth );
	printf( "%d %d %d¥n", vif->img.width, vif->img.height, vif->img.depth );

	// ピッチ読み込み
	fscanf( inp, "%s %lf %lf %lf", vif->pitch.data_name,
		&vif->pitch.x, &vif->pitch.y, &vif->pitch.z );
	printf( "%lf %lf %lf¥n", vif->pitch.x, vif->pitch.y, vif->pitch.z );

	// データタイプ読み込み
	fscanf( inp, "%s %d", vif->data_tp.data_name, &vif->data_tp.type );
	printf( "%d¥n", vif->data_tp.type );

	fclose( inp );
	return 0;
}

/**
/* ボクセルデータ読み込み
/* @param vol構造体
/* @return 0 or -1 or -2
**/
int loadVol( Vifdata *vif, Voldata *vol ) {
	FILE *inp;
						// ファイルポインタ
	char filename[MAXWORD];
	// volファイル名

	size_t size;

	//
	//
	// 読み込むファイル名入力
	printf( "Road volfile name?->" );
	scanf( "%s",filename );

	// データファイルオープン
	if( ( inp = fopen( filename, "rb" ) )  == NULL ) return -1;

	// 総データ数
	vol->numofdata =  vif->img.width * vif->img.height * vif->img.depth;

	// 領域確保＆データ読み込み（！！？）
	switch ( vif->data_tp.type ) {
	case 1:
		vol->type_c = ( char * )malloc( sizeof( char ) * vol->numofdata );
		if( vol->type_c == NULL ) return -2;
		size = fread( vol->type_c, sizeof( char ), vol->numofdata, inp );
		break;
	case 2:
		vol->type_usi = ( unsigned short int * )
			malloc( sizeof( unsigned short int ) * vol->numofdata );
		if( vol->type_usi == NULL ) return -2;
		size = fread( vol->type_usi, sizeof( unsigned short int ),
			vol->numofdata, inp );
		break;
	case 3:
		vol->type_si = ( short * )malloc( sizeof( short ) * vol->numofdata );
		if( vol->type_si == NULL ) return -2;
		size = fread( vol->type_si, sizeof( short ), vol->numofdata, inp );
```

```
                break;
        case 4:
                vol->type_i = ( int * )malloc( sizeof( int ) * vol->numofdata );
                if( vol->type_i == NULL ) return -2;
                size = fread( vol->type_i, sizeof( int ), vol->numofdata, inp );
                break;
        default:
                return -1;
        }

        fclose( inp );
        return 0;
}


/**
/* vifファイル書き込み
/* @param vif構造体
/* @return 0 or -1
**/
int saveVif( Vifdata *vif ) {
        FILE *outp;
        // ファイルポインタ
        char filename[MAXWORD];                 // vifファイル名

        //
        //
        // 読み込むファイル名入力
        printf( "Write viffile name?->" );
        scanf( "%s",filename );

        // データファイルオープン
        if( ( outp = fopen( filename, "wb" ) )  == NULL ) return -1;

        fprintf( outp, "%s", vif->str_header );
        fprintf( outp, "%s  %lf %lf %lf¥n", vif->start_pt.data_name,
                vif->start_pt.x, vif->start_pt.y, vif->start_pt.z );
        fprintf( outp, "%s  %d %d %d¥n", vif->img.data_name,
                vif->img.width, vif->img.height, vif->img.depth );
        fprintf( outp, "%s  %lf %lf %lf¥n", vif->pitch.data_name,
                vif->pitch.x, vif->pitch.y, vif->pitch.z );
        fprintf( outp, "%s  %d¥n", vif->data_tp.data_name, vif->data_tp.type );

        fclose( outp );
        return 0;
}


/**
/* volファイル書き込み
/* @param vol構造体
/* @return 0 or -1
**/
int saveVol( Vifdata *vif, Voldata *vol ) {
        FILE *outp;
        // ファイルポインタ
        char filename[MAXWORD];                 // vifファイル名

        //
        // 読み込むファイル名入力
        printf( "Write volfile name?->" );
        scanf( "%s",filename );

        // データファイルオープン
        if( ( outp = fopen( filename, "wb" ) )  == NULL ) return -1;
```

```
        // ボクセルデータ書き込み
        switch ( vif->data_tp.type ) {
        case 1:
                fwrite( vol->type_c, sizeof( char ), vol->numofdata, outp );
                free( vol->type_c );
                break;
        case 2:
                fwrite( vol->type_usi, sizeof( unsigned short int ), vol->numofdata,
outp );
                free( vol->type_usi );
                break;
        case 3:
                fwrite( vol->type_si, sizeof( short int ), vol->numofdata, outp );
                free( vol->type_si );
                break;
        case 4:
                fwrite( vol->type_i, sizeof( int ), vol->numofdata, outp );
                free( vol->type_i );
                break;
        default:
                return -1;
        }

        fclose( outp );
        return 0;
}
```

## 付録3. 3次元画像フィルタサンプル (sobel.cpp)

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#define MIN_ARGC 1


#define getDotPos(x,y,z,X,XY) ( (x)+(y)*(X)+(z)*(XY) )

/* コマンドラインの第一引数が元ファイル名，第二引数が出力ファイル名．*/
#define SRC_F_NAME        1
#define DST_F_NAME        2

/* 読み込めるデータタイプをshortに限定する*/
#define USABLE_DATA_TYPE 3
/* データタイプに対応*/
typedef short DataType;
/*
1        unsigned char
2        unsigned short
3        short
4        int
*/


typedef enum _ERROR_TYPE{
        SUCCESS,
        ERROR_ARG,
        ERROR_MEMORY,
        ERROR_VIF_READ,
        ERROR_VIF_NOTFOUND,
        ERROR_VIF_OPENFALT,
        ERROR_VIF_SIZEX,
        ERROR_VIF_SIZEY,
        ERROR_VIF_SIZEZ,
        ERROR_VIF_TYPE,
        ERROR_VOL_NOTFOUND,
        ERROR_VOL_OPENFALT,
        ERROR_FILTER_FALT,
} ERROR_TYPE;

char* skipSpace(char* charBuffer);
char* skipChar(char* charBuffer);

ERROR_TYPE VIFReader(char* VIFname,int* sizeX,int* sizeY,int* sizeZ,int* data_type);
ERROR_TYPE sobel_main(DataType* OriginalImg,DataType* ResultImg,int sizeX,int sizeY,int
sizeZ);
void copySlice(DataType** temp_faces,int tZIndex,DataType* OriginalData,int oZIndex,int
sizeX,int sizeY,int orgX_Len,int org_Area,int tmpX_Len,int tmp_Area);



/*
実行方法
 $ Filter SrcFileName DstFileName
 SrcFileName ... 必須フィルタ適用前のvif/volファイル．ただし拡張子は含めない
 DstFileName ... 必須フィルタ適用後に出力されるファイル名
*/
int main(int argc,char* argv[])
{
        char* FileName = NULL;/* 入出力ファイル名(各引数に拡張子を結合して作成) */
        FILE* fpVol=NULL;     /*入力vol，出力volファイルに使用するファイルポインタ*/
        FILE* fpVif=NULL;
```

```
        size_t readSize;

        int sizeX=0;     /* X,Y,Z各方向のボクセル数(vifファイルから取得) */
        int sizeY=0;
        int sizeZ=0;
        int data_type;   /* データの形式(vifファイルから取得) */
        ERROR_TYPE err_flag;


        DataType* ImageData = NULL;/* volファイル内のデータを全て格納する領域へのポイ
ンタ*/
        DataType* ResultImg = NULL;

        /* 引数チェック引数はSrcFileName,DstFileNameの二つのみ　それ以上は無視*/
        if(argc < MIN_ARGC)
        {
                fprintf(stderr,"引数の数が不正です. ¥n");
                return ERROR_ARG;
        }

        /* 入力vifファイル名(正式)を保存するメモリを確保*/
        FileName =
(char*)(malloc((strlen(argv[SRC_F_NAME])+strlen(".vif")+1)*sizeof(char)));
        if(FileName == NULL)
        {
                fprintf(stderr,"メモリを確保できませんでした. ¥n");
                return ERROR_MEMORY;
        }

        /* 実際に入出力の対象となるファイルの正式なファイル名を得る*/
        strcpy(FileName,argv[SRC_F_NAME]);
        strcat(FileName,".vif");

        err_flag = VIFReader(FileName,&sizeX,&sizeY,&sizeZ,&data_type);
        switch(err_flag){
                case ERROR_VIF_NOTFOUND:
                        fprintf(stderr,"vifファイルを読み込めません. ¥n");
                        break;
                case ERROR_VIF_SIZEX:
                case ERROR_VIF_SIZEY:
                case ERROR_VIF_SIZEZ:
                        fprintf(stderr,"サイズの読み込みが不正です. ");
                        break;
                case ERROR_VIF_TYPE:
                        fprintf(stderr,"ファイルタイプが不正です. ");
                        break;
        }

        free((void*)(FileName));/* 入力用vifファイル名は使用しないので領域を破棄*/

        if(err_flag != SUCCESS)  return ERROR_VIF_READ;/* vifの読み込みに失敗していた
らプログラム終了*/

        /* 入力volファイル名(正式)を保存するメモリを確保*/
        FileName =
(char*)(malloc((strlen(argv[SRC_F_NAME])+strlen(".vol")+1)*sizeof(char)));
        if(FileName == NULL)
        {
                fprintf(stderr,"メモリを確保できませんでした. ¥n");
                return ERROR_MEMORY;
        }

        /* フィルタをかけたい画像ファイルを開く*/
        strcpy(FileName,argv[SRC_F_NAME]);
        strcat(FileName,".vol");
        if( (fpVol = fopen(FileName,"rb")) == NULL)
```

```
{
        fprintf(stderr,"volファイルが見つかりません. ¥n");
        free((void*)(FileName));
        return ERROR_VOL_NOTFOUND;
}


/* 画像を確認するメモリ領域を確保*/
ImageData = (DataType*)(malloc(sizeX*sizeY*sizeZ*sizeof(DataType)));
if(ImageData == NULL)
{
        fprintf(stderr,"メモリを確保できませんでした. ¥n");
        free((void*)(FileName));
        fclose(fpVol);
        return ERROR_MEMORY;
}

/* フィルタ結果を格納するメモリ領域を確保*/
ResultImg = (DataType*)(malloc(sizeX*sizeY*sizeZ*sizeof(DataType)));
if(ResultImg == NULL)
{
        fprintf(stderr,"メモリを確保できませんでした. ¥n");
        free((void*)(ImageData));
        free((void*)(FileName));
        fclose(fpVol);
        return ERROR_MEMORY;
}

/* 画像を読み込む*/
readSize =
   fread((void*)(ImageData),sizeof(DataType),sizeX*sizeY*sizeZ,fpVol);
if(readSize != sizeX*sizeY*sizeZ)
{
        fprintf(stderr,"volファイルの読み込みに失敗しました. ");
        free((void*)(ImageData));
        free((void*)(ResultImg));
        free((void*)(FileName));
        fclose(fpVol);
        return ERROR_VOL_NOTFOUND;
}
fclose(fpVol);
free((void*)(FileName));

/* フィルタを適用*/
err_flag = sobel_main(ImageData,ResultImg,sizeX,sizeY,sizeZ);
if(err_flag != SUCCESS)
{
        fprintf(stderr,"フィルタ処理に失敗しました. ");
        free((void*)(ImageData));
        free((void*)(ResultImg));
        return ERROR_FILTER_FALT;
}

/* フィルタ結果を出力*/
FileName =
  (char*)(malloc((strlen(argv[DST_F_NAME])+strlen(".vol")+1)*sizeof(char)));
strcpy(FileName,argv[DST_F_NAME]);
strcat(FileName,".vol");
if( (fpVol = fopen(FileName,"wb")) == NULL)
{
        fprintf(stderr,"出力用volファイルを開けません. ");
        free((void*)(ImageData));
        free((void*)(ResultImg));
        return ERROR_VOL_OPENFALT;
}
fwrite(ResultImg,sizeof(DataType),sizeX*sizeY*sizeZ,fpVol);
```

```
        fclose(fpVol);
        free((void*)FileName);

        /* 新しいvifファイルを書き込む*/
        FileName =
        (char*)(malloc((strlen(argv[DST_F_NAME])+strlen(".vif")+1)*sizeof(char)));
        strcpy(FileName,argv[DST_F_NAME]);
        strcat(FileName,".vif");
        if( (fpVif = fopen(FileName,"w")) == NULL)
        {
                fprintf(stderr,"出力用vifファイルを開けません. ");
                free((void*)(ImageData));
                free((void*)(ResultImg));
                return ERROR_VIF_OPENFALT;
        }
        else{
                char tmp[100];
                char* vifName =
                  (char*)(malloc((strlen(argv[SRC_F_NAME])+strlen(".vif")+1)
                    *sizeof(char)));
                FILE* fpoVIF = NULL;
                if(vifName == NULL){
                        fprintf(stderr,"メモリを確保できませんでした. ¥n");
                        free((void*)(FileName));
                        return ERROR_MEMORY;
                }
                strcpy(vifName,argv[SRC_F_NAME]);
                strcat(vifName,".vif");
                if( (fpoVIF = fopen(vifName,"r")) == NULL)
                {
                        fprintf(stderr,"出力用vifファイルを開けません. ");
                        free((void*)(ImageData));
                        free((void*)(ResultImg));
                        free((void*)(FileName));
                        free((void*)(vifName));
                        return ERROR_VIF_OPENFALT;
                }
                while(fgets(tmp,100,fpoVIF))fputs(tmp,fpVif);
                fclose(fpoVIF);
                free((void*)(vifName));
        }
        fclose(fpVif);

        free((void*)(ImageData));
        free((void*)(ResultImg));
        return 0;
}


/* vifファイル内の情報を取得する*/
ERROR_TYPE VIFReader(char* VIFname,int* sizeX,int* sizeY,int* sizeZ,int* data_type){
        FILE* fpFile = NULL;
        char FileInputBuffer[256];/* vifファイルの読み込み用のバッファ*/
        char* lookat=NULL;/* 入力された文字列を捜査するためのポインタ*/

        /* フィルタ適用前のvifファイルを読み込み*/
    if( (fpFile = fopen(VIFname,"r")) == NULL)
                /* 読み取り専用. ファイルがなければエラー終了*/
                return ERROR_VIF_NOTFOUND;

        fgets(FileInputBuffer,256,fpFile);/* 一行目VIF 1.0 .... の行は飛ばす*/
        fgets(FileInputBuffer,256,fpFile);/* 二行目start_pt .... の行もフィルタには関
係ない*/
        fgets(FileInputBuffer,256,fpFile);/* 三行目size ....の行*/
        lookat = skipSpace(FileInputBuffer);
        if(strncmp(lookat,"size",4) == 0)/* 最初の文字が"size"であればサイズを読み込
```

```
む*/
        {
                lookat = skipChar(lookat);/* 項目名「size」を無視*/
                lookat = skipSpace(lookat);
                if( (*sizeX = atoi(lookat)) == 0){
                        return ERROR_VIF_SIZEX;
                };
                lookat = skipChar(lookat);/*次の項目へ*/
                lookat = skipSpace(lookat);
                if( (*sizeY = atoi(lookat)) == 0){
                        return ERROR_VIF_SIZEY;
                };
                lookat = skipChar(lookat);/*次の項目へ*/
                lookat = skipSpace(lookat);
                if( (*sizeZ = atoi(lookat)) == 0){
                        return ERROR_VIF_SIZEZ;
                };
        }

        fgets(FileInputBuffer,256,fpFile);/* 四行目pitch .... の行を飛ばす*/
        fgets(FileInputBuffer,256,fpFile);/* 五行目data_type .... の行*/

        lookat = skipSpace(FileInputBuffer);
        if(strncmp(lookat,"data_type",9) == 0)/* 最初の文字が"size"であればサイズを読
み込む*/
        {
                lookat = skipChar(lookat);/* 項目名「data_type」は無視*/
                lookat = skipSpace(lookat);
                if( (*data_type = atoi(lookat)) != USABLE_DATA_TYPE)
                {
                        return ERROR_VIF_TYPE;
                }
        }

        fclose(fpFile);
        return SUCCESS;
}


ERROR_TYPE sobel_main(DataType* OriginalData,DataType* ResultData,int sizeX,int
sizeY,int sizeZ){
        DataType data_t,data_t1,data_t2;

        int     i, j, k, p, q, r;
        long fnum, fx, fy, fz;
        double* fil1;
        double* fil2;
        double filter1,filter2;
        int filX_Len;
        int fil_Area;

        DataType** temp_faces=   NULL;
        DataType* temp_face1=    NULL;
        DataType* temp_face2=    NULL;
        DataType* temp_face3=    NULL;

        int orgX_Len; /* 元画像のX幅 */
        int org_Area; /* 元画像の面積*/

        int tmpX_Len; /* 作業領域の幅*/
        int tmp_Area; /* 作業領域の面積*/

        int filpos;
        int imgpos;

        /* sobel filterのサイズ*/
```

```
fx = 3; fy = 3; fz = 3;
fnum = fx*fy*fz;

filX_Len = fx;
fil_Area = fx*fy;

orgX_Len = sizeX; /* 元画像のX幅*/
org_Area = sizeX*sizeY; /* 元画像の面積*/

tmpX_Len = (sizeX+2);
tmp_Area = (sizeX+2)*(sizeY+2);

//フィルタ用のメモリ領域
fil1 = (double*)malloc(sizeof(double)*fnum);
if(fil1 == NULL){
        return ERROR_MEMORY;
}
fil2 = (double*)malloc(sizeof(double)*fnum);
if(fil2 == NULL){
        free((void*)(fil1));
        return ERROR_MEMORY;
}

temp_faces = (DataType**)(malloc(sizeof(DataType*)*3));
temp_face1 = (DataType*)(malloc(sizeof(DataType)*tmp_Area));
temp_face2 = (DataType*)(malloc(sizeof(DataType)*tmp_Area));
temp_face3 = (DataType*)(malloc(sizeof(DataType)*tmp_Area));
if( temp_faces == NULL ||
        temp_face1 == NULL ||
        temp_face2 == NULL ||
        temp_face3 == NULL){
                if(temp_faces)free((void*)(temp_faces));
                if(temp_face1)free((void*)(temp_face1));
                if(temp_face2)free((void*)(temp_face2));
                if(temp_face3)free((void*)(temp_face3));
                return ERROR_MEMORY;
}


/* ▼<-- ソーベルフィルタを作成する-------------------- */
for(k=0; k<fz; k++){
        for(j=0; j<fy; j++){
                for(i=0; i<fx; i++){
                        fil1[getDotPos(i,j,k,filX_Len,fil_Area)] = 0.0;
                        fil2[getDotPos(i,j,k,filX_Len,fil_Area)] = 0.0;
                }
        }
}
/* #1 */
fil1[getDotPos(0,1,0,filX_Len,fil_Area)] =
fil1[getDotPos(2,1,0,filX_Len,fil_Area)] =
fil1[getDotPos(0,0,1,filX_Len,fil_Area)] =
fil1[getDotPos(2,0,1,filX_Len,fil_Area)] =-1.0;

fil1[getDotPos(0,2,1,filX_Len,fil_Area)] =
fil1[getDotPos(2,2,1,filX_Len,fil_Area)] =
fil1[getDotPos(0,1,2,filX_Len,fil_Area)] =
 fil1[getDotPos(2,1,2,filX_Len,fil_Area)] = 1.0;
fil1[getDotPos(1,1,0,filX_Len,fil_Area)] =
fil1[getDotPos(1,0,1,filX_Len,fil_Area)] = -2.0;
fil1[getDotPos(1,2,1,filX_Len,fil_Area)] =
fil1[getDotPos(1,1,2,filX_Len,fil_Area)] = 2.0;

/* #2 */

fil2[getDotPos(1,0,0,filX_Len,fil_Area)] =
```

```
fil2[getDotPos(1,2,0,filX_Len,fil_Area)] =
fil2[getDotPos(2,0,1,filX_Len,fil_Area)] =
fil2[getDotPos(2,2,1,filX_Len,fil_Area)] = -1.0;

fil2[getDotPos(0,0,1,filX_Len,fil_Area)] =
fil2[getDotPos(0,2,1,filX_Len,fil_Area)] =
fil2[getDotPos(1,0,2,filX_Len,fil_Area)] =
fil2[getDotPos(1,2,0,filX_Len,fil_Area)] = 1.0;

fil2[getDotPos(1,1,0,filX_Len,fil_Area)] =
fil2[getDotPos(2,1,1,filX_Len,fil_Area)] = -2.0;

fil2[getDotPos(0,1,1,filX_Len,fil_Area)] =
fil2[getDotPos(1,1,2,filX_Len,fil_Area)] = 2.0;
/* ▲-------------------- フィルタ作成ここまで--> */


/* ▼フィルタをかける-------------------------------- */
for(k=0; k<sizeZ; k++){

        /* 以下フィルタに影響するスライスを，z方向に上下一枚を含め枚取得する*/

        /* フィルタ処理の対象がz方向について最初の一枚の時*/
        if(k==0){
                temp_faces[0] = temp_face1;
                temp_faces[1] = temp_face2;
                temp_faces[2] = temp_face3;
                copySlice(temp_faces,0,OriginalData,0,sizeX,sizeY,
                orgX_Len,org_Area,tmpX_Len,tmp_Area);
                copySlice(temp_faces,1,OriginalData,0,sizeX,sizeY,
                orgX_Len,org_Area,tmpX_Len,tmp_Area);
                copySlice(temp_faces,2,OriginalData,1,sizeX,sizeY,
                orgX_Len,org_Area,tmpX_Len,tmp_Area);
        }
        /* 最後の一枚を指す時*/
        else if(k==sizeZ-1){
                temp_faces[0] = temp_face1;
                temp_faces[1] = temp_face2;
                temp_faces[2] = temp_face3;
                copySlice(temp_faces,0,OriginalData,k-1   ,sizeX,
                sizeY,orgX_Len,org_Area,tmpX_Len,tmp_Area);
                copySlice(temp_faces,1,OriginalData,k     ,sizeX,
                sizeY,orgX_Len,org_Area,tmpX_Len,tmp_Area);
                copySlice(temp_faces,2,OriginalData,k     ,sizeX,
                sizeY,orgX_Len,org_Area,tmpX_Len,tmp_Area);
        }
        else{
                temp_faces[0] = temp_faces[1];
                temp_faces[1] = temp_faces[2];
                temp_faces[2] = temp_faces[0];
                copySlice(temp_faces,2,OriginalData,k+1   ,sizeX,
                sizeY,orgX_Len,org_Area,tmpX_Len,tmp_Area);
        }

        /* フィルタリング(取得した三枚のうち，中央一枚をフィルタ処理対象とする．)*/
        for(j=1; j<sizeY+1; j++){
                for(i=1; i<sizeX+1; i++){
                        data_t = 0;
                        data_t1= 0;
                        data_t2= 0;

                        /* for 3x3x3 */
                        for(r=-1; r<2; r++)
                        {
                                for(q=-1; q<2; q++)
                                {
```

```
                                        for(p=-1; p<2; p++)
                                        {

                                                filpos =
                                    getDotPos((p+1),(q+1),(r+1),filX_Len,fil_Area);
                                                imgpos =
                                    getDotPos((i+p),(j+q),0,tmpX_Len, tmp_Area );

                                                filter1 = fil1[filpos];
                                                data_t1 +=
                                    (DataType)(filter1*temp_faces[(r+1)][imgpos]);

                                                filter2 = fil2[filpos];
                                                data_t2 +=
                                    (DataType)(filter2*temp_faces[(r+1)][imgpos]);
                                        }
                                    }
                                }
                                data_t = (DataType)(sqrt(data_t1*data_t1
                                        + data_t2*data_t2));
                                data_t = abs(data_t);

        ResultData[getDotPos((i-1),(j-1),k,orgX_Len,org_Area)] = data_t;
                            }
                    }
            }
            /* ▲フィルタリング------------------------------ */

            /* フィルタ用のメモリを解放*/
            free((void*)(fil1));
            free((void*)(fil2));

            /* 作業用のメモリを解放*/
            free((void*)(temp_faces));
            free((void*)(temp_face1));
            free((void*)(temp_face2));
            free((void*)(temp_face3));

            return SUCCESS;
}


void copySlice(DataType** temp_faces, int tZIndex,DataType* OriginalData, int oZIndex, int
sizeX, int sizeY, int orgX_Len, int org_Area, int tmpX_Len, int tmp_Area){
            int dx,dy;
            for(dy=0 ; dy<sizeY ; dy++){
                    for(dx=0 ; dx<sizeX ; dx++)

                    temp_faces[tZIndex][getDotPos(dx+1,dy+1,0,tmpX_Len,tmp_Area)]
                                    =
                    OriginalData[getDotPos(dx,dy,oZIndex,orgX_Len,org_Area)];
            }

            /* 四辺を埋める*/
            for(dy=0 ; dy < sizeY+2 ; dy++){
                    temp_faces[tZIndex][getDotPos(0,dy,0,tmpX_Len,tmp_Area)] =
                    temp_faces[tZIndex][getDotPos(1,dy,0,tmpX_Len,tmp_Area)];
            }
            for(dy=0 ; dy < sizeY+2 ; dy++){
                    temp_faces[tZIndex][getDotPos(sizeX+1,dy,0,tmpX_Len,tmp_Area)] =
                    temp_faces[tZIndex][getDotPos(sizeX,dy,0,tmpX_Len,tmp_Area)] ;
            }
            for(dx=0 ; dx < sizeX+2 ; dx++){
                    temp_faces[tZIndex][getDotPos(dx,0,0,tmpX_Len,tmp_Area)] =
                    temp_faces[tZIndex][getDotPos(dx,1,0,tmpX_Len,tmp_Area)] ;
            }
```

```
		for(dx=0 ; dx < sizeX+2 ; dx++){
			temp_faces[tZIndex][getDotPos(dx,sizeY+1,0,tmpX_Len,tmp_Area)] =
			temp_faces[tZIndex][getDotPos(dx,sizeY,0,tmpX_Len,tmp_Area)] ;
		}
}


/* 行中の先頭にある一つ以上の半角スペースへのポインタを取得*/
char* skipSpace(char* charBuffer)
{
		while(*charBuffer == ' ' && *charBuffer)charBuffer++;
		return charBuffer;
}

/* 行中の先頭から最も近いスペース以外の文字へのポインタを取得*/
char* skipChar(char* charBuffer)
{
		while(*charBuffer != ' ' && *charBuffer)charBuffer++;
		return charBuffer;
}
```

## 付録4. 3次元画像ファイルの出力サンプル

VDF, VOL, VIF の出力サンプルコードを下記に記します.

※ エラー処理等は考慮されていませんので、ご注意下さい

```
///
/// Raw データの書き出し
/// 下記のメソッド内で使用します。
///
private: bool SaveRAW(BinaryWriter^ bw, array<unsigned char>^ data){
    // Raw データ書き出し
    int count = 0;
    while(count < data->Length){
        bw->Write(data[count]);
        count++;
    }
    return true;
}

///
/// VOL ファイルの書き出し
///
private: bool SaveVOL(
  String^ path,                // 出力先のファイルパス
  array<unsigned char>^ data  // 出力元データ（1 次元配列のボリュームデータ）
){
    // VOL 書き出し
    FileStream^ fs = gcnew FileStream(
                    path, System::IO::FileMode::Create,
                    System::IO::FileAccess::Write, System::IO::FileShare::None);
    BinaryWriter^ bw = gcnew BinaryWriter(fs);

    // VOL ファイルは、実際のところ Raw データそのもの
    SaveRAW(bw, data);

    bw->Close();
    fs->Close();

    return true;
}



///
/// VIF ファイルの書き出し
///
private: bool SaveVIF(
  String^ path,                        // 出力先のファイルパス
  int x, int y, int z,                 // データグリッドサイズ
```

```
    double spx, double spy, double spz, // データ開始位置
    double ptx, double pty, double ptz  // データピッチ
){
    // VIF 書き出し
    FileStream^ fs = gcnew FileStream(
                        path, System::IO::FileMode::Create,
                        System::IO::FileAccess::Write,
                    System::IO::FileShare::None);
    StreamWriter^ sw = gcnew StreamWriter(fs);
    String^ tmpstr = L"";
    sw->NewLine = L"¥r¥n";    // 改行コード
    //全部で 5 行
    // 1 行目
    sw->WriteLine(L"VIF 1.0 VE12.8");
    // 2 行目
    tmpstr = L"start_pt  "
           + spx.ToString() + L" " + spy.ToString() + L" " + spz.ToString();
    sw->WriteLine(tmpstr);
    // 3 行目
    tmpstr = L"size  "
           + x.ToString() + L" " + y.ToString() + L" " + z.ToString();
    sw->WriteLine(tmpstr);
    // 4 行目
    tmpstr = L"pitch  "
           + ptx.ToString() + L" " + pty.ToString() + L" " + ptz.ToString();
    sw->WriteLine(tmpstr);
    // 5 行目
    tmpstr = L"data_type  " + L"1";   // サンプルでは unsigned char 型に固定
    sw->WriteLine(tmpstr);

    sw->Close();
    fs->Close();

    return true;
}


///
/// VDF ファイルの書き出し
///
private: bool SaveVDF(
  String^ path,                         // 出力先のファイルパス
  int x, int y, int z,                  // データグリッドサイズ
  double spx, double spy, double spz,   // データ開始位置
  double ptx, double pty,double ptz,    // データピッチ
  array<unsigned char>^ data  // 出力元データ（1 次元配列のボリュームデータ）
){
    // VDF 書き出し
    FileStream^ fs = gcnew FileStream(
                path, System::IO::FileMode::Create,
                System::IO::FileAccess::Write, System::IO::FileShare::None);
```

```
        BinaryWriter^ bw = gcnew BinaryWriter(fs);

        // ヘッダ文字列     unsigned char 型の例（データタイプ=1。ヘッダ長さは 256 固定）
        String^ str_header = L"VDF_1.0_VE12.8"
                + " sp "     + spx.ToString() + " " + spy.ToString() + " " + spz.ToString()
                + " n "      + x.ToString()   + " " + y.ToString()   + " " + z.ToString()
                + " pitch " + ptx.ToString() + " " + pty.ToString() + " " + ptz.ToString()
                + " dt "     + "1"
                + "¥n";

        // String を Char クラス配列に変換
        array<Char>^ transfer_header = str_header->ToCharArray();
        // 実際に書き出す Char クラス配列
        array<Char>^ put_header_array = gcnew array<Char>(256);
        // 書き出しに使用する配列に、データを転送
        Array::Copy(transfer_header, put_header_array, transfer_header->Length);

        // 未使用列は、0 で埋める
        int count = transfer_header->Length;
        while(count < 256){
            put_header_array[count] = 0;
            count++;
        }

        // ヘッダを書き出す
        count = 0;
        while(count < 256){
            bw->Write(put_header_array[count]);
            count++;
        }

        // データ部分を書き出す。データ部は、Raw データと同じ
        SaveRAW(bw, data);

        bw->Close();
        fs->Close();
        return true;
    }
```

Volume Extractor 3.0 開発編（画像フィルタ作成マニュアル）
2010 年 10 月 24 日 第 1.0 版発行

製作・著作　株式会社 i-Plants Systems
info@i-plants.jp
ve_support@i-plants.jp（VE サポート専用窓口）
019 – 694 – 3103（代表）

http://www.i-plants.jp/hp