

CTView 2.0 入門

-- 画像処理・C++ 教育用 可視化ツール --

1. はじめに

1.1 CTView とは

CTView は、元々医用画像分野で 3 次元 CT (Computed Tomography) 画像処理の研究を行っていた作者が、処理データの可視化用に開発したツールである。その後作者は、諸処の事情により 3 次元 CT 画像処理の研究からは遠ざかってしまったが、大学で画像処理の授業を行うに当たり、CTView を大幅に改造し、汎用の画像処理用可視化ツールとした。

画像処理プログラミングの講義を行うに当たり問題となるのが、「画像処理のプログラムコード(アルゴリズム)」と「画像表示のプログラムコード」の混在である。「画像表示のプログラム」はプラットフォームに依存し、従来「画像処理のプログラム」とは関係の無い物である。講義において教師は「画像処理のアルゴリズム」を学生に教えたいと考えるが、通常はそれと同時に「画像表示のプログラムコード」も提示しなければならず、プログラムが複雑になり「画像処理のアルゴリズム」の本質が見えにくくなる。

CTView はこの問題を解決するために、処理データの可視化の部分を一手に引き受ける。完全とは言えないが、CTView を使用することにより「画像処理のプログラムコード (アルゴリズム)」と「画像表示のプログラムコード」をある程度分離することが可能で、「画像処理のアルゴリズム」の学習を行い易くする。CTView では、画像処理の機能は全てプラグイン (DLL: Dynamic Link Library) として読み込む。

学習者は画像処理を行うプラグイン (DLL) を作成し CTView に読み込ませるだけで良く、処理用データの読み込みと表示、さらに処理後データの表示や保存などは全て CTView 側が行ってくれる。

CTView v2.0 からはカラー対応がサポートされ、JPEG ファイルのデータの読み込み、保存が可能となっている。

1.2 CTView にできること

CTView は幾種類かの画像データファイルを読み込むことが可能である (メジャーなものとしては JPEG)。読み込んだ画像データは自動的に表示される。

CTView は画像データと共に、第三者の作成した DLL をプラグインとして読み込むことが可能である。通常 プラグインは読み込んだ画像データに対して処理を行うが、読み込みデータなしに新しい画像データを生成することも可能である。

読み込んだ画像データもしくは処理後の画像データで、Z 方向成分をもつ画像に対しては、マルチスライス表示やサムネイル表示、3 次元のボリューム表示 (サーフェイスレンダリング) など可能である。また、処理後の画像データの保存も幾種類かのフォーマットで行うことができる。

2. CTVIEW の概要

2.1 セットアップ

2.1.1 ダウンロードとインストール

CTView の最新版は <http://www.nsl.tuis.ac.jp/xoops/modules/xpwiki/?CTView> 内のリンクからダウンロード可能である。2011 年 7 月の段階で MicroSoft Visual Studio (以下 VS) 2005 用と VS 2008 用があるので、自分の開発環境にあったパッケージをダウンロードする。なお、VS 2010 用は用意されていないが、プログラム全体を VS 2010 で再コンパイルし直せば、VS 2010 でも使用可能である。

CTViewではインストール作業は特に必要はない。ダウンロードしたパッケージを任意のフォルダで解凍すれば、解凍したディレクトリでそのまま実行可能である。

2.1.2 CTVIEW の動作環境

CTViewを使用するためには DirectXのランタイムが必要である。DirectXがインストールされていない、もしくはインストールされている DirectX のバージョンが古い場合には「d3dx9_##.dll が見つからない」というようなメッセージが出て起動しない場合がある(##の部分には数字が入る)。この場合は DirectXの最新版をインストールするか、<http://jp.dll-download-system.com/> から足りない DLL をダウンロードして、CTView.exe と同じディレクトリに配置する。

例えば d3dx9_43.dll の場合は <http://jp.dll-download-system.com/docman/d-dlls-not-system-/d3dx9-43.dll/details.html> より d3dx9_43.dll をダウンロードする。

2.1.3 プラグインの開発環境

プラグインを開発するためには、VS 2005 もしくは VS 2008 が必要である。ただし、VS 2005、VS 2008 以外でも、プログラム全体をコンパイルし直せば、その VS で使用することは可能である(2.1.4 参照)。

VS のデバッグモード (Debug のソリューション構成) でコンパイルを行う場合は、別途 **MagaDebug32 (magdb142)** が必要である。MagaDebug32 は下記 URL よりダウンロード可能である。

<http://s-gikan2.maizuru-ct.ac.jp/old/magadbg.html>

<http://www.vector.co.jp/soft/win95/prog/se065884.html>

ダウンロード後、解凍してできたディレクトリ (magdb142) を CTVIEW のディレクトリ内 (JunkBox_Lib++ などと同じ階層) に設置する。

2.1.4 CTVIEW の再コンパイル

CTViewを配布されているバージョン以外の VS で使用する場合などには、CTView全体を再コンパイルを行う必要がある。CTViewを再コンパイルするには、配布しているものの他に、**DirectX の SDK**、**libjpeg**、**MagaDebug32** が必要である。

libjpeg は <http://www.ijg.org/> からダウンロード可能であるが、パッケージを解凍後、手動でコンパイルする必要がある。VS のコマンドプロンプトを起動して、解凍ディレクトリに移動後、以下のコマンドを実行する。

```
> copy jconfig.vc jconfig.h
> nmake -f makefile.vc
```

以上の追加パッケージをインストール後、JunkBox_Lib++、JunkBox_Win_Lib、CT、CTView¥CTMFC_Lib、CTView¥CTView の順にコンパイルを行う。ただし、インストールした DirectX の SDK、libjpeg、MagaDebug32 のバージョンによっては、各プロジェクトファイルのインクルードファイルのパス (プ

ロパティ→C/C++→全般→追加のインクルードディレクトリ) およびライブラリファイルのパス (プロパティ→リンカ→全般→追加のライブラリディレクトリ)を変更する必要がある(特にDirectX SDKのパス)。

2.2 CTVIEW のデータ構造

CTViewのデータの構造を図2-1に示す。図2-1のメインデータは、処理対照または処理後のデータでCTViewの中に定常的に保存されるデータである。他の一時的画像データ、一時的ボリュームデータ、表示用データなどは、表示ウィンドウがクローズされれば自動的に削除されるが、メインデータは表示ウィンドウがクローズされても、明示的に削除コマンドを実行するか、データを置き換える(図2-1の⑥)までメモリ中に存在し続ける。

図2-1中の「プラグイン」の部分で、学習者が作成するプラグインである。CTView自身はほとんど画像データの処理は行わないので、メインの画像処理はここで行うことになる。

プラグインは DLLとして実装され、CTViewは起動時に特定ディレクトリ (pluginsディレクトリ)にある複数のプラグインを自動的に読み込む。また、起動後にプラグインを差し替えるために、プラグインの開放や再読み込み機能を持つ。

2.3 サポートする画像データ

CTView v2.0 がサポートする画像データは以下の通りである。

- ・Common Head 独自形式。他の画像データのヘッダ情報を内包できる。
- ・Moon (CT) 独自形式。2次元CT画像用。Dicomのサブセット版
- ・Dicom 医用画像フォーマット
- ・Sun Raster 8・16bit/pixel 圧縮なし。
- ・JPEG 各種JPEGデータ。保存時の品質は常に100%。libjpegを使用 (<http://www.i.jg.org/>)。
- ・3D VOL 独自形式。3次元画像(ボリュームデータ)用

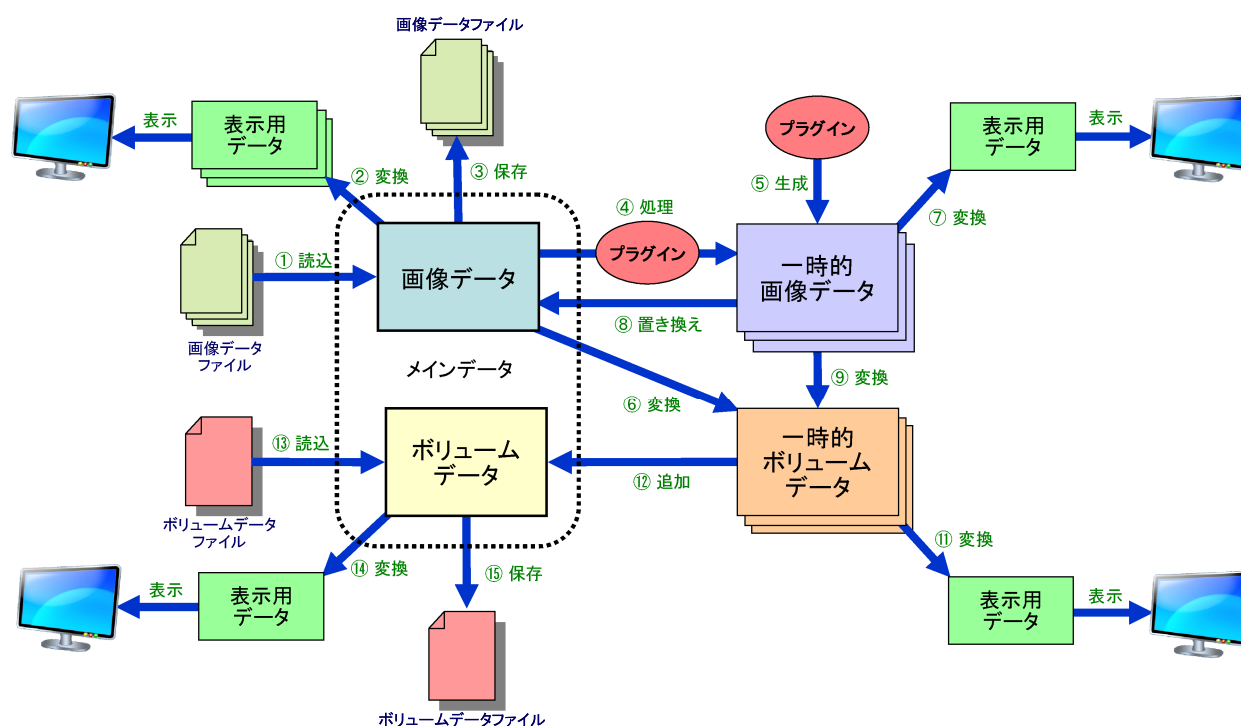


図2-1 CTVIEW のデータ遷移 (図中の番号は処理順序ではない)

その他、非圧縮データであればヘッダの構造指定により読み込み可能である。また、CTViewは内部的にはアルファチャンネルも取り扱い可能であるが、現時点ではアルファチャンネルをサポートするPNGファイルなどの読み込みや保存には対応していない（今後対応予定）。

2.4 操作の概要

2.4.1 ファイルメニュー「新規ファイル読み込み」

ファイルメニュー（図2-2）の「新規ファイル読み込み」を選択すると、指定された画像ファイルのデータがメインの画像データとして読み込まれる（図2-1の①）。画像ファイルのフォーマットは自動認識され、CT画像の場合は、さらにマルチスライス（複数データ）を読み込むためのダイアログが表示される（図2-3）。

この「ファイル読み込み設定」ダイアログでは、画像ファイル名に %d など指定することにより、ファイル名を可変にすることができる（例えば file.1 から file.32 を読み込む場合は、ファイル名を file.%d とし、From に 1, To に 32を指定する）。

ファイル読み込み時に既にメインの画像データが存在し、そのデータを使用したウィンドウが表示されている場合は、それらのウィンドウを全て閉じるように要請するダイアログが表示される。メインの画像データを使用したウィンドウが表示されていない場合は、何の警告もなしにメインの画像データは上書きされる。読み込んだデータは自動的にウィンドウ上に表示される（図2-1の②）。

なお、「2.3 サポートする画像データ」に挙げたファイルフォーマット以外でも、DirectXが表示可能なデータは自動的にウィンドウ上に表示されるが、これはDirectXの内部処理により表示されているのだけであって、CTView内のメインの画像データとして取り込まれている訳ではない（例えばPNGファイルなど）。従ってこの場合は、タイトルバーに「未サポートファイル形式[処理不可]」と表示され、読み込んだデータに対してプラグインなどによる加工処理等を行うことはできない。

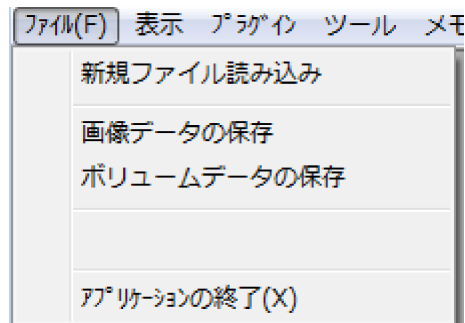


図2-2 ファイルメニュー

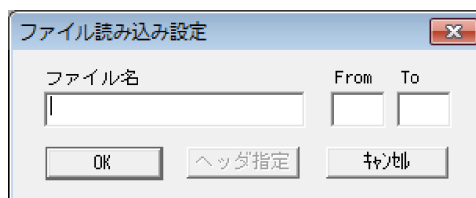


図 2-3 ファイル読み込み設定ダイアログ

2.4.2 ファイルメニュー「画像データの保存」

「画像データの保存」を選択すると、メインの画像データのファイルへの保存が行われる。このメニューを選択した場合、まず保存ディレクトリと仮の保存ファイル名を指定するダイアログが表示される。その後、保存するファイルのフォーマットなどの詳細を指定するためのダイアログが表示される（図2-4）。

なお、JPEG形式で保存する場合、クオリティは常に100%となる。また、16bitカラーのデータをJPEG(24bitカラー)で保存した場合には、色の精度が低下するので注意が必要である。

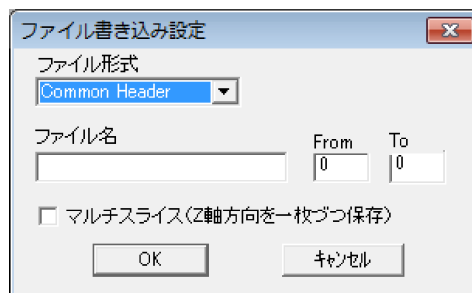


図 2-4 ファイル書き込み設定ダイアログ

2.4.3 ファイルメニュー「ポリウムデータの保存」

「ポリウムデータの保存」を選択すると、メインのポリウムデータのファイルへの保存が行われる。保存形式（フォーマット）は独自形式で、他のアプリケーションで読み込むことはできない。

2.4.4 ファイルメニュー「アプリケーションの終了」

このメニューを選択すると、プログラムは終了する。プログラム実行中に変更した設定等の保存は行われない。

2.4.5 表示メニュー「画像データの表示」

表示メニュー（図2-3）の「画像データの表示」を選択すると、メインの画像データの表示が行われる（図2-1の②）。

なお、CTViewでは内部の画像データは常に16bitで扱われるが、モノクロデータの場合、8bitに変換されて表示が行われる（256階調）。

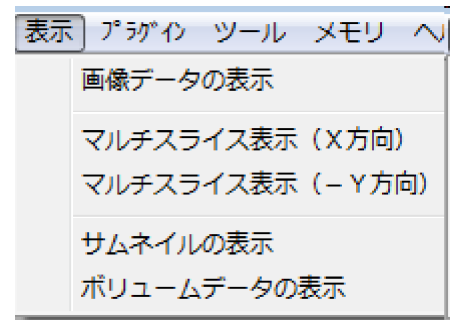


図2-5 表示メニュー

2.4.6 表示メニュー「マルチスライス表示」

「マルチスライス表示」を選択した場合、メインの画像データがマルチスライス（3次元画像）であれば、X方向または-Y方向から見た画像が表示される（右手系）。

2.4.7 表示メニュー「サムネイルの表示」

「サムネイルの表示」を選択した場合、メインの画像データのサムネイルが表示される。RBG, ARGB, RGBAカラーの場合は、各チャネル（モノクロ）のサムネイルが表示される。各サムネイルをクリックすると、その画像が表示されるが、この画像も必ずモノクロとなる。従って、RBG, ARGB, RGBAカラーのサムネイルをクリックした場合は、各チャネルの画像（モノクロ）を表示させることができる。

2.4.8 表示メニュー「ボリュームデータの表示」

「ボリュームデータの表示」を選択した場合、メインのボリュームデータの3次元表示が行われる。表示はサーフェイスレンダリングによって行われ、物体（ボリュームデータ）の内部は実際には処理されない（速度向上のため）。表示中にマウスのホイールで表示の拡大・縮小が可能であるが、拡大しすぎるとボクセル間の隙間ができて、向こう側が透けてしまう場合がある（バグ？）。

2.4.9 プラグインメニュー

CTViewにプラグインを読み込ませた場合、ここにタイトルが表示され、それを選択することによりプラグインが機能する。

2.4.10 ツールメニュー「DLLの開放」

ツールメニュー（図2-6）の「DLLの開放」では、読み込んだプラグインが開放される。プログラムの作動中にプラグインを削除したい場合などに使用する（プラグインが読み込まれている間は、ファイルがロックされるので、そのプラグインを削除することはできない）。



図2-6 ツールメニュー

2.4.11 ツールメニュー「DLL読み直し」

「DLL読み直し」を選択した場合は、読み込んだプラグインが一旦開放され、その後再読み込みされる。プログラムの作動中にプラグインを追加する場合などに使用する。

2.4.12 ツールメニュー「ファイルヘッダ設定」

「ファイルヘッダ設定」を選択すると、ファイル読み込みのためのヘッダ指定のダイアログが表示される。

サポートするファイル以外で、ファイルデータの構造が分かっている場合には、このダイアログでデータファイルの構造を指定してからファイルを読み込ませることができる。なおこの設定は、「ヘッダの指定を有効にする」にチェックを入れないと有効にはならない（図 2-7）。

追加機能として、ファイルのデータがリトルエンディアンの場合には「Little Endian」にチェックを入れることで対応できる。また「基底値の指定」では、ここで指定した値が、読み込んだ画像データの全画素値に自動的に足し算される（主に CT 画像用）。

2.4.13 ツールメニュー「動作設定」

このメニューではプログラムの動作モードを指定することができる。ただし、「ボリュームの色付表示」以外は殆ど変更する必要はない（図 2-8）。

「ボリュームの色付表示」にチェックを入れると、ボリュームデータの表示時に各ボクセルの値(16bit)がA4R4G4B4カラーとして扱われ、表示される。

2.4.14 メモリメニュー「画像データの削除」

メモリメニュー（図 2-9）の「画像データの削除」を選択すると、メインの画像データの削除が行われる。ただし、メインの画像データを使用したウィンドウが表示されている場合は、それらのウィンドウを全て閉じるように要請するダイアログが表示される。

2.4.15 メモリメニュー「ボリュームデータの削除」

「ボリュームデータの削除」では、メインのボリュームデータが削除される。メインのボリュームデータを使用したウィンドウが表示されている場合は、それらのウィンドウを全て閉じるように要請するダイアログが表示される。

2.4.16 メモリメニュー「メモリ参照数の確認」

CTViewはメモリ節約のためいくつかのデータを共有使用している。このメニューは、その共有状態が表示される。尤もこれはデバッグ用なので、一般ユーザは気にする必要はない。

2.4.17 ヘルプメニュー「画像データ情報」

ヘルプメニュー（図 2-10）の「画像データ情報」では、メインの画像データの情報が表示される。

2.4.18 ヘルプメニュー「CTVIEW のバージョン情報」

CTViewのバージョン情報が表示される。

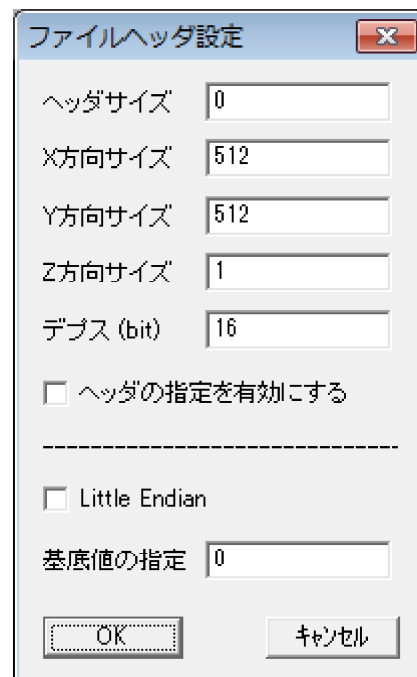


図 2-7 ファイルヘッダ設定

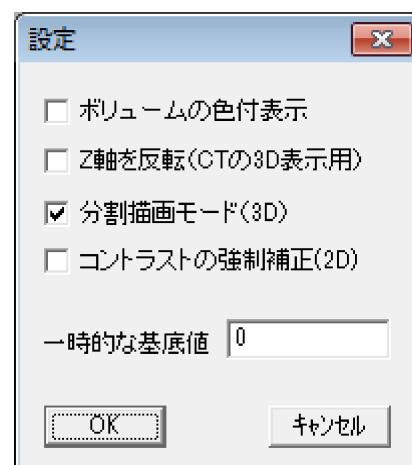


図 2-8 動作設定

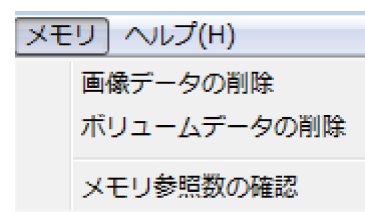


図2-9 メモリメニュー

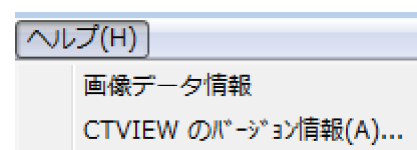


図2-10 ヘルプメニュー

3. CTVIEW 操作例


3.1 起動

CyTView は、「2.1.1 ダウンロードとインストール」によりダウンロード・解凍したディレクトリにある CTVIEW.exe をダブルクリックにより起動する。もし DirectX のバージョンに関するエラーが表示された場合には、「2.1.2 CTVIEW の動作環境」を参考にする。また、DirectX の初期化のエラーが表示された場合は、起動タイミングの問題である場合が大半なので、再起動を試みる。

3.2 画像データファイルの読み込み

CTView が起動したら、ファイルメニューから「新規ファイル読み込み」を選択し、CTView のディレクトリに存在する Sample_data\subaru.ras を読み込んでみる（図 2-1 の①）。このデータファイルは 8bit の SUN RASTER である。

3.3 画像データのコントラストの変更

読み込んだ画像データの表示コントラストを変更することが可能である。画像データを表示するウィンドウのツールボックスの左端のボタン  をクリックすると、コントラスト変更用のダイアログが表示される（図 3-1）。このダイアログのスライダーを動かすか、または直接数値を入力することにより、表示データのコントラストをコントロールすることが可能である。

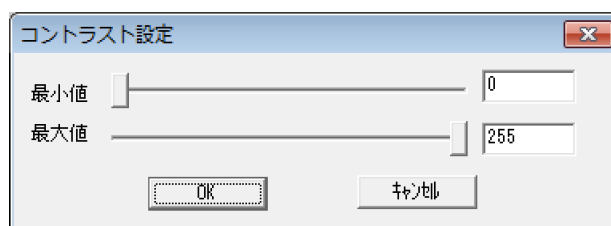



図 3-1 コントラスト調整用ダイアログ

ただし、この処理により変更されるのは表示用データのコントラストであり、メインの画像データそのものは何の変更も受けない。

3.4 二値化プラグインの適用

ダウンロードした CTVIEW には、予め「二値化」と「3D 図形」のプラグインが組み込まれているが、ここではプラグインメニューから「二値化」を選択する（図 2-1 の④）。すると、値を入力するためのダイアログが表示されるので、二値化のしきい値（255 以下）を入力すれば、その処理結果の画像データが表示される（図 3-2。図ではしきい値 150 の結果が示されている）。

3.5 メインの画像データの置き換え

読み込んだメインの画像（元画像）をクローズし、処理後の画像のツールボックスの「黄色の左矢印」  をクリックすれば、二値化後の画像がメインの画像データとなる（図 2-1 の⑧）。

3.6 画像データの保存

二値化後の画像がメインの画像になったら、ファイルメニューから「画像データの保存」を選択する（図 2-1 の③）。保存ディレクトリと仮の保存ファイル名を決めるダイアログ



図 3-2 元画像と二値化後の画像（しきい値 150）

が表示されるので、それらを決める。

その後、保存の詳細を設定するための「ファイル書き込み設定」ダイアログが表示されるので、ファイルの形式（例えば SUN RASTER）、ファイル名を最終的に決定し、OK ボタンをクリックする。


なお、今回は2次元画像なので、マルチスライスに関する設定は不要である。

3.7 ボリュームデータファイルの読み込み

次に、ボリュームデータを読み込んで表示するために、ツールメニューから「動作設定」を選択し、「ボリュームの色付表示」にチェックを入れ、OK ボタンをクリックする。

ファイルメニューから「新規ファイル読み込み」を選択し、Sample_data¥sphere.vol を読み込む（図 2-1 の⑬）。これは、球が少し欠けた形をした3次元画像である（図 3-3. 200x200x200 なのでボクセルが少し荒い）。

3.8 ボリュームデータ作成プラグインの適用

プラグインメニューから、今度は「3D画形」を選択すると（図 2-1 の⑤）、真っ黒な一時的画像データのウィンドウが表示される。ウィンドウのツールボックスの「緑色のボックス」をクリックすると（図 2-1 の⑨）、この画像の3次元のボリュームデータ（一時的なボリュームデータ）が表示される（図 3-3）。

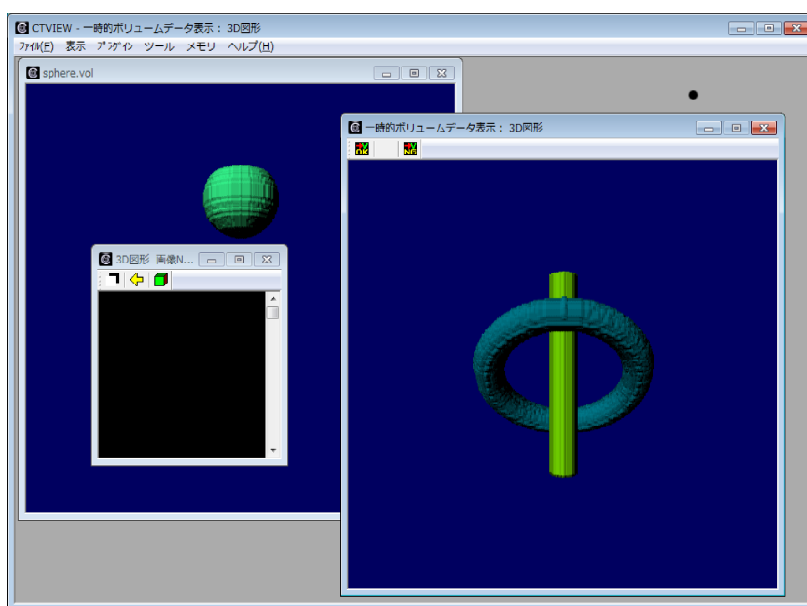



図 3-3 ボリュームデータの表示

3.9 ボリュームデータの合成

一時的なボリュームデータのツールボックスの左側の OK の文字があるボタン  をクリックすると、このデータをファイルから読み込んだボリュームデータに合成することができる（図 2-1 の⑫）。合成したデータはその場で表示される（図 2-1 の⑭）。

4. プラグインの作成の概要

4.1 開発環境

プラグインの開発環境では VS 2005 または VS 2008 が必要である。詳細については、「2.1.3 プラグインの開発環境」を参照すること。

4.2 プラグインの構成要素

各プラグインには変数名 **vp** の画像データインスタンスが渡される（図 4-1）。プラグイン中でこの画像データから新しい画像データ（図 4-1 では **xp**）を生成して、リターンすれば良い。なお、**vp** はメインの画像データであるので、プラグイン中でこのデータの内容を書き換えて

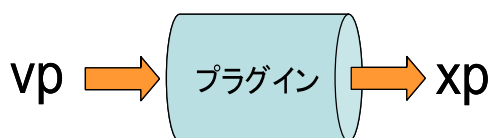


図 4-1 プラグインへの入力と出力

はいけない（書き換えた場合の動作は保証されない）。

画像データの新しいインスタンスは、以下命令で作成することができる。

```
MSGraph<sWord>* xp = new MSGraph<sWord>(xs, ys);
```

ここで、**sWord** は符号付の 16bit を意味し、画像データ **xp** の各ピクセル（画素）が 16bit であることを表す。**sWord** は CTVIEW 内で定義された型であり、符号なしの 16bit の場合は **uWord** を使用する。CTVIEW 内で使用される画像データは（表示の場合を除いて）1 ピクセル（画素）が 16bit であるので、プラグイン内では画像データの型として必ず **sWord** か **uWord** を使用する。

xs, **ys** は画像データ **xp** の X 方向サイズと Y 方向サイズである。3 次元画像や RGB, ARGB, RGBA カラーを扱う場合は、さらに Z 方向のサイズを付け加えることもできる。なお、3 次元画像の場合、座標は右手系である。

プラグインに渡される画像データ **vp** で頻繁に使用されるメンバ変数及びメンバ関数は以下の通りである。

vp->xs	: 画像データの X 方向サイズ。
vp->ys	: 画像データの Y 方向サイズ。
vp->zs	: 画像データの Z 方向サイズ。2D 画像の場合は 1。
vp->max	: 画像データの画素値の最大値。
vp->min	: 画像データの画素値の最小値。
vp->color	: 画像データのカラーモード。
vp->point(i, j)	: 座標 (i, j) の画素値への参照。i, j に続いて、Z 方向の座標も指定可。
vp->free()	: 画像データを開放する。

4.3 プラグイン DLL の作成

4.3.1 プラグイン DLL の雛形

プラグインを作成する場合、**Sample_src\General** フォルダを雛形とすればよい。**Sample_src** フォルダ内で **General** フォルダをコピーし、フォルダに適当な名前を付け直す。その後、フォルダ内の **VS** のプロジェクトファイルをダブルクリックすれば **VS** が起動する。

プロジェクトの名前等は変更しても構わないが、特に変更しなくともプラグインの開発には何の影響も及ぼさない。

4.3.2 雛形の書き換え

ソースコードで書き換えるのは **General.cpp** のみで、他は特に書き換える必要はない。まず、**General.cpp** の **get_info** 関数で「ここにメニュータイトルを書く」という箇所に、プラグインのタイトル名を書く。このタイトル名が CTVIEW のプラグインメニューに表示される。

続いてメインの処理関数である **general_proc** 関数で、「以下に処理コードを書く」から「処理コードここまで」の間に処理コードを記述し、新しい画像データのインスタンスを生成して返せばよい。

もし、数値データの入力が必要なら、「以下に処理コードを書く」のコメントの前の「数値入力用ダイアログ」のコメントを外す。**General.cpp** の数値入力ダイアログでは、入力された数値（整数）は **val** という変数に格納される。

4.3.3 DLL の生成と配置

get_info と **general_proc** の記述が終わったら、ソリューション構成を **Release** にして、ビルドメニューから「General のビルド」を選択する。ソリューション構成に **Debug** を選択しても構わないが、この場合は「2.1.3 プラグインの開発環境」にあるように、**MagaDebug32** が必要となる。

初めてコンパイルする場合は、幾つかのファイルを保存するようダイアログがポップアップするが、この場合はそのままファイルの保存を行う。

コンパイル時の出力には「警告: Warning」が大量に出るかも知れないが、エラーでないかぎり

無視してもかまわない。コンパイル・リンクが正常に終了すれば、**Release**ディレクトリ（ソリューション構成が **Debug** の場合は **Debug** ディレクトリ）に **General.dll** が生成されるので、名前を変更して **CTView.exe** と同じ階層にある **plugins** ディレクトリにコピーする。

CTView は起動時または、ツールメニューの「DLL の読み直し」選択時に、自分と同じ階層にある **plugins** ディレクトリの中を検索して、読み込み可能な DLL が存在すれば自動的に読み込む。

5. プラグイン例1（二値化）

二値化処理（図 5-1）のプラグインの **get_info** 関数と メインの処理関数である **bin_proc** の例を図 5-2 と図 5-3 に示す。メイン関数の **bin_proc** は「4.3.2 雛形の書き換え」の説明中にあるメイン関数（**general_proc**）と名前が違っているが、実はこの関数の名前は **get_info** 関数の中で自由に行うことが可能あり、一定の名前にしなければならないという制約はない。

図 5-3 の二値化では、入力されたしきい値以上の画素値は全て元の画像の最大値に、しきい値より小さい画素値は全て元の画像の最小値に変換されている（図 5-1）。

図 5-3 の二値化プラグインは、3次元画像にも対応している。また **vp->color** にはメインの画像データのカラーモードが格納されており、処理結果の画像データにそれを引き継がせることにより、カラー画像（JPEG）の二値化にも対応している。

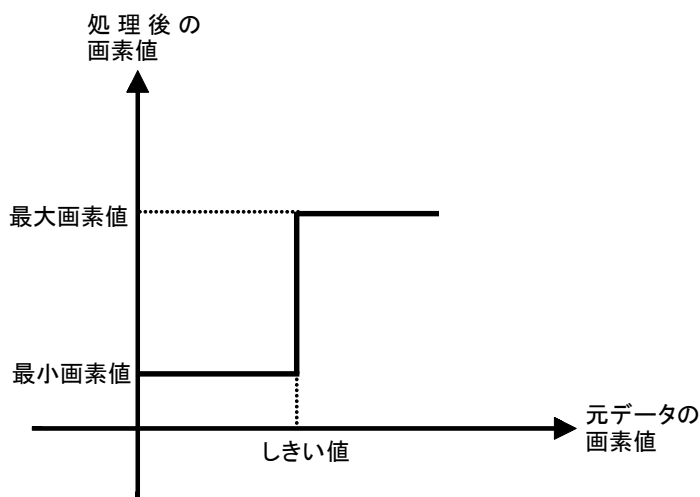


図 5-1 二値化処理による画素値の変換

```

BIN_API char* get_info(int n)
{
    switch(n) {
        case 0 :
            return "二値化";          // ここにメニュータイトルを書く
        case 1 :
            return "bin_proc";        // 処理用関数の名前
        case 2 :
            return "bin_free";        // 処理後のメモリ開放用関数の名前
        case 3 :
            return "bin_active";      // アクティブ判定の関数の名前
        default:
            return NULL;
    }
}

```

図 5-2 二値化プラグインの get_info() 関数

```

BIN_API MSGraph<sWord>* bin_proc(MSGraph<sWord>* vp)
{
    MSGraph<sWord>* xp = NULL;

    // 整数値入力用ダイアログ
    int val = 0;
    BOOL isok = InputNumDLG("しきい値", &val);
    if (!isok) {
        xp = new MSGraph<sWord>();
        xp->state = ERROR_GRAPH_CANCEL;
        return xp;
    }

    //////////////////////////////////////
    // 以下に処理コードを書く
    //////////////////////////////////////

    xp = new MSGraph<sWord>(vp->xs, vp->ys, vp->zs);

    for (int k=0; k<xp->zs; k++) {
        for (int j=0; j<xp->ys; j++) {
            for (int i=0; i<vp->xs; i++) {
                if (vp->point(i, j, k) >= val) xp->point(i, j, k) = vp->max;
                else xp->point(i, j, k) = vp->min;
            }
        }
    }

    xp->color = vp->color;          // カラーモード

    //////////////////////////////////////
    // 処理コードはここまで
    //////////////////////////////////////

    return xp;
}

```

図 5-3 二値化プラグインのメイン関数 bin_proc()

6. プラグイン例2（画像の微分）

6.1 エッジ抽出

画像のエッジを抽出するためには、画素値の微分を計算する。画素値の微分と言っても、画像処理では隣接するピクセル間の差分値を計算すれば良い。ただし、差分値の計算の仕方には幾通りもあが、ここでは尤も単純な方法を紹介する。

$f(i, j)$ の X 方向の差分は $f(i, j) - f(i-1, j)$ で計算できるが、よく考えればこれは ピクセル (i, j) と ピクセル $(i-1, j)$ の中間の差分値となる。従って ピクセル (i, j) の差分値を求めるためには、 $f(i+1, j) - f(i, j)$ との平均を求めれば良い（図 6-1）。つまり、ピクセル (i, j) での X 方向の差分値（微分値） $fx(i, j)$ は以下ようになる。

$$fx(i, j) = \{f(i+1, j) - f(i, j) + f(i, j) - f(i-1, j)\} * 1/2 = \{f(i+1, j) - f(i-1, j)\} * 1/2$$

同様に Y 方向の微分 $fy(i, j)$ は以下ようになる

$$fy(i, j) = \{f(i+1, j) - f(i, j) + f(i, j) - f(i-1, j)\} * 1/2 = \{f(i+1, j) - f(i-1, j)\} * 1/2$$

X 方向の微分（差分）をプログラムで表すと、プラグインのメイン処理関数は図 6-2 のようになる。差分値には負の数も表れるので、ここでは差分値の絶対値を計算してる。絶対値を計算しない場合、CTView は自動的にコントラストを調整し、画素値の中に負の数が現れないように、画素値の底上げを行う。

図 6-3 に X 方向の微分（差分）の結果を示す。

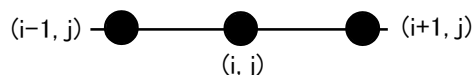


図 6-1 X 方向の差分

```
xp = new MSGraph<sWord>(vp->xs, vp->ys, vp->zs);
for (int j=1; j<vp->ys-1; j++) {
    for (int i=1; i<vp->xs-1; i++) {
        xp->point(i, j) = (vp->point(i+1, j) - vp->point(i-1, j))/2;
    }
}

// 絶対値の計算
for (int j=0; j<vp->ys; j++) {
    for (int i=0; i<vp->xs; i++) {
        if (xp->point(i, j)<0) xp->point(i, j) = - xp->point(i, j);
    }
}
```

図 6-2 X 方向の差分値の計算

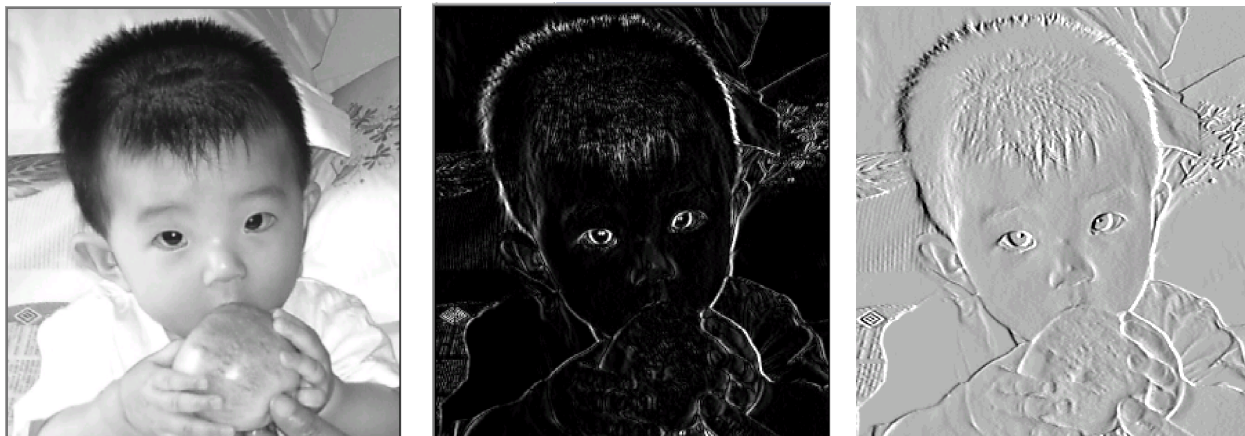


図 6-3 X 方向の微分（差分） 左側：元画像，中央：差分の絶対値，右側：負の値を底上げ
中央と右側の結果は印刷のためにコントラスト調整を行っている

6.2 フィルタ

フィルタとは、画像変換において元の画像に対する叩き込み（コンボリューション）の係数の重さを表した行列である。

例えば図6-4において、変換後（処理後）の画像の画素Xが、元画像の画素値（a, b, c, d, e, f, g, h, i）から $X = A*a + B*b + C*c + D*d + E*e + F*f + G*g + H*h + I*i$ のように計算される時、（A, B, C, D, E, F, G, H, I）の行列（図6-4の中央）を3×3のフィルタと呼ぶ。

前項のX方向の微分の計算式を3×3のフィルタで表せば、以下のようになる。

$$\begin{array}{ccc} 0 & 0 & 0 \\ -1/2 & 0 & 1/2 \\ 0 & 0 & 0 \end{array}$$

同様に Y方向の微分のフィルタは以下のようになる。

$$\begin{array}{ccc} 0 & -1/2 & 0 \\ 0 & 0 & 0 \\ 0 & 1/2 & 0 \end{array}$$

CTView ではフィルタ形式でプログラムを作成することも可能で、例えばX方向の微分の場合は図6-5のようになる（サンプルプログラム Sample¥Different も参照せよ）

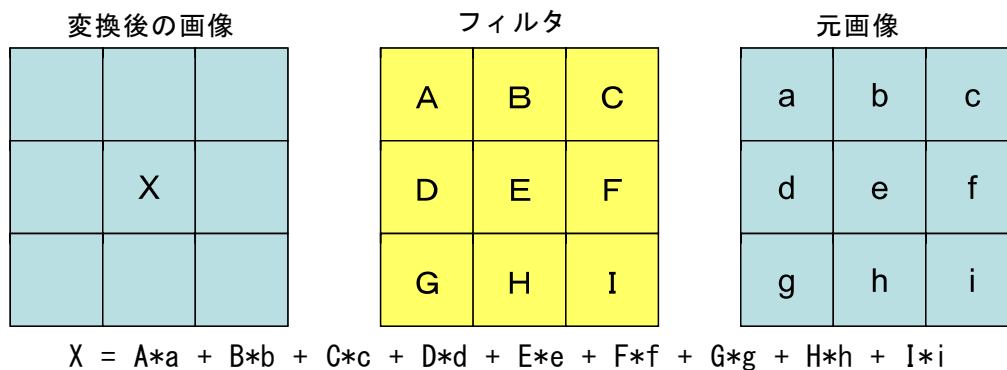


図6-4 フィルタ

```
sWord mask[] = { 0, 0, 0,
                 -1, 0, 1,
                 0, 0, 0 };

// Filter の作成
MSGraph<sWord>* filter = new MSGraph<sWord>(3, 3); // フィルターのサイズを正確に指定する
filter->set_array(mask); // フィルターへコピー
filter->norm = 2.0; // 規格化の係数

xp = new MSGraph<sWord>();
*xp = MSMaskFilter(*vp, *filter);

// Filter の開放
filter->free();
delete(filter);
```

図6-5 フィルタを使用したX方向の差分値の計算

6.3 | ∇f の計算

画像 f の X 方向微分と Y 方向微分の両方を考慮したものはベクトル (∇f : ∇ はナブラ) となる. 従って, 画素 (i, j) に於ける画素値の勾配は ∇f の絶対値 $|\nabla f|$ となる.

$$\nabla f(i, j) = f_x(i, j) \cdot \mathbf{ex} + f_y(i, j) \cdot \mathbf{ey} \quad \mathbf{ex}, \mathbf{ey} \text{ はそれぞれ } X \text{ 方向と } Y \text{ 方向の単位ベクトル}$$
$$\therefore |\nabla f(i, j)| = \sqrt{f_x(i, j)^2 + f_y(i, j)^2}$$

$|\nabla f|$ を計算するためのプログラム (プラグインのメイン処理部分) を図 6-6 に示す. またそのプログラムによる処理結果を図 6-7 に示す. 図 6-7 は, 見やすくするためにコントラスト調整を行っているが, 図 6-3 の中央の X 方向微分のみの結果と比較して, (Y 方向の微分による) 横 (X) 方向のエッジが鮮明に抽出されている.

```
xp = new MSGraph<SWord>(vp->xs, vp->ys, vp->zs);

double dx, dy;
for (int j=1; j<vp->ys-1; j++) {
    for (int i=1; i<vp->xs-1; i++) {
        dx = (vp->point(i+1, j) - vp->point(i-1, j))/2.;
        dy = (vp->point(i, j+1) - vp->point(i, j-1))/2.;
        xp->point(i, j) = (SWord)(sqrt(dx*dx + dy*dy) + 0.5);
    }
}
```

図 6-6 | ∇f の計算プログラム



図 6-7 | ∇f の処理結果
印刷のためにコントラストの調整を行っている

6.4 ラプラシアン計算と画像の鮮鋭化

関数 $f(i, j)$ の2階微分 $\Delta f(i, j)$ は以下のように記述できる.

$$\Delta f(i, j) = \nabla^2 f(i, j) = f_{xx}(i, j) + f_{yy}(i, j)$$

Δ (ラプラシアン) を画像に適用した場合のイメージは図 6-8 を参照されたい. この図 6-8 から分かる通り, 元画像からラプラシアンを引き算するとエッジを強調することになる (ただしラプラシアンの重みは任意).

最も単純な $f(i, j)$ の2階微分の計算では, $f(i-1, j)$ と $f(i, j)$ の差分値である $f(i, j) - f(i-1, j)$ と, $f(i, j)$ と $f(i+1, j)$ の差分値である $f(i+1, j) - f(i, j)$ のさらなる差分を計算する.

つまり, 以下ようになる.

$$\begin{aligned} f_{xx}(i, j) &= \{f(i+1, j) - f(i, j)\} - \{f(i, j) - f(i-1, j)\} \\ &= f(i+1, j) - 2 * f(i, j) + f(i-1, j) \end{aligned}$$

$f_{yy}(i, j)$ も考慮すると, 図 6-9 の 4 近傍ラプラシアンフィルタの 3×3 のフィルタが得られる. またこれを 8 近傍に拡張すれば, 図 6-10 の 8 近傍ラプラシアンのフィルタが得られる. また, Sobel の2階微分をテイラー展開により求めた場合の 5×5 のラプラシアンフィルタを図 6-11 に示す.

図 6-12 に 4 近傍ラプラシアンフィルタによる画像の鮮鋭化のプログラムを示す. 他のプログラムについてはパッケージに添付されている `Sample_src\Sharpness` ディレクトリを参照されたい.

また図 6-13 にこれらのラプラシアンフィルタによる画像の鮮鋭化の処理結果を示す.

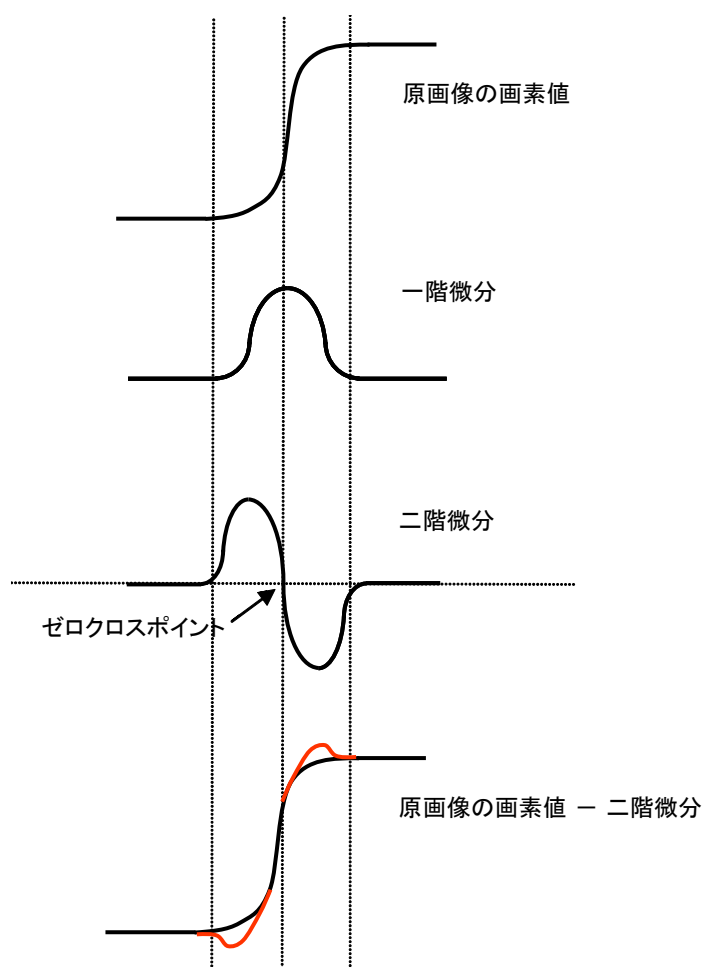


図 6-8 ラプラシアンのイメージ

$$\begin{array}{ccc} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{array} \quad \text{規格化係数 } 1$$

図 6-9 4 近傍ラプラシアンフィルタ

$$\begin{array}{ccc} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{array} \quad \text{規格化係数 } 1$$

図 6-10 8 近傍ラプラシアンフィルタ

$$\begin{array}{ccccc} 2 & 4 & 4 & 4 & 2 \\ 4 & 0 & -8 & 0 & 4 \\ 4 & -8 & -24 & -8 & 4 \\ 4 & 0 & -8 & 0 & 4 \\ 2 & 4 & 4 & 4 & 2 \end{array} \quad \text{規格化係数 } 32$$

図 6-11 Sobel の2階微分をテイラー展開により求めた場合のラプラシアンフィルタ


```

sWord mask[] = { 0, 1, 0,
                 1, -4, 1,
                 0, 1, 0 };

MSGraph<sWord>* filter = new MSGraph<sWord>(3, 3);
filter->set_array(mask);
filter->norm = 1.0;

xp = new MSGraph<sWord>();
*xp = MSMaskFilter(*vp, *filter);

filter->free();
delete(filter);

for (int j=0; j<vp->ys; j++) {
    for (int i=0; i<vp->xs; i++) {
        xp->point(i, j) = vp->point(i, j) - xp->point(i, j);
        if (xp->point(i, j)>vp->max) xp->point(i, j) = vp->max;
        else if (xp->point(i, j)<vp->min) xp->point(i, j) = vp->min;
    }
}

```

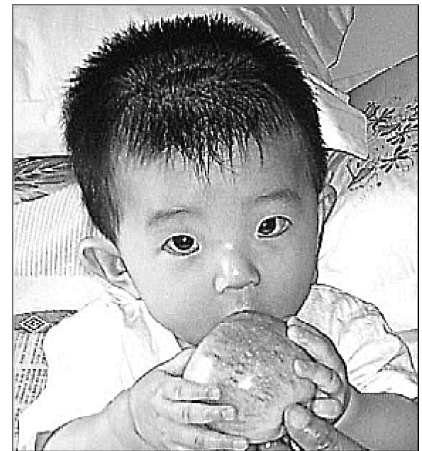
図 6-12 4近傍ラプラシアンフィルタによる画像の鮮鋭化プログラム



元画像



4近傍ラプラシアン



8近傍ラプラシアン



Sobel のラプラシアン

図 6-13 各種のラプラシアンを用いた画像の鮮鋭化。目の付近に注目。
ただし、ラプラシアンの重みを変えた場合は、結果は殆ど変わらない

7. プラグイン例3（フラクタル）

7.1 フラクタル図形

フラクタルは「砕けた石」という意味を持つ言葉で、フラクタル図形の定義を簡単に述べると、

- 1) 非整数次元を持つ図形
- 2) 自己相似性を持つ図形

となる。1)の図形とは、例えば平面上において無限の長さをもつ線や、3次元空間において無限の面積を持つ図形のことを指す。無限の長さをもつ線により描かれた図形は、1次元より上の次元を持つと考えられるが、平面上を全て覆い尽くすわけではないので2次元であるとも言えない。このことからそのような図形は非整数次元（フラクタル次元）を持つと考えられている。

2)では、ある図形を拡大していったときに、元の図形と相似な図形が無限に繰り返されることを指す。

フラクタル図形は、単純な原理（数式）を使って非常に複雑な図形を生成できるため、自然を表す図形であるとも言われている。

7.2 マンデルブロー集合

マンデルブロー集合は、1985年に Mandelbrotにより発見されたフラクタル図形である（Mandelbrotの最後の t を発音するかしないかは、フランス語とドイツ語の違いのようである）。

マンデルブロー集合とは、式 7-1 の漸化式において複素数 Z_n が発散しないような複素数 C の集合である。この図形を図 7-1 にその拡大図を図 7-2 に示す。図 7-1 の中央の達磨のような黒い領域がマンデルブロー集合である。マンデルブロー集合の周囲は、漸化式の発散の速度によって色分けされている。

ここで、マンデルブロー集合を拡大していくと（図 7-2）、無限に自己相似的な図形が現れてくる。また、マンデルブロー集合の境界領域も無限に拡大可能なので、境界線の長さも無限大となる。つまり、マンデルブロー集合はフラクタル図形の要件を満たしており、フラクタル図形の一つであると言える。

$$\begin{aligned} Z_{n+1} &= Z_n^2 + C \\ Z_0 &= 0 \end{aligned}$$

式 7-1 複素数 Z_n の漸化式



図7-1 マンデルブロー集合

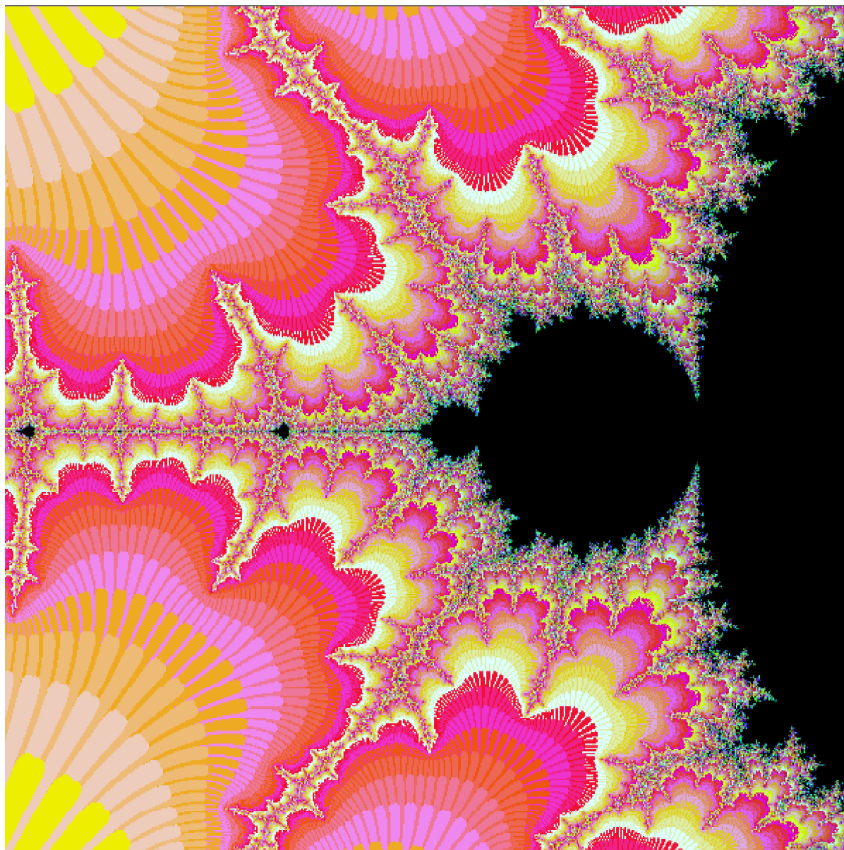


図7-2 マンデルブロー集合の拡大 $(-1.45, -0.05) - (-1.35, 0.05)$

```
MANDEL_API char* get_info(int n)
{
    switch(n) {
        case 0 :
            return "マンデルブロー集合";
        case 1 :
            return "mandel_proc";
        case 2 :
            return "mandel_free";
        case 3 :
            return "mandel_active";
        default:
            return NULL;
    }
}
```

図7-3 マンデルブロー集合プラグインの名前定義

```
/*
    TRUE を返すと、入力データ無しでもアクティブになる.
*/
MANDEL_API BOOL mandel_active(void)
{
    return TRUE;
}
```

図7-4 マンデルブロー集合プラグインの有効化処理

```

MANDEL_API MSGraph<sWord>* mandel_proc(MSGraph<sWord>* vp)
{
    MSGraph<sWord>* xp = NULL;
    int    Xsize  = 600;      // スクリーンのXサイズ
    double Infnty = 1.0e8;    // 無限大
    double Xmin   = -2.5;     // X軸（実数部）の最小値の初期値
    double Xmax   = 1.0;      // X軸（実数部）の最大値の初期値
    double Ymin   = -1.5;     // Y軸（虚数部）の最小値の初期値
    double Ymax   = 1.5;      // Y軸（虚数部）の最大値の初期値
    double val    = 200;      // 最大繰り返し回数の初期値

    // 数値入力用ダイアログ
    BOOL isok = InputMultiNumDLG("X軸の最小値", &Xmin, "X軸の最大値", &Xmax,
                                "Y軸の最小値", &Ymin, "Y軸の最大値", &Ymax,
                                "最大繰り返し回数", &val);

    if (!isok) {
        xp = new MSGraph<sWord>();
        xp->state = ERROR_GRAPH_CANCEL;
        return xp;
    }

    // 以下に処理コードを書く
    int    xsize = Xsize;
    int    ysize = (int)(xsize*(Ymax-Ymin)/(Xmax-Xmin) + 0.5);
    double dC    = (Xmax-Xmin)/xsize;      // 刻み幅

    xp = new MSGraph<sWord>(xsize, ysize); // (xsize × ysize) の画用紙を用意
    int  i, j, k;
    double Cr, Ci, Zr, Zi, Zrp, Zip;

    Ci = Ymax;
    for (j=0; j<xp->ys; j++) {
        Cr = Xmin;
        for (i=0; i<xp->xs; i++) {
            Zrp = 0.0;
            Zip = 0.0;
            k = 0;

            // 漸化式の計算  $Z(n+1) = Z(n) + C$ 
            do {
                Zr = Zrp*Zrp - Zip*Zip + Cr;
                Zi = 2.*Zrp*Zip + Ci;
                Zrp = Zr;
                Zip = Zi;
                k++;
            } while (Zr<Infnty && Zi<Infnty && k<=(int)val); // 発散のチェック

            xp->point(i, j) = (sWord)((double)(val-k+1)*4095/val);
            Cr += dC;
        }
        Ci -= dC;
    }

    xp->color = GRAPH_COLOR_ARGB16;      // カラーモード
    // 処理コードはここまで

    return xp;
}

```

図7-5 マンデルブロー集合のプラグインのメイン処理関数

マンデルブロー集合を描くためのプラグインのソースコードを図 7-3 ～図 7-5 に示す．図 7-2 の `get_info` 関数ではプラグインのタイトル名と各種の処理関数の名前を定義している．

また図 7-3 の関数が `TRUE` を返しているのは、このプラグインが読み込んだ画像データを処理する関数ではなく、画像データを生成するプラグインであることを示す（図 2-1 の⑤）．`mandel_active` が `TRUE` を返すと、そのプラグインは画像データを読み込まなくても有効になる．

図 7-5 において、`InputMultiNumDLG` 関数は、6 個までの `double` 型の数値を入力するためのダイアログを表示する．

処理の漸化式部分を図 7-6 に示す．ここでは、式 7-1 の漸化式を実数部と虚数部に分けて計算している．図 7-6 で `Zr` が Z_{n+1} の実数部で `Zi` が虚数部である．また `Cr` が式 7-1 の C の実数部であり、`Ci` が C の虚数部である．

```
// 漸化式の計算  $Z(n+1) = Z(n) + C$ 
do {
    Zr = Zrp*Zrp - Zip*Zip + Cr;
    Zi = 2.*Zrp*Zip      + Ci;
    Zrp = Zr;
    Zip = Zi;
    k++;
} while (Zr<Infnty && Zi<Infnty && k<=(int)val);    // 発散のチェック
```

図 7-6 処理関数の漸化式部分

変数 `val` は繰り返しが無限ループに陥るのを防ぐための、最大繰り返し回数である．

処理結果に色を付けるために、漸化式の繰り返し回数 (`k`) を元に最大 4095 の値を各画素に与えている．この値は、図形がウィンドウに表示されるときに `A4R4G4B4` (`GRAPH_COLOR_ARGB16`) のカラーデータとして解釈されて表示される．

7.3 その他のフラクタル図形

その他のフラクタル図形を図 7-7 から図 7-10 に示す．それぞれのプログラムについては、パッケージに添付されている `Sample_src` ディレクトリ内の各サンプルディレクトリを参照されたい．

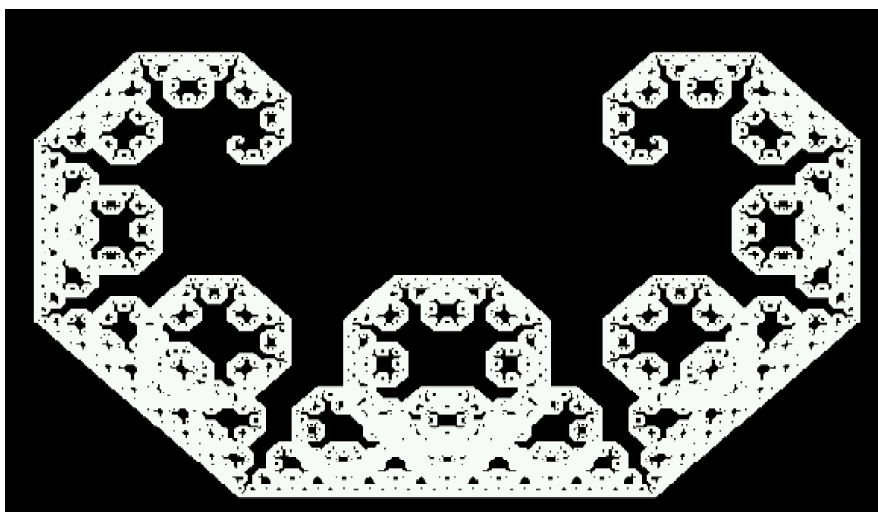


図 7-7 Cカーブ

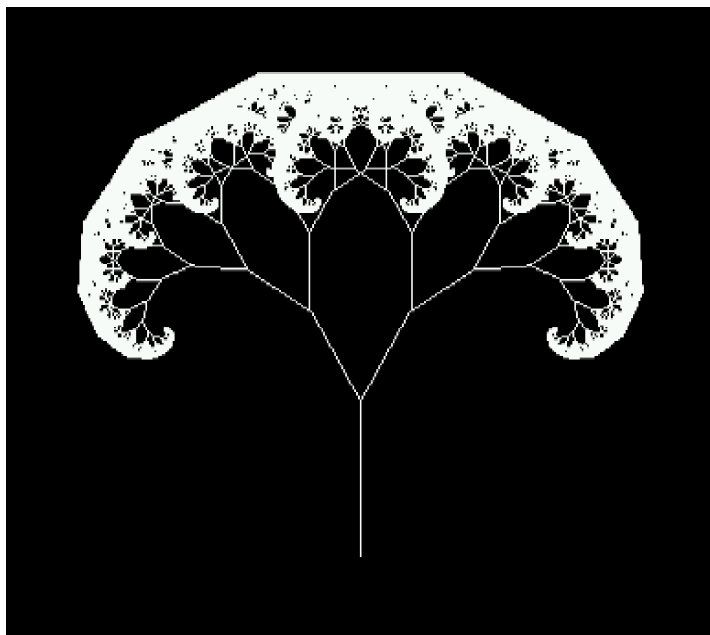


図 7-8 ツリー

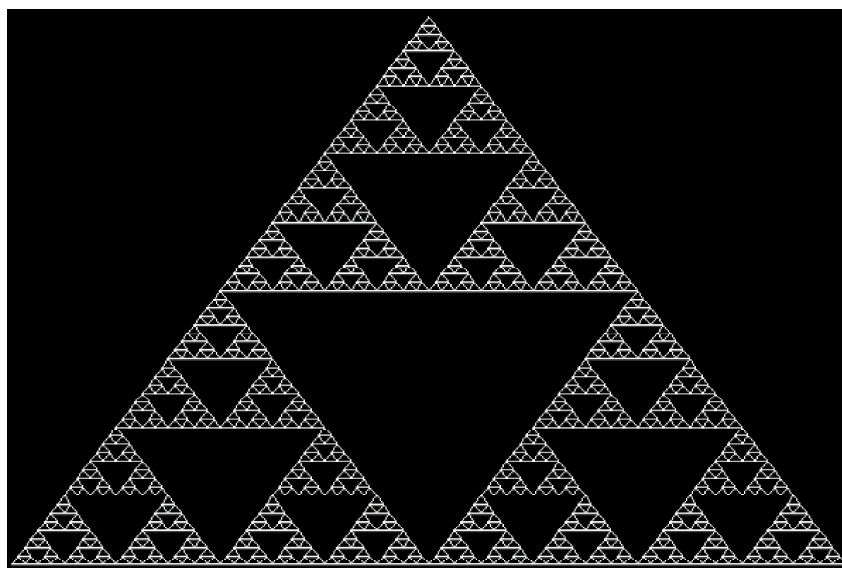


図7-9 シェルピンスキーの三角形

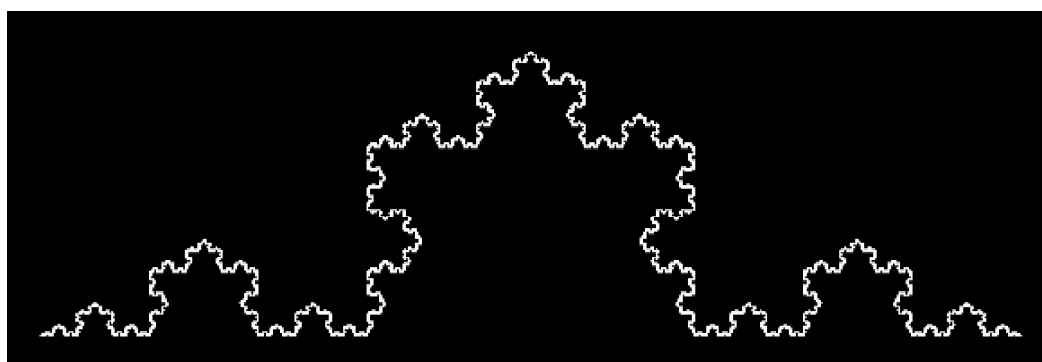


図 7-10 コッホ曲線

8. プラグイン例3（3次元図形）

8.1 3次元ボクセル表示の概要

CTViewでは3次元のボクセル画像を扱うことも可能である。CTviewで使用している3次元ライブラリは Microsoft の DirectX（左手系）であるが、内部で座標変換を行っているので、右手系で使うことが可能である（図8-1）。

3次元の物体を描画する場合、重要なのが法線ベクトルである（図8-2）。法線ベクトルが定まらなると、光の反射を計算できず、描画された3次元物体はメリハリのないものとなり、およそ3次的には見えなくなってしまう。

なお、CTViewでは3次元物体の表面のみを取り扱っている（サーフェスレンダリング）。これは、3次元物体内部は法線ベクトルを計算できないためであるのと、処理時間の短縮を行うためである。

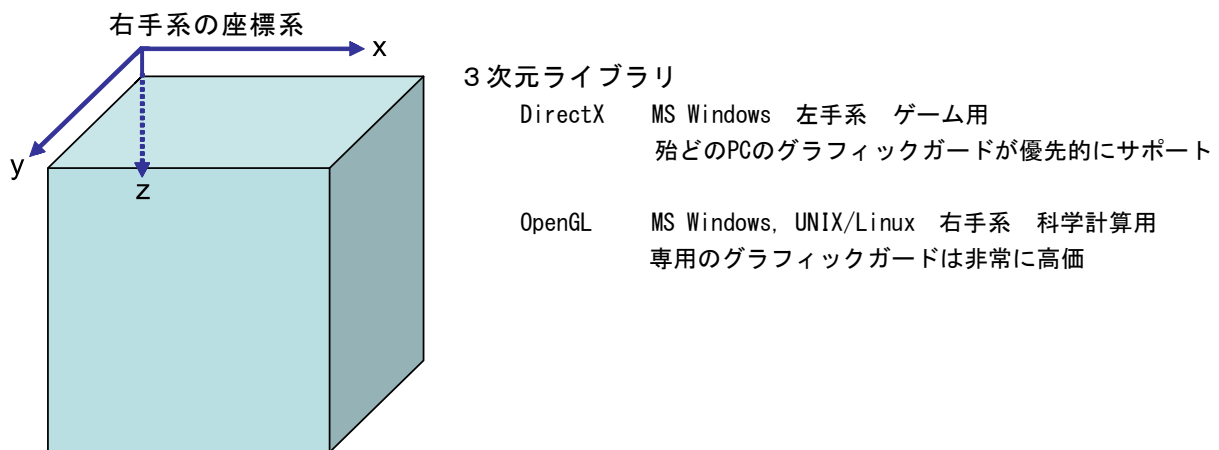


図 8-1 グラフィックライブラリと座標系

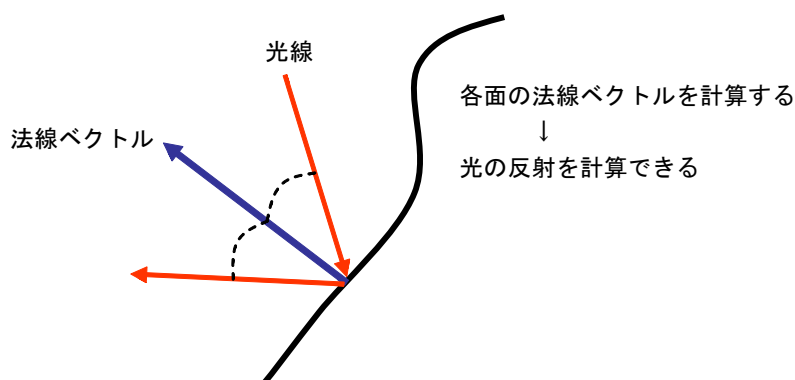


図 8-2 法線ベクトルと光の反射

8.2 3次元図形をサポートする関数

以下に、CTViewにおいて3次元図形をサポートする関数の紹介を行う。

```
void MSGraph_Circle3D(MSGraph<T> vp, Vector<> ox, Vector<> ex, int rr, int cc, int mode=OFF);
```

機能: 3D的な円の描画。

引数: vp -- 操作対象となるグラフィックデータ構造体。

ox -- 円の中心の座標ベクトル。

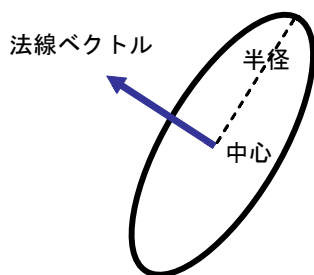
ex -- 円の中心の法線ベクトル。

rr -- 円の半径。

cc -- 線の濃度。

mode -- ON なら円の内部の0～ccをccで塗りつぶす。

戻り値: なし



```
Vector<> ox, ex;  
xp = new MSGraph<sWord>(200, 200, 200);  
  
ox.set(100, 100, 100);  
ex.set(100, 100, 180);  
MSGraph_Circle3D(*xp, ox, ex, 30, 100, OFF);
```

図 8-3 円 (板) の描画

```
void MSGraph_Pool(MSGraph<T> vp, Vector<> a, Vector<> b, int rr, int cc);
```

機能: 3D的な円柱の描画。中身はccで塗りつぶされる。

引数: vp -- 操作対象となるグラフィックデータ構造体。

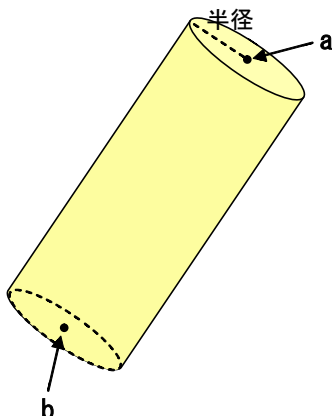
a -- 円柱の一方の底面の中心の座標ベクトル。

b -- 円柱のもう一方の底面の中心の座標ベクトル。

rr -- 円柱の半径。

cc -- 線と塗りつぶしの濃度。

戻り値: なし



```
Vector<> a, b;  
xp = new MSGraph<sWord>(200, 200, 200);  
  
a.set(100, 100, 100);  
b.set(100, 100, 180);  
MSGraph_Pool(*xp, a, b, 10, 150);
```

図 8-4 ポールの描画

```
void MSGraph_Torus(MSGraph<T> vp, Vector<> ox, Vector<> ex, int rr, int ra, int cc);
```

機能: 3D 的なトーラスの描画. 中身は cc で塗りつぶされる.

引数: vp -- 操作対象となるグラフィックデータ構造体.

ox -- トーラスの中心の座標ベクトル.

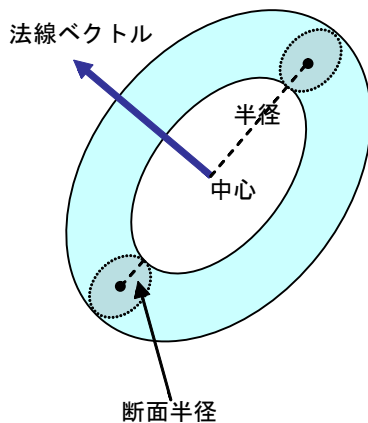
ex -- トーラスの中心の法線ベクトル.

rr -- トーラスの半径(トーラスの中心から断面の円の中心まで).

ra -- トーラスの断面の円の半径

cc -- 線と塗りつぶしの濃度.

戻り値: なし



```
Vector<> ox, ex;
xp = new MSGraph<sWord>(200, 200, 200);

ox.set(100, 100, 100);
ex.set(100, 80, 180);
MSGraph_Torus(*xp, ox, ex, 50, 5, 200);
```

図 8-5 トーラスの描画

```
void MSGraph_Sphere(MSGraph<T> vp, Vector<> a, int r, int cc, int mode=1);
```

機能: 球の描画.

引数: vp -- 操作対象となるグラフィックデータ構造体.

ox -- 球の中心の座標ベクトル.

r -- 球の半径.

cc -- 線と塗りつぶしの濃度(mode=1 のとき)

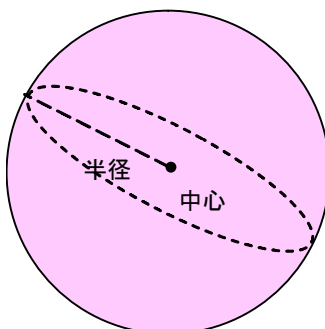
mode -- モード.

1: 円を重ねて球を作る. 中身は cc で塗りつぶされる.

-1: 極座標で球を作る. vp との境界に壁を作る.

それ以外: 極座標で球を作る.

戻り値: なし



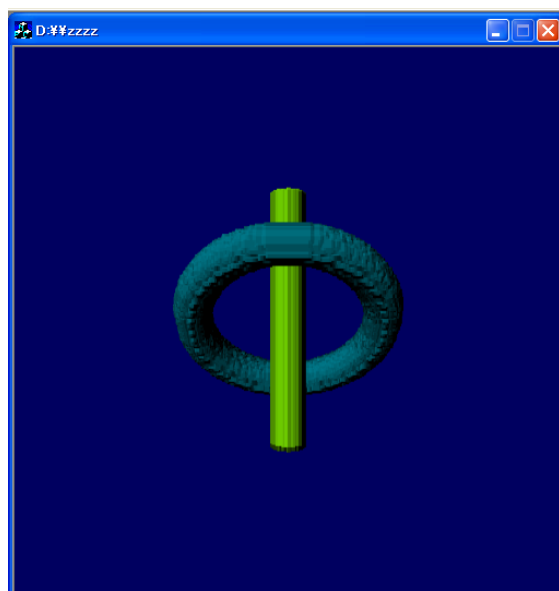
```
Vector<> ox;
xp = new MSGraph<sWord>(200, 200, 200);

ox.set(100, 50, 50);
MSGraph_Sphere(*xp, ox, 30, 300, 0);
```

図 8-6 球の描画

8.3 サンプル画像

図 8-7 にポールとトーラスを使用した 3 次元画像のサンプルを示す.



```
Vector<> va, vb, vt, et;  
xp = new MGraph<sWord>(200, 200, 200);  
  
va.set(100, 100, 20);  
vb.set(100, 100, 180);  
MGraph_Pool(*xp, va, vb, 10, 2000);  
  
vt.set(100, 100, 100);  
et.set(0, 1, 1);  
MGraph_Torus(*xp, vt, et, 60, 10, 120);
```

図 8-7 サンプル 3D 画像