VTK勉強会資料

大野

地球シミュレータセンター在籍時の勉強会資料 (2006 年ごろ)

Contents

Chapter 1

基本と2次元データの可視化

1.1 勉強会の目標

多くのサンプルプログラムを通して VTK のプログラミング方法を習得し，自分のシミュレーション結果を自作ソフトで可視化できるようになる。
(「等値面を表示させてマウスでグリグリまわす」，このレベルのソフトなら作れるようになります)

1.2 VTK

VTK (VISUALIZATION TOOLKIT) とは，フリーの可視化ライブラリである。これを利用すると，等値面生成などの可視化アルゴリズムや OpenGL などの CG プログラミングの知識がさほどなくとも，自分用の 3 次元可視化ソフトを容易に開発することができる。プログラミングは，C++, Tcl, JAVA, Python でおこなえる（この勉強会では C++を使う）。また，この勉強会では，Version 4.2 で作成したサンプルプログラムを使用する。

1.2.1 使用可能な可視化手法

カラーコンター，等高線，等値面，ボリューム・レンダリング，流線など，基本的なものはすべて使用可能。これ以外にも，チューブ状流線など面白い可視化手法が用意されている。

1.2.2 数値データ

型 Unsigned Char, Short, Int, Float, Double など
座標 等間隔，Rectilinear, StructuredGrid, UnstructuredGrid など

1.2.3 パイプライン

VTK を利用したプログラミングは，AVS のネットワークのように，VTK のオブジェクト同士を接続していくだけである。当然であるが，オブジェクト同士の接続においては，受け渡しするデータの型が一致していなければならない（接続可能かどうかは，オンライン・リファレンスを調べるとわかる）。パイプラインの基本的な流れは，

Data → Filter → Mapper → Actor → Renderer → Window

Filterの部分で，等値面処理などを行う。

1.3 データの可視化 序章

本章では，ETOPO2から日本近海を切り出した(VTKのデータ形式ではない)生データ[zenkoku.dem : 900×900, short型]をカラーコンターと等高線で可視化しつつ，VTKのプログラミングの雰囲気をつかむ。

1.3.1 可視化の手順

等高線による可視化を行うプログラムをOpenGLを使って自作するとする。どのような作業をプログラムにさせたらよいだろうか？

次の手順は一つの例である(VTKのプログラムでも同様の手順をふむ)。

1. 可視化するデータを読み込む(サイズや座標の情報も取得する)
2. 等高線作成のアルゴリズムでポリゴンデータ(線分の集まり。カラーコンターなら三角形や四角形の集まり)を作る
3. 手順2で求めたポリゴンデータをOpenGLで描画する

注意すべき点は，手順1のカッコ内である。データがあっても，そのサイズ(200×150など)や座標の情報(等間隔なのかRectilinearなのか? その刻み幅は? など)がなければ，可視化(手順2)は実行できない(AVSでもコントロールファイルに，それらの情報を記述しなければならない)。

VTKのプログラムでは，手順1でデータそのものと座標やサイズをあわせ持ったデータセットを作らなければならない(あるいは，データをVTK形式に変換しておかなければならない)。データセットを作って初めて，手順2で等高線や等値面のポリゴンデータを生成するフィルタを使うことができる。

1.4 カラーコンター

等間隔メッシュを使って，日本近海の標高データを色で表現してみる。

1.4.1 サンプルプログラム1

```
 1  /* ColorContour.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkShortArray.h>
 4  #include <vtkPointData.h>
 5  #include <vtkLookupTable.h>
 6  #include <vtkImageDataGeometryFilter.h>
 7  #include <vtkPolyDataMapper.h>
 8  #include <vtkRenderWindow.h>
 9  #include <vtkActor.h>
10  #include <vtkRenderer.h>
```

```
11
12  #include <unistd.h>
13
14  #define MAP_X 900
15  #define MAP_Y 900
16
17  short sdata[MAP_X * MAP_Y];
18
19  int main(int argc, char *argv[])
20  {
21      int size[3] = { MAP_X, MAP_Y, 1 }; /* 2次元データ */
22      FILE *fpi;
23
24      /* ハードディスクからデータを読み込む */
25      if ((fpi = fopen("./zenkoku.dem", "rb")) == NULL) {
26          puts("cannot open");
27          exit(1);
28      }
29       fread(sdata, sizeof(short), MAP_X * MAP_Y, fpi);
30
31      fclose(fpi);
32
33      /* 読み込んだデータをsarrayに代入 */
34      vtkShortArray *sarray = vtkShortArray::New();
35       sarray->SetArray(sdata, MAP_X * MAP_Y, 1);
36
37      /* 等間隔データを作成 */
38      vtkImageData *imgData = vtkImageData::New();
39       imgData->SetScalarTypeToShort();
40       imgData->SetDimensions(size);
41
42       float sp = 0.1;
43
44       imgData->SetSpacing(sp, sp, 0);
45
46       /* 等間隔メッシュの箱に，データを代入 */
47       imgData->GetPointData()->SetScalars(sarray);
48
49      /* カラーテーブルを作成 */
50      vtkLookupTable *lut = vtkLookupTable::New();
51       lut->SetHueRange(0.667, 0.0); /* 青 -> 赤 */
52       lut->Build();
53
54      /* imgDataを基にポリゴンデータを作成*/
55      vtkImageDataGeometryFilter *igf
56          = vtkImageDataGeometryFilter::New();
57       igf->SetInput(imgData);
```

```
58
59      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
60       Mapper->SetInput(igf->GetOutput());
61       Mapper->SetLookupTable(lut);
62       Mapper->SetColorModeToMapScalars();
63       Mapper->SetScalarRange(-500.0, 3800.0);
64
65      vtkActor *Actor = vtkActor::New();
66       Actor->SetMapper(Mapper);
67
68      vtkRenderer *vren = vtkRenderer::New();
69       vren->AddActor(Actor);
70       vren->SetBackground(0.0, 0.0, 0.0); /* 背景を黒に設定 */
71
72      vtkRenderWindow *renWin = vtkRenderWindow::New();
73       renWin->AddRenderer(vren);
74       renWin->SetSize(600, 600); /*Window サイズを 600x600 に設定*/
75
76      /* 10 秒程度表示 */
77      for(int i=0; i<10; i++){
78        renWin->Render();
79        sleep(1);
80      }
81
82      sarray->Delete();
83      imgData->Delete();
84      lut->Delete();
85      Mapper->Delete();
86      Actor->Delete();
87      vren->Delete();
88      renWin->Delete();
89
90     return 0;
91  }
```

1.4.2 サンプルプログラム 1 の簡単な解説

このプログラムは，sdata に日本近海の標高データ (zenkoku.dem) を読み込み，それを VTK の等間隔メッシュのデータにして，さらにそれを基にポリゴンを生成し，スカラーの値 (標高値) に応じて色づけして表示する (補足 : 2 次元データの可視化であるが，透視射影で表示している)。

1 - 12 : ヘッダファイルの読み込み
17 : 標高データを入れる short 型変数 “sdata” を宣言
21, 40 : Z 方向のサイズが “1” なので，このデータは 2 次元である
34 - 35 : VTK 用の short 型配列 (sarray) に sdata をセット
VTK のクラスは，new, delete 演算子は使わず，〜::New(), 〜->Delete() を使用する。**35** 行目の SetArray の 1 番目の引数はセットする配列データのポインタ，2 番目はサイズである。int, float,

double 型には，それぞれ vtkIntArray, vtkFloatArray, vtkDoubleArray が用意されている。
37 - 47 : VTK の等間隔メッシュデータ格納用のクラス (imgData) に sarray をセット
39 行でデータ型を，**40** 行でデータのサイズを，**44** 行でメッシュの幅を指定。原点の位置も指定できるが，このプログラムでは指定していない。
ようやく，(座標の情報を含んだ)VTK の等間隔メッシュのデータが完成。
49 - 52 : 色（青 → 赤）にセット
明度・彩度の指定や，自分でカラーテーブルを作ることもできる。
54 - 57 : imgData をポリゴン情報に変換
imgData(vtkImageData) をそのまま vtkPolyDataMapper に渡すことはできない。
59 - 63 : 色情報などと統合
このプログラムの可視化パイプラインを図で示すと，Figure 1.1 のようになる。

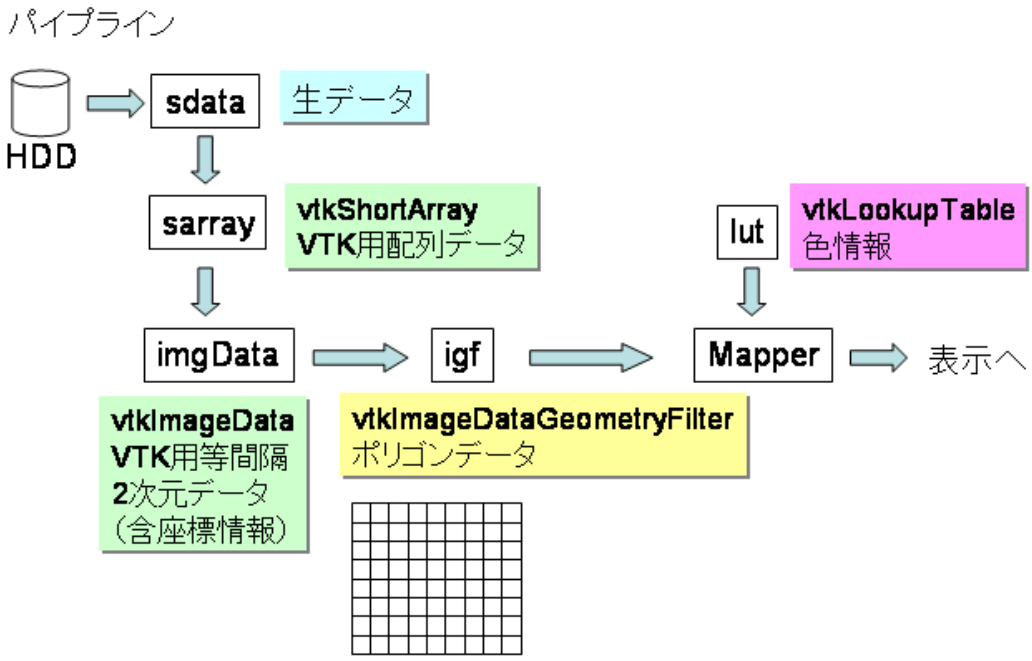

Figure 1.1: サンプルプログラム 1 のパイプライン構造

あるデータから別のデータに変換するオブジェクト (プロセス・オブジェクト) の出力は，a->GetOutput() と憶えて差し支えない (**57** 行，**60** 行を参考に)。であるから接続部分は，

入力側->SetInput(出力側->GetOutput());

の形となる。
63 行で色づけするスカラーの範囲を指定しているが，-500.0(色づけ指定した範囲の最小値) より小さい値は青（最小値の色），3800.0（色づけ指定した範囲の最大値）より大きい値は，赤（最大値の色）になる。
70 : 背景の色を黒 (RGB = (0.0, 0.0, 0.0)) に指定
74 : Window の大きさを 600 × 600 に指定
カメラの位置などをまったく指定していないが，本サンプルプログラムの場合はデータ全体が表示されるように，VTK が自動的にカメラの位置・その他を決めている。もちろん指定することも可能である (たとえば下記リスト)。

```
1       vren->GetActiveCamera()->SetPosition(30, 30, 100);
```

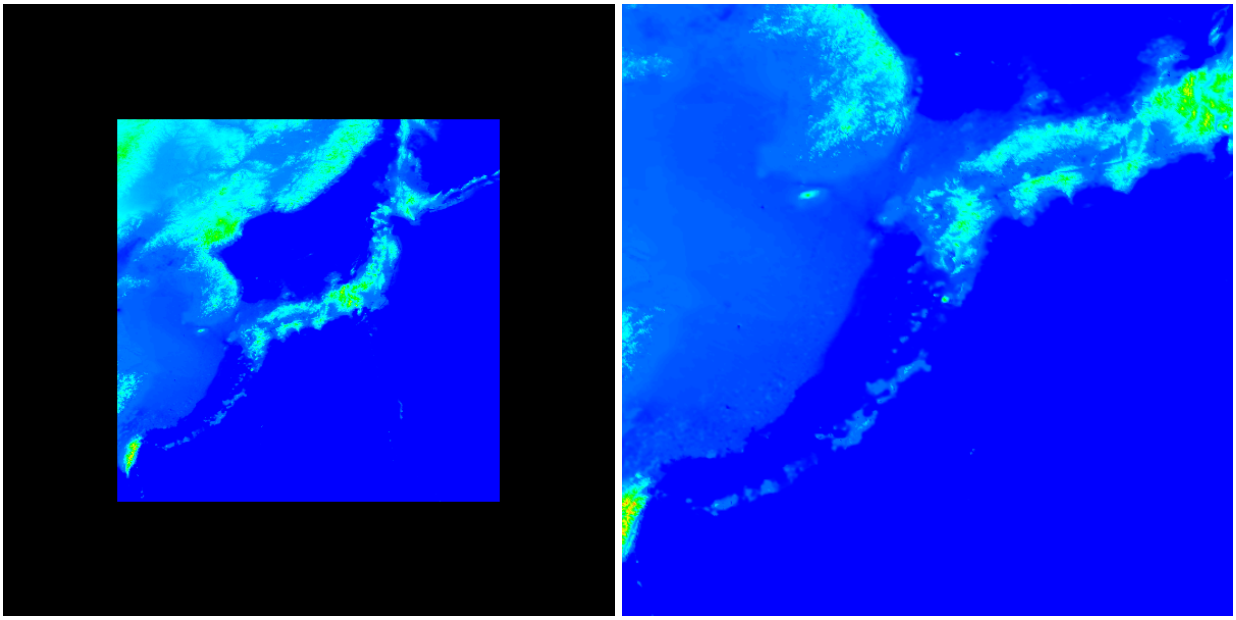

Figure 1.2: カラーコンター ((左) カメラ位置指定なし, (右) カメラ位置指定)

```
2       vren->GetActiveCamera()->SetFocalPoint(30, 30, 0);
3       vren->GetActiveCamera()->SetClippingRange(10, 200);
```

Figure 1.2 は，カメラ位置指定なしの場合と，(vtkCamera.h をインクルードして) 上記リストを **71** 行目に挿入してカメラ位置を指定した場合に出力される画像。カメラ関係については，次章でもう少し詳しく説明する。

1.4.3 画像保存

表示させた画像を保存しておきたい場合には，vtkRendererSource.h, vtkBMPWriter.h をインクルードして，下記リストを **81** 行目，プログラムの最後に rs− >Delete(), bw− >Delete() を加える。

```
1     vtkRendererSource *rs = vtkRendererSource::New();
2      rs->SetInput(vren);
3      rs->WholeWindowOn();
4      rs->Modified();
5
6     vtkBMPWriter *bw = vtkBMPWriter::New();
7      bw->SetInput(rs->GetOutput());
8      bw->SetFileName("Map.bmp");
9      bw->Write();
```

このリストを挿入すると，Window に表示された画像が，BMP 形式でファイル名 “Map.bmp” で保存される。vtkBMPWriter の類似のクラスとして，vtkJPEGWriter, vtkPNGWriter, vtkPostScriptWriter, vtkTIFFWriter などがある。

1.5 等高線

次に，vtkContourFilter というフィルタを使って，等高線を描いてみる。このフィルタは，等間隔でも Rectiliner のデータでも使用可能で，2 次元データを渡すと等高線，3 次元データを渡すと等値面を生成する。今度は，Rectilinear のデータを使用してみる。

1.5.1 サンプルプログラム 2

```
 1  /* ContourLines.cxx */
 2  #include <vtkRectilinearGrid.h>
 3  #include <vtkFloatArray.h>
 4  #include <vtkShortArray.h>
 5  #include <vtkPointData.h>
 6  #include <vtkContourFilter.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkPolyDataMapper.h>
 9  #include <vtkRenderWindow.h>
10  #include <vtkActor.h>
11  #include <vtkRenderer.h>
12
13  #include <unistd.h>
14
15  #define MAP_X 900
16  #define MAP_Y 900
17  #define NCONTLINES 10
18
19  short sdata[MAP_X * MAP_Y];
20
21  int main(int argc, char *argv[])
22  {
23      int size[3] = { MAP_X, MAP_Y, 1 };
24      FILE *fpi;
25
26      if ((fpi = fopen("./zenkoku.dem", "rb")) == NULL) {
27          puts("cannot open");
28          exit(1);
29      }
30       fread(sdata, sizeof(short), MAP_X * MAP_Y, fpi);
31
32      fclose(fpi);
33
34      vtkShortArray *sarray = vtkShortArray::New();
35       sarray->SetArray(sdata, MAP_X * MAP_Y, 1);
36
37      /* 座標データを格納するための配列 */
38      vtkFloatArray *x_coord = vtkFloatArray::New();
```

```
39      vtkFloatArray *y_coord = vtkFloatArray::New();
40      vtkFloatArray *z_coord = vtkFloatArray::New();
41
42      float z = 0.0;
43      float x[MAP_X];
44      float y[MAP_Y];
45
46      for (int i = 0; i < MAP_X; i++) {
47          x[i] = (float) i *0.1;
48          y[i] = (float) i *0.1;
49      }
50
51       x_coord->SetArray(x, MAP_X, 1);
52       y_coord->SetArray(y, MAP_Y, 1);
53       z_coord->SetArray(&z, 1, 1);
54
55      vtkRectilinearGrid *RectData
56              = vtkRectilinearGrid::New();
57       RectData->SetXCoordinates(x_coord); /* 座標データをセット*/
58       RectData->SetYCoordinates(y_coord);
59       RectData->SetZCoordinates(z_coord);
60       RectData->SetDimensions(size);
61       RectData->GetPointData()->SetScalars(sarray);
62
63      vtkLookupTable *lut = vtkLookupTable::New();
64       lut->SetHueRange(0.667, 0.0);
65       lut->Build();
66
67      /* 等高線を生成する */
68      vtkContourFilter *contour = vtkContourFilter::New();
69       contour->SetInput(RectData);
70
71       /* 0.0 - 3800.0 の間で, NCONTLINES 本等高線を生成 */
72       contour->GenerateValues(NCONTLINES, 0.0, 3800.0);
73
74      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
75       Mapper->SetInput(contour->GetOutput());
76       Mapper->SetLookupTable(lut);
77       Mapper->SetColorModeToMapScalars();
78       Mapper->SetScalarRange(-500.0, 3800.0);
79
80      vtkActor *Actor = vtkActor::New();
81       Actor->SetMapper(Mapper);
82
83      vtkRenderer *vren = vtkRenderer::New();
84       vren->AddActor(Actor);
85       vren->SetBackground(0.0, 0.0, 0.0);
```

```
 86
 87       vtkRenderWindow *renWin = vtkRenderWindow::New();
 88        renWin->AddRenderer(vren);
 89        renWin->SetSize(600, 600);
 90
 91       for(int i=0; i<10; i++){
 92         renWin->Render();
 93         sleep(1);
 94       }
 95
 96       sarray->Delete();
 97       x_coord->Delete();
 98       y_coord->Delete();
 99       z_coord->Delete();
100       RectData->Delete();
101       lut->Delete();
102       contour->Delete();
103       Mapper->Delete();
104       Actor->Delete();
105       vren->Delete();
106       renWin->Delete();
107
108      return 0;
109   }
```

1.5.2 サンプルプログラム 2 の簡単な説明

37 - 53 : Rectilinear の座標データ用の配列確保と，座標値の代入（結局等間隔にしています）
55 - 61 : 上記の座標データと sdata を Rectilinear のデータ (RectData) にセット。Rectilinear の 2 次元データ完成
68 - 72 : contour(vtkContourFilter) に RectData (vtkRectilinearGrid) を代入し，**72** 行で 0.0 〜 3800.0 の間で等間隔に NCONTLINES 本 (10 本) 等高線を生成させる。等高線の値を直接指定することも可能である
このサンプルプログラムの可視化パイプラインは，Figure 1.3 のようになる。
　結果は Figure 1.4 。

1.5.3 パイプラインの接続可否

vtkContourFilter を例に考えてみる。
　69 行 “contour->SetInput(RectData)”で，vtkContourFilter に RectilinearGrid のデータセット (のポインタ) を代入しているが，オンライン・リファレンスで SetInput の引数の定義を見てみると，SetInput(**vtkRectilinearGrid ***) ではなく，SetInput(**vtkDataSet ***) となっている。多重定義されているわけでもない。これでもパイプラインが接続できる理由は，vtkRectilinearGrid が vtkDataSet の派生クラス (Figure 1.5) だからである。ちなみに vtkImageData も vtkDataSet の派生クラスである。
　このように，パイプラインの接続可否は，オンライン・リファレンスで該当クラスの継承関係を調べれるとわかる。§1.4.2 でも述べたが，例えば a と b を接続する場合，基本的に

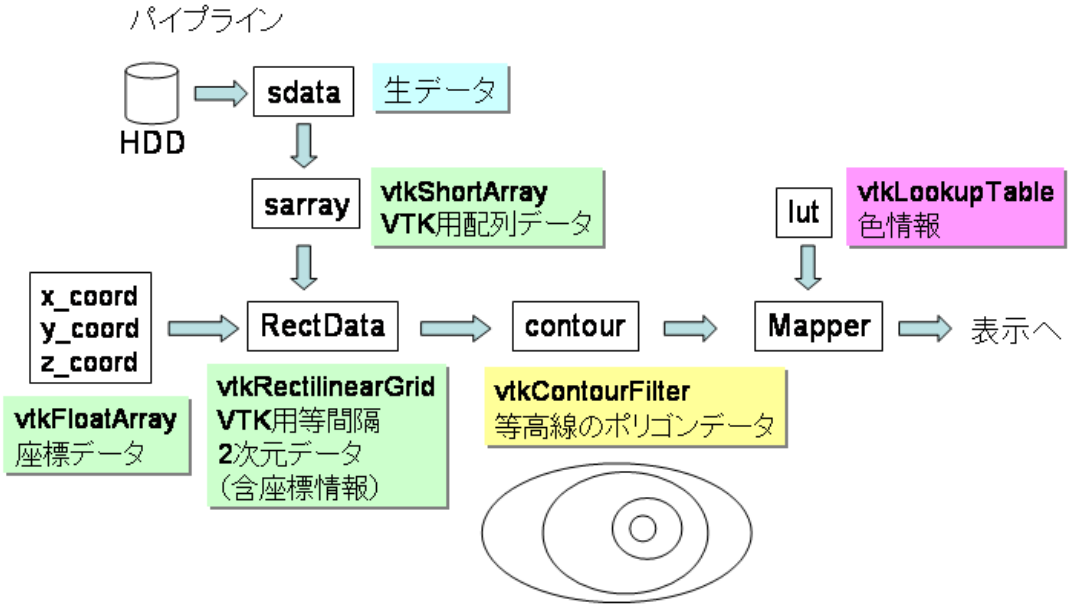

Figure 1.3: サンプルプログラム 2 のパイプライン構造

b->SetInput(a->GetOutput());

という形をとるので，入力側の **SetInput** メソッドの引数 と出力側の **GetOutput** メソッドの返り値のデータ型に注目する。

1.6 同一の Window で複数の結果を表示

最後に「カラーコンター」,「等高線」,「その重ね合わせ」の 3 つの結果を同一の Window に表示させる。

1.6.1 サンプルプログラム 3

```
 1  /* Contour3.cxx */
 2  #include <vtkRectilinearGrid.h>
 3  #include <vtkFloatArray.h>
 4  #include <vtkShortArray.h>
 5  #include <vtkPointData.h>
 6  #include <vtkContourFilter.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkRectilinearGridGeometryFilter.h>
 9  #include <vtkPolyDataMapper.h>
10  #include <vtkRenderWindow.h>
11  #include <vtkCamera.h>
12  #include <vtkActor.h>
13  #include <vtkProperty.h>
14  #include <vtkRenderer.h>
```

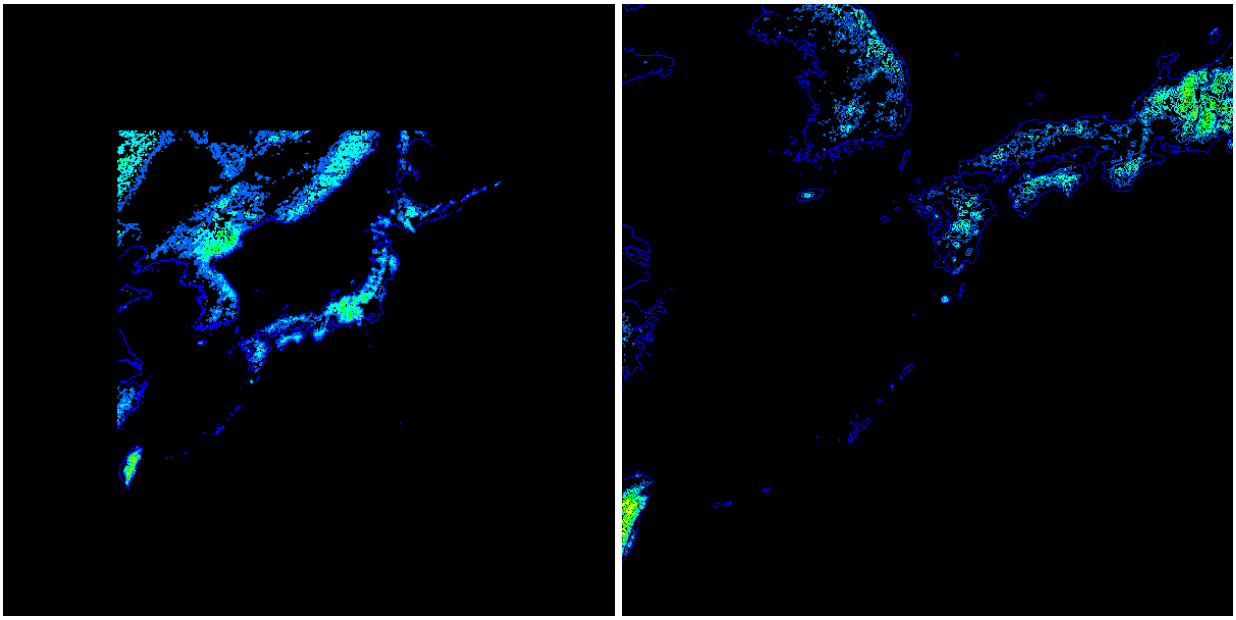

Figure 1.4: 等高線 ((左) カメラ位置指定なし, (右) カメラ位置指定)

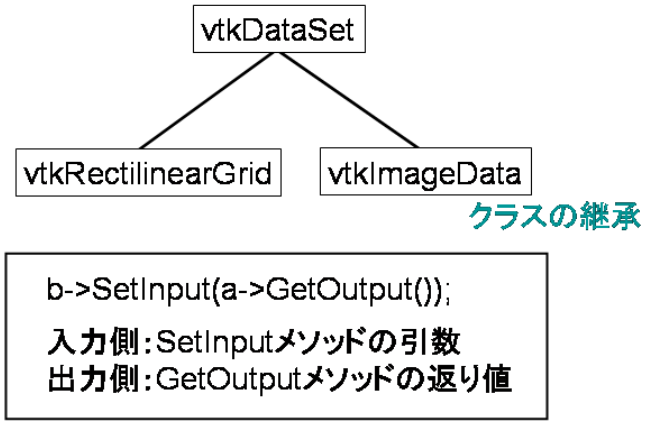

Figure 1.5: クラスの継承

```
15
16  #include <unistd.h>
17
18  #define MAP_X 900
19  #define MAP_Y 900
20
21  short sdata[MAP_X*MAP_Y];
22
23  int main( int argc, char *argv[] )
24  {
25      int size[3] = {MAP_X, MAP_Y, 1};
26      FILE *fpi;
27
```

```
28      if ((fpi = fopen("./zenkoku.dem", "rb")) == NULL) {
29          puts("cannot open");
30          exit(1);
31      }
32        fread(sdata, sizeof(short), MAP_X*MAP_Y, fpi);
33
34      fclose(fpi);
35
36      vtkShortArray *sarray = vtkShortArray::New();
37       sarray->SetArray(sdata, MAP_X*MAP_Y, 1);
38
39      vtkFloatArray *x_coord = vtkFloatArray::New();
40      vtkFloatArray *y_coord = vtkFloatArray::New();
41      vtkFloatArray *z_coord = vtkFloatArray::New();
42
43      vtkRectilinearGrid *RectData = vtkRectilinearGrid::New();
44
45      float z = 0.0;
46      float x[MAP_X];
47      float y[MAP_Y];
48
49        for(int i=0;i<MAP_X;i++){
50            x[i] = (float)i*0.1;
51            y[i] = (float)i*0.1;
52        }
53
54       x_coord->SetArray(x, MAP_X, 1);
55       y_coord->SetArray(y, MAP_Y, 1);
56       z_coord->SetArray(&z, 1, 1);
57
58       RectData->SetXCoordinates(x_coord);
59       RectData->SetYCoordinates(y_coord);
60       RectData->SetZCoordinates(z_coord);
61       RectData->SetDimensions(size);
62       RectData->GetPointData()->SetScalars(sarray);
63
64      vtkLookupTable *lut = vtkLookupTable::New();
65       lut->SetHueRange(0.667, 0.0);
66       lut->Build();
67
68      vtkContourFilter *contour = vtkContourFilter::New();
69       contour->SetInput(RectData);
70       contour->SetValue(0, 0.0); /* スカラー値0.0の等高線を生成 */
71
72      vtkRectilinearGridGeometryFilter *rgf
73             = vtkRectilinearGridGeometryFilter::New();
74       rgf->SetInput(RectData);
```

```
 75
 76      /* 等高線用の Mapper */
 77      vtkPolyDataMapper *Mapper1 = vtkPolyDataMapper::New();
 78       Mapper1->SetInput(contour->GetOutput());
 79
 80       /* スカラー値による色づけをしない */
 81       Mapper1->ScalarVisibilityOff();
 82
 83      /* カラーコンター用の Mapper */
 84      vtkPolyDataMapper *Mapper2 = vtkPolyDataMapper::New();
 85       Mapper2->SetInput(rgf->GetOutput());
 86       Mapper2->SetLookupTable(lut);
 87       Mapper2->SetColorModeToMapScalars();
 88       Mapper2->SetScalarRange(-500.0, 3800.0);
 89
 90      /* 等高線用の Actor */
 91      vtkActor *Actor1 = vtkActor::New();
 92       Actor1->SetMapper(Mapper1);
 93
 94       /* 等高線の色を白に設定 */
 95       Actor1->GetProperty()->SetColor(1.0, 1.0, 1.0);
 96
 97      /* カラーコンター用の Actor */
 98      vtkActor *Actor2 = vtkActor::New();
 99       Actor2->SetMapper(Mapper2);
100
101      /* カラーコンターと等高線を同時に映す */
102      vtkRenderer *vren= vtkRenderer::New();
103       vren->AddActor( Actor2 );
104       vren->AddActor( Actor1 );
105       vren->SetBackground( 0.0, 0.0, 0.0 );
106       vren->GetActiveCamera()->SetPosition(30, 30, 100);
107       vren->GetActiveCamera()->SetFocalPoint(30, 30, 0);
108       vren->GetActiveCamera()->SetClippingRange(10, 200);
109       vren->SetViewport(0.66, 0.0, 1.0, 1.0);
110
111      /* 等高線を映す */
112      vtkRenderer *vren1= vtkRenderer::New();
113       vren1->AddActor( Actor1 );
114       vren1->SetBackground( 0.0, 0.0, 0.0 );
115       vren1->SetViewport(0.33, 0.0, 0.66, 1.0);
116
117      /* カラーコンターを映す */
118      vtkRenderer *vren2= vtkRenderer::New();
119       vren2->AddActor( Actor2 );
120       vren2->SetBackground( 0.0, 0.0, 0.0 );
121       vren2->SetViewport(0.0, 0.0, 0.33, 1.0);
```

```
122
123       vtkRenderWindow *renWin = vtkRenderWindow::New();
124        renWin->AddRenderer( vren1 );
125        renWin->AddRenderer( vren2 );
126        renWin->AddRenderer( vren  );
127        renWin->SetSize( 900, 300 );
128
129       for(int i=0; i<10; i++){
130         renWin->Render();
131         sleep(1);
132       }
133
134       sarray->Delete();
135       x_coord->Delete();
136       y_coord->Delete();
137       z_coord->Delete();
138       RectData->Delete();
139       lut->Delete();
140       contour->Delete();
141       Mapper1->Delete();
142       Mapper2->Delete();
143       Actor1->Delete();
144       Actor2->Delete();
145       vren->Delete();
146       vren1->Delete();
147       vren2->Delete();
148       renWin->Delete();
149
150      return 0;
151   }
```

1.6.2 サンプルプログラム 3 の簡単な説明

70 : 等高線の値を直接指定 (1 本目 (0 番) の等高線の値を 0.0 に指定。2 本目は 1 番, 3 本目は 2 番になる)。GenerateValues と違い，ピンポイントで指定できる
72 - 74 : サンプルプログラム 1 の **54 - 57** 行参照
81, 95 : スカラーによる色づけはせず，等高線を白 (RGB = (1.0, 1.0, 1.0)) で表示
103 - 104 : 同一の Renderer に Actor1(等高線) と Actor2(カラーコンター) をセットすることで，カラーコンターと等高線を両方表示させる
124 - 126 : 同一の vtkRenderWindow に各 vtkRenderer をセットすることで，同一 Window に 3 つの結果を表示させる
109, 115, 121 : Window 内での位置を決定
WIndow の左下が (0.0, 0.0)，右上が (1.0, 1.0)。**121** 行では，Actor2 を映す範囲を Window の左側 1/3 使用するように SetViewport で設定している
127 : 3 つの結果を表示したいので，Window の大きさを 900 × 300 と横長に設定

Actor 以降のパイプラインは，Figure 1.6 のようになる。

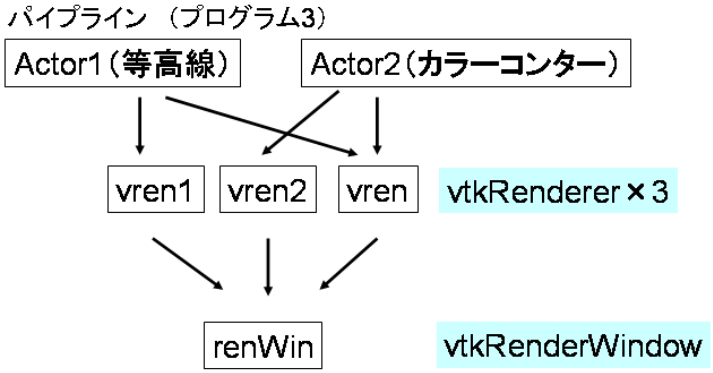

Figure 1.6: サンプルプログラム 3 のパイプライン構造

結果と ViewPort の座標を Figure 1.7 に示す。データ同士の比較や，一つのデータをいろいろな角度から映した画像を並べるのに便利である。

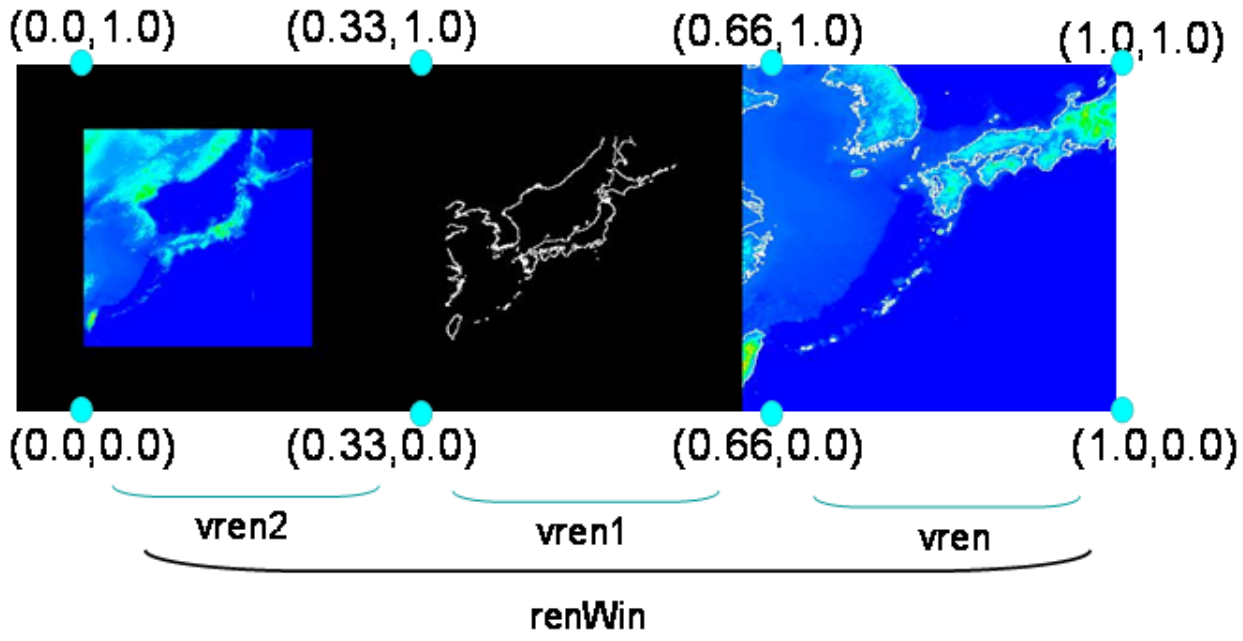

Figure 1.7: 同一 Window 内での 3 パターンの可視化結果の表示（(左）カラーコンター,（中）等高線,（右）組み合わせ）

1.7 この章のまとめ

- データの読み込み
- 等間隔，Rectilinear
- パイプラインの接続

本章で紹介したクラスは，Table 1.1 の通り。

Table 1.1: 本章て紹介したクラスおよびそのメソッド

配列	vtkShortArray vtkFloatArray : SetArray(short *, vtkIdType, int) / vtkShortArray
データセット	vtkImageData, vtkRectilinearGrid : SetDimensions(int, int, int) : SetSpacing(float, float, float) : GetPointData()(->SetScalars(vtkDataArray*)) : SetXCoordinates(vtkDataArray *) / vtkRectilinearGrid : SetYCoordinates(vtkDataArray *) / vtkRectilinearGrid : SetZCoordinates(vtkDataArray *) / vtkRectilinearGrid
ポリゴン化	vtkImageDataGeometryFilter vtkRectilinearGridGeometryFilter
カラーテーブル	vtkLookupTable : SetHueRange(float, float) : Build()
等高線	vtkContourFilter : GenerateValues(int N, float min, float max) : SetValue(int, float)
Mapper	vtkPolyDataMapper : SetColorModeToMapScalars() : SetLookupTable(vtkScalarsToColors *) : SetScalarRange(float, float) : ScalarVisibilityOff()
Actor	vtkActor : GetProperty()(->SetColor(float, float, float))
Renderer	vtkRenderer : SetViewPort(float, float, float, float)
画像保存	vtkRendererSource vtkBMPWriter, vtkPNGWriter, vtkJPEGWriter, vtkTIFFWriter
Window	vtkRenderWindow

Chapter 2

基本と2次元データの可視化その2

2.1 本章の概要

前章に引き続き 2 次元の標高データを可視化するが，本章では 2 次元データを基に 3 次元オブジェクトを作成する。また，射影やカメラ，ポリゴンの法線ベクトルのことにも触れる。

2.2 透視射影と正射影

VTK は，透視射影と正射影のいずれもサポートしている。透視射影と正射影は，それぞれ以下のような特徴を持つ。

- 透視射影
  物体の大きさが同じ場合，視点から近い物体ほど大きく，遠くにある物体ほど小さく画面に描く (遠近法で描いてくれる)
- 正射影
  近くとも遠くとも同じ大きさで画面に描く

2.2.1 透視射影

透視射影の場合は，Figure 2.1 で near と far の間の錐台の中が，画面に映し出される。この錘台の中に，映したいオブジェクトが入るようにカメラ関係のパラメータの設定をする。パラメータの設定は，vtkCamera と vtkRenderer で行える (サンプルプログラム 1 参照)。

- カメラの位置 (視点の位置) : SetPosition(double, double, double)
- 焦点 : SetFocalPoint(double, double, double) [カメラの位置と焦点を結ぶ方向が，視線方向。焦点より後ろが映らなくなるわけではない]
- 上方向 : SetViewUp(double, double, double)
- near, far : SetClippingRange(double near, double far) [near と far はカメラからの距離]
- ViewAngle : SetViewAngle(double)

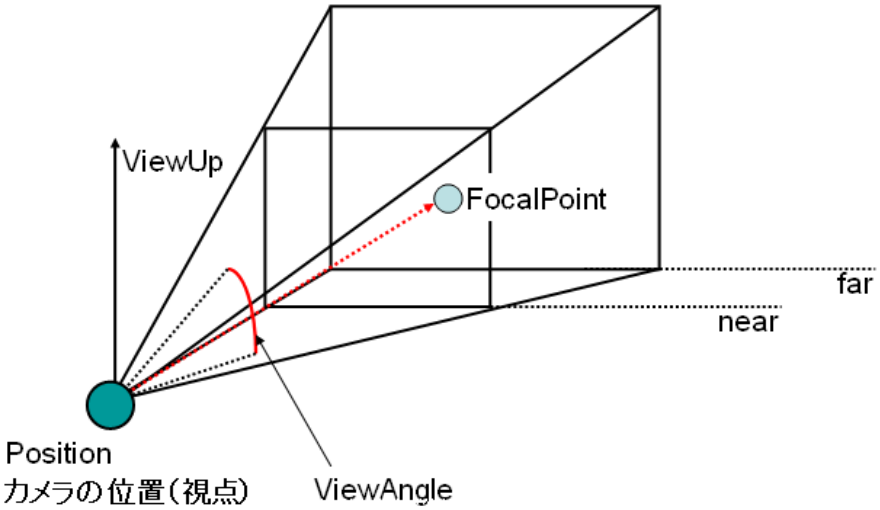

Figure 2.1: 透視射影

上記は，vtkCamera のメソッドである。カメラの位置などを設定するには，まず vtkCamera を生成し，上記のメソッドでパラメータを設定，その後 vtkRenderer の SetActiveCamera(vtkCamera *) を介して，Renderer にカメラ情報を渡す (SetViewAngle を除いて多重定義されているので，上記以外の設定方法もある)。カメラ位置などは，透視射影・正射影などかかわりなく，default では，Figure 2.2 のように設定されている。

2.2.2 正射影

正射影の場合は，映し出される空間が錐台から直方体になる (Figure 2.3)。正射影では，映し出される画像は ViewAngle の影響を受けないが，映し出す範囲 (画面の縦の大きさ) を vtkCamera の下記メソッドで設定する必要がある。

- SetParallelScale(double scale)

scale の値を変えることにより，映し出すオブジェクトに対する相対的な画面のサイズを変更できる。映し出すオブジェクトの高さが 6.0 の場合，画面いっぱいに映そうと思ったら，scale の値を 3.0(3 倍にする。画面の縦の長さは 2.0) にすればよい。正射影と透視射影の切り替えは，やはり vtkCamera の

- ParallelProjectionOn()
- ParallelProjectionOff()

で行う。default では，透視射影である。

2.2.3 サンプルプログラム 1

このサンプルプログラムでは，vtkCamera と vtkRenderer でカメラ位置などを設定して，画面全体に可視化画像が表示されるようにしている。本章のサンプルデータ (MtFuji.dem) は，GTOPO30 から富士山付近を切り出したサイズ 75 × 60 の Short 型バイナリデータである。

```
1  /* ParallelPro.cxx */
```

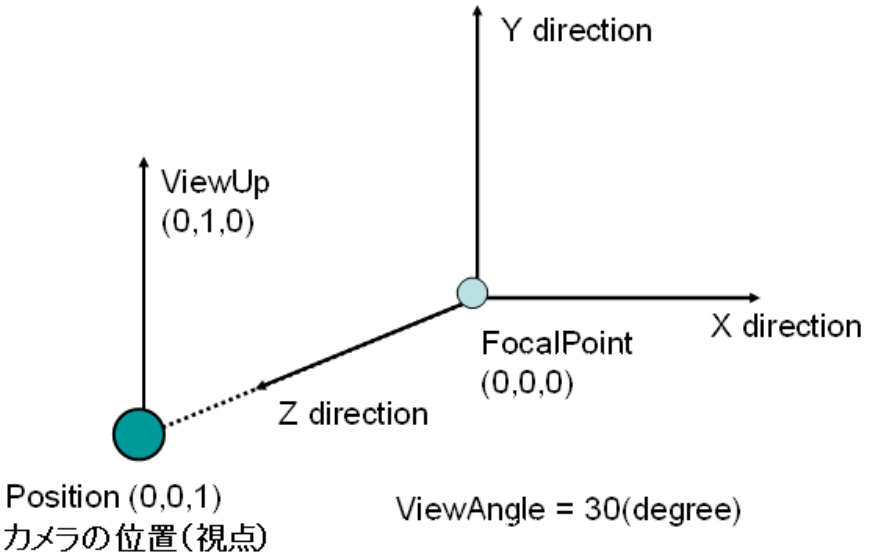

Figure 2.2: 座標

```
 2  #include <vtkShortArray.h>
 3  #include <vtkImageData.h>
 4  #include <vtkPointData.h>
 5  #include <vtkContourFilter.h>
 6  #include <vtkLookupTable.h>
 7  #include <vtkPolyDataMapper.h>
 8  #include <vtkRenderWindow.h>
 9  #include <vtkCamera.h>
10  #include <vtkActor.h>
11  #include <vtkRenderer.h>
12
13  #include <unistd.h>
```

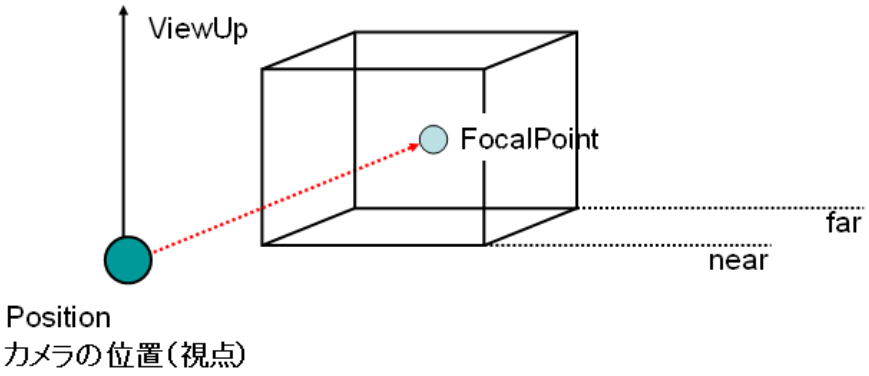

Figure 2.3: 正射影

```
14
15  #define MAP_X 75
16  #define MAP_Y 60
17  #define NCONTLINES 37
18
19  short sdata[MAP_X*MAP_Y];
20
21  int main( int argc, char *argv[] )
22  {
23      int size[3] = {MAP_X, MAP_Y, 1};
24      FILE *fpi;
25
26      if ((fpi =
27          fopen("./MtFuji.dem", "rb")) == NULL) {
28          puts("cannot open");
29          exit(1);
30      }
31        fread(sdata, sizeof(short), MAP_X*MAP_Y, fpi);
32
33      fclose(fpi);
34
35      vtkShortArray *sarray = vtkShortArray::New();
36       sarray->SetArray(sdata, MAP_X*MAP_Y, 1);
37
38      vtkImageData *imgData = vtkImageData::New();
39       imgData->SetSpacing(0.1,0.1,0);
40       imgData->SetDimensions(size);
41       imgData->GetPointData()->SetScalars(sarray);
42
43      vtkLookupTable *lut = vtkLookupTable::New();
44       lut->SetHueRange(0.667, 0.0);
45       lut->Build();
46
47      vtkContourFilter *contour = vtkContourFilter::New();
48       contour->SetInput(imgData);
49       contour->GenerateValues(NCONTLINES, 0.0, 3776.0);
50
51      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
52       Mapper->SetInput(contour->GetOutput());
53       Mapper->SetLookupTable(lut);
54       Mapper->SetColorModeToMapScalars();
55       Mapper->SetScalarRange(0.0, 3776.0);
56
57      vtkActor *Actor = vtkActor::New();
58       Actor->SetMapper(Mapper);
59
60      vtkRenderer *vren= vtkRenderer::New();
```

```
61        vren->AddActor( Actor );
62        vren->SetBackground( 0.0, 0.0, 0.0 );
63
64       /* カメラの設定 */
65       vtkCamera *camera = vtkCamera::New();
66        camera->SetPosition(3.7, 2.95, 100); /* カメラ位置 */
67        camera->SetFocalPoint(3.7, 2.95, 0); /* 焦点 */
68        camera->SetViewUp(0, 1, 0);          /* カメラの上方向 */
69        camera->ParallelProjectionOn();      /* 正射影 */
70        camera->SetParallelScale(2.95);      /* 画面サイズ*/
71        camera->SetClippingRange(50, 120);   /* Near, Far */
72
73        /* Rendererにカメラ情報を代入 */
74        vren->SetActiveCamera(camera);
75
76       vtkRenderWindow *renWin = vtkRenderWindow::New();
77        renWin->AddRenderer( vren );
78        renWin->SetSize( 740, 590 );
79
80       for(int i=0;i<10;i++){
81         renWin->Render();
82         sleep(1);
83       }
84
85       sarray->Delete();
86       imgData->Delete();
87       lut->Delete();
88       contour->Delete();
89       Mapper->Delete();
90       Actor->Delete();
91       vren->Delete();
92       camera->Delete();
93       renWin->Delete();
94
95     return 0;
96   }
```

2.2.4 サンプルプログラム 1 の簡単な説明

このプログラムは，カメラの X,Y 座標の位置をデータ (等高線) の中心 (XY 平面での) に据え，さらに画面の大きさもデータの大きさと同じにして，画面いっぱいに可視化像が映るようにしている。

カメラ関係の設定は，**64 - 73** 行目で行っている。

65 : camera(vtkCamera) の生成

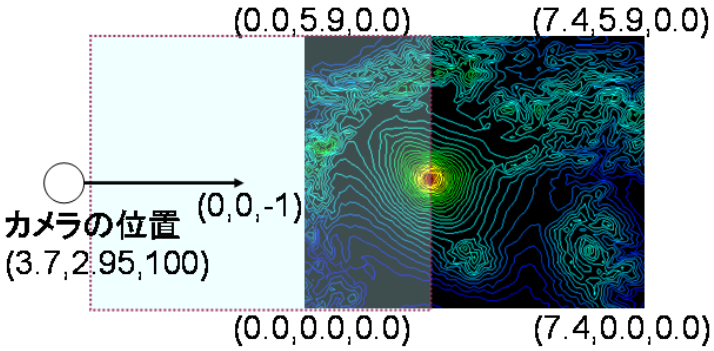

Figure 2.4: カメラとオブジェクトの位置関係

66 : カメラの位置設定 [1]
67 : カメラの焦点設定
68 : カメラの上方向設定
69 : 射影方法を正射影に設定
70 : 画面 (表示するオブジェクト) の相対的なサイズを設定 (表示されるオブジェクトの高さは 5.9 なので，引数に 2.95 を代入している)
71 : near, far を設定
74 : **66-71** 行で設定した camera の内容を vtkRenderer に代入

カメラの位置関係は Figure 2.4，表示は Figure 2.5。

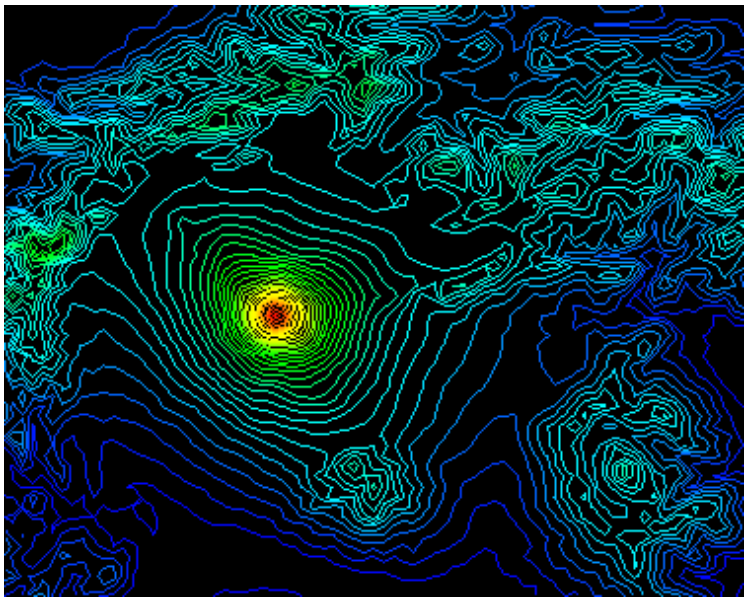

Figure 2.5: 富士山周辺の等高線：右下の山は箱根山

[1] vtkImageData のメソッド SetOrigin(float, float, float) でデータの原点を (-3.7,-2.95,0.0) に，カメラと焦点の X, Y 座標の値を 0.0 に設定しても，同じ画像が表示される。

2.2.5　カメラと焦点の位置の回転

FocalPoint の周りで，カメラの位置を簡単に回転させることができる。そのための vtkCamera のメソッドは，

- Azimuth(double angle)
- Elevation(double angle)

である。angle の単位は degree。これを使うと，カメラがオブジェクトの周りを周るアニメー

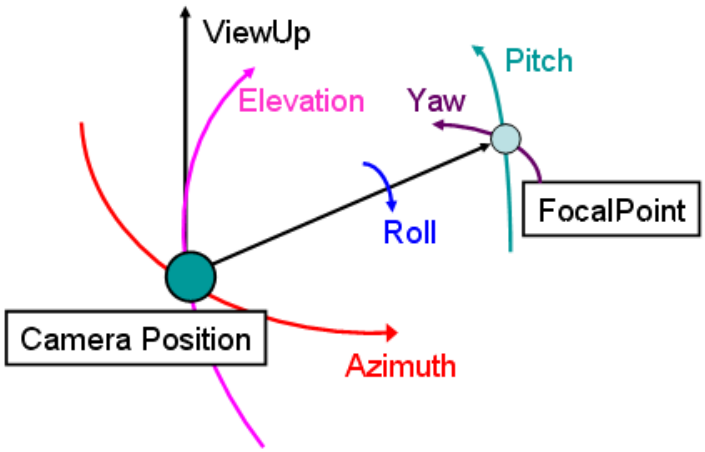

Figure 2.6: カメラ位置と焦点の回転方向

ションが簡単に作れる。サンプルプログラム 1 の **65 - 83** 行目を下記のリストに差し替えると (vtkRenderSource.h と vtkBMPWriter.h のインクルードも忘れずに)，透視射影で等高線の周りを一週するアニメーションが Window に現れ，平行して各スナップショットの画像が (anim1 ディレクトリの下に)BMP 形式で保存される。この連番画像を使いムービーを作れば，VTK がなくともアニメーションを見ることができる。

```
 1       vtkCamera *camera = vtkCamera::New();
 2        camera->SetPosition(3.7, 2.95, 15);
 3        camera->SetFocalPoint(3.7, 2.95, 0);
 4        camera->SetViewUp(0, 1, 0);
 5        camera->SetClippingRange(1, 20);
 6
 7        vren->SetActiveCamera(camera);
 8
 9       vtkRenderWindow *renWin = vtkRenderWindow::New();
10        renWin->AddRenderer( vren );
11        renWin->SetSize( 750, 600);
12
13       vtkRendererSource *rs = vtkRendererSource::New();
14        rs->SetInput(vren);
15        rs->WholeWindowOn();
16
```

```
17      vtkBMPWriter *bw = vtkBMPWriter::New();
18       bw->SetInput(rs->GetOutput());
19
20      char FileName[128];
21
22      for(int i=0;i<180;i++){
23
24      sprintf(FileName,"anim1/FujiAnim%d.bmp",i);
25
26      renWin->Render();
27      rs->Modified();
28
29      bw->SetFileName(FileName);
30      bw->Write();
31
32      /* Azimuth 方向に 2 度カメラを移動 */
33      vren->GetActiveCamera()->Azimuth(2.0);
34
35     }
```

カメラの設定は，vtkRenderer のメソッドを使って，直接変えることもできる (**33** 行)。**24 - 33** 行で，カメラの位置を 2 度ずつ回転させ，そのたびに表示される画像を BMP 形式で保存している。上記のリストを差し替えたバージョンのサンプルプログラムも RotateLines.cxx として保存されている。

33 行の Azimuth を Elevation に変えて実行すると，表示される画像がおかしくなる。これは，視線方向 (カメラと焦点を結ぶ方向) と ViewUp が垂直でなくなるからである。これを回避するには，**34** 行目に，

vren->GetActiveCamera()->OrthogonalizeViewUp();

を加えて，常に垂直に保てばよい。

焦点の位置をカメラの位置を中心として回転させるメソッドもある。これを使うと，カメラの向き (視線方向) を変化させることができる。回転は，下記のメソッドで行う。

- Pitch(double angle)
- Yaw(double angle)

Pitch でカメラの向きを上下方向に，Yaw で左右方向に変化させることができる。

カメラと焦点の位置は固定で，視線方向の周りでカメラを回転させるメソッド

- Roll(double angle)

もある。

2.3 凹凸つきスライス

前セクションと同じデータ (MtFuji.dem:Short 型, 75×60) を 3 次元オブジェクトとして可視化しつつ，そのオブジェクトを Window の中でマウスを使って，回転・移動・ズームイン/アウトさせることができるプログラムを作る。同時に、カラーテーブルの作り方，ポリゴンの法線ベクトルについても知識を増やす。

2.3.1 サンプルプログラム 2

前章のサンプルプログラム 1 は，標高 (スカラー値) に応じて色づけしただけであったが，本章は，ポリゴンの頂点を Z 方向に移動させることで，スライスに凹凸をつけている。さらに，Window 上でマウスのボタンを押しながらマウスカーソルを動かすと，オブジェクトが移動や回転をするようにしている。

```
 1  /* BumpySlice.cxx */
 2  #include <vtkShortArray.h>
 3  #include <vtkImageData.h>
 4  #include <vtkImageDataGeometryFilter.h>
 5  #include <vtkWarpScalar.h>
 6  #include <vtkPointData.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkPolyData.h>
 9  #include <vtkPolyDataMapper.h>
10  #include <vtkRenderWindow.h>
11  #include <vtkActor.h>
12  #include <vtkRenderer.h>
13  #include <vtkProperty.h>
14  #include <vtkRenderWindowInteractor.h>
15  #include <vtkInteractorStyleTrackballCamera.h>
16
17  #define MAP_X 75
18  #define MAP_Y 60
19
20  short sdata[MAP_X*MAP_Y];
21
22  int main( int argc, char *argv[])
23  {
24      int size[3] = {MAP_X, MAP_Y, 1};
25      FILE *fpi;
26
27      if ((fpi =
28          fopen("./MtFuji.dem", "rb")) == NULL) {
29          puts("cannot open");
30          exit(1);
31      }
32        fread(sdata, sizeof(short), MAP_X*MAP_Y, fpi);
33
34      fclose(fpi);
35
36      vtkShortArray *sarray = vtkShortArray::New();
37       sarray->SetArray(sdata, MAP_X*MAP_Y, 1);
38
39      vtkImageData *imgData = vtkImageData::New();
40       imgData->SetSpacing(0.1, 0.1, 0);
41       imgData->SetDimensions(size);
```

```
42       imgData->GetPointData()->SetScalars(sarray);
43
44      vtkLookupTable *lut = vtkLookupTable::New();
45       lut->SetHueRange(0.7, 0.0);
46       lut->Build();
47
48      vtkImageDataGeometryFilter *igf
49           = vtkImageDataGeometryFilter::New();
50       igf->SetInput(imgData);
51
52      /* 頂点をスカラー値に応じて移動 */
53      vtkWarpScalar *warp = vtkWarpScalar::New();
54       warp->SetInput(igf->GetOutput());
55       warp->SetScaleFactor(1.0/3776.0);
56
57      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
58       Mapper->SetInput(warp->GetPolyDataOutput());
59       Mapper->SetLookupTable(lut);
60       Mapper->SetColorModeToMapScalars();
61       Mapper->SetScalarRange(0.0, 3776.0);
62
63      vtkActor *Actor = vtkActor::New();
64       Actor->SetMapper(Mapper);
65
66      vtkRenderer *vren= vtkRenderer::New();
67       vren->AddActor( Actor );
68       vren->SetBackground( 0.0, 0.0, 0.0 );
69
70      vtkRenderWindow *renWin = vtkRenderWindow::New();
71       renWin->AddRenderer( vren );
72       renWin->SetSize( 750, 600 );
73
74      /* Windowをインタラクティブにする */
75      vtkRenderWindowInteractor *iwin
76          = vtkRenderWindowInteractor::New();
77       iwin->SetRenderWindow(renWin);
78
79      vtkInteractorStyleTrackballCamera *trackball =
80       vtkInteractorStyleTrackballCamera::New();
81
82       iwin->SetInteractorStyle(trackball);
83       iwin->Initialize();
84       iwin->Start();
85
86       sarray->Delete();
87       lut->Delete();
88       igf->Delete();
```

```
89        warp->Delete();
90        Mapper->Delete();
91        Actor->Delete();
92        vren->Delete();
93        iwin->Delete();
94        trackball->Delete();
95        renWin->Delete();
96
97      return 0;
98    }
```

(等間隔メッシュではなく，Rectilinear のデータの場合は，vtkImageDataGeometryFilter を vtkRectilinearGridGeometryFilter に変えればよい。)

2.3.2 サンプルプログラム 2 の簡単な説明

このプログラムは，vtkImageDataGeometryFilter と vtkPolyDataMapper の間に，頂点を移動させるフィルタを入れて，スライスに凹凸をつける処理をしている。また，Window をインタラクティブにしている。

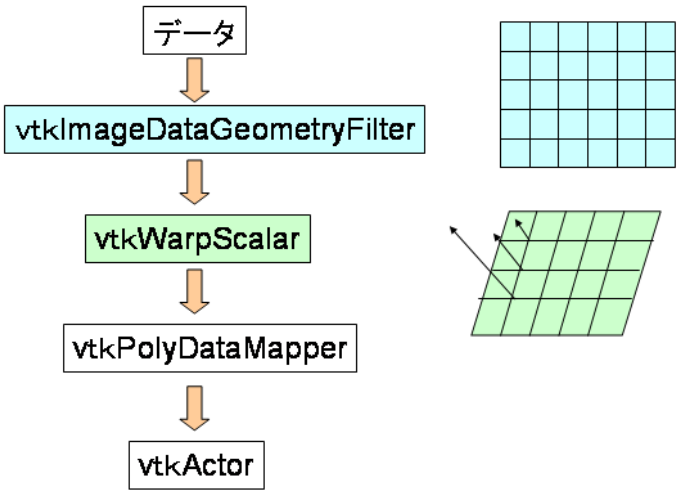

Figure 2.7: サンプルプログラム 2 のパイプライン構造

可視化パイプラインの構造を Figure 2.7 に示す。
このプログラムの第一のポイントは，**52 - 55** 行目である。

- vtkWarpScalar クラス

を使って，頂点をスカラー値に応じて Z 方向に移動させている。**55** 行目で，移動の度合を調整している。

第二のポイントは，**74 - 84** 行目である。renWin->Render() の代わりに，**74 - 84** 行目を書くだけで，Window がインタラクティブになる (簡単な操作方法を，Table 2.1 にまとめた)。**79 - 80** 行目をみると，このプログラムは vtkInteractorStyleTrackballActor を使用しているのがわかるが，vtkInteractorStyle は，これ以外にも数種類用意されている (オンライン・リファレンスでご確認を)。表示は，Figure 2.8(高さ方向は，かなり強調している)。

Table 2.1: Window の使い方

左ボタン + マウスカーソル上下左右	オブジェクトの回転 (Elevation, Azimuth)
中ボタン + マウスカーソル上下左右	オブジェクトの移動
右ボタン + マウスカーソル上下	ズームイン・ズームアウト
CTRL + 左ボタン + マウスカーソル上下左右	オブジェクトの回転 (Roll)
W, S	W:ワイヤフレーム表示，S:面での表示
U	ユーザー定義コマンド (最終章参照)
E, Q	終了

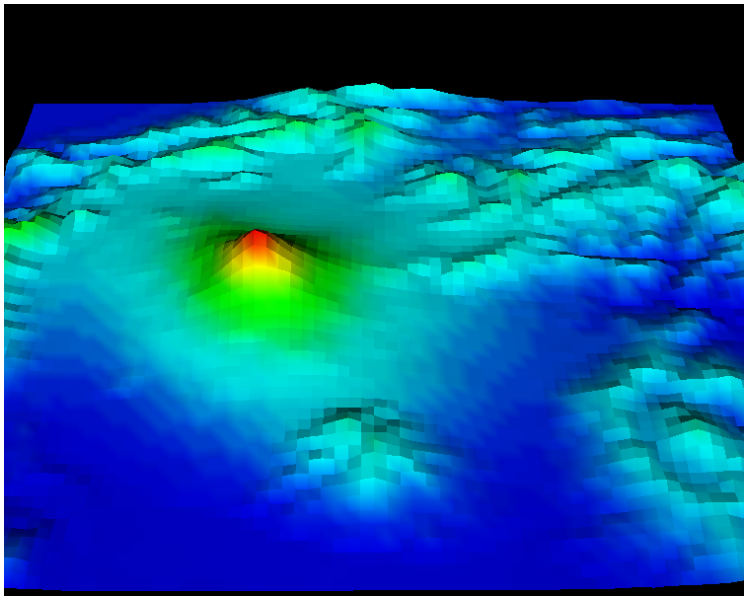

Figure 2.8: 富士山付近

2.3.3 ポリゴンの法線ベクトル

Figure 2.8 をみると，妙にポリゴンが目立っていることに気づくと思う。なぜこのようなことが起きているのか？　まず，ポリゴンへの“陰”のつけ方を紹介する。Figure 2.9 の拡散光と鏡面反射光に注目していただきたい。環境光だけでは，映しているものが 3 次元構造を持っていても，のっぺりとしか映らない。拡散光・鏡面反射光があってこそ，暗い部分や明るい部分ができて，3 次元的に見える。その拡散光と鏡面反射光の影響は，いずれもポリゴンの法線ベクトルの方向に依存している。サンプルプログラム 2 で，表面がガタガタして見える理由は，各ポリゴンの頂点に与えられている法線ベクトルが，同じ頂点を共有する隣のポリゴンの法線ベクトルと方向が異なっているので，陰のつけ方がポリゴンごとに不連続になるためである (Figure 2.10 の真ん中の図)。このガタガタを消すには，Figure 2.10 の右の図のような法線ベクトルの与え方をすればよい (このような法線ベクトルのセットの仕方をすれば，少ないポリゴンで綺麗な曲面を表現できる。)。これを実現してくれるのが，

- vtkPolyDataNormals クラス

である。vtkWarpScalar から直接 vtkPolyDataMapper にパイプラインをつながず，それらの間に，この Filter を入れればよい。サンプルプログラム 2 の **56 - 58** 行目を下記リストで書き換える

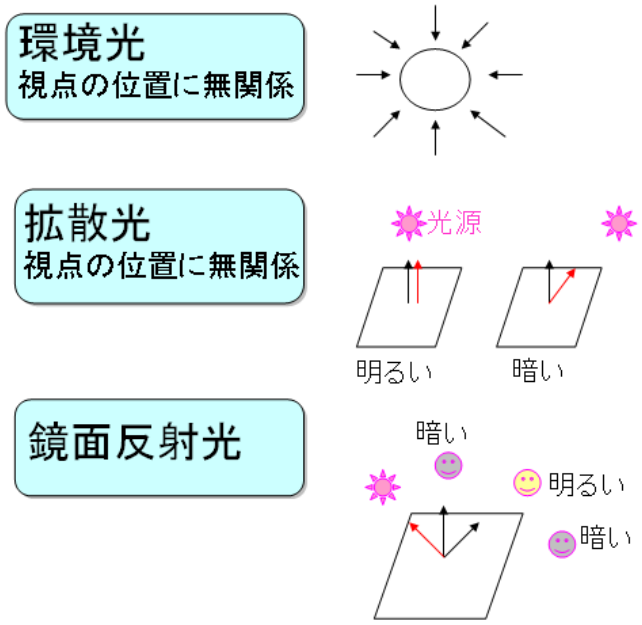

Figure 2.9: 陰のつけ方

(vtkPolyDataNormals.h のインクルードも忘れずに) とガタガタがかなりおさまる (Figure 2.11)。

```
1       vtkPolyDataNormals *PNormals = vtkPolyDataNormals::New();
2        PNormals->SetInput(warp->GetPolyDataOutput());
3
4       vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
5        Mapper->SetInput(PNormals->GetOutput());
```

書き換えられているバージョンは，BumpySlice2.cxx として保存されている。(建物などを描いている場合は，スムーズにしないほうが良い)

2.3.4 光源

光源の位置 (照らす方向) を変えて，陰のつけ方を変えてみる。vtkLight と vtkRenderer を使う。

```
1      vtkLight *light1 = vtkLight::New();
2       light1->SetPosition(0.0, -1.0, 1.0);
3       light1->SetFocalPoint(0.0, 0.0, 0.0);
4
5      vtkLight *light2 = vtkLight::New();
6       light2->SetPosition(0.0, 0.0, -1.0);
7       light2->SetFocalPoint(0.0, 0.0, 0.0);
8
9      vtkRenderer *vren = vtkRenderer::New();
10       vren->AddActor(Actor);
11       vren->AddLight(light1); /* Light1 をセット*/
```

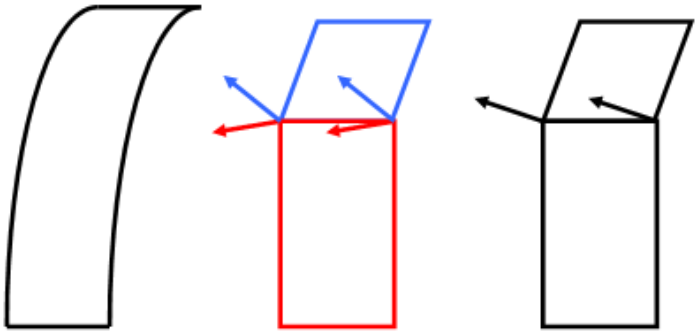

Figure 2.10: ポリゴンの法線ベクトル : 左の図のような曲面を 2 枚のポリゴンで描く場合。真ん中の図のように，同じ頂点でもポリゴンにより法線ベクトルの方向が異なるとガタガタが目立つ。右の図のように法線ベクトルの方向が共通だと，スムーズになる。

```
12        vren->AddLight(light2); /* Light2 をセット*/
```

BumpySlice.cxx や BumpySlice2.cxx の該当箇所を上記のリストで書き換えると，陰のつき方が変わる。

上記リストでは，光源の位置と焦点を与えて，光の方向のみ(Figure 2.12) を設定している。光源の位置の後ろにあるオブジェクトにも，陰がつく。

2.3.5 カラーテーブル

カラーテーブルは，vtkLookupTable で簡単に指定できるが，自分で作りたい場合もある。下記リストは，vtkLookupTable に自分で作ったカラーテーブルの渡し方の一例である。サンプルプログラム 2 の **45** 行目を下記のものと差し替えればよい。

```
1         lut->SetNumberOfColors(NCOLORS);
2         for(int i=0;i<NCOLORS;i++)
3              lut->SetTableValue(i, FujiColor[i]);
```

ただし，NCOLORS は使用する色の数，float FujiColor[NCOLORS][4] にはあらかじめ RGBA が 0.0 - 1.0 の範囲で入っていなければならない。サンプルプログラム 2 の書き換えバージョンの BumpySlice3.cxx では，可視化されたスライスが，山っぽく見えるような色を set_color 関数で FujiColor[NCOLORS][4] に入れるようにしてある。BumpySlice3.cxx を実行すると，Figure 2.13 の左側の図が表示される。なお，右側の絵は，Actor のプロパティを変更する

Actor->GetProperty()->SetRepresentationToWireframe();

の 1 行をプログラムに加えて，面でなくワイヤフレームで表示している [2]。

[2]W キーでのワイヤフレーム表示と違い，アクターに入っているオブジェクトごとにワイヤフレームか否かを指定できる。

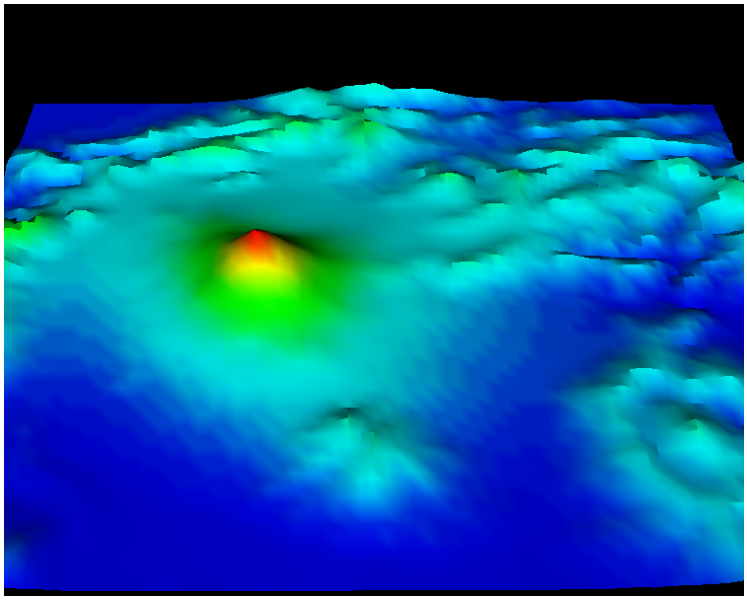

Figure 2.11: 富士山周辺：法線ベクトル修正

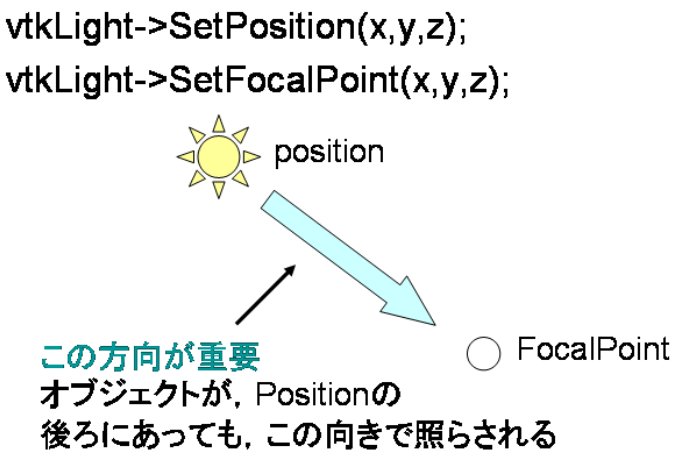

Figure 2.12: 光の方向

2.3.6 補足

Figure 2.13 のワイヤフレームを見るとわかるが，この凹凸つきスライスは，四角のポリゴンで構成されている。この場合，各ポリゴンの 4 つの頂点が同一平面上にあるとは限らない。ゆえに，vtkTriangleFilter (四角などのポリゴンを三角のポリゴンに変換してくれる) をパイプラインの中に入れて，四角のポリゴンをすべて三角のポリゴンに直しておくのが望ましいと思う。

2.4 高さつき等高線

VTK には，可視化した 3 次元オブジェクトを VRML2.0 形式や Inventor 形式で保存する便利な機能がある。高さつき等高線のプログラムを見ながら使い方をみてみる。

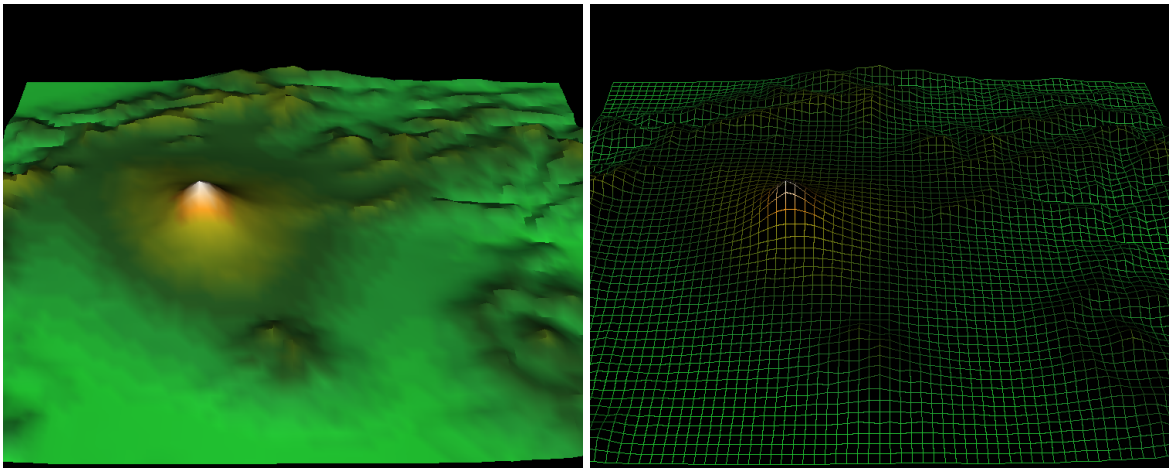

Figure 2.13: 富士山周辺：(左) 色づけ変更, (右) ワイヤフレーム版

2.4.1 サンプルプログラム 3

```
 1  /* UnevenCLines.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkWarpScalar.h>
 4  #include <vtkShortArray.h>
 5  #include <vtkPointData.h>
 6  #include <vtkContourFilter.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkPolyData.h>
 9  #include <vtkVRMLExporter.h>
10  #include <vtkPolyDataMapper.h>
11  #include <vtkRenderWindow.h>
12  #include <vtkActor.h>
13  #include <vtkRenderer.h>
14  #include <vtkProperty.h>
15  #include <vtkRenderWindowInteractor.h>
16  #include <vtkInteractorStyleTrackballCamera.h>
17
18  #define MAP_X 75
19  #define MAP_Y 60
20  #define NCONTLINES 37
21
22  short sdata[MAP_X*MAP_Y];
23
24  int main( int argc, char *argv[] )
25  {
26      FILE *fpi;
27
28      if ((fpi =
29          fopen("./MtFuji.dem", "rb")) == NULL) {
30          puts("cannot open");
```

```
31          exit(1);
32      }
33        fread(sdata, sizeof(short), MAP_X*MAP_Y, fpi);
34
35      fclose(fpi);
36
37      vtkShortArray *sarray = vtkShortArray::New();
38       sarray->SetArray(sdata, MAP_X*MAP_Y, 1);
39
40      vtkImageData *imgData = vtkImageData::New();
41       imgData->SetDimensions(MAP_X, MAP_Y, 1);
42       imgData->SetSpacing(0.1, 0.1, 0);
43       imgData->GetPointData()->SetScalars(sarray);
44
45      vtkContourFilter *contour = vtkContourFilter::New();
46       contour->SetInput(imgData);
47       contour->GenerateValues(NCONTLINES, 0.0, 3776.0);
48
49      vtkLookupTable *lut = vtkLookupTable::New();
50       lut->SetHueRange(0.7, 0.0);
51       lut->Build();
52
53      /* 頂点を移動して，高さをつける*/
54      vtkWarpScalar *warp = vtkWarpScalar::New();
55       warp->SetInput(contour->GetOutput());
56       warp->SetScaleFactor(1.0/3776.0);
57
58      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
59       Mapper->SetInput(warp->GetPolyDataOutput());
60       Mapper->SetLookupTable(lut);
61       Mapper->SetColorModeToMapScalars();
62       Mapper->SetScalarRange(0.0, 3776.0);
63
64      vtkActor *Actor = vtkActor::New();
65       Actor->SetMapper(Mapper);
66
67      vtkRenderer *vren= vtkRenderer::New();
68       vren->AddActor( Actor );
69       vren->SetBackground( 0.0, 0.0, 0.0 );
70
71      vtkRenderWindow *renWin = vtkRenderWindow::New();
72       renWin->AddRenderer( vren );
73       renWin->SetSize( 750, 600 );
74
75      vtkRenderWindowInteractor *iwin
76         = vtkRenderWindowInteractor::New();
77       iwin->SetRenderWindow(renWin);
```

```
 78
 79      vtkInteractorStyleTrackballCamera *trackball
 80         = vtkInteractorStyleTrackballCamera::New();
 81
 82       iwin->SetInteractorStyle(trackball);
 83       iwin->Initialize();
 84       iwin->Start();
 85
 86      /* 高さつき等高線を VRML2.0 のファイルとして保存 */
 87      vtkVRMLExporter * vrml = vtkVRMLExporter::New();
 88       vrml->SetFileName("ContourLines.wrl");
 89       vrml->SetInput(renWin);
 90       vrml->Write();
 91
 92       sarray->Delete();
 93       imgData->Delete();
 94       lut->Delete();
 95       warp->Delete();
 96       contour->Delete();
 97       Mapper->Delete();
 98       Actor->Delete();
 99       vren->Delete();
100       iwin->Delete();
101       trackball->Delete();
102       renWin->Delete();
103       vrml->Delete();
104
105    return 0;
106  }
```

このプログラムを実行すると，Figure 2.14 のように高さのついた等高線が表示される。vtkImageData を vtkRectlinearGrid に変えれば，特に何も直さずとも，Rectilinear のデータを扱える。

2.4.2 サンプルプログラム 3 の簡単な説明

等高線に高さをつけるには，vtkContourFilter で等高線を描いた後，vtkWarpScalar で頂点を移動させるだけでよい。パイプライン構造は，Figure 2.15。

VRML2.0 形式でオブジェクトを保存するのも簡単で，vtkVRMLExporter を使えばよい (**86 - 90** 行を加えるだけである。拡張子は wrl)。vtkBMPWriter は単に画像を保存するだけであるが，こちらは 3 次元構造も保存するので，学会その他の発表時に VRML ビューアでいろいろな角度から可視化オブジェクトを聴講者に見せることができる [3]。類似のクラスとして，vtkOBJExporter (Wavefront 形式)[4], vtkIVExporter (Inventor 形式), vtkRIBExporter (RenderMan 形式) がある。

Inventor 形式 (.iv) のファイルは，OpenGLPerformer を使うと，簡単に CAVE の VR 空間に表示すことがでる。

[3] Cosmo Player や Cortona では見れなかったので，ホームページ掲載は少々問題ありかも。
[4] 色情報が保存されませんでした。

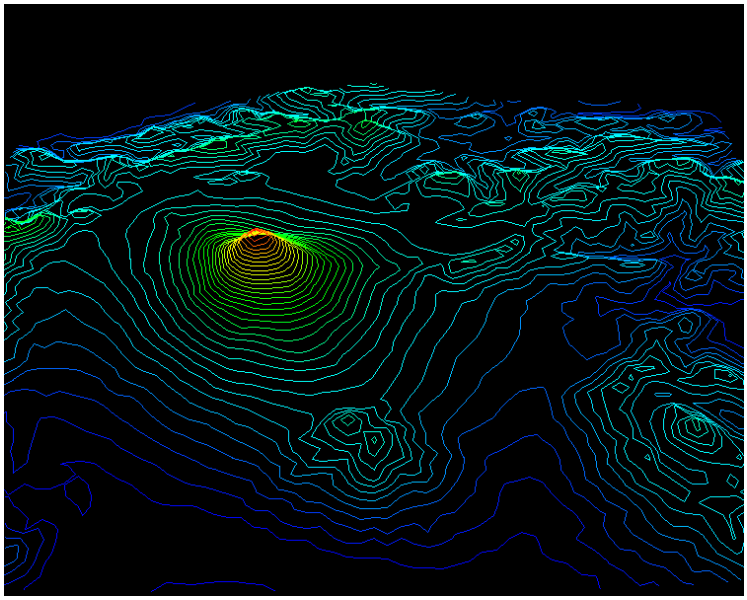

Figure 2.14: 高さつき等高線

2.5 発展

GTOPO30 を使って凹凸つきスライスを作るときに，

- スカラー値と RGB を対応させつつカラーテーブルを作りたい
- 凹凸と色付けを別々のスカラーデータで行いたい

2.5.1 サンプルプログラム 4

前述した 2 件について，次のサンプルプログラム 4 で方法を説明する。

```
 1  /* BumpySlice4.cxx */
 2  #include <vtkShortArray.h>
 3  #include <vtkFieldData.h>
 4  #include <vtkImageData.h>
 5  #include <vtkImageDataGeometryFilter.h>
 6  #include <vtkWarpScalar.h>
 7  #include <vtkPointData.h>
 8  #include <vtkPolyData.h>
 9  #include <vtkColorTransferFunction.h>
10  #include <vtkTriangleFilter.h>
11  #include <vtkPolyDataMapper.h>
12  #include <vtkRenderWindow.h>
13  #include <vtkActor.h>
14  #include <vtkRenderer.h>
15  #include <vtkProperty.h>
16  #include <vtkRenderWindowInteractor.h>
17  #include <vtkInteractorStyleTrackballCamera.h>
18
```

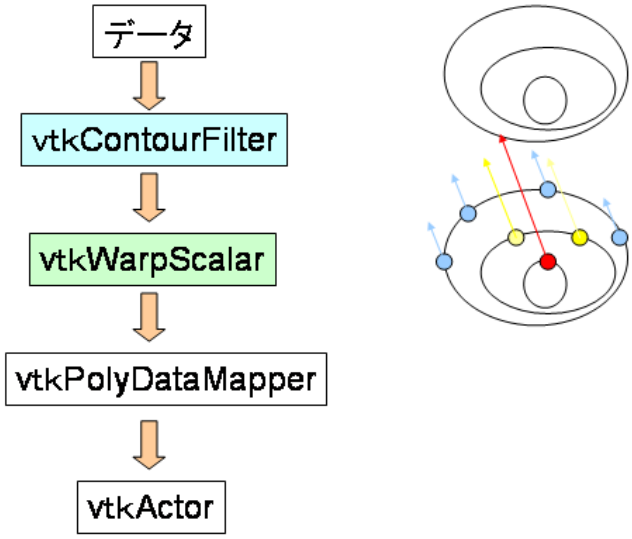

Figure 2.15: サンプルプログラム 3 のパイプライン構造

```
19  #define MAP_X 75
20  #define MAP_Y 60
21
22  short sdata[MAP_X*MAP_Y];
23  short sdata_m[MAP_X*MAP_Y];
24
25  int main( int argc, char *argv[])
26  {
27      int i, j;
28      int size[3] = {MAP_X, MAP_Y, 1};
29      FILE *fpi;
30
31      if ((fpi =
32          fopen("./MtFuji.dem", "rb")) == NULL) {
33          puts("cannot open");
34          exit(1);
35      }
36        fread(sdata, sizeof(short), MAP_X*MAP_Y, fpi);
37
38      fclose(fpi);
39
40      /* 色づけ用のデータを作る */
41      for(j=0;j<MAP_Y;j++){
42       for(i=0;i<MAP_X;i++){
43
44       if(i < MAP_X/2) sdata_m[i+MAP_X*j] = - sdata[i+MAP_X*j];
45              else sdata_m[i+MAP_X*j] = sdata[i+MAP_X*j];
46       }
47      }
```

```
48
49      vtkShortArray *sarray = vtkShortArray::New();
50       sarray->SetArray(sdata, MAP_X*MAP_Y, 1);
51       sarray->SetName("Picard"); /* Picard と名づける */
52
53      /* 色づけ用のデータ */
54      vtkShortArray *sarray2 = vtkShortArray::New();
55       sarray2->SetArray(sdata_m, MAP_X*MAP_Y, 1);
56       sarray2->SetName("Riker"); /* Riker と名づける */
57
58      /* FieldDataに 2つの配列を代入 */
59      vtkFieldData *fd = vtkFieldData::New();
60       fd->AddArray(sarray);
61       fd->AddArray(sarray2);
62
63      vtkImageData *imgData = vtkImageData::New();
64       imgData->SetSpacing(0.1, 0.1, 0);
65       imgData->SetDimensions(size);
66       imgData->GetPointData()->ShallowCopy(fd); /* fd をセット */
67
68       /* スカラーデータとして, Picard を使用 */
69       imgData->GetPointData()->SetActiveScalars("Picard");
70
71      /* スカラー値と RGB を対応させながらカラーテーブルを作る */
72      vtkColorTransferFunction *ctf
73          = vtkColorTransferFunction::New();
74       /*------------- スカラー値,   R,   G,   B ---------*/
75       ctf->AddRGBPoint(-3800.0, 0.0, 0.0, 1.0);
76       ctf->AddRGBPoint( -500.0, 0.0, 1.0, 1.0);
77       ctf->AddRGBPoint(    0.0, 1.0, 1.0, 1.0);
78       ctf->AddRGBPoint(  500.0, 1.0, 1.0, 0.0);
79       ctf->AddRGBPoint( 1800.0, 1.0, 0.0, 0.0);
80       ctf->Build();
81
82      vtkImageDataGeometryFilter *igf
83          = vtkImageDataGeometryFilter::New();
84       igf->SetInput(imgData);
85
86      vtkTriangleFilter *tri = vtkTriangleFilter::New();
87       tri->SetInput(igf->GetOutput());
88
89      vtkWarpScalar *warp = vtkWarpScalar::New();
90       warp->SetInput(tri->GetOutput());
91       warp->SetScaleFactor(1.0/3776.0);
92
93      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
94       Mapper->SetInput(warp->GetPolyDataOutput());
```

```
 95       Mapper->SetLookupTable(ctf);
 96
 97       /* 色づけ用のデータは，FieldData のRikerを使う */
 98       Mapper->SetScalarModeToUsePointFieldData();
 99       Mapper->ColorByArrayComponent("Riker", 0);
100
101      vtkActor *Actor = vtkActor::New();
102       Actor->SetMapper(Mapper);
103
104      vtkRenderer *vren= vtkRenderer::New();
105       vren->AddActor( Actor );
106       vren->SetBackground( 0.0, 0.0, 0.0 );
107
108      vtkRenderWindow *renWin = vtkRenderWindow::New();
109       renWin->AddRenderer( vren );
110       renWin->SetSize( 750, 600 );
111
112      vtkRenderWindowInteractor *iwin
113          = vtkRenderWindowInteractor::New();
114       iwin->SetRenderWindow(renWin);
115
116      vtkInteractorStyleTrackballCamera *trackball =
117       vtkInteractorStyleTrackballCamera::New();
118
119       iwin->SetInteractorStyle(trackball);
120       iwin->Initialize();
121       iwin->Start();
122
123       sarray->Delete();
124       sarray2->Delete();
125       imgData->Delete();
126       fd->Delete();
127       ctf->Delete();
128       igf->Delete();
129       tri->Delete();
130       warp->Delete();
131       Mapper->Delete();
132       Actor->Delete();
133       vren->Delete();
134       iwin->Delete();
135       trackball->Delete();
136       renWin->Delete();
137
138    return 0;
139  }
```

2.5.2 カラーテーブル作成

色付けに関しては，今まで使用してきた vtkLookupTable ではなく，vtkColorTransferFunction を使えばよい。**71 - 80** 行目で，スカラー値と RGB 値を対応させつつカラーテーブルを作成している。たとえば，**75** 行目ではスカラー値 -3800.0 に RGB = (0.0, 0.0, 1.0) を割り当てている。このようにして作ったカラーテーブルを **95** 行目で vtkLookupTable と同じ関数を使って，vtkPolyDataMapper に渡している。
(作成したカラーテーブルは，負の値で青っぽく，正の値で赤っぽくなる。)

2.5.3 別のスカラーで色づけ

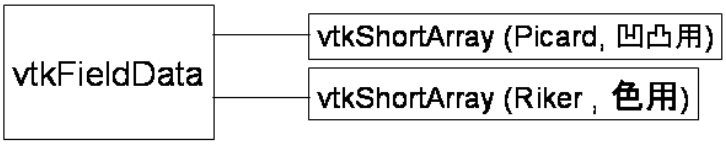

Figure 2.16: Field Data：複数のデータを持つことができる

凹凸をつけるスカラーデータとは別のスカラーデータで色づけするには，vtkFieldData というクラスを使えばよい。このクラスは，配列を複数まとめて持つことができる (FIgure 2.16)。
40 - 47 : 色づけ用のデータを元データから作成，sdata_m に代入。このスカラーデータで色づけをする。実際には，ファイルから温度などのデータを読み込むことになる
49 - 56 : 凹凸用データを sarray に，色づけ用データを sarray2 にセット。それぞれに “Picard”, “Riker”と SetName で名前をつけている
58 - 61 : sarray と sarray2 を fd(vtkFieldData) にセット
66 : 等間隔メッシュデータに，fd(sarray, sarray2 を保持) をセット
68 - 69 : SetActiveScalars で，ポリゴンデータを作るスカラーデータとして，“Picard”を使うように指示
97 - 99 : 色付けに fd にあるデータ・“Riker”を使うように指示
パイプライン構造は Figure 2.17，実行結果は Figure 2.18。X が小さいところと大きいところで，色 (青っぽい → 赤っぽい) が変わっているのが確認できる。これは，圧力の等値面を温度で色づけする，などに応用できる。

2.6 この章のまとめ

- 透視射影・正射影・カメラ設定
- インタラクティブな Window
- 頂点移動・ポリゴンの法線ベクトル・光源の位置
- VRML 2.0 形式での保存
- カラーテーブル作成
- ポリゴンデータをつくるのとは別のデータで，色づけする

本章で紹介したクラスは，Table 2.2 の通り。

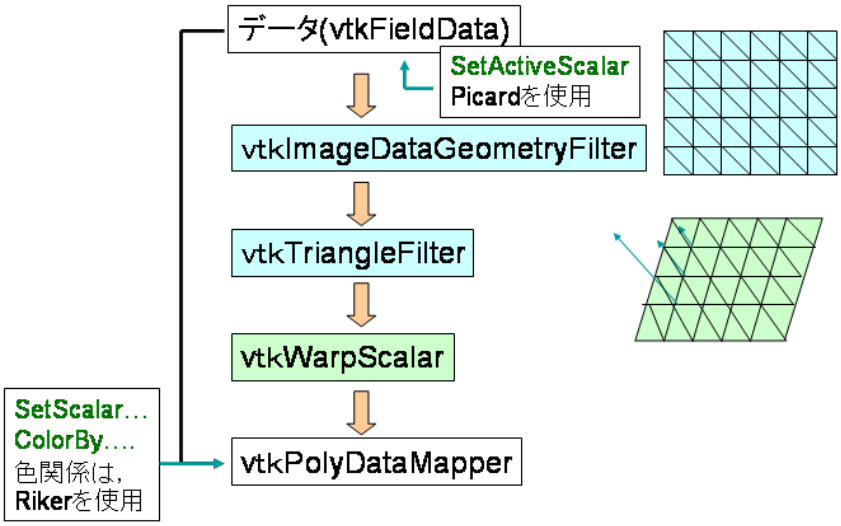

Figure 2.17: サンプルプログラム 4 のパイプライン構造

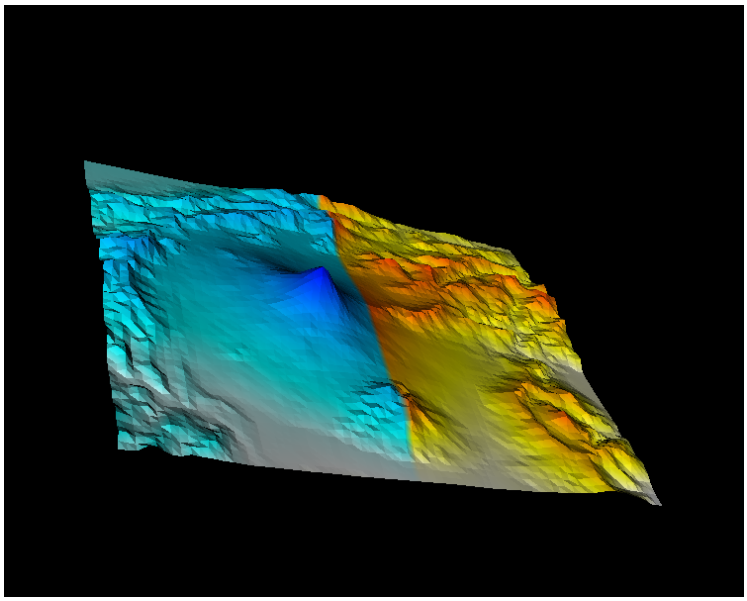

Figure 2.18: 富士山周辺：地面の凹凸は “Picard”，色づけは “Riker”を使用

Table 2.2: 本章で紹介したクラスおよびそのメソッド

頂点移動	vtkWarpScalar
法線ベクトル計算	vtkPolyDataNormals
ポリゴンの三角化	vtkTriangleFilter
照明設定	vtkLight : SetPosition(float, float, float) : SetFocalPoint(float, float, float)
配列	vtkFloatArray : SetName(const char *)
配列関係	vtkFieldData : AddArray(vtkDataArray *)
データセット	vtkImageData : GetPointData()(->ShallowCopy(vtkFieldData *)) : GetPointData()(->SetActiveScalars(const char *))
カラーテーブル	vtkLookupTable : SetNumberOfColors(int n) : SetTableValue(vtkIdType, float [4])
カラーテーブル	vtkColorTransferFunction : AddRGBPoint(float value, float r, float g, float b)
Mapper	vtkPolyDataMapper : SetScalarModeToUsePointFieldData() : ColorByArrayComponent(char *, int)
カメラ設定	vtkCamera : SetPosition(double, double, double) : SetFocalPoint(double, double, double) : SetViewUp(double,double, double) : SetClippingRange(double, double) : SetViewAngle(double) /degree で指定 : ParallelProjectionOn(), 〜Off() : SetParallelScale(double) : Azimuth(double) / degree で指定 : Elevation(double) / degree で指定 : Roll(double) / degree で指定 : Pitch(double) / degree で指定 : Yaw(double) / degree で指定 : OrthogonalizeViewUp()
Window 関係	vtkRenderWindowInteractor, vtkInteractiveStyleTrackball
VRML 他	vtkVRMLExporter, vtkIVExporter, vtkOBJExporter, vtkRIBExporter

Chapter 3

VTK 形式のデータ

3.1 本章の概要

VTK のデータ形式の解説をする。この形式のデータを作ると，vtk～Reader というクラスで読み込むことができるようになり，自ら C/C++ の関数を使いデータを読み込んむ部分を可視化プログラムに組み込まなくて良い。また，データの格子サイズなどをプログラムの中に書き込まなくても良いので，プログラムの汎用性が高まる。

ここでは等間隔，Rectilinear Grid について，VTK 形式のデータの作り方を紹介する。

VTK 形式のデータは，アスキーやバイナリの生データの前に，ヘッダ (格子点のサイズなど) がついているだけである。

3.2 等間隔メッシュのデータ

等間隔メッシュのデータは，vtkImageData ではなく，その派生クラスの vtkStructuredPoints を使用する。ただ，このクラスは将来廃止される予定で，すでに独自のメソッドは，まったくないといってよい。

等間隔メッシュデータ (Float 型のバイナリデータ) のヘッダ部分は，下記のようになる。

```
 1  # vtk DataFile Version 2.0
 2  等間隔メッシュのデータ
 3  BINARY
 4  DATASET STRUCTURED_POINTS
 5  DIMENSIONS 50 100 50
 6  ORIGIN 0.0 0.0 0.0
 7  SPACING 0.1 0.1 0.1
 8  POINT_DATA 250000
 9  SCALARS NameOfData_S float 1
10  LOOKUP_TABLE default
11  スカラーのバイナリデータ
12  VECTORS NameOfData_V float
13  ベクトルのバイナリデータ
```

2行目はコメント，**8**行目の数字は，格子点の総数である。このサンプルの場合，50×100×50=250000。**9** 行目の NameOfData_S はデータの名前 (前章の “Picard”, “Riker”にあたる)，“1”は要素数である。Float 型ではなく，Double 型，Int 型や Unsigned Char 型であれば，**9** 行目の float を double，

int や unsigned_char と書けばよい。**13** 行目のベクトルデータは，X Y Z 成分が $x_1y_1z_1x_2y_2z_2$ と順番に並んでいなければならない。データがバイナリ (Fortran の書式なしにあたる) ではなくアスキー (エディタで開ける形式) で与えられている場合は，**3** 行目を ASCII として，**11, 13** 行目の部分に数値を一つずつスペースで区切りながら，必要に応じて改行しつつ書いていく。

下記のプログラムは，スカラーのバイナリデータ (sample.dat) とベクトルのバイナリデータ (vector_x,dat, vector_y.dat, vector_z.dat) を読み込み，VTK 形式のデータ (sample.vtk) を出力するプログラムである。**69, 112** 行目を入れると，Big Endian のデータが Little Endian に変換される [1]。

```
 1  /* makevtkdata.cxx */
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4  #include <vtkByteSwap.h>
 5
 6  typedef float DATA_TYPE;
 7  char data_type[] = "float";
 8
 9  char output[]    = "./sample.vtk";
10  char comment[]   = "SampleData";
11  int size[3]      = { 50, 100, 50 };
12  float spacing[3] = { 0.1, 0.1, 0.1 };
13  float origin[3]  = { 0.0, 0.0, 0.0 };
14
15  char s_input[]    = "./scalar.dat";
16  char scalarname[] = "ScalarName";
17  int scalar_comp   = 1;
18  DATA_TYPE *s_data;
19  char v_input[3][128];
20  char v_input1[] = "./vector_x.dat";
21  char v_input2[] = "./vector_y.dat";
22  char v_input3[] = "./vector_z.dat";
23  char vectorname[] = "VectorName";
24  DATA_TYPE *v_data;
25  DATA_TYPE *v_data_temp;
26
27  int main(int argc, char **argv)
28  {
29      FILE *fp, *fp2;
30      int total_size;
31
32      total_size = size[0] * size[1] * size[2];
33
34      if ((fp = fopen(output, "wb")) == NULL) {
35          puts("cannot open");
36          exit(1);
```

[1] vtkByteSwap には，逆の変換を行うメソッドも用意されている。また，Double 型などでは，別なメソッドを使用しなければならない。

```
37      }
38
39      fprintf(fp, "# vtk DataFile Version 2.0\n");
40      fprintf(fp, "%s\n", comment);
41      fprintf(fp, "BINARY\n");
42      fprintf(fp, "DATASET STRUCTURED_POINTS\n");
43      fprintf(fp, "DIMENSIONS %d %d %d\n",
44              size[0], size[1], size[2]);
45      fprintf(fp, "ORIGIN %f %f %f\n",
46              origin[0], origin[1], origin[2]);
47      fprintf(fp, "SPACING %f %f %f\n",
48              spacing[0], spacing[1], spacing[2]);
49      fprintf(fp, "POINT_DATA %d\n", total_size);
50
51      fprintf(fp, "SCALARS %s %s %d\n",
52              scalarname, data_type, scalar_comp);
53      fprintf(fp, "LOOKUP_TABLE default\n");
54
55      s_data = (DATA_TYPE *) malloc(sizeof(DATA_TYPE) * total_size);
56
57      if ((fp2 = fopen(s_input, "rb")) == NULL) {
58          puts("cannot open");
59          free(s_data);
60          fclose(fp2);
61          exit(1);
62      }
63
64      fread(s_data, sizeof(DATA_TYPE), total_size, fp2);
65
66      fclose(fp2);
67
68       /* BE から LE に変更する場合に必要 */
69       vtkByteSwap::Swap4BERange(s_data, total_size);
70
71      fwrite(s_data, sizeof(DATA_TYPE), total_size, fp);
72      free(s_data);
73
74    fprintf(fp, "\n");
75
76
77      fprintf(fp, "VECTORS %s %s\n", vectorname, data_type);
78
79      v_data = (DATA_TYPE *) malloc(
80                            sizeof(DATA_TYPE) * total_size * 3);
81
82      v_data_temp = (DATA_TYPE *) malloc(
83                            sizeof(DATA_TYPE) * total_size);
```

```
 84
 85      sprintf(v_input[0], "%s", v_input1);
 86      sprintf(v_input[1], "%s", v_input2);
 87      sprintf(v_input[2], "%s", v_input3);
 88
 89
 90     for(int j=0;j<3;j++){
 91
 92      if ((fp2 = fopen(v_input[j], "rb")) == NULL) {
 93          puts("cannot open");
 94          free(v_data);
 95          free(v_data_temp);
 96          fclose(fp);
 97          exit(1);
 98      }
 99
100       fread(v_data_temp, sizeof(DATA_TYPE), total_size, fp2);
101
102      fclose(fp2);
103
104
105       for(int i=0;i<total_size;i++){
106           v_data[3*i + j] = v_data_temp[i];
107       }
108
109      }
110
111       /* BE から LE に変更する場合に必要 */
112       vtkByteSwap::Swap4BERange(v_data, total_size*3);
113
114      fwrite(v_data, sizeof(DATA_TYPE), total_size*3, fp);
115      free(v_data);
116      free(v_data_temp);
117
118      fprintf(fp, "\n");
119
120      fclose(fp);
121
122  }
```

このようにして作成した VTK 形式のデータは，vtkStructuredPointsReader を利用して読み込むことができる。

3.3 Rectilinear Grid のデータ

RectilinearGrid のヘッダは，下記のようになる。

```
 1  # vtk DataFile Version 2.0
 2  Rectilinear Gridのデータ
 3  BINARY
 4  DATASET RECTILINEAR_GRID
 5  DIMENSIONS 50 100 50
 6  X_COORDINATES float 50
 7  X座標のバイナリデータ
 8  Y_COORDINATES float 100
 9  Y座標のバイナリデータ
10  Z_COORDINATES float 50
11  Z座標のバイナリデータ
12  POINT_DATA 250000
13  SCALARS NameOfData_S float 1
14  LOOKUP_TABLE default
15  スカラーのバイナリデータ
16  VECTORS NameOfData_V float
17  ベクトルのバイナリデータ
```

ORIGIN と SPACING が，X,Y, Z_COORDINATES に置き換えられるだけで，あとは等間隔メッシュのデータと同じである。このようにして作ったデータは，vtkRectilinearGridReader を利用して読み込むことができる。

3.4 この章のまとめ

- 等間隔，RectilinearGrid の VTK 形式のデータ
- エンディアンの変換

本章で紹介したクラスは，Table 3.1 の通り。

Table 3.1: 本章で紹介したクラスおよびそのメソッド

リーダー	vtkStructuredPointsReader vtkRectilinearGridReader
エンディアンの変換	vtkByteSwap ：Swap4BERange(float *, int)

Chapter 4

3次元スカラーデータの可視化 その1

4.1 本章の概要

3 次元のスカラーデータを可視化する。同時に，データの切り出し・間引きの方法にも触れる。

4.2 等値面

3 次元スカラーデータを等値面を使って可視化する。前章までの知識で，なんの問題もなく可視化できる。

4.2.1 サンプルプログラム 1

地球磁気圏のプラズマの温度データ (plasma_data.vtk：等間隔メッシュ(間隔はすべての方向で 0.1)，Float 型，サイズ 256×128×128) を読み込み，等値面を作成し，その等値面を 4 方向から映した画像を同じ Window 内に表示する。

```
 1  /* Isosurface.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkContourFilter.h>
 7  #include <vtkOutlineFilter.h>
 8  #include <vtkPolyDataMapper.h>
 9  #include <vtkRenderWindow.h>
10  #include <vtkCamera.h>
11  #include <vtkActor.h>
12  #include <vtkRenderer.h>
13  #include <vtkProperty.h>
14
15  #include <unistd.h>
16
17  int main( int argc, char *argv[] )
18  {
19      /* カメラ位置 */
```

```
20      float position[4][3] ={ {-40.0, 6.35, 6.35},
21                              {12.75, -40.0, 6.35},
22                              {12.75, 6.35, 52.75},
23                              {20.0, -20.0, 23.75}};
24      /* カメラの上方 */
25      float viewup[4][3] ={{0.0, 0.0, 1.0}, {0.0, 0.0, 1.0},
26                           {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}};
27      /* ビューポート */
28      float viewport[4][4] = {{0.0, 0.0, 0.5, 0.5},
29                              {0.5, 0.0, 1.0, 0.5},
30                              {0.0, 0.5, 0.5, 1.0},
31                              {0.5, 0.5, 1.0, 1.0}};
32      int i;
33
34      vtkStructuredPointsReader *reader
35         = vtkStructuredPointsReader::New();
36       reader->SetFileName("./plasma_data.vtk");
37
38      vtkImageData *imgData;
39       imgData = reader->GetOutput();
40
41      /* 3次元データから等値面を生成する */
42      vtkContourFilter *contour = vtkContourFilter::New();
43       contour->SetInput(imgData);
44       contour->SetValue(0, 12.0);  /* 12.0の等値面作成 */
45       contour->ComputeNormalsOn(); /* 法線ベクトルを計算する */
46
47      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
48       Mapper->SetInput(contour->GetOutput());
49       Mapper->ScalarVisibilityOff();
50
51      vtkActor *Actor = vtkActor::New();
52       Actor->SetMapper(Mapper);
53       Actor->GetProperty()->SetColor(0.5, 0.75, 0.6);
54
55      /* 外枠 */
56      vtkOutlineFilter *outline = vtkOutlineFilter::New();
57       outline->SetInput(imgData);
58
59      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
60       OLMapper->SetInput(outline->GetOutput());
61
62      vtkActor *OLActor = vtkActor::New();
63       OLActor->SetMapper(OLMapper);
64
65      vtkRenderer *vren;
66      vtkRenderWindow *renWin = vtkRenderWindow::New();
```

```
 67
 68        for(i=0;i<4;i++){
 69         vren = vtkRenderer::New();
 70         vren->AddActor(Actor);
 71         vren->AddActor(OLActor);
 72         vren->SetBackground( 0.0, 0.0, 0.0 );
 73         vren->GetActiveCamera()->SetPosition(position[i]);
 74         vren->GetActiveCamera()->SetFocalPoint(12.75, 6.35, 6.35);
 75         vren->GetActiveCamera()->SetViewUp(viewup[i]);
 76         vren->GetActiveCamera()->OrthogonalizeViewUp();
 77         vren->GetActiveCamera()->SetClippingRange(30.0, 200.0);
 78         vren->SetViewport(viewport[i]);
 79         renWin->AddRenderer( vren );
 80         vren->Delete();
 81        }
 82
 83         renWin->SetSize( 800, 600 );
 84
 85       for(i=0;i<10;i++){
 86        renWin->Render();
 87        sleep(1);
 88       }
 89
 90       reader->Delete();
 91       contour->Delete();
 92       outline->Delete();
 93       Mapper->Delete();
 94       Actor->Delete();
 95       OLMapper->Delete();
 96       OLActor->Delete();
 97       renWin->Delete();
 98
 99     return 0;
100   }
```

4.2.2 サンプルプログラム 1 の簡単な説明

前章・前々章に使用した vtkContourFilter[1] に，3 次元のデータを代入しているだけであるし，カメラの設定は前章，ViewPort については前々章で触れているので，何の問題もないはずである。新規箇所は，vtkOutlineFilter で，データの外枠を作るところくらいである。

19 - 31 : カメラ関係の設定値
44 : SetValue メソッドで，12.0 のレベルの等値面を作成する。GenerateValues と違い，ピンポイントで値を指定できる

[1] 3 次元等間隔メッシュデータに特化した vtkMarchingCubes という等値面生成専用クラスも存在する。

45 : 等値「面」であるので，法線も計算する
49 : ScalarVisibilityOff で，スカラー値による色づけはしないことを指示
53 : 等値面の色を指定 (上記と併用する)
56 - 63 : vtkOutlineFilter を使いデータの外枠を作る。作り方は簡単で，データを代入するだけで (**57** 行)，GetOutput() で外枠のポリゴンデータを取り出せる (**60** 行)
実行結果は，Figure 4.1。

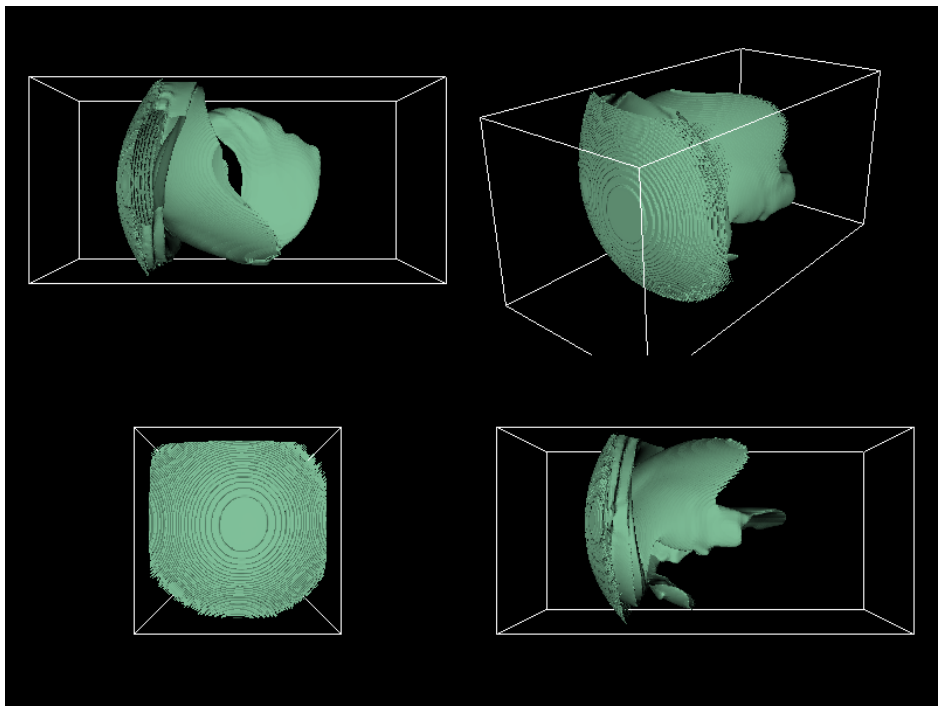

Figure 4.1: 等値面：4 方向から

4.3 データの切り出し：断面切り

VTK を使うと，軸に垂直な面だけでなく，任意の断面で 3 次元データのカラーコンターや等高線を作ることができる。

使用するクラスは，vtkPlane と vtkCutter である。vtkPlane で平面を指定して，vtkCutter にターゲットとなるデータとともに vtkPlane を介して平面の情報を渡すと，断面のポリゴンデータを切り出してくれる。

さらに，カラーバーを表示させてみる。vtkScalarBarActor を使用すれば，容易に画面上にカラーバーを表示させることができる。また，このカラーバーは，オブジェクトを回転させても，静止したままである。

4.3.1 サンプルプログラム 2

斜め断面でカラーコンターを作る。また，vtkScalarBarActor を用いてカラーバーも同時に表示させる。

```
1  /* CrossSection.cxx */
2  #include <vtkImageData.h>
```

```
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkPlane.h>
 7  #include <vtkCutter.h>
 8  #include <vtkLookupTable.h>
 9  #include <vtkOutlineFilter.h>
10  #include <vtkPolyDataMapper.h>
11  #include <vtkRenderWindow.h>
12  #include <vtkActor.h>
13  #include <vtkRenderer.h>
14  #include <vtkScalarBarActor.h>
15  #include <vtkRenderWindowInteractor.h>
16  #include <vtkInteractorStyleTrackballCamera.h>
17
18  int main( int argc, char *argv[] )
19  {
20      float range[2];
21      char datafile[]="./plasma_data.vtk";
22      float plane_normal[3] = {-1.0, 0.5, 0.75};
23      float plane_origin[3] = {10.0, 6.35, 6.35};
24
25      vtkStructuredPointsReader *reader
26         = vtkStructuredPointsReader::New();
27       reader->SetFileName(datafile);
28
29      vtkImageData *imgData;
30       imgData = reader->GetOutput();
31       imgData->Update();
32       imgData->GetScalarRange(range);
33
34      vtkLookupTable *lut = vtkLookupTable::New();
35       lut->SetHueRange(0.7, 0.0);
36       lut->Build();
37
38      /* 任意の断面でスライスを作る */
39      /* 面の情報 */
40      vtkPlane *plane = vtkPlane::New();
41       plane->SetOrigin(plane_origin); /* 面の原点 */
42       plane->SetNormal(plane_normal); /* 面の法線ベクトル */
43
44      vtkCutter *cutter = vtkCutter::New();
45       cutter->SetInput(imgData);
46       cutter->SetCutFunction(plane);
47
48      /* 等高線の場合は，下記もパイプラインに入れる */
49      /*
```

```
50      vtkContourFilter *contour = vtkContourFilter::New();
51       contour->SetInput(cutter->GetOutput());
52       contour->GenerateValues(15, range[0], range[1]);
53      */
54
55      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
56       Mapper->SetInput(cutter->GetOutput());
57       Mapper->SetLookupTable(lut);
58       Mapper->SetColorModeToMapScalars();
59       Mapper->SetScalarRange(range[0], range[1]);
60
61      vtkActor *Actor = vtkActor::New();
62       Actor->SetMapper(Mapper);
63
64      /* データの外枠  */
65      vtkOutlineFilter *outline = vtkOutlineFilter::New();
66       outline->SetInput(imgData);
67
68      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
69       OLMapper->SetInput(outline->GetOutput());
70
71      vtkActor *OLActor = vtkActor::New();
72       OLActor->SetMapper(OLMapper);
73
74       /* カラーバーを表示する */
75      vtkScalarBarActor *scalarbar = vtkScalarBarActor::New();
76       scalarbar->SetTitle("Temperature");
77       scalarbar->SetLookupTable(lut);
78       scalarbar->SetOrientationToVertical();
79       scalarbar->SetWidth(0.075);
80       scalarbar->SetHeight(0.75);
81
82      vtkRenderer *vren= vtkRenderer::New();
83       vren->AddActor(Actor);
84       vren->AddActor(OLActor);
85       vren->AddActor(scalarbar);
86       vren->SetBackground( 0.0, 0.0, 0.0 );
87
88      vtkRenderWindow *renWin = vtkRenderWindow::New();
89       renWin->AddRenderer( vren );
90       renWin->SetSize( 750, 600 );
91
92      vtkRenderWindowInteractor *iwin
93          = vtkRenderWindowInteractor::New();
94       iwin->SetRenderWindow(renWin);
95
96      vtkInteractorStyleTrackballCamera *trackball =
```

```
 97        vtkInteractorStyleTrackballCamera::New();
 98
 99        iwin->SetInteractorStyle(trackball);
100        iwin->Initialize();
101        iwin->Start();
102
103        reader->Delete();
104        lut->Delete();
105        outline->Delete();
106        Mapper->Delete();
107        Actor->Delete();
108        OLMapper->Delete();
109        OLActor->Delete();
110        scalarbar->Delete();
111        iwin->Delete();
112        trackball->Delete();
113        vren->Delete();
114        renWin->Delete();
115
116     return 0;
117   }
```

4.3.2 サンプルプログラム 2 の簡単な説明

40 - 42 : 平面の原点 (plane_origin) と平面の法線ベクトル (plane_normal) を与えて平面を決定する。plane_origin と plane_normal は，**22 - 23** 行で与えられている
44 - 46 : 上記の平面データとターゲットのデータを vtkCutter に渡す
48 - 53 : vtkCutter から直接 vtkPolyDataMapper につながず，vtkContourFilter を通してつなぐと，等高線を生成できる。コメントアウトした部分を参考に
74 - 80 : カラーバーの表示。vtkScalarBarActor という名のとおり，そのまま vtkRenderer につなげばよい (**85** 行)
76 : タイトル
77 : カラーテーブルを入力
78 : SetOrientationToHorizontal() で，バーを横にできる
79 - 80 : 引数の単位は，ViewPort のそれと (0.0 - 1.0) 同じである

データが Rectilinear でも，(リーダーを変えるだけで) 特に何も直さずとも動作する。
vtkPlane と vtkCutter のパイプライン接続は，Figure 4.2 のようになる。実行結果は，Figure 4.3。

4.4 データの切り出し : Volume Of Interest (VOI)

データが大きくなればなるほど，データ全体ではなく，可視化したい部分 (VOI : Volume Of Interest) を切り出して可視化する。VTK には，データを切り出すためのクラスが用意されてい

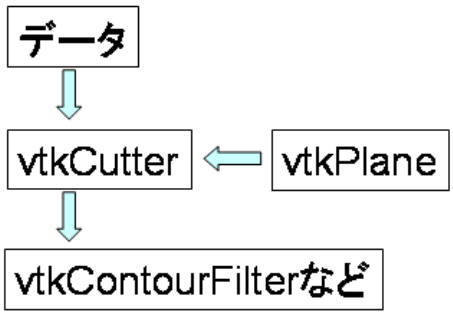

Figure 4.2: vtkPlane と vtkCutter の接続

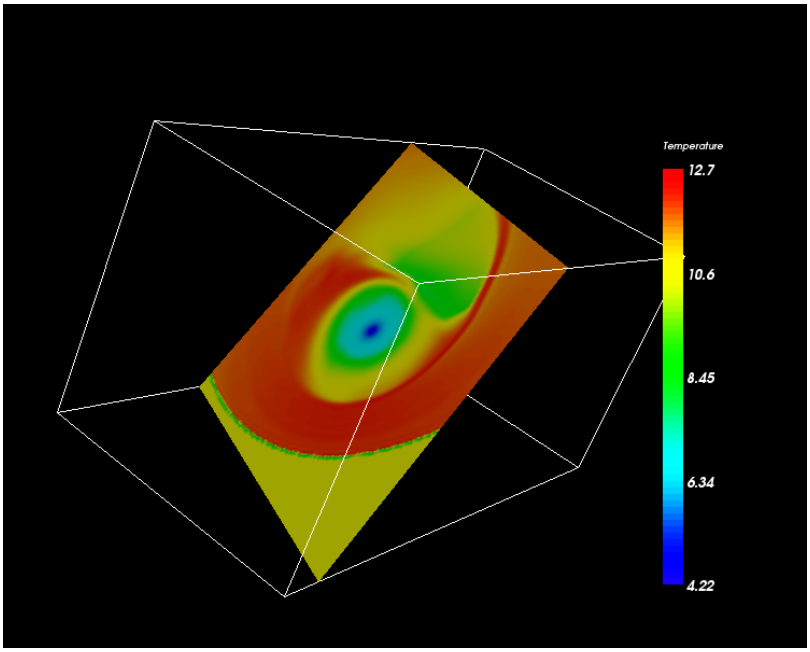

Figure 4.3: 斜め断面とカラーバー

るので，自分で切り出す必要はない[2]。

データの切り出しは，vtkExtractVOI というクラスで行うことができる。このクラスをデータと可視化のためのクラスの間に挿入すればよい (Figure 4.4)。切り出しには，このクラスのメソッド

- SetVOI($i_1, i_2, j_1, j_2, k_1, k_2$)

で，格子点の番号 (0 以上の整数) で指定する (Figure 4.5)。たとえば，本章のサンプルデータで，X 方向の後半を半分切り出すなら

- SetVOI(128, 255, 0, 127, 0,127)

$y = 64$ の平面を切り出すなら，

- SetVOI(0, 255, 64, 64, 0,127)

とすればよい。

[2]計算機に乗せるにはあまりにも大きすぎる場合は，この限りではない。

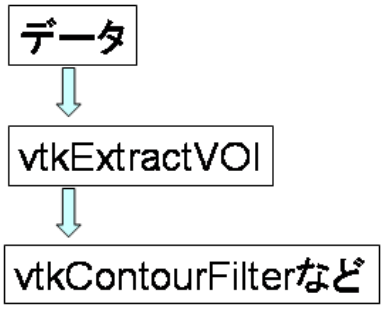

Figure 4.4: vtkExtractVOI の接続位置

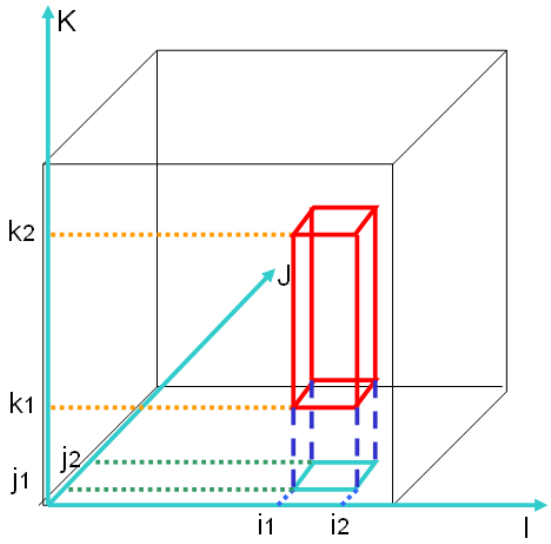

Figure 4.5: 格子点の指定

座標軸に垂直な面でのカラーコンターや等高線を描かせたい場合は，前セクションのように vtkPlane と vtkCutter を利用する方法よりも本セクションのように，平面を切り出して vtkContourFilter を使うほうが，処理時間は短くすむであろう。

4.4.1 サンプルプログラム 3

このプログラムは，サンプルデータを読み込み，そこから vtkExtractVOI で 2 次元および 3 次元のデータを切り出して，それぞれ等高線・等値面で可視化する。さらに，2 次元の面 (切り出す部分) を，オリジナルデータ内で移動させて，そのたびに可視化した画像を連番画像として保存する。

```
1  /* ExtractData.cxx */
2  #include <vtkImageData.h>
3  #include <vtkStructuredPoints.h>
4  #include <vtkStructuredPointsReader.h>
5  #include <vtkPointData.h>
```

```
 6  #include <vtkExtractVOI.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkContourFilter.h>
 9  #include <vtkOutlineFilter.h>
10  #include <vtkAppendPolyData.h>
11  #include <vtkPolyDataMapper.h>
12  #include <vtkRenderWindow.h>
13  #include <vtkCamera.h>
14  #include <vtkActor.h>
15  #include <vtkRenderer.h>
16  #include <vtkProperty.h>
17  #include <vtkRendererSource.h>
18  #include <vtkBMPWriter.h>
19
20  #define NCONTLINES 15
21  #define ISOSURFACE_VALUE 11.0
22
23  int main( int argc, char *argv[] )
24  {
25      float range[2];
26      char datafile[] = "./plasma_data.vtk";
27      char FileName[128];
28
29      /* 切り出す範囲 */
30      int extent_xy[6] = {50, 200, 20, 107, 0, 0};
31      int extent_yz[6] = {0, 0, 20, 107, 20, 107};
32      int extent_zx[6] = {0, 255, 0, 0, 0, 127};
33      int extent_vol[6] = {75, 150, 15, 45, 15, 65};
34
35      vtkStructuredPointsReader *reader
36         = vtkStructuredPointsReader::New();
37       reader->SetFileName(datafile);
38
39      vtkImageData *imgData;
40       imgData = reader->GetOutput();
41       imgData->Update();
42       imgData->GetScalarRange(range);
43
44      vtkLookupTable *lut = vtkLookupTable::New();
45       lut->SetHueRange(0.7, 0.0);
46       lut->Build();
47
48      /* 元データから2次元データの切り出し */
49      vtkExtractVOI *voi_xy = vtkExtractVOI::New();
50       voi_xy->SetInput(imgData);
51
52      vtkExtractVOI *voi_yz = vtkExtractVOI::New();
```

```
53       voi_yz->SetInput(imgData);
54
55      vtkExtractVOI *voi_zx = vtkExtractVOI::New();
56       voi_zx->SetInput(imgData);
57
58      /* 切り出した2次元データの等高線生成 */
59      vtkContourFilter *contour_xy = vtkContourFilter::New();
60       contour_xy->SetInput(voi_xy->GetOutput());
61       contour_xy->GenerateValues(NCONTLINES, range[0], range[1]);
62
63      vtkContourFilter *contour_yz = vtkContourFilter::New();
64       contour_yz->SetInput(voi_yz->GetOutput());
65       contour_yz->GenerateValues(NCONTLINES, range[0], range[1]);
66
67      vtkContourFilter *contour_zx = vtkContourFilter::New();
68       contour_zx->SetInput(voi_zx->GetOutput());
69       contour_zx->GenerateValues(NCONTLINES, range[0], range[1]);
70
71      /* 3次元データを切り出して，等値面を生成 */
72      vtkExtractVOI *voi = vtkExtractVOI::New();
73       voi->SetInput(imgData);
74       voi->SetVOI(extent_vol);
75
76      vtkContourFilter *isosurf = vtkContourFilter::New();
77       isosurf->SetInput(voi->GetOutput());
78       isosurf->SetValue(0, ISOSURFACE_VALUE);
79       isosurf->ComputeNormalsOn();
80
81      /* 等高線と等値面のポリゴンデータを一つにまとめる */
82      vtkAppendPolyData *contours = vtkAppendPolyData::New();
83       contours->AddInput(contour_xy->GetOutput());
84       contours->AddInput(contour_yz->GetOutput());
85       contours->AddInput(contour_zx->GetOutput());
86       contours->AddInput(isosurf->GetOutput());
87
88      vtkPolyDataMapper *CMapper = vtkPolyDataMapper::New();
89       CMapper->SetInput(contours->GetOutput());
90       CMapper->SetLookupTable(lut);
91       CMapper->SetColorModeToMapScalars();
92       CMapper->SetScalarRange(range[0], range[1]);
93
94      vtkActor *CActor = vtkActor::New();
95       CActor->SetMapper(CMapper);
96
97      /* データの外枠 */
98      vtkOutlineFilter *outline_xy = vtkOutlineFilter::New();
99       outline_xy->SetInput(voi_xy->GetOutput());
```

```
100
101     vtkOutlineFilter *outline_yz = vtkOutlineFilter::New();
102      outline_yz->SetInput(voi_yz->GetOutput());
103
104     vtkOutlineFilter *outline_zx = vtkOutlineFilter::New();
105      outline_zx->SetInput(voi_zx->GetOutput());
106
107     vtkOutlineFilter *outline = vtkOutlineFilter::New();
108      outline->SetInput(imgData);
109
110     vtkOutlineFilter *outline_voi = vtkOutlineFilter::New();
111      outline_voi->SetInput(voi->GetOutput());
112
113     /* 外枠のポリゴンデータを一つにまとめる */
114     vtkAppendPolyData *outlines = vtkAppendPolyData::New();
115      outlines->AddInput(outline->GetOutput());
116      outlines->AddInput(outline_voi->GetOutput());
117      outlines->AddInput(outline_xy->GetOutput());
118      outlines->AddInput(outline_yz->GetOutput());
119      outlines->AddInput(outline_zx->GetOutput());
120
121     vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
122      OLMapper->SetInput(outlines->GetOutput());
123
124     vtkActor *OLActor = vtkActor::New();
125      OLActor->SetMapper(OLMapper);
126
127     vtkCamera *camera = vtkCamera::New();
128      camera->SetPosition(-35.0, -15.0, 20.0);
129      camera->SetFocalPoint( 10.0, 6.0, 3.0);
130      camera->SetViewUp( 0.0, 0.0, 1.0);
131      camera->SetClippingRange( 1.0, 100.0);
132      camera->OrthogonalizeViewUp();
133
134     vtkRenderer *vren= vtkRenderer::New();
135      vren->AddActor(CActor);
136      vren->AddActor(OLActor);
137      vren->SetActiveCamera(camera);
138      vren->SetBackground( 0.0, 0.0, 0.0 );
139
140     vtkRenderWindow *renWin = vtkRenderWindow::New();
141      renWin->AddRenderer( vren );
142      renWin->SetSize( 750, 600 );
143
144     vtkRendererSource *rs = vtkRendererSource::New();
145      rs->SetInput(vren);
146      rs->WholeWindowOn();
```

```
147
148      vtkBMPWriter *bw = vtkBMPWriter::New();
149       bw->SetInput(rs->GetOutput());
150
151     for(int i=0;i<128;i++){
152       /* 切り出す場所を変えて，等高線を作り直す */
153       voi_xy->SetVOI(extent_xy[0], extent_xy[1], extent_xy[2],
154                      extent_xy[3], i, i);
155       voi_yz->SetVOI(i*2, i*2, extent_yz[2], extent_yz[3],
156                      extent_yz[4], extent_yz[5]);
157       voi_zx->SetVOI(extent_zx[0], extent_zx[1], i, i,
158                      extent_zx[4], extent_zx[5]);
159
160       renWin->Render();
161
162      sprintf(FileName,"bmps/Anim%d.bmp",i);
163
164      renWin->Render();
165      rs->Modified();
166
167      bw->SetFileName(FileName);
168      bw->Write();
169
170     }
171
172     reader->Delete();
173     lut->Delete();
174     contour_xy->Delete();
175     contour_yz->Delete();
176     contour_zx->Delete();
177     outline->Delete();
178     outline_xy->Delete();
179     outline_yz->Delete();
180     outline_zx->Delete();
181     CActor->Delete();
182     OLActor->Delete();
183     CMapper->Delete();
184     OLMapper->Delete();
185     voi_xy->Delete();
186     voi_yz->Delete();
187     voi_zx->Delete();
188     voi->Delete();
189     isosurf->Delete();
190     contours->Delete();
191     outlines->Delete();
192     vren->Delete();
193     renWin->Delete();
```

```
194      rs->Delete();
195      bw->Delete();
196
197     return 0;
198   }
```

4.4.2 サンプルプログラム 3 の簡単な説明

このプログラムは，vtkExtractVOI クラスを用いて，データから興味のある部分を切り出し，それを vtkContourFilter で等高線や等値面で可視化している。なお，vtkExtractVOI は，等間隔メッシュのデータ専用のクラスで，Rectilinear のデータの場合は，vtkExtractRectilinearGrid, Structured Grid のデータの場合は，vtkExtractGrid をそれぞれ使わなければならない。また，vtkAppendPolyData と言うクラスを使うと，ポリゴンデータを一つにまとめることができて便利である。

29 - 33 : VOI の指定範囲を表す変数
42 : スカラーの範囲 (最大値・最小値) を調べる。range[2] に結果が入る
48 - 56 : 平面データ切り出し用に vtkExtractVOI を 3 つ生成
58 - 69 : vtkExtractVOI で切り出した平面データを vtkContourFilter に渡して，等高線を生成する
71 - 79 : vtkExtractVOI で 3 次元データを切り出し，等値面を生成
81 - 86 : vtkAppendPolyData で，等高線・等値面のポリゴンデータを一つにまとめる。こうすると，Mapper と Actor が一つで済む
97 - 111 : 切り出した範囲それぞれに vtkOutLineFilter で外枠を作る
(**162** : 画像は，bmps というディレクトリ下に保存される)
VOI の範囲の指定は，描画の直前 (**152 - 158** 行) で行っているのにもかかわらず，i=0 でもうまくいく。これは，VTK が描画を指示したときに初めて計算を行うからである。i=1 以降では，すべてを計算しなおすのではなく，変化したところだけ再計算する。なお，**41** 行の「UpDate()」は，計算を強制的に実行させるためのメソッドである。

実行結果は，Figure 4.6。

4.5 データの間引き

可視化したいデータが巨大な場合，あるいはポリゴン数が膨大になって，表示するのが困難な場合は，データあるいはポリゴンを間引くとよい。その方法を見てみる。

4.5.1 サンプルプログラム 4

このプログラムは，

- 間引いたデータで等値面を作成
- オリジナルデータで等値面を作成
- オリジナルデータで等値面を作成してポリゴンデータを間引く

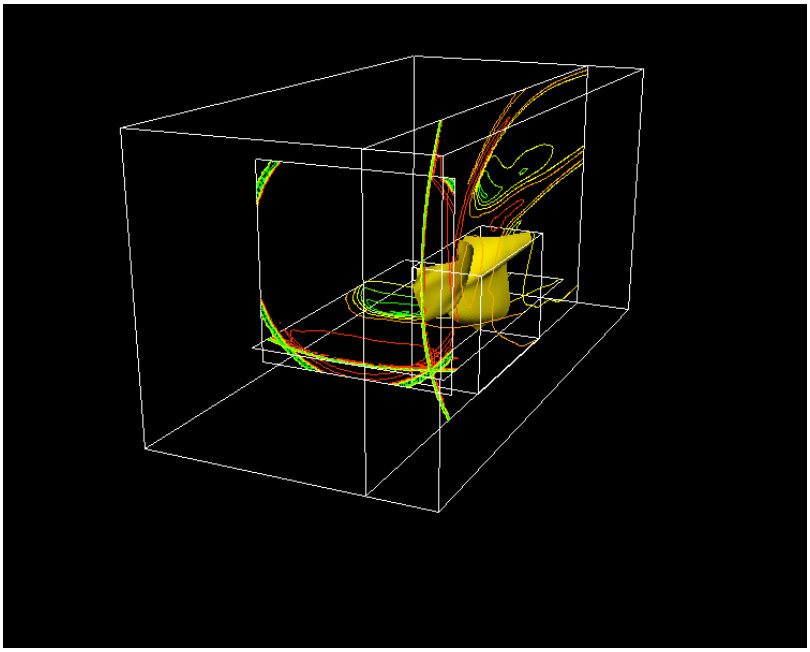

Figure 4.6: 2 次元 & 3 次元データの切り出し

の 3 種類の等値面を同じ Window に表示する。データの格子点の間引きは vtkExtractVOI の SetSampleRate というメソッドで，ポリゴンの間引きは vtkDecimatePro というクラスでできる。これを例えば vtkContourFilter と vtkPolyDataMapper の間に挿入すれば (Figure 4.7)，ポリゴンを間引いた等値面が表示される。

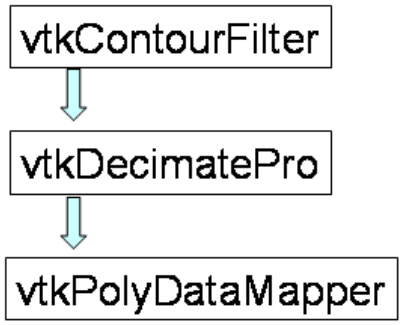

Figure 4.7: vtkDecimatePro の接続位置

```
 1  /* Decimate.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkExtractVOI.h>
 7  #include <vtkDecimatePro.h>
 8  #include <vtkContourFilter.h>
 9  #include <vtkOutlineFilter.h>
10  #include <vtkAppendPolyData.h>
```

```
11  #include <vtkPolyDataMapper.h>
12  #include <vtkPolyDataNormals.h>
13  #include <vtkRenderWindow.h>
14  #include <vtkActor.h>
15  #include <vtkRenderer.h>
16  #include <vtkProperty.h>
17  #include <vtkRenderWindowInteractor.h>
18  #include <vtkInteractorStyleTrackballCamera.h>
19
20  #define ISOSURF_VALUE 12.0
21
22  int main( int argc, char *argv[] )
23  {
24      int size[3];
25      char datafile[] = "./plasma_data.vtk";
26
27      vtkStructuredPointsReader *reader
28         = vtkStructuredPointsReader::New();
29       reader->SetFileName(datafile);
30
31      vtkImageData *imgData;
32       imgData = reader->GetOutput();
33       imgData->Update();
34       imgData->GetDimensions(size);
35
36      /* 1/8 に間引いたデータで等値面を生成 */
37      vtkExtractVOI *voi_all = vtkExtractVOI::New();
38       voi_all->SetInput(imgData);
39       voi_all->SetVOI(0, size[0]-1, 0, size[1]-1, 0, size[2]-1);
40       voi_all->SetSampleRate(2, 2, 2); // 各方向を 1/2 に間引く
41
42      vtkContourFilter *contour1 = vtkContourFilter::New();
43       contour1->SetInput(voi_all->GetOutput());
44       contour1->SetValue(0, ISOSURF_VALUE);
45       contour1->ComputeNormalsOn();
46
47      vtkPolyDataMapper *Mapper1 = vtkPolyDataMapper::New();
48       Mapper1->SetInput(contour1->GetOutput());
49       Mapper1->ScalarVisibilityOff();
50
51      vtkActor *Actor1 = vtkActor::New();
52       Actor1->SetMapper(Mapper1);
53       Actor1->GetProperty()->SetColor(0.75, 0.35, 0.35);
54
55      /* 元データで等値面を生成 */
56      vtkContourFilter *contour2 = vtkContourFilter::New();
57       contour2->SetInput(imgData);
```

```
 58       contour2->SetValue(0, ISOSURF_VALUE);
 59
 60      vtkPolyDataNormals *surf2 = vtkPolyDataNormals::New();
 61       surf2->SetInput(contour2->GetOutput());
 62
 63      vtkPolyDataMapper *Mapper2 = vtkPolyDataMapper::New();
 64       Mapper2->SetInput(surf2->GetOutput());
 65       Mapper2->ScalarVisibilityOff();
 66
 67      vtkActor *Actor2 = vtkActor::New();
 68       Actor2->SetMapper(Mapper2);
 69       Actor2->GetProperty()->SetColor(0.35, 0.75, 0.35);
 70
 71      /* 元データで作った等値面のポリゴンを 7/8 間引く */
 72      vtkDecimatePro *decimate = vtkDecimatePro::New();
 73       decimate->SetInput(contour2->GetOutput());
 74       decimate->PreserveTopologyOn();
 75       decimate->SetTargetReduction(7.0/8.0); // 7/8 間引く
 76
 77      vtkPolyDataNormals *surf3 = vtkPolyDataNormals::New();
 78       surf3->SetInput(decimate->GetOutput());
 79
 80      vtkPolyDataMapper *Mapper3 = vtkPolyDataMapper::New();
 81       Mapper3->SetInput(surf3->GetOutput());
 82       Mapper3->ScalarVisibilityOff();
 83
 84      vtkActor *Actor3 = vtkActor::New();
 85       Actor3->SetMapper(Mapper3);
 86       Actor3->GetProperty()->SetColor(0.35, 0.35, 0.75);
 87
 88      /* データの外枠 */
 89      vtkOutlineFilter *outline = vtkOutlineFilter::New();
 90       outline->SetInput(imgData);
 91
 92      vtkPolyDataMapper *OLAMapper = vtkPolyDataMapper::New();
 93       OLAMapper->SetInput(outline->GetOutput());
 94
 95      vtkActor *OLAActor = vtkActor::New();
 96       OLAActor->SetMapper(OLAMapper);
 97
 98      /* 間引いたデータから作った等値面 */
 99      vtkRenderer *vren1= vtkRenderer::New();
100       vren1->AddActor(Actor1);
101       vren1->AddActor(OLAActor);
102       vren1->SetBackground( 0.0, 0.0, 0.0 );
103       vren1->SetViewport( 0.0, 0.0, 0.333, 1.0);
104
```

```
105      /* 元データから作った等値面 */
106      vtkRenderer *vren2= vtkRenderer::New();
107       vren2->AddActor(Actor2);
108       vren2->AddActor(OLAActor);
109       vren2->SetBackground( 0.0, 0.0, 0.0 );
110       vren2->SetViewport( 0.333, 0.0, 0.666, 1.0 );
111
112      /* 元データから作った等値面のポリゴンを間引いたもの */
113      vtkRenderer *vren3= vtkRenderer::New();
114       vren3->AddActor(Actor3);
115       vren3->AddActor(OLAActor);
116       vren3->SetBackground( 0.0, 0.0, 0.0 );
117       vren3->SetViewport( 0.666, 0.0, 1.0, 1.0);
118
119      vtkRenderWindow *renWin = vtkRenderWindow::New();
120       renWin->AddRenderer( vren1 );
121       renWin->AddRenderer( vren2 );
122       renWin->AddRenderer( vren3 );
123       renWin->SetSize( 1100, 500 );
124
125      vtkRenderWindowInteractor *iwin
126           = vtkRenderWindowInteractor::New();
127       iwin->SetRenderWindow(renWin);
128
129      vtkInteractorStyleTrackballCamera *trackball =
130       vtkInteractorStyleTrackballCamera::New();
131
132       iwin->SetInteractorStyle(trackball);
133       iwin->Initialize();
134       iwin->Start();
135
136       reader->Delete();
137       surf1->Delete();
138       surf2->Delete();
139       surf3->Delete();
140       contour1->Delete();
141       contour2->Delete();
142       decimate->Delete();
143       Mapper1->Delete();
144       Actor1->Delete();
145       Mapper2->Delete();
146       Actor2->Delete();
147       Mapper3->Delete();
148       Actor3->Delete();
149       vren1->Delete();
150       vren2->Delete();
151       vren3->Delete();
```

```
152         iwin->Delete();
153         trackball->Delete();
154         renWin->Delete();
155
156       return 0;
157     }
```

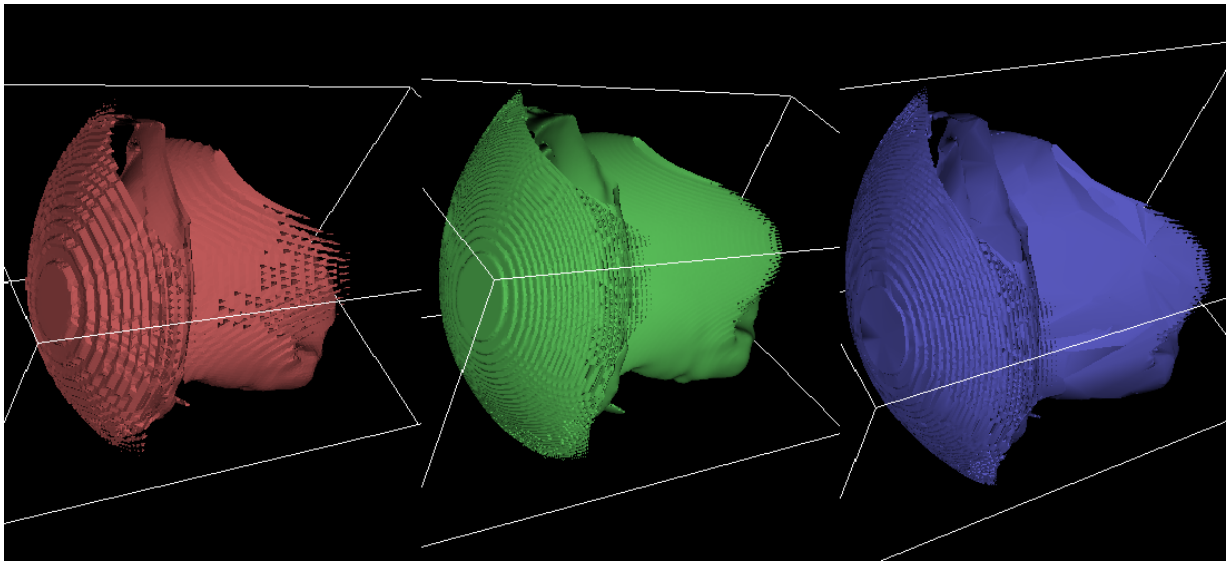

Figure 4.8: 3 種類の等値面 ((左):間引きデータによる等値面，(中):オリジナルデータによる等値面，(右):中の結果のポリゴンを間引いたもの)

4.5.2 サンプルプログラム 4 の簡単な説明

36 - 40 : データ全体を取り出しているが，**40** 行目で，格子点を一つ置きに取るように指定している。だから，取り出したデータの格子点は，オリジナルの 1/8 である。SampleRate を 3 にすると，3 個に 1 個だけ格子点をとることになる [3]。このパラメータは，i, j, k 独立に与えることができる
71 - 75 : **55 - 58** 行目で (オリジナルデータで) 作った等値面のポリゴンデータを間引きする。**75** 行目の SetTragetReduction で，ポリゴン数を 7/8 消去 (1/8 にする) するよう指定している。なお，このクラスは，三角形のポリゴンしか受け付けないので，四角形ポリゴン等が入っている場合は，vtkTriangleFilter などで，あらかじめ三角形ポリゴンに直しておかなければならない
119 - 134 : 3 つとも別々に回転させることができる

実行結果は，Figure 4.8。実行結果を見ると，元のデータを間引きして等値面を作ると多数の穴が開いているが (左)，オリジナルデータから作った等値面のポリゴンデータを間引きしたものには，そのようなものはない。ただし，計算時間は 3 つの中で最もかかる。

前セクションの内容と組み合わせると，データ全体を間引いて，切り出した部分はオリジナルでそれぞれ等値面を作り，同じ Window に表示させることができる。

[3] 指定した範囲の境界の格子点を必ず拾うとは限らない。vtkExtractRectilinearGrid と vtkExtractGrid には，IncludeBoundaryOn() という，境界の格子点を必ず拾わせるメソッドがある。

4.5.3 補足：vtkImageShrink3D

このセクションで紹介した vtkExtractVOI によるデータの間引きは，平均操作はせずに，単純に格子点を飛ばしながら拾っていく。しかし，等間隔メッシュのデータに限り，格子点に与えられているデータの平均を取りつつ，縮小するクラスがある。それが，vtkImageShrink3D である。サンプルプログラムは用意していないので，リファレンスを参照して欲しい。

4.6 この章のまとめ

- データの外枠・カラーバー表示
- 斜め断面
- データの切り出し・間引き
- ポリゴンの間引き

本章で紹介したクラスは，Table 4.1 の通り。

Table 4.1: 本章で紹介したクラスおよびそのメソッド

リーダー	vtkStructuredPointsReader : SetFileName(const char *)
データセット	vtkImageData : GetScalarRange(float [2])
外枠表示	vtkOutlineFilter
断面表示	vtkPlane : SetOrigin(float o[3]) : SetNormal(float n[3])
断面表示	vtkCutter : SetCutFunction(vtkImplicitFunction *)
カラーバー	vtkScalarBarActor : SetTitle(const char *) : SetOrientationTo〜(), Vertical, Horizontal : SetLookupTable(vtkScalarsToColors *) : SetWidth(float) : SetHeight(float)
ポリゴンをまとめる	vtkAppendPolyData
データ切り出し・間引き	vtkExtractVOI (vtkExtractRectilinearGrid, vtkExtractGrid) : SetVOI(int [6]) : SetSampleRate(int, int, int)
データ間引き	vtkImageShrink3D
ポリゴンの間引き	vtkDecimatePro : SetTargetReduction(float)

Chapter 5

3次元スカラーデータの可視化 その2

5.1 本章の概要

ボリューム・レンダリングでスカラーデータを可視化する。また，テクスチャ・マッピングについても触れる。

5.2 ボリューム・レンダリング：レイ・キャスティング法

ボリューム・レンダリングとは，(スカラー) データ全体を半透明な形で可視化する手法である (等値面は，指定されたレベルのみ)。実行方法は，Figure 5.1 にあるとおりシンプルである。視点から視線を延ばして行き，データに入ったところで，サンプリング点における色をその点の不透明度 (伝達関数によって，スカラー値を色と不透明度に変換する) を重みとして加算して行き，投影面のピクセルの色を決める。この方法は，レイ・キャスティング法と呼ばれる。この方法はシンプルではあるが，計算量が多く，可視化像が出力されるまでに一般的には時間がかかる。

VTK には，このレイ・キャスティング法による方法と，後述するテクスチャ・マッピングを利用した方法が実装されている[1]。ただし，いずれの場合も，等間隔メッシュのデータで，Short 型あるいは Unsigned Char 型のデータのみのサポートとなっている。

5.2.1 サンプルプログラム 1

このサンプル・プログラムは，レイ・キャスティング法によって，ボリューム・レンダリングを行うプログラムである。サンプルデータは，球状トカマクの圧力 (p128.vtk : Unsigned Char 型 (0 - 255)，サイズ 128×128×128) である。

```
1  /* VolumeRendering_rc.cxx */
2  #include <vtkImageData.h>
3  #include <vtkStructuredPoints.h>
4  #include <vtkStructuredPointsReader.h>
5  #include <vtkPointData.h>
6  #include <vtkPiecewiseFunction.h>
7  #include <vtkColorTransferFunction.h>
8  #include <vtkVolumeProperty.h>
```

[1]ボリュームプロという，ボリューム・レンダリング専用ハードウェアを利用するクラスも用意されている。

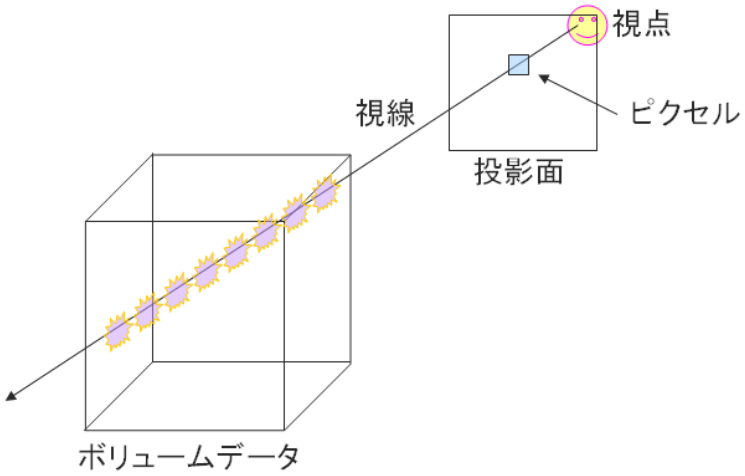

Figure 5.1: ボリューム・レンダリングの実行方法 (レイ・キャスティング法)

```
 9  #include <vtkVolumeRayCastCompositeFunction.h>
10  #include <vtkVolumeRayCastMapper.h>
11  #include <vtkVolume.h>
12  #include <vtkCubeAxesActor2D.h>
13  #include <vtkOutlineFilter.h>
14  #include <vtkPolyDataMapper.h>
15  #include <vtkRenderWindow.h>
16  #include <vtkActor.h>
17  #include <vtkRenderer.h>
18  #include <vtkCamera.h>
19  #include <vtkRenderWindowInteractor.h>
20  #include <vtkInteractorStyleTrackballCamera.h>
21
22  int main( int argc, char *argv[] )
23  {
24      char datafile[] = "./p128.vtk";
25
26      vtkStructuredPointsReader *reader
27         = vtkStructuredPointsReader::New();
28       reader->SetFileName(datafile);
29
30      vtkImageData *imgData;
31       imgData = reader->GetOutput();
32
33       /* 色の伝達関数を作成 */
34       vtkColorTransferFunction *tf4color
35               = vtkColorTransferFunction::New();
36        tf4color->AddHSVPoint(0,  0.7,                  1.0, 1.0);
```

```
37        tf4color->AddHSVPoint(63, 0.7*(1.0-63.0/255.0), 1.0, 1.0);
38        tf4color->AddHSVPoint(127,0.7*(1.0-127.0/255.0),1.0, 1.0);
39        tf4color->AddHSVPoint(190,0.7*(1.0-190.0/255.0),1.0, 1.0);
40        tf4color->AddHSVPoint(255,0.0,                  1.0, 1.0);
41
42       /* 不透明度の伝達関数を作成 */
43       vtkPiecewiseFunction *tf4opacity
44                    = vtkPiecewiseFunction::New();
45        tf4opacity->AddPoint(0,   0.0);
46        tf4opacity->AddPoint(1,   0.01);
47        tf4opacity->AddPoint(128, 0.02);
48        tf4opacity->AddPoint(200, 0.02);
49        tf4opacity->AddPoint(255, 0.05);
50
51       /* 伝達関数を vtkVolumeProperty に代入 */
52       vtkVolumeProperty *vp = vtkVolumeProperty::New();
53        vp->SetColor(tf4color);
54        vp->SetScalarOpacity(tf4opacity);
55        vp->SetInterpolationTypeToLinear();
56  //    vp->ShadeOn(); /* 陰つけができる */
57  //    vp->SetAmbient(0.7);
58  //    vp->SetDiffuse(0.3);
59  //    vp->SetSpecular(0.3);
60
61      /* ボリュームレンダリング (Ray Casting) 用 Mapper */
62       vtkVolumeRayCastMapper *vMapper
63                = vtkVolumeRayCastMapper::New();
64
65       vtkVolumeRayCastCompositeFunction *cfunction
66         = vtkVolumeRayCastCompositeFunction::New();
67
68        vMapper->SetVolumeRayCastFunction(cfunction);
69        vMapper->SetSampleDistance(0.5);
70        vMapper->SetInput(imgData);
71
72        /* データの一部切り取りなどができる */
73  //    vMapper->SetCroppingRegionPlanes(0.0, 90.0,
74  //                                     0.0, 127.0, 0.0, 127.0);
75  //    vMapper->SetCroppingRegionFlagsToSubVolume();
76  //    vMapper->CroppingOn();
77
78       /* ボリュームレンダリング用の Actor */
79       vtkVolume *Volume = vtkVolume::New();
80        Volume->SetMapper(vMapper);
81        Volume->SetProperty(vp);
82
83      /* データの外枠 */
```

```
 84      vtkOutlineFilter *outline = vtkOutlineFilter::New();
 85       outline->SetInput(imgData);
 86
 87      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
 88       OLMapper->SetInput(outline->GetOutput());
 89
 90      vtkActor *OLActor = vtkActor::New();
 91       OLActor->SetMapper(OLMapper);
 92
 93     /* 軸の生成 */
 94      vtkCubeAxesActor2D *Axes = vtkCubeAxesActor2D::New();
 95       Axes->SetInput(imgData);
 96
 97      vtkCamera *camera = vtkCamera::New();
 98       camera->SetPosition(-200.0, 63.5, 200.0);
 99       camera->SetFocalPoint(63.5, 63.5, 63.5);
100       camera->SetViewUp(0.0, 1.0, 0.0);
101       camera->OrthogonalizeViewUp();
102       camera->SetClippingRange(30.0, 2000.0);
103
104      vtkRenderer *vren= vtkRenderer::New();
105       vren->AddVolume(Volume);
106       vren->AddActor (OLActor);
107       vren->AddActor (Axes);
108       vren->SetActiveCamera (camera);
109       vren->SetBackground( 0.0, 0.0, 0.0 );
110
111       Axes->SetCamera(vren->GetActiveCamera()); /* これも忘れずに */
112
113      vtkRenderWindow *renWin = vtkRenderWindow::New();
114       renWin->AddRenderer( vren );
115       renWin->SetSize( 400, 300 );
116
117      vtkRenderWindowInteractor *iwin
118                  = vtkRenderWindowInteractor::New();
119       iwin->SetRenderWindow(renWin);
120
121      vtkInteractorStyleTrackballCamera *trackball =
122       vtkInteractorStyleTrackballCamera::New();
123
124       iwin->SetInteractorStyle(trackball);
125       iwin->Initialize();
126       iwin->Start();
127
128       reader->Delete();
129       tf4color->Delete();
130       tf4opacity->Delete();
```

```
131       vp->Delete();
132       cfunction->Delete();
133       vMapper->Delete();
134       Volume->Delete();
135       outline->Delete();
136       OLActor->Delete();
137       Axes->Delete();
138       iwin->Delete();
139       trackball->Delete();
140       vren->Delete();
141       camera->Delete();
142       renWin->Delete();
143
144     return 0;
145   }
```

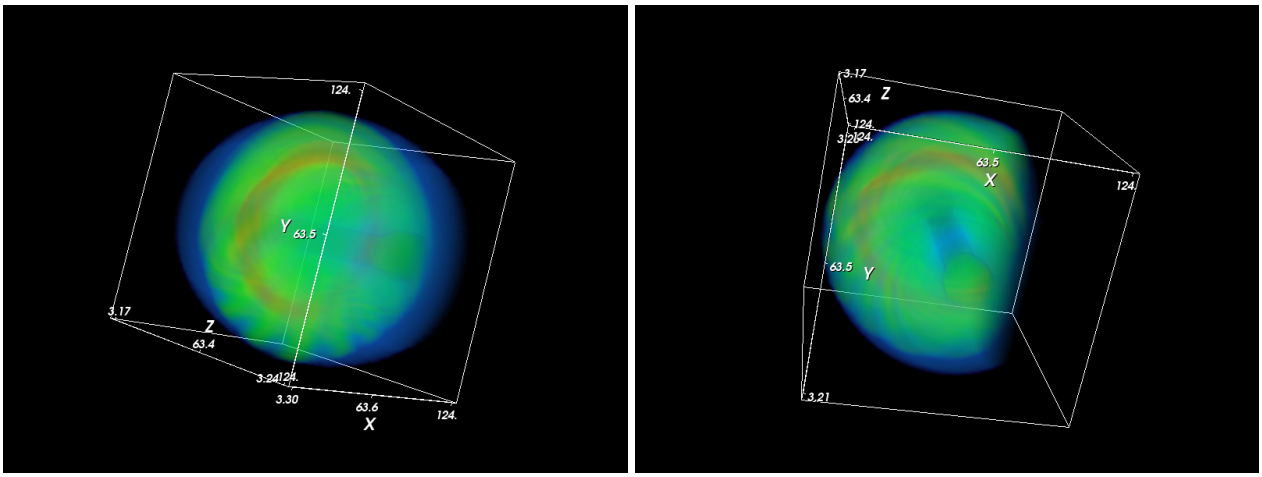

Figure 5.2: 球状トカマクの圧力分布：右図は，一部を Cropping している

5.2.2 伝達関数

ボリューム・レンダリングを実行するには，伝達関数を定義しなければならない。伝達関数とは，スカラー値と色・不透明度の関係を表した関数である。伝達関数という名称は，本章で始めて出たが，カラーコンターや等高線などで，第 1 章から使っている。スカラー値が小さいところは青っぽく，大きいところは赤っぽくするよう vtkLookupTable で設定してきたが，まさにそれである。ただし，ボリューム・レンダリングには，色のほかに，スカラー値と不透明度 (アルファ値:RGBA の A) の関係も決めなければならない。不透明度の設定しだいで，得られる画像はかなり違ったものとなる。はじめは，消したいスカラー値の不透明度を 0.0 に近く，強調したい (重要な) スカラー値の不透明度を大きく (といっても 0.3 程度でよい) 設定してみるとよい。

VTK では，色の伝達関数を vtkColorTransferFunction で，不透明度を vtkPiecewiseFunction で設定する。

5.2.3 サンプルプログラム 1 の簡単な説明

前章までのサンプルプログラムと大きく違うところは,

- vtkPolyDataMapper → vtkVolumeRayCastMapper,
- vtkActor → vtkVolume,

となっているところである。ポリゴンデータを作って映す，という形ではないので，Mapper と Actor も別物が用意されている (Figure 5.3)。

ボリューム・レンダリングに必要なパラメータとして，先述した伝達関数がある。伝達関数で，スカラー値と色・不透明度の関係を決める。色の伝達関数は **33 - 40** 行で，不透明度の伝達関数は **42 - 49** 行で作成している。そしてそれらの伝達関数を vtkVolumeProperty を介して vtkVolume に渡している。また vtkCubeAxesActor2D を使って，軸を描いている。表示される画像は，Figure 5.2

33 - 40 : vtkColorTransferFunction を使って，スカラー値と色の関係を決めている。前々章と違うのは，RGB ではなく HSV で色を指定しているところである。ちなみに **36** と **40** 行だけでは，色の変化は青 → 紫 → 赤となる。理由は RGB の値が補間されるからである
42 - 49 : vtkPiecewiseFunction を使って，不透明度の伝達関数を作成している。AddPoint メソッドでスカラー値と不透明度の関係を決めている (一つ目の引数がスカラー値，二つ目が不透明度)。消したいスカラー値の不透明度を 0.0 にして，強調したいスカラー値の不透明度を大きくする
53 - 55 : 伝達関数を vtkVolumeProperty にセットして，**81** 行で vtkVolume(Actor にあたる) にわたしている。
69 : レイの刻み幅を指定している。刻みが小さいほど絵は綺麗になるが，処理時間が大きくなる
93 - 95, 111 : これだけで，軸を出せる

VTK のボリューム・レンダリングは，陰つけやデータの切り出しも簡単にできる。例えば，vtkVolumeProperty で ShadeOn() とするだけで陰がつくし (**56** 行)，**73 - 76** 行を入れるだけで，Figure 5.2 の右図のように一部を消すことができる (**73 - 74** 行での数値は，vtkExtractVOI のように格子番号ではなく，座標値を入れる)。

このボリューム・レンダリングは，計算負荷が高いので，PC で実行するときは，インタラクティブな Window で見るのではなく，カメラ設定で見たい角度からの画像を保存するという使い方の方が，良いかもしれない。

5.3 カラーコンター再び : テクスチャ・マッピングの利用

第 1 章で，カラーコンターの表示方法を紹介したが，データが等間隔の場合，別な方法もある。おそらく，以下に紹介する方法の方が，(表示された後は) 高速である。

5.3.1 テクスチャ・マッピング

OpenGL などで建物を描くとき，単なる直方体ではなく，窓やドアを描きたい。それらを描くとき，ポリゴンの頂点に色を指定して描いていくというのは，途方も無い作業であるし，ポリゴン数が膨大になり描画に多大な時間を要する。これを解決してくれるのが，テクスチャ・マッピングである。これは，簡単に言うと，ポリゴンに画像を貼り付ける機能である。建物を描く

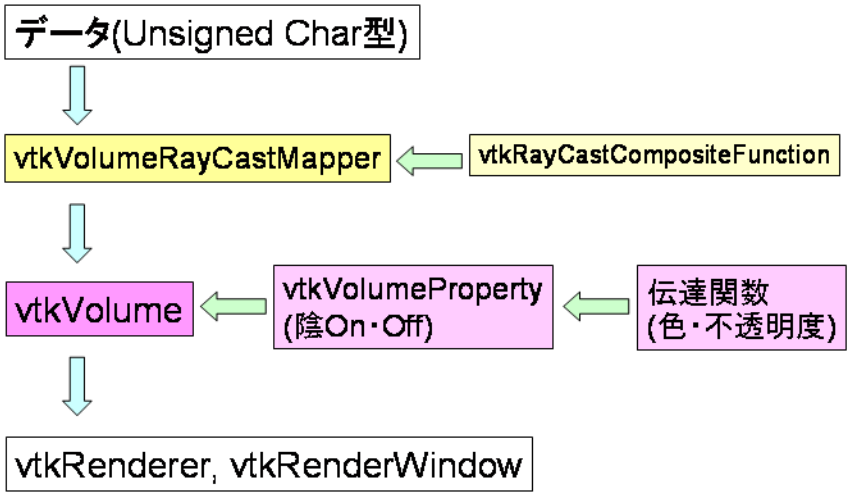

Figure 5.3: サンプルプログラム 1 のパイプライン構造

とき，建物の外形をポリゴンで作って，そのポリゴンにデジタルカメラで撮った写真 (画像) を貼り付ければ，簡単に窓やドアをつけれるし，質感も出せる (Figure 5.4)。この機能を使っても，カラーコンターが可能なことが理解していただけると思う。

サンプルプログラム 2 を引数を “1”にして実行すると，スライス (y=127) のテクスチャを使ったカラーコンターが表示される。

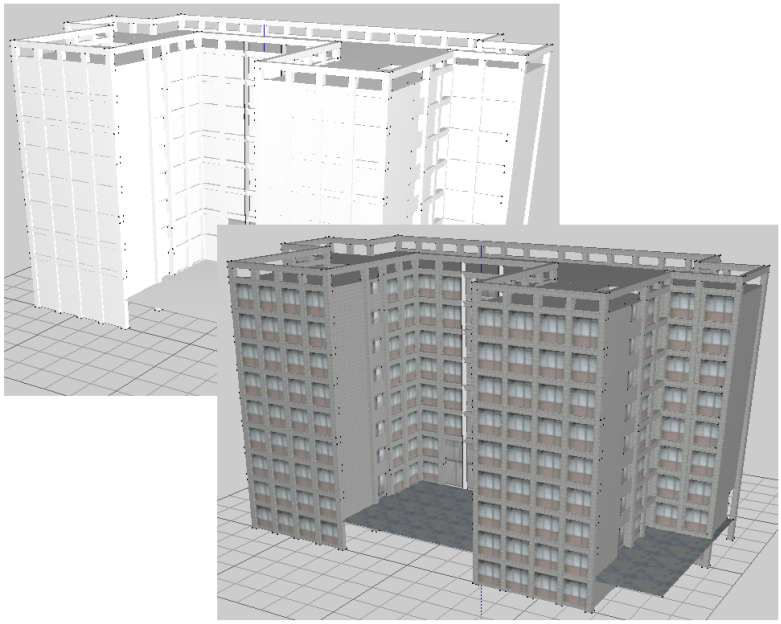

Figure 5.4: 建物 (テクスチャマップされる前後)

5.3.2 サンプルプログラム 2

このプログラムは，テクスチャマッピングを使ったカラーコンターを表示するプログラムである。とりあえず，引数を “1”にして実行していただきたい (データは，前章とおなじ地球磁気圏

の温度データである)。

```
 1  /* Colorcontour_again.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkImageDataGeometryFilter.h>
 6  #include <vtkPointData.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkOutlineFilter.h>
 9  #include <vtkImageActor.h>
10  #include <vtkImageMapToColors.h>
11  #include <vtkPolyDataMapper.h>
12  #include <vtkRenderWindow.h>
13  #include <vtkActor.h>
14  #include <vtkRenderer.h>
15  #include <vtkRenderWindowInteractor.h>
16  #include <vtkInteractorStyleTrackballCamera.h>
17
18  #define MAX_SLICES 120
19
20  int main( int argc, char *argv[] )
21  {
22      int size[3];
23      float range[2];
24      char datafile[] = "./plasma_data.vtk";
25      float color_Table[4];
26      int i, nslices, dy;
27      float fdy;
28
29      if(argc != 2){
30        printf("Input the Number of slices (1-%d)\n", MAX_SLICES);
31        return 1;
32       }
33
34      nslices = atoi(argv[1]);
35
36      if(nslices < 1 || nslices > MAX_SLICES){
37         printf("Set 1 - %d\n", MAX_SLICES);
38         return 1;
39      }
40
41      vtkStructuredPointsReader *reader
42         = vtkStructuredPointsReader::New();
43       reader->SetFileName(datafile);
44
45      vtkImageData *imgData;
```

```
46       imgData = reader->GetOutput();
47       imgData->Update();
48       imgData->GetDimensions(size);
49       imgData->GetScalarRange(range);
50
51      vtkLookupTable *lut = vtkLookupTable::New();
52       lut->SetHueRange(0.7, 0.0);
53       lut->SetNumberOfTableValues(256);
54       lut->SetRange(range[0], range[1]);
55       lut->Build();
56
57       /* lut の不透明度 (RGBA の A) のみ変更する */
58       for(i=0;i<256;i++){
59        lut->GetTableValue(i, color_Table);
60        color_Table[3] = 1.0/nslices;
61        lut->SetTableValue(i, color_Table);
62       }
63
64      /* スカラー値 -> 色 */
65      vtkImageMapToColors *colordata;
66
67      /* テクスチャを使用する場合の Actor */
68      vtkImageActor *IActor;
69
70      if(nslices == 1) fdy = 0.0;
71         else fdy = (size[1]-1.0)/(nslices-1.0);
72
73      vtkRenderer *vren= vtkRenderer::New();
74
75     for(i=0;i<nslices;i++){
76       colordata = vtkImageMapToColors::New();
77       colordata->SetInput(imgData);
78       colordata->SetLookupTable(lut);
79
80       IActor = vtkImageActor::New();
81       IActor->SetInput(colordata->GetOutput());
82       dy = (int)(fdy* i);
83       IActor->SetDisplayExtent(0, size[0]-1,
84         size[1] -1 - dy, size[1]-1-dy, 0, size[2]-1);
85
86       vren->AddActor(IActor);
87       IActor->Delete();
88       colordata->Delete();
89      }
90
91      /* データの外枠 */
92      vtkOutlineFilter *outline = vtkOutlineFilter::New();
```

```
 93       outline->SetInput(imgData);
 94
 95      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
 96       OLMapper->SetInput(outline->GetOutput());
 97
 98      vtkActor *OLActor = vtkActor::New();
 99       OLActor->SetMapper(OLMapper);
100
101       vren->AddActor(OLActor);
102       vren->SetBackground( 0.0, 0.0, 0.0 );
103
104      vtkRenderWindow *renWin = vtkRenderWindow::New();
105       renWin->AddRenderer( vren );
106       renWin->SetSize( 800, 600 );
107
108      vtkRenderWindowInteractor *iwin
109             = vtkRenderWindowInteractor::New();
110       iwin->SetRenderWindow(renWin);
111
112      vtkInteractorStyleTrackballCamera *trackball =
113       vtkInteractorStyleTrackballCamera::New();
114
115       iwin->SetInteractorStyle(trackball);
116       iwin->Initialize();
117       iwin->Start();
118
119       reader->Delete();
120       lut->Delete();
121       outline->Delete();
122       OLMapper->Delete();
123       OLActor->Delete();
124       iwin->Delete();
125       trackball->Delete();
126       vren->Delete();
127       renWin->Delete();
128
129     return 0;
130  }
```

5.3.3 サンプルプログラム 2 の簡単な説明

引数を “1”にして実行したときの表示が，Figure 5.5 である。

このサンプルプログラムは，vtkImageMapToColors というクラスで，データを色情報に変換，vtkImageActor というテクスチャ・マッピングされたポリゴンを表示する Actor でカラーコンター (画像) を表示している。表示するスライスは，**83 - 84** 行目で，指定している。

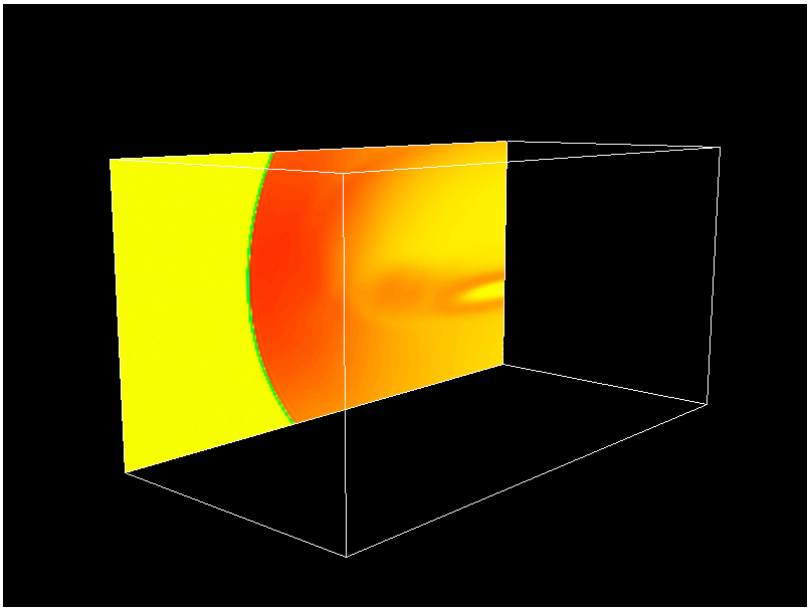

Figure 5.5: テクスチャ・マッピングを利用したカラーコンター

5.3.4 不透明度の設定

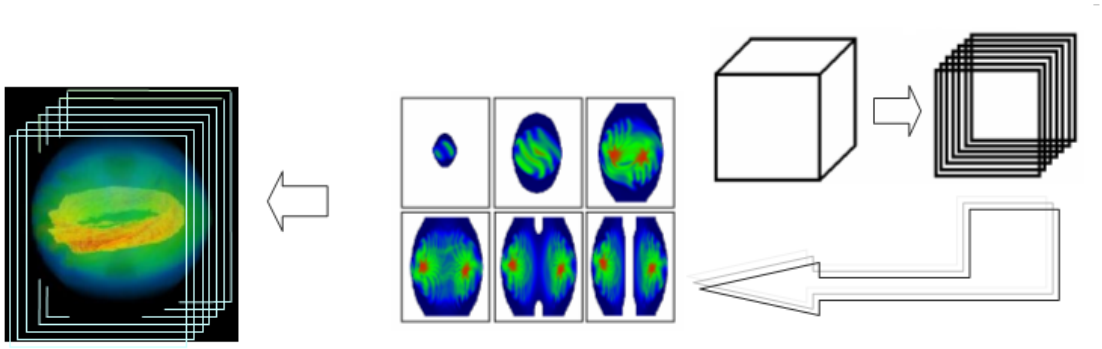

Figure 5.6: テクスチャ・マッピングを利用したボリューム・レンダリング

テクスチャ・マッピングでは，画像のピクセルに不透明度を設定することも可能である。さらに，半透明にしたスライスを前後に並べることで，混合された絵を表示することも可能である。これを利用すると，簡易的にボリューム・レンダリングができる。つまり，3 次元データをスライスしてそれぞれを画像化，不透明度を与えて前後に並べる (Figure 5.6)。

サンプルプログラム 2 で，これを実験できる。引数は，スライスの数である。また，不透明度は，**60** 行目で (1.0/スライス数) になるように設定している。スライス数が上がるにつれ，ボリューム・レンダリングらしくなっていくのがわかる (Figure 5.7)。このプログラムは，Y(の負方向から見た) 方向からしか混合された絵を表示できないが，X, Z 方向用のテクスチャセットも作って，視点の位置に応じて切り替えるようにすれば，全方向に対応できる。これが，次に紹介する (2 次元) テクスチャ・マッピングを利用したボリューム・レンダリングである。

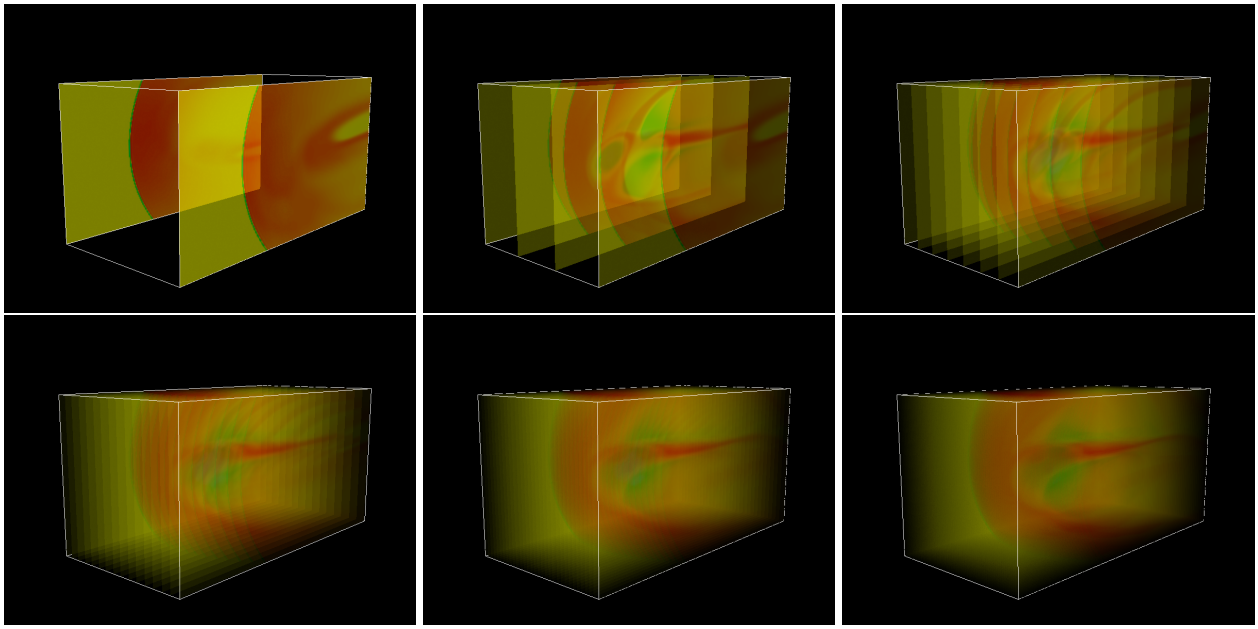

Figure 5.7: テクスチャ・マッピングを利用したボリューム・レンダリング実験; 左上からスライス数が，2, 4, 8, 16, 32, 64 枚

5.4 ボリューム・レンダリング : テクスチャ・マッピングの利用

テクスチャ・マッピングを利用したボリュームレンダリングを行うためのクラスが，VTK には用意されている。一般的には，レイ・キャスティング法よりも高速である。

5.4.1 サンプルプログラム 3

テクスチャ・マッピング利用のボリュームレンダリングを使う場合は，レイ・キャスティングバージョンのプログラムの vtkVolumeRayCastMapper を vtkVolumeTextureMapper2D に変えるだけで良い。さらに，このプログラムで，Float 型 (Unsigned Char 型以外なら Float でなくても良い) のデータを読み込み，それを VTK のフィルタで Unsigned Char 型に直す方法も紹介する。また，球体のポリゴンデータ生成やビルボード処理されたテキストの表示方法も紹介する。データは，引き続き地球磁気圏の温度 (ただし間隔は 1.0) である。

```
 1  /* VolumeRendering_2dt.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkSphereSource.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkOutlineFilter.h>
 9  #include <vtkImageShiftScale.h>
10  #include <vtkPiecewiseFunction.h>
11  #include <vtkColorTransferFunction.h>
12  #include <vtkVolumeProperty.h>
13  #include <vtkVolumeTextureMapper2D.h>
```

```
14  #include <vtkVolume.h>
15  #include <vtkPolyData.h>
16  #include <vtkPolyDataMapper.h>
17  #include <vtkRenderWindow.h>
18  #include <vtkActor.h>
19  #include <vtkRenderer.h>
20  #include <vtkProperty.h>
21  #include <vtkCamera.h>
22  #include <vtkScalarBarActor.h>
23  #include <vtkVectorText.h>
24  #include <vtkFollower.h>
25  #include <vtkRenderWindowInteractor.h>
26  #include <vtkInteractorStyleTrackballCamera.h>
27
28  int main( int argc, char *argv[] )
29  {
30      int i;
31      float range[2];
32      char datafile[] = "./tempe200.vtk";
33      float shift, scale;
34      float opacity_table[256];
35      float color_table[3*256], temp_color[4];
36      float earth_position[3] = {100.5, 63.5, 63.5};
37
38      vtkStructuredPointsReader *reader
39         = vtkStructuredPointsReader::New();
40       reader->SetFileName(datafile);
41
42      vtkImageData *imgData;
43       imgData = reader->GetOutput();
44       imgData->Update();
45       imgData->GetScalarRange(range);
46
47       /* Float 型のデータを Unsigned Char 型に変換する */
48       shift = -range[0];
49       scale = 255.0/(range[1]-range[0]);
50
51       vtkImageShiftScale *f2uc = vtkImageShiftScale::New();
52        f2uc->SetShift(shift); // 最小値を 0 にする
53        f2uc->SetScale(scale); // 値域を 0 - 255 にする
54        f2uc->SetOutputScalarTypeToUnsignedChar();
55        f2uc->SetInput(imgData);
56
57       /* vtkLookupTable を利用して色の伝達関数を作る */
58       vtkLookupTable *lut = vtkLookupTable::New();
59        lut->SetHueRange(0.7, 0.0);
60        lut->SetNumberOfTableValues(256);
```

```
 61        lut->SetRange(range);
 62        lut->Build();
 63
 64       /* vtkLookupTable から色を取り出す */
 65       for(i=0;i<256;i++){
 66        lut->GetTableValue(i, temp_color);
 67        color_table[i*3]   = temp_color[0];
 68        color_table[i*3+1] = temp_color[1];
 69        color_table[i*3+2] = temp_color[2];
 70       }
 71
 72       vtkColorTransferFunction *tf4color
 73             = vtkColorTransferFunction::New();
 74        tf4color->SetColorSpaceToRGB();
 75        tf4color->BuildFunctionFromTable(0, 255, 256, color_table);
 76
 77       /* 不透明度の伝達関数作成 */
 78       for(i=0;i<256;i++){
 79        if(i > 187) opacity_table[i] = i/255.0 * 0.025;
 80          else opacity_table[i] = 0.0;
 81       }
 82
 83       vtkPiecewiseFunction *tf4opacity
 84                    = vtkPiecewiseFunction::New();
 85        tf4opacity->BuildFunctionFromTable(
 86                     0, 255, 256, opacity_table, 1);
 87
 88       /* vtkVolumeProperty に伝達関数を代入 */
 89       vtkVolumeProperty *vp = vtkVolumeProperty::New();
 90        vp->SetColor(tf4color);
 91        vp->SetScalarOpacity(tf4opacity);
 92        vp->SetInterpolationTypeToLinear();
 93
 94       /* ボリュームレンダリング(2D Texture Technique)用Mapper*/
 95       vtkVolumeTextureMapper2D *vMapper
 96                  = vtkVolumeTextureMapper2D::New();
 97        vMapper->SetInput(f2uc->GetOutput());
 98        vMapper->SetMaximumNumberOfPlanes(256);
 99        vMapper->SetTargetTextureSize(256, 128);
100
101       vtkVolume *Volume = vtkVolume::New();
102        Volume->SetMapper(vMapper);
103        Volume->SetProperty(vp);
104
105       /* データの外枠 */
106       vtkOutlineFilter *outline = vtkOutlineFilter::New();
107        outline->SetInput(imgData);
```

```
108
109         vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
110          OLMapper->SetInput(outline->GetOutput());
111
112         vtkActor *OLActor = vtkActor::New();
113          OLActor->SetMapper(OLMapper);
114
115         /* 青い球体を地球の位置に表示する */
116         vtkSphereSource *earth = vtkSphereSource::New();
117          earth->SetRadius(3.3);
118
119         vtkPolyDataMapper *EMapper = vtkPolyDataMapper::New();
120          EMapper->SetInput(earth->GetOutput());
121
122         vtkActor *EActor = vtkActor::New();
123          EActor->SetMapper(EMapper);
124          EActor->SetPosition(earth_position);
125          EActor->GetProperty()->SetColor(0.0, 0.0, 1.0);
126
127         /* ビルボード処理されたテキストを表示する */
128         /* EARTH という文字のポリゴンデータ生成 */
129         vtkVectorText *EText = vtkVectorText::New();
130          EText->SetText("   EARTH");
131
132         vtkPolyDataMapper *TMapper = vtkPolyDataMapper::New();
133          TMapper->SetInput(EText->GetOutput());
134
135         /* ビルボード処理用Actor */
136         vtkFollower *TActor = vtkFollower::New();
137          TActor->SetMapper(TMapper);
138          TActor->SetScale(5.0, 5.0, 5.0);
139          TActor->AddPosition(earth_position);
140
141         /* カラーバー */
142         vtkScalarBarActor *scalarbar = vtkScalarBarActor::New();
143          scalarbar->SetTitle("Temperature");
144          scalarbar->SetLookupTable(lut);
145          scalarbar->SetOrientationToVertical();
146          scalarbar->SetWidth(0.075);
147          scalarbar->SetHeight(0.75);
148
149         vtkCamera *camera = vtkCamera::New();
150          camera->SetPosition(-200.0, -200.0, 237.5);
151          camera->SetFocalPoint(127.5, 63.5, 63.5);
152          camera->SetViewUp(0.0, 0.0, 1.0);
153          camera->OrthogonalizeViewUp();
154          camera->SetClippingRange(30.0, 2000.0);
```

```
155
156       vtkRenderer *vren= vtkRenderer::New();
157        vren->AddVolume(Volume);
158        vren->AddActor( EActor);
159        vren->AddActor( OLActor);
160        vren->AddActor( TActor);
161        vren->AddActor( scalarbar);
162        vren->SetActiveCamera (camera);
163        vren->SetBackground( 0.0, 0.0, 0.0 );
164
165       TActor->SetCamera(vren->GetActiveCamera()); /* これも忘れずに */
166
167       vtkRenderWindow *renWin = vtkRenderWindow::New();
168        renWin->AddRenderer( vren );
169        renWin->SetSize( 800, 600 );
170
171       vtkRenderWindowInteractor *iwin
172                = vtkRenderWindowInteractor::New();
173        iwin->SetRenderWindow(renWin);
174
175       vtkInteractorStyleTrackballCamera *trackball =
176        vtkInteractorStyleTrackballCamera::New();
177
178        iwin->SetInteractorStyle(trackball);
179        iwin->Initialize();
180        iwin->Start();
181
182        reader->Delete();
183        f2uc->Delete();
184        lut->Delete();
185        tf4color->Delete();
186        tf4opacity->Delete();
187        vp->Delete();
188        vMapper->Delete();
189        Volume->Delete();
190        outline->Delete();
191        OLMapper->Delete();
192        OLActor->Delete();
193        earth->Delete();
194        EMapper->Delete();
195        EActor->Delete();
196        EText->Delete();
197        TMapper->Delete();
198        TActor->Delete();
199        scalarbar->Delete();
200        iwin->Delete();
201        trackball->Delete();
```

```
202         vren->Delete();
203         camera->Delete();
204         renWin->Delete();
205
206     return 0;
207   }
```

5.4.2 サンプルプログラム3の簡単な説明

ボリューム・レンダリングに関しては，Mapper がレイ・キャスティングバージョンと違うだけである。(新規事項ではないが，色の伝達関数作成を工夫した (**57 - 75** 行))

95 - 96 : vtkVolumeRayCastMapper ではなく，vtkVolumeTextureMapper2D を使う
97 : Unsigned Char 型に直したデータをセット
98 : スライス数の最大値設定
99 : テクスチャの画像解像度設定。Default では，512×512。ターゲットのデータサイズは，256×128×128 であるので，256×128 に設定している。なお，このテクスチャのサイズは，2 のべき乗 (おそらく 64 以上の) でなければならない。

以上が，レイ・キャスティングバージョンからの主な変更点であるが，このプログラムには以下の新しい事項が含まれている。

- Float 型から Unsigned Char 型への変換
- 球体のポリゴン生成
- ビルボード処理されたテキスト

以下，順に説明する。
Float 型から Unsigned Char 型への変換 (47 - 55 行)
vtkImageShiftScale を使うと良い (クラス名からわかるが，等間隔データ用である)。
48, 52 : 最小値が 0.0 になるように，データのスカラー値を “Shift” (range[0] には，データの最小値が代入されている)。データの値域は，[0.0, Max - Min] となっている
49, 53 : 値が，[0.0, 255.0] になるように Scale を変更
54 : 出力データを Unsigned Char 型に指定

球体のポリゴン生成 (115 - 117 行)
vtkSphereSource で球体のポリゴン (正確には球面のポリゴン) が生成できるので，それを vtkPolyDataMapper, vtkActor とつなげれば，簡単に球体を映すことができる。さらに，VTK には球体だけでなく，立方体，コーン，矢印などのポリゴンを生成するクラスも用意されている (Figure 9.3 参照。左上から vtkConeSource, vtkArrowSource, vtkCubeSource, vtkSphereSource, vtkCylinderSource, vtkDiskSource を使ってポリゴンデータを生成した)。本プログラムでは，これを利用して，地球の位置 (正確でないかもしれません) に青い球体を表示するようにしている。次章のベクトル場の可視化では，矢印を使う予定である

ビルボード処理されたテキスト (127 - 139, 165 行)
常に画面に平行な面にテキストを表示させる方法がある。vtkVectorText, vtkPolyDataMapper,

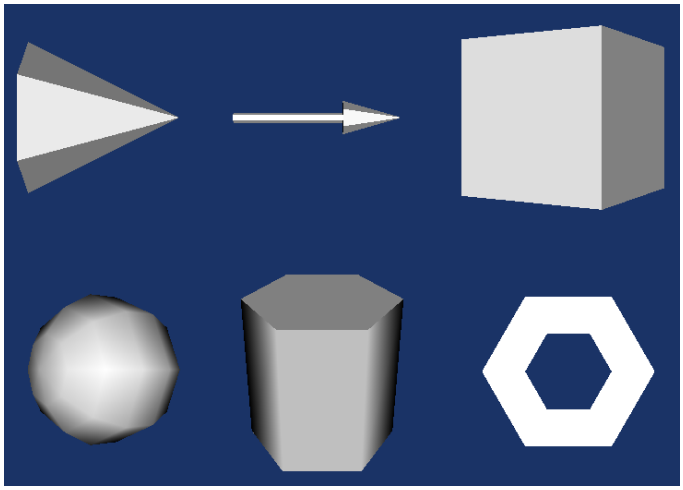

Figure 5.8: ポリゴンの生成

vtkFollower(vtkActor の代わり) を利用すればよい。vtkVectorText は，テキストのポリゴンデータを生成するだけなので，vtkFollower ではなく vtkActor でも表示できるが，それだと，カメラの視線方向が変わっても，テキストは回転しない。ちなみに，vtkFollower は，vtkActor の派生クラス。本プログラムでは，青い球体の横に “EARTH”と表示させている

このプログラムを実行すると，Figure 5.9 のような画像が表示される。

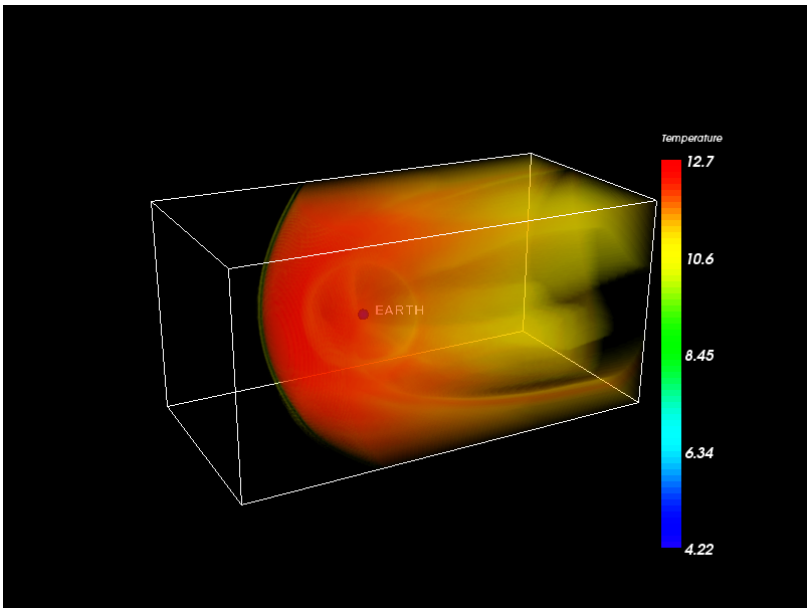

Figure 5.9: 地球磁気圏の温度分布

5.5 この章のまとめ

- レイ・キャスティング法によるボリュームレンダリング

- テクスチャマッピングを利用したカラーコンター
- テクスチャマッピングを利用したボリューム・レンダリング
- Float 型から Unsigned Char 型への変換
- 球体 (面) の表示，軸の表示，ビルボード処理されたテキスト表示

本章で紹介したクラスは，Table 5.1 の通り。

Table 5.1: 本章で紹介したクラスおよびそのメソッド

データセット	vtkImageData : GetDimensions(int [3])
データ変換	vtkImageShiftScale : SetShift(float) : SetScale(float) : SetOutputScalarTypeTo〜()
	vtkImageMapToColors
カラーテーブル関係	vtkPiecewiseFunction : AddPoint(float, float) : BuildFunctionFromTable(float min, float max, int size, float *, 1)
カラーテーブル関係	vtkColorTransferFunction : BuildFunctionFromTable(float min, float max, int size, float *)
球体その他の表示	vtkSphereSource, vtkConeSource, vtkArrowSource vtkCubeSource, vtkCylinderSource, vtkDiskSource
軸	vtkCubeAxesActor
テクスチャ用 Actor	vtkImageActor
テキスト表示	vtkVectorText, vtkFollower
ボリュームのプロパティ	vtkVolumeProperty : SetColor(vtkColorTransferFunction *) : SetOpacity(vtkPiecewiseFunction *) : SetInterpolationTypeToLinear() : ShadeOn()
(VR 用 Mapper)	vtkVolumeRayCastMapper vtkVolumeTextureMapper2D ————————- : SetSampleDistance(float) / RayCast : SetMaximumNumberOfPlanes(int) / Texture : SetTargetTextureSize(int, int) / Texture : SetCroppingRegionPlanes(float, float, float, float, float, float) : SetCroppingRegionFlagsToSubVolume() : CroppingOn()
(VR 用 Actor)	vtkVolume : SetProperty(vtkVolumeProperty *)

Chapter 6

3次元ベクトルデータの可視化 その1

6.1 本章の概要

ベクトルデータを矢印やスライスを使って可視化する。

6.2 HedgeHog(針鼠) によるベクトルデータの可視化

ここでは，VTK 用のベクトルデータの生成などを中心に話を進める。本章のサンプルデータは，私が自作したものである。Float 型・サイズ 300×300×100 で，x, y, z 成分が別々のファイル (vector_x.dat, vector_y.dat, vector_z.dat) になっている。また，ベクトルの絶対値のファイル (vector_abs.dat) も用意した。

このサンプルプログラムは，データ内からランダムに点を選び出して，そこでのベクトルを基に線分や矢印を描くことでベクトル場を可視化する。ベクトル場の可視化で良く用いられる手法である。

6.2.1 サンプルプログラム 1

```
 1  /* viz_vect.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkFloatArray.h>
 4  #include <vtkPointData.h>
 5  #include <vtkMaskPoints.h>
 6  #include <vtkHedgeHog.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkOutlineFilter.h>
 9  #include <vtkPolyData.h>
10  #include <vtkPolyDataMapper.h>
11  #include <vtkRenderWindow.h>
12  #include <vtkActor.h>
13  #include <vtkRenderer.h>
14  #include <vtkRenderWindowInteractor.h>
15  #include <vtkInteractorStyleTrackballCamera.h>
16
17  #define SIZE_X 300
```

```
18  #define SIZE_Y 300
19  #define SIZE_Z 100
20
21  float vector[SIZE_X*SIZE_Y*SIZE_Z][3];
22  float vector_abs[SIZE_X*SIZE_Y*SIZE_Z];
23
24  char datafile_x[]   = "./vector_x.dat";
25  char datafile_y[]   = "./vector_y.dat";
26  char datafile_z[]   = "./vector_z.dat";
27  char datafile_abs[] = "./vector_abs.dat";
28
29  float *vector_x;
30  float *vector_y;
31  float *vector_z;
32
33  void load_data();
34
35  int main( int argc, char *argv[] )
36  {
37      int size[3] = {SIZE_X, SIZE_Y, SIZE_Z};
38      float range[2];
39
40      load_data(); /* ベクトル・スカラーデータの読み込み */
41
42      vtkFloatArray *varray = vtkFloatArray::New();
43       varray->SetNumberOfComponents(3); /* 要素数は 3 */
44
45       /* 格子点の総数 */
46       varray->SetNumberOfTuples(SIZE_X* SIZE_Y* SIZE_Z);
47       varray->SetArray(vector[0], SIZE_X* SIZE_Y* SIZE_Z* 3, 1);
48
49      vtkFloatArray *sarray = vtkFloatArray::New();
50       sarray->SetArray(vector_abs, SIZE_X* SIZE_Y* SIZE_Z, 1);
51
52      vtkImageData *imgData = vtkImageData::New();
53       imgData->SetDimensions(size);
54       imgData->SetSpacing(1.0, 1.0, 1.0);
55       imgData->SetScalarTypeToFloat();
56       imgData->GetPointData()->SetScalars(sarray);
57       /* ベクトルデータの代入 */
58       imgData->GetPointData()->SetVectors(varray);
59       imgData->Update();
60       imgData->GetScalarRange(range);
61
62      vtkLookupTable *lut = vtkLookupTable::New();
63       lut->SetHueRange(0.7, 0.0);
64       lut->Build();
```

```
 65
 66      /* ランダムに格子点を取り出す */
 67      vtkMaskPoints *mask = vtkMaskPoints::New();
 68       mask->SetInput(imgData);
 69       mask->SetOnRatio(1000);
 70       mask->RandomModeOn();
 71
 72      /* 上記で取り出した格子点のところに,
 73         ベクトルデータを基に線分のポリゴンデータを生成 */
 74      vtkHedgeHog *hedgehog = vtkHedgeHog::New();
 75       hedgehog->SetInput(mask->GetOutput());
 76       hedgehog->SetScaleFactor(5.0/range[1]);
 77
 78      vtkPolyDataMapper * vecMapper = vtkPolyDataMapper::New();
 79       vecMapper->SetInput(hedgehog->GetOutput());
 80       vecMapper->SetScalarRange(range[0], range[1]);
 81       vecMapper->SetLookupTable(lut);
 82
 83      vtkActor *Actor = vtkActor::New();
 84       Actor->SetMapper(vecMapper);
 85
 86     /* データの外枠 */
 87      vtkOutlineFilter *outline = vtkOutlineFilter::New();
 88       outline->SetInput(imgData);
 89
 90      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
 91       OLMapper->SetInput(outline->GetOutput());
 92
 93      vtkActor *OLActor = vtkActor::New();
 94       OLActor->SetMapper(OLMapper);
 95
 96      vtkRenderer *vren= vtkRenderer::New();
 97       vren->AddActor(Actor);
 98       vren->AddActor(OLActor);
 99       vren->SetBackground( 0.0, 0.0, 0.0 );
100
101      vtkRenderWindow *renWin = vtkRenderWindow::New();
102       renWin->AddRenderer( vren );
103       renWin->SetSize( 500, 375 );
104
105      vtkRenderWindowInteractor *iwin
106             = vtkRenderWindowInteractor::New();
107       iwin->SetRenderWindow(renWin);
108
109      vtkInteractorStyleTrackballCamera *trackball =
110       vtkInteractorStyleTrackballCamera::New();
111
```

```
112       iwin->SetInteractorStyle(trackball);
113       iwin->Initialize();
114       iwin->Start();
115
116       varray->Delete();
117       sarray->Delete();
118       imgData->Delete();
119       lut->Delete();
120       mask->Delete();
121       hedgehog->Delete();
122       vecMapper->Delete();
123       Actor->Delete();
124       outline->Delete();
125       OLMapper->Delete();
126       OLActor->Delete();
127       iwin->Delete();
128       trackball->Delete();
129       vren->Delete();
130       renWin->Delete();
131
132     return 0;
133   }
134
135   void load_data(){
136
137       int i,j,k;
138       FILE *fpi;
139
140       vector_x = (float *)malloc(sizeof(float)*SIZE_X*SIZE_Y*SIZE_Z);
141       vector_y = (float *)malloc(sizeof(float)*SIZE_X*SIZE_Y*SIZE_Z);
142       vector_z = (float *)malloc(sizeof(float)*SIZE_X*SIZE_Y*SIZE_Z);
143
144       if ((fpi = fopen(datafile_x, "rb")) == NULL) {
145           puts("cannot open");
146           exit(1);
147       }
148
149         fread(vector_x, sizeof(float), SIZE_X*SIZE_Y*SIZE_Z, fpi);
150
151       fclose(fpi);
152
153       if ((fpi = fopen(datafile_y, "rb")) == NULL) {
154           puts("cannot open");
155           exit(1);
156       }
157
158         fread(vector_y, sizeof(float), SIZE_X*SIZE_Y*SIZE_Z, fpi);
```

```
159
160       fclose(fpi);
161
162       if ((fpi = fopen(datafile_z, "rb")) == NULL) {
163           puts("cannot open");
164           exit(1);
165       }
166
167         fread(vector_z, sizeof(float), SIZE_X*SIZE_Y*SIZE_Z, fpi);
168
169       fclose(fpi);
170
171       if ((fpi = fopen(datafile_abs, "rb")) == NULL) {
172           puts("cannot open");
173           exit(1);
174       }
175
176         fread(vector_abs, sizeof(float), SIZE_X*SIZE_Y*SIZE_Z, fpi);
177
178       fclose(fpi);
179
180      for(k=0;k<SIZE_Z;k++){
181       for(j=0;j<SIZE_Y;j++){
182        for(i=0;i<SIZE_X;i++){
183
184          vector[i + j* SIZE_X + k*SIZE_X*SIZE_Y][0]
185               = vector_x[i+j*SIZE_X + k*SIZE_X*SIZE_Y];
186
187          vector[i + j* SIZE_X + k*SIZE_X*SIZE_Y][1]
188               = vector_y[i+j*SIZE_X + k*SIZE_X*SIZE_Y];
189
190          vector[i + j* SIZE_X + k*SIZE_X*SIZE_Y][2]
191               = vector_z[i+j*SIZE_X + k*SIZE_X*SIZE_Y];
192
193        }
194       }
195      }
196
197      free(vector_x);
198      free(vector_y);
199      free(vector_z);
200   }
```

6.2.2 サンプルプログラム 1 の簡単な説明 1: VTK 用のベクトルデータ

vtkImageData や vtkRectilinearGrid にベクトル場をセットしなければ，可視化のためのフィルタを使うことはできない。VTK 形式のデータ作成の項で，VTK 形式のベクトルデータの作り方

はすでに述べたので，ここでは，マニュアルでベクトルデータをセットする方法をサンプルプログラムを通して説明する。

VTK のベクトルデータは，X, Y, Z 成分のデータがメモリ上に X Y Z の順番で並んでいなければならない。具体的にサンプルプログラム 1 の例で言うと，float vector[SIZE_X*SIZE_Y*SIZE_Z][3] の中に，X, Y, Z の各成分が入ったものを用意して (load_data 関数)，それを vtkFloatArray を介して，imgData(等間隔メッシュ) に代入してベクトルデータを作っている。

サンプルプログラム 1 の load_data 関数 (**135 - 200** 行) は，別々のファイルから，それぞれ x, y, z 成分のデータを読み込み，float vector[SIZE_X*SIZE_Y*SIZE_Z][3] にまとめている。その後，**42 - 47** 行で，varray (vtkFloatArray) にこのデータを渡している。スカラーのときは，
47 : varray->SetArray(vector[0], SIZE_X*SIZE_Y*SIZE_Z*3, 1);
とするだけであったが，今回はベクトルなので，
43 : varray->SetNumberOfComponent(3); /* 成分の数を指定 */
46 : varray->SetNumberOfTuples(SIZE_X*SIZE_Y*SIZE_Z); /* 格子点の総数を指定 */
としている (Figure 6.1)。こうして用意した VTK の配列を，**58** 行目で等間隔メッシュのデータの箱に

```
imgData->GetPointData()->SetVectors(varray);
```

として代入している (このサンプルプログラム 1 では，ベクトルデータと同時に，スカラーデータ (ベクトルの絶対値のデータ) も同じ箱 (imgData) に代入している)。

この imgData を使うことで，ようやく，VTK でのベクトル場の可視化が可能になる。

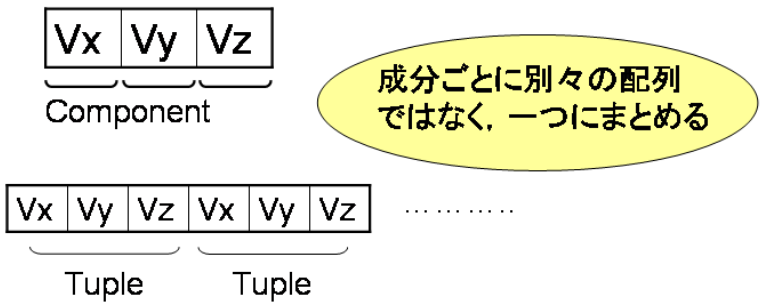

Figure 6.1: Component と Tuple

6.2.3 サンプルプログラム 1 の簡単な説明 2: 可視化部分

ベクトル場に，線分や矢印をちりばめるには，一部の点をデータ内から取り出さなければならない (すべての点で線分を作ると，何がなんだかわからなくなるので)。それには，

- vtkMaskPoints クラス [1]

を使えばよい。vtkMaskPoints は，ランダムあるいは周期的に，データの格子点を取り出してくれる。また，

- vtkHedgeHog クラス

[1] このクラスのほかに，vtkThresholdPoints というスカラー値の閾値を与えて，点を取り出すクラスもある。例えば，高温なところの点のみを取り出すなどに使える。vtkMaskPoints と組み合わせても良い。

は，入力された点に，その点のベクトルデータを考慮して，線分のポリゴンデータを生成する。このクラスに，vtkMaskPoints で取り出した点を入力すればよい。

このサンプルプログラムの可視化処理部分は，vtkMaskPoints を使い全格子点の 1/1000 の点をランダムに取り出し，その点で線分のポリゴンデータを vtkHedgeHog を使い生成して表示する。色づけは，スカラーデータを使っている (このプログラムでは，ベクトルの絶対値をスカラーデータとして持っているので，ベクトルの絶対値が大きいところでは赤っぽくなり，小さいところでは青っぽくなる)。

66 - 70 : imgData から，1/1000 の割合で格子点を取り出す。
69 : SetOnRatio で，1/1000 の割合で格子点を取り出すよう指定
70 : RandomModeOn() で，ランダムに点を選び出す。default では，周期的に選び出す
74 - 76 : 取り出した点でのベクトルの値を基に，線分のポリゴンデータを作成する

その他は，前章までの知識で，理解していただけると思う。パイプラインを Figure 6.2 に示す。

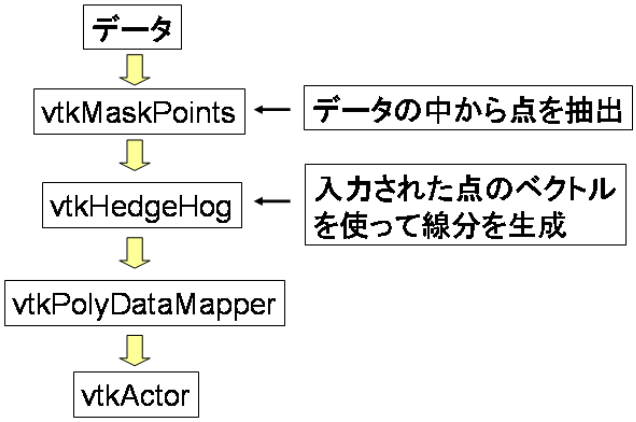

Figure 6.2: サンプルプログラム 1 のパイプライン構造

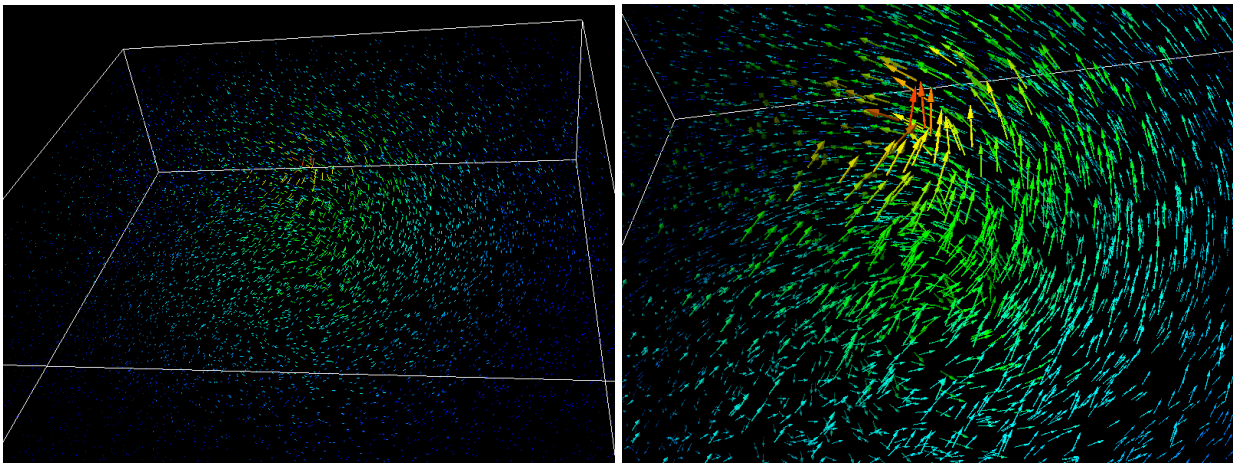

Figure 6.3: 線分・矢印によるベクトル場表示

表示は，Figure 6.3(左)。

6.3 矢印・コーンを使う

HedgeHog による可視化では，ベクトルの向きまではわからない。そこで，線分の代わりに，矢印やコーンを使って，方向までわかるようにしてみる。それには，

- vtkGlyph3D クラス

という，SetInput で入力された点に，SetSource で入力されたポリゴンデータを方向も考慮に入れて配置していくクラスを vtkHedgeHog と差し替えればよい。

```
1       /* 矢印のポリゴンデータ生成 */
2       vtkArrowSource *arrow = vtkArrowSource::New();
3         arrow->SetTipResolution(4.0);
4         arrow->SetShaftResolution(4.0);
5
6       /* mask で取り出した点に矢印のポリゴンデータを配置する */
7       vtkGlyph3D *glyphs = vtkGlyph3D::New();
8        glyphs ->SetInput (mask ->GetOutput());
9        glyphs ->SetSource (arrow ->GetOutput());
10       glyphs ->SetScaleFactor (2.0);
11       glyphs ->SetScaleModeToScaleByVector();
12       glyphs ->SetColorModeToColorByScalar();
13
14      vtkPolyDataMapper * vecMapper = vtkPolyDataMapper::New();
15       vecMapper->SetInput(glyphs->GetOutput());
16       vecMapper->SetScalarRange(range[0], range[1]);
17       vecMapper->SetLookupTable(lut);
```

サンプルプログラム 1 の **72 - 81** 行を上記のリストと入れ替えると，線分が矢印に変わる (入れ替えたプログラムは，viz_vect2.cxx として保存されている)。

1 - 4 : 矢印のポリゴンデータ生成
8 : 取り出した点を入力
9 : 矢印のポリゴンデータを入力
11 : 矢印を表示するとき，大きさはベクトルの大きさと連動するようにする。SetScaleModeToScalingOff() で，ベクトルの大きさにかかわらず，同じ大きさの矢印を使うことができる

パイプラインを Figure 6.4 に示す。表示は，Figure 6.3(右)。矢印ではなく，コーンを使いたいときは，vtkConeSource でコーンのポリゴンデータを生成して，**9** 行目の SetSource でそれを入力すればよい。ここまでの知識を組み合わせると，Figure 6.5 のように，等値面上の点でのベクトルを矢印で表現することもできる。気圧の等値面を描き，それを温度で色付けして，その面上での風速を矢印で表示，なども可能である。

この 3 次元の矢印やコーンを利用すると，計算機によっては，描画が遅くなる場合がある。このような場合は，2 次元の矢印を使うと良い。2 次元の矢印のポリゴンデータは，vtkGlyphSource2D で生成することができる。vtkGlyphSource2D を利用すると，Figure 6.6 のようなポリゴンデータが得られる。

これ以降本章のサンプルプログラムでは，リーダーを使いデータを読み込む方法をとる。

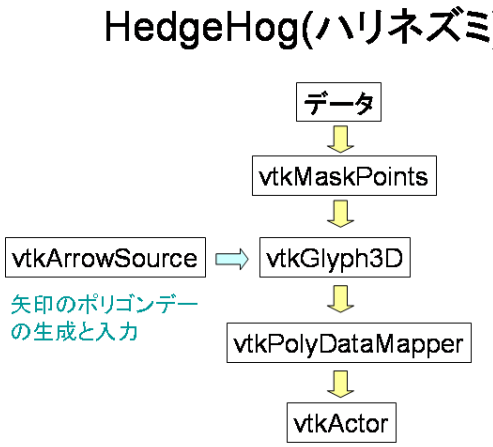

Figure 6.4: 矢印を使う場合のパイプライン構造

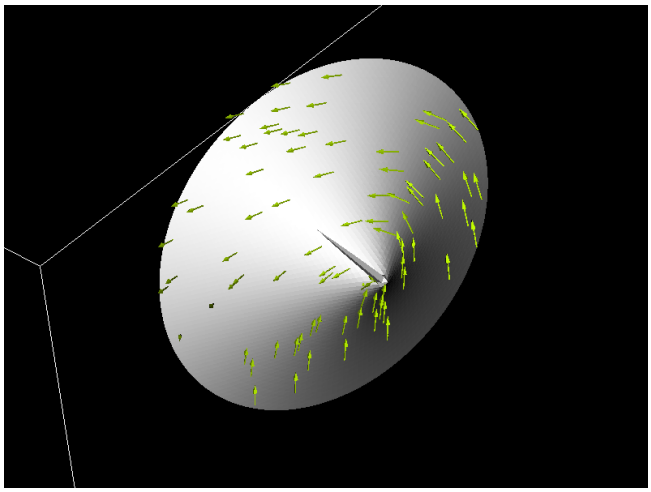

Figure 6.5: ベクトルを矢印で表現：等値面上の点を選んである

補足

言うまでもないが，vtkHedgeHog や vtkGlyph3D に入力する点には，ベクトルデータが付随していなければならない。

6.4 ベクトル版凹凸つきスライス

スカラーデータの可視化で (第 2 章)，vtkWarpScalar というクラスを使い，頂点をスカラーの値に応じて移動させることで，スライスに凹凸をつけて富士山を可視化した。この vtkWarpScalar のベクトル版で， vtkWarpVector というクラスがある。これは，ベクトルの方向と大きさで頂点を移動させるクラスである。

6.4.1 サンプルプログラム 2

```
1  /*Vec_BumpySlice.cxx */
```

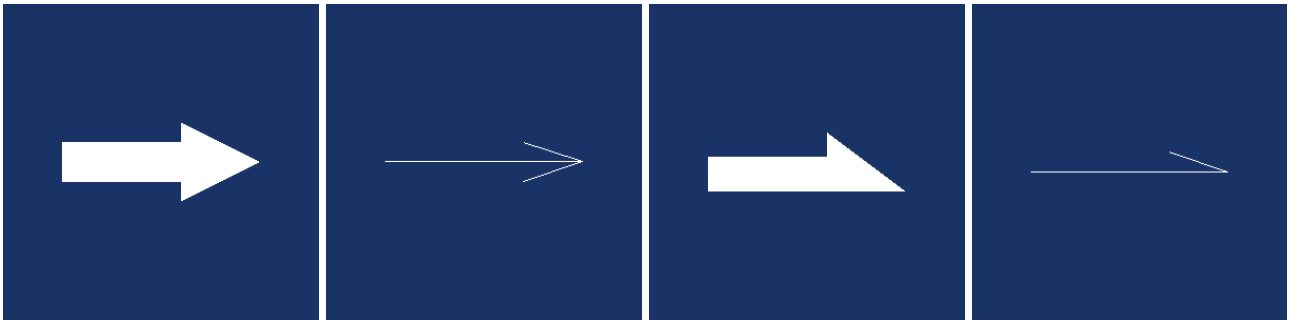

Figure 6.6: 2 次元グリフ

```
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkImageDataGeometryFilter.h>
 7  #include <vtkWarpVector.h>
 8  #include <vtkPolyDataNormals.h>
 9  #include <vtkTriangleFilter.h>
10  #include <vtkLookupTable.h>
11  #include <vtkOutlineFilter.h>
12  #include <vtkPolyData.h>
13  #include <vtkPolyDataMapper.h>
14  #include <vtkRenderWindow.h>
15  #include <vtkActor.h>
16  #include <vtkRenderer.h>
17  #include <vtkProperty.h>
18  #include <vtkRenderWindowInteractor.h>
19  #include <vtkInteractorStyleTrackballCamera.h>
20
21  int main( int argc, char *argv[] )
22  {
23      float range[2];
24      char datafile[]   = "./vector.vtk";
25
26      vtkStructuredPointsReader *reader
27         = vtkStructuredPointsReader::New();
28       reader->SetFileName(datafile);
29
30      vtkImageData *imgData;
31       imgData = reader->GetOutput();
32       imgData->Update();
33       imgData->GetScalarRange(range);
34
35      vtkLookupTable *lut = vtkLookupTable::New();
36       lut->SetHueRange(0.7, 0.0);
```

```
37       lut->Build();
38
39      vtkImageDataGeometryFilter *igf
40              = vtkImageDataGeometryFilter::New();
41       igf->SetInput(imgData);
42       igf->SetExtent(75, 225, 75, 225, 80, 80);
43
44      vtkTriangleFilter *tri = vtkTriangleFilter::New();
45       tri->SetInput(igf->GetOutput());
46
47      /* ベクトルデータを基に頂点を移動 */
48      vtkWarpVector *v_carpet = vtkWarpVector::New();
49       v_carpet->SetInput(tri->GetOutput());
50       v_carpet->SetScaleFactor(10.0);
51
52      vtkPolyDataNormals *normals = vtkPolyDataNormals::New();
53       normals->SetInput(v_carpet->GetPolyDataOutput());
54
55      vtkPolyDataMapper * vecMapper = vtkPolyDataMapper::New();
56       vecMapper->SetInput(normals->GetOutput());
57       vecMapper->SetScalarRange(range[0], range[1]);
58       vecMapper->SetLookupTable(lut);
59
60      vtkActor *Actor = vtkActor::New();
61       Actor->SetMapper(vecMapper);
62
63      vtkOutlineFilter *outline = vtkOutlineFilter::New();
64       outline->SetInput(imgData);
65
66      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
67       OLMapper->SetInput(outline->GetOutput());
68
69      vtkActor *OLActor = vtkActor::New();
70       OLActor->SetMapper(OLMapper);
71
72      vtkPolyDataMapper *gridMapper = vtkPolyDataMapper::New();
73       gridMapper->SetInput(igf->GetOutput());
74       gridMapper->ScalarVisibilityOff();
75
76      vtkActor *GActor = vtkActor::New();
77       GActor->SetMapper(gridMapper);
78       GActor->GetProperty()->SetColor(1.0,1.0,1.0);
79       GActor->GetProperty()->SetAmbient(0.5);
80       GActor->GetProperty()->SetDiffuse(0.5);
81       GActor->GetProperty()->SetRepresentationToWireframe();
82
83      vtkRenderer *vren= vtkRenderer::New();
```

```
 84       vren->AddActor(Actor);
 85       vren->AddActor(GActor);
 86       vren->AddActor(OLActor);
 87       vren->SetBackground( 0.0, 0.0, 0.0 );
 88
 89      vtkRenderWindow *renWin = vtkRenderWindow::New();
 90       renWin->AddRenderer( vren );
 91       renWin->SetSize( 500, 375 );
 92
 93      vtkRenderWindowInteractor *iwin
 94           = vtkRenderWindowInteractor::New();
 95       iwin->SetRenderWindow(renWin);
 96
 97      vtkInteractorStyleTrackballCamera *trackball =
 98       vtkInteractorStyleTrackballCamera::New();
 99
100       iwin->SetInteractorStyle(trackball);
101       iwin->Initialize();
102       iwin->Start();
103
104       reader->Delete();
105       lut->Delete();
106       igf->Delete();
107       tri->Delete();
108       v_carpet->Delete();
109       normals->Delete();
110       vecMapper->Delete();
111       Actor->Delete();
112       outline->Delete();
113       OLMapper->Delete();
114       OLActor->Delete();
115       gridMapper->Delete();
116       GActor->Delete();
117       vren->Delete();
118       renWin->Delete();
119       iwin->Delete();
120       trackball->Delete();
121
122     return 0;
123   }
```

6.4.2 サンプルプログラム 2 の簡単な説明

vtkWarpVector (**47 - 50**) は，vtkWarpScalar から使い方を類推できると思う。実行結果は，Figure 6.7。下のほうに見える白い網は，移動する前の頂点の位置を表している。2 つを見比べると，渦巻いているのがわかる。

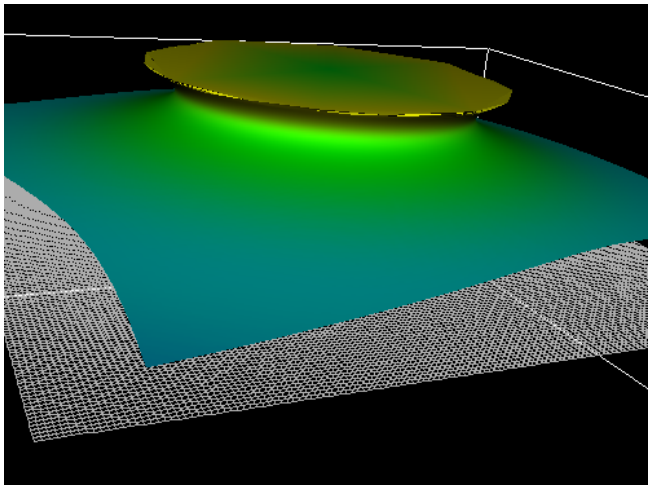

Figure 6.7: 凹凸つきスライスでベクトル場の一部を表示

6.5 この章のまとめ

- 線分・矢印によるベクトル場の可視化
- 凹凸つきスライスによるベクトル場の可視化

本章で紹介したクラスは，Table 6.1 の通り。

Table 6.1: 本章で紹介したクラスおよびそのメソッド

配列	vtkFloatArray : SetNumberOfComponents(int) : SetNumberOfTuples(const vtkIdType)
点の抽出	vtkMaskPoints : SetOnRatio(int) : RandomModeOn()
点の抽出	vtkThresholdPoints
ベクトル場の線分作成	vtkHedgeHog
2 次元の矢印	vtkGlyphSource2D
ポリゴンデータ配置	vtkGlyph3D
頂点移動	vtkWarpVector

Chapter 7

3次元ベクトルデータの可視化 その2

7.1 本章の概要

ベクトルデータを流線や流線と他のフィルタを組み合わせて可視化する。

7.2 流線 1 : 流線

当然であるが，VTK には，流線追跡機能がある。単純に曲線 (実際は細かい線分の集まり) を描かせることもできるし，他のフィルタと組み合わせて，面白い可視化をすることもできる。

流線計算には，vtkStreamLine (破線版は，vtkDashedStreamLine) を使う。このクラスに，データと出発点，Integrator を代入して刻み幅などのパラメータを設定すれば，流線のポリゴンを出力してくれる。

7.2.1 サンプルプログラム 1

下記が，複数の流線を描くプログラムである。

```
 1  /* StreamLines.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkPoints.h>
 7  #include <vtkPointSet.h>
 8  #include <vtkRungeKutta4.h>
 9  #include <vtkStreamLine.h>
10  #include <vtkDashedStreamLine.h>
11  #include <vtkSphereSource.h>
12  #include <vtkLookupTable.h>
13  #include <vtkOutlineFilter.h>
14  #include <vtkPolyData.h>
15  #include <vtkPolyDataMapper.h>
16  #include <vtkRenderWindow.h>
17  #include <vtkActor.h>
18  #include <vtkRenderer.h>
```

```
19  #include <vtkProperty.h>
20  #include <vtkRenderWindowInteractor.h>
21  #include <vtkInteractorStyleTrackballCamera.h>
22
23  #define NSP 10 /* 流線の出発点の数 */
24
25  int main( int argc, char *argv[] )
26  {
27      int i;
28      float range[2];
29
30      /* 流線の出発点 */
31      float startpoints[NSP][3]={
32          {100.0, 100.0, 15.0},
33          {110.0, 110.0, 15.0},
34          {120.0, 120.0, 15.0},
35          {130.0, 130.0, 15.0},
36          {140.0, 140.0, 15.0},
37          {150.0, 150.0, 15.0},
38          {160.0, 160.0, 15.0},
39          {170.0, 170.0, 15.0},
40          {180.0, 180.0, 15.0},
41          {190.0, 190.0, 15.0}    };
42
43      char datafile[]   = "./vector.vtk";
44
45      vtkStructuredPointsReader *reader
46         = vtkStructuredPointsReader::New();
47       reader->SetFileName(datafile);
48
49      vtkImageData *imgData;
50       imgData = reader->GetOutput();
51       imgData->Update();
52       imgData->GetScalarRange(range);
53
54      vtkLookupTable *lut = vtkLookupTable::New();
55       lut->SetHueRange(0.7, 0.0);
56       lut->Build();
57
58      /* 出発点のセッティング */
59      vtkPoints *points = vtkPoints::New();
60       points->Allocate(NSP);
61
62      for (i = 0; i < NSP; i++) points->InsertPoint(i, startpoints[i]);
63
64      vtkPolyData *pset = vtkPolyData::New();
65       pset->SetPoints(points);
```

```
 66
 67      /* 流線の計算 */
 68      vtkStreamLine *slines = vtkStreamLine::New();
 69      vtkRungeKutta4 *integ = vtkRungeKutta4::New();
 70
 71       slines->SetInput(imgData);
 72       slines->SetSource(pset);
 73       slines->SetIntegrator(integ);
 74       slines->SetMaximumPropagationTime(300);
 75       slines->SetIntegrationStepLength(0.1);
 76       slines->SetStepLength(0.5);
 77
 78      vtkPolyDataMapper * slineMapper = vtkPolyDataMapper::New();
 79       slineMapper->SetInput(slines->GetOutput());
 80       slineMapper->SetScalarRange(range[0], range[1]);
 81       slineMapper->SetLookupTable(lut);
 82
 83      vtkActor *Actor = vtkActor::New();
 84       Actor->SetMapper(slineMapper);
 85
 86      /* 出発点に赤い球体を配置 */
 87      vtkSphereSource *sphere = vtkSphereSource::New();
 88       sphere->SetRadius(1.0);
 89
 90      vtkPolyDataMapper *SMapper = vtkPolyDataMapper::New();
 91       SMapper->SetInput(sphere->GetOutput());
 92
 93      vtkActor *SActor[NSP];
 94      vtkRenderer *vren= vtkRenderer::New();
 95
 96      for(i=0;i<NSP;i++){
 97       SActor = vtkActor::New();
 98       SActor->SetMapper(SMapper);
 99       SActor->GetProperty()->SetColor(1.0, 0.0, 0.0);
100       SActor->SetPosition(startpoints[i]);
101       vren->AddActor(SActor);
102       SActor->Delete();
103      }
104
105      /* データの外枠 */
106      vtkOutlineFilter *outline = vtkOutlineFilter::New();
107       outline->SetInput(imgData);
108
109      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
110       OLMapper->SetInput(outline->GetOutput());
111
112      vtkActor *OLActor = vtkActor::New();
```

```
113       OLActor->SetMapper(OLMapper);
114
115       vren->AddActor(Actor);
116       vren->AddActor(OLActor);
117       vren->SetBackground( 0.0, 0.0, 0.0 );
118
119      vtkRenderWindow *renWin = vtkRenderWindow::New();
120       renWin->AddRenderer( vren );
121       renWin->SetSize( 500, 375 );
122
123      vtkRenderWindowInteractor *iwin
124           = vtkRenderWindowInteractor::New();
125       iwin->SetRenderWindow(renWin);
126
127      vtkInteractorStyleTrackballCamera *trackball =
128       vtkInteractorStyleTrackballCamera::New();
129
130       iwin->SetInteractorStyle(trackball);
131       iwin->Initialize();
132       iwin->Start();
133
134       reader->Delete();
135       lut->Delete();
136       points->Delete();
137       pset->Delete();
138       slines->Delete();
139       integ->Delete();
140       slineMapper->Delete();
141       Actor->Delete();
142       sphere->Delete();
143       SMapper->Delete();
144       outline->Delete();
145       OLMapper->Delete();
146       OLActor->Delete();
147       iwin->Delete();
148       trackball->Delete();
149       vren->Delete();
150       renWin->Delete();
151
152    return 0;
153  }
```

7.2.2 サンプルプログラム 1 の簡単な説明

新規な部分は，**58 - 76** 行である。**58 - 65** では，startpoints[NSP][3] に代入しておいた流線の出発点を vtkPolyData の点データとして pset に保持させている (もっとスマートな方法があるかも

しれません)。**67 - 76** 行で，4 次のルンゲ・クッタ法を用いて，積分計算している。**vtkStream-Line は，基本的に，速度場 を可視化するためのクラスです。**なお，vtkStreameLine の代わりに vtkDashedStreameLine を 使うと，破線が得られる。

23 : NSP = 出発点の数
31 - 41 : float startpoints[NSP][3] に出発点の座標値をセット
58 - 62 : startpoints[NSP][3] を points(vtkPoints) に代入
64 - 65 : points を pset(vtkPolyData) に代入 (もう少しスマートな方法があるかもしれないです)
71 : ベクトルのデータをセット
72 : 出発点 (pset) をセット
73 : 4 次のルンゲ・クッタを使用する (2 次も用意されている)
74 : 積分を続ける長さを時間で指定。これが大きいほど，流線は長くなるが，計算時間もかかる (引数が同じ値でも，入力されているベクトル場により，流線の長さは変わる)
75 : 積分の細かさを指定する。小さいほど正確になる。セルの大きさにたいする割合 (0.1 ならセルの 1/10 刻みで流線計算する)
76 : ポリゴン (線分の集まり) の頂点を出力する時間間隔。小さいほど多くの点を出力して綺麗なカーブを描くが，描画が遅くなる。また，速度が小さいところほど，多くの点を出力する
86 - 100 : vtkSphereSource を使い，出発点に赤い球体を表示させる

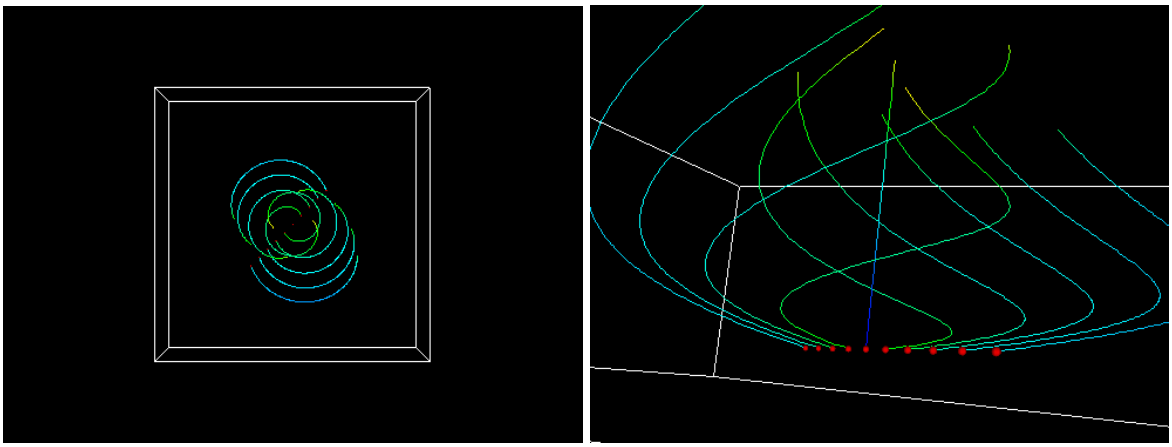

Figure 7.1: 流線による可視化

結果は，Figure 7.1。パイプラインを Figure 7.2 に示す。

「積分の細かさ」「点を出力する時間間隔」では，意味がよく分からないと思うので，IntegrationStepLength と StepLength を大きくした場合，流線にどのような変化がおきるか，具体的な結果を Figure 7.3 に示す。色つきが，サンプルプログラム 1 と同じパラメータで，白い流線がそれぞれの StepLength を大きくしたものである。IntegrationSteLength を大きくした Figure 7.3(左) をみると，元の線とずれてしまっている (ポリゴンの頂点の数は，減っていない)。理由は，積分が荒くなったためである。StepLength を大きくした (右) をみると，ポリゴンの点自体は，元の線上に存在するのがわかる。

速度場以外を可視化する場合は特に,「時間」や「刻み幅」の設定を気をつけなければならない。

積分する方向は，SetIntegrationDirectionTo〜(Forward(), Backward, BothDirections()) で，指定できる。

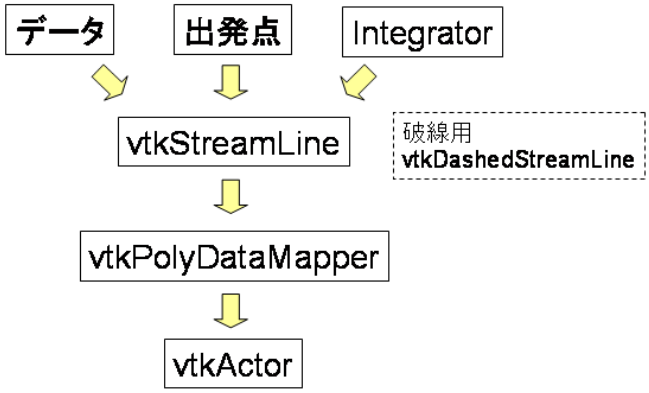

Figure 7.2: 流線のパイプライン構造

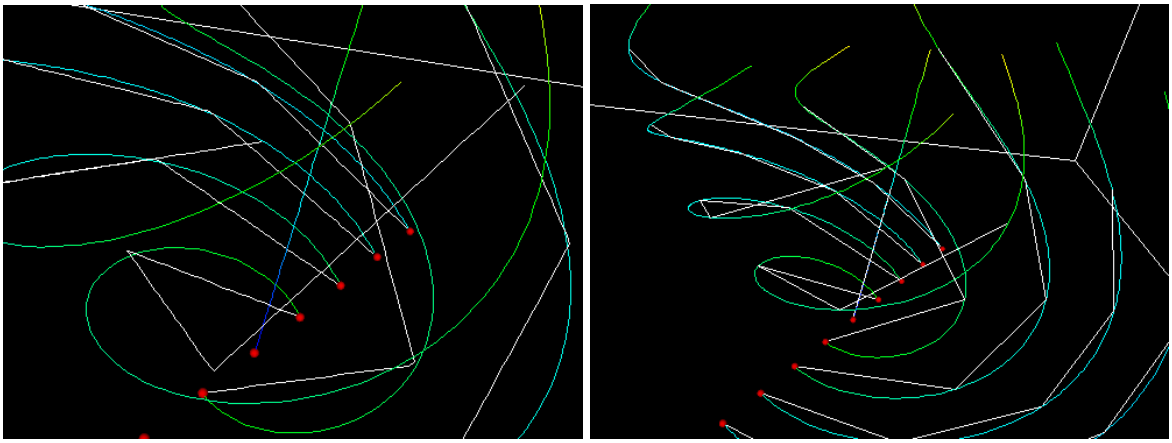

Figure 7.3: StepLength : (左)IntegratonStepLength を大きくした場合，(右)StepLength を大きくした場合

補足

前章の vtkHedgeHog や vtkGlypho3D の場合とは違い，流線計算のためのベクトルデータは別に入力するので，vtkStreamLine に入力する点にはベクトルデータが付随している必要はない。

7.3 流線と矢印の組み合わせ

前章のサンプルプログラム 2 とこのサンプルプログラム 1 を組み合わせると，Figure 7.4(左) にあるように，流線上に矢印を配置することもできるし，(右) のように流線上の点に別のベクトル場の矢印を配置することもできる。

前者は，流線のポリゴンデータを vtkMaskPoints に代入した後に，その出力を vtkGlyph3D に入力すればよい。

後者は，少々工夫が必要である。 vtkGlyph3D に入力する点には，ベクトルデータが付随していなければならないが，流線計算の出力をそのまま代入したのでは，流線と同じベクトルデータが付随しているので，別のベクトル場の矢印を表示することはできない。この問題を解決するのが，vtkProbeFilter である。このフィルタは，点とデータセットを入力すると，入力点にデータセットから (格子点上になければ補間して) スカラーやベクトルのデータを付随させた

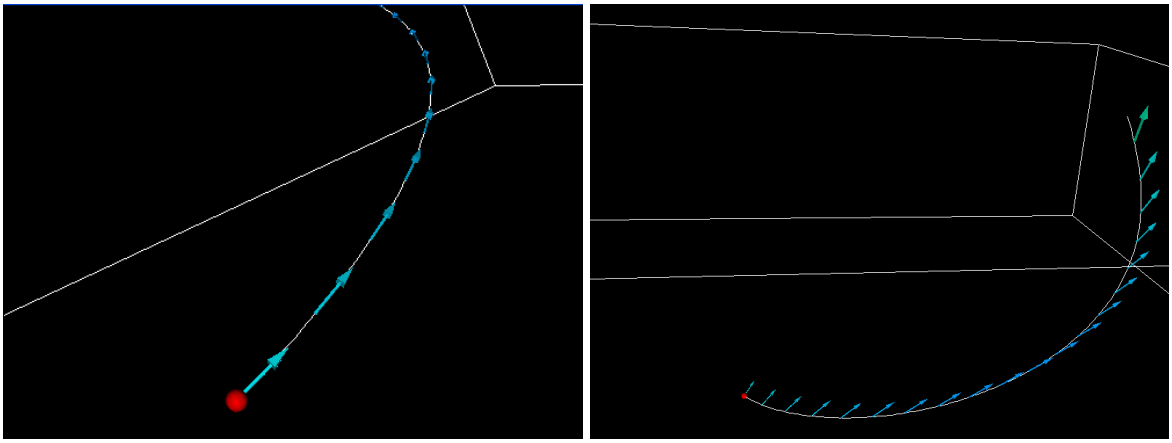

Figure 7.4: 流線上に矢印を配置

上で出力してくれる。具体的に，プログラムの一部を示す。

```
 1       vtkStreamLine *sline = vtkStreamLine::New();
 2       vtkRungeKutta4 *integ = vtkRungeKutta4::New();
 3        sline->SetInput(imgData);
 4        sline->SetSource(pset);
 5        sline->SetIntegrator(integ);
 6        sline->SetMaximumPropagationTime(300);
 7        sline->SetIntegrationStepLength(0.1);
 8        sline->SetStepLength(0.5);
 9
10       vtkMaskPoints *mask = vtkMaskPoints::New();
11        mask->SetInput(sline->GetOutput());
12        mask->SetOnRatio(50);
13
14       vtkProbeFilter *probe = vtkProbeFilter::New();
15        probe->SetInput(mask->GetOutput());
16        probe->SetSource(imgData2);
17
18       vtkGlyph3D *glyphs = vtkGlyph3D::New();
19        glyphs ->SetInput (probe ->GetOutput());
20        glyphs ->SetSource (arrow ->GetOutput());
```

1-8 : imgData のベクトルデータで流線を計算
10 - 12 : 流線のポリゴンデータの頂点を一部を抜き取る
14 - 16 : **15** 行で入力された点に imgData2 のベクトルやスカラーデータを付随させて，出力する (**19** 行)

7.4　流線 2 : チューブ・リボン

VTK には，線 (分) をチューブで覆う vtkTubeFilter というクラスやリボン上に変換する vtkRibbonFilter というクラスがある。これを使うと，単なる線での表示よりも，奥行き感などがでる。

いずれのクラスも，線 (分) のポリゴンデータを入力すると，チューブやリボンのポリゴンデータを出力する。これらのクラスと流線を組み合わせると，単なる曲線での表示よりも構造を把握しやすくなる。

7.4.1 サンプルプログラム 2

サンプルプログラム 1 と違い，vtkPointSource を使い出発点をランダムに決めている。

```
 1  /* StreamLines2.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkPointSet.h>
 7  #include <vtkRungeKutta4.h>
 8  #include <vtkStreamLine.h>
 9  #include <vtkSphereSource.h>
10  #include <vtkPointSource.h>
11  #include <vtkTubeFilter.h>
12  #include <vtkLookupTable.h>
13  #include <vtkOutlineFilter.h>
14  #include <vtkPolyData.h>
15  #include <vtkPolyDataMapper.h>
16  #include <vtkRenderWindow.h>
17  #include <vtkActor.h>
18  #include <vtkRenderer.h>
19  #include <vtkProperty.h>
20  #include <vtkRenderWindowInteractor.h>
21  #include <vtkInteractorStyleTrackballCamera.h>
22
23  #define NSP 10      /* 出発点の数 */
24
25  /* 球体の中からランダムに NSP 点選び出発点とする */
26  #define Bradius 5.0 /* 球体の半径 */
27  float Bposition[3] = {160, 149.0, 10.0}; /* 球体の中心 */
28
29  int main( int argc, char *argv[] )
30  {
31      float range[2];
32      char datafile[]   = "./vector.vtk";
33
34      vtkStructuredPointsReader *reader
35         = vtkStructuredPointsReader::New();
36       reader->SetFileName(datafile);
37
38      vtkImageData *imgData;
39       imgData = reader->GetOutput();
```

```
40        imgData->Update();
41        imgData->GetScalarRange(range);
42
43       vtkLookupTable *lut = vtkLookupTable::New();
44        lut->SetHueRange(0.7, 0.0);
45        lut->Build();
46
47       /* 球体の中からNSP点ランダムに点を選ぶ */
48       vtkPointSource *pset = vtkPointSource::New();
49         pset->SetCenter(Bposition);
50         pset->SetRadius(Bradius);
51         pset->SetNumberOfPoints(NSP);
52
53       /* 流線計算 */
54       vtkStreamLine *slines = vtkStreamLine::New();
55       vtkRungeKutta4 *integ = vtkRungeKutta4::New();
56
57        slines->SetInput(imgData);
58        slines->SetSource(pset->GetOutput());
59        slines->SetIntegrator(integ);
60        slines->SetMaximumPropagationTime(300);
61        slines->SetIntegrationStepLength(0.1);
62        slines->SetIntegrationDirectionToForward();
63        slines->SetStepLength(0.5);
64
65       /* 流線をチューブで囲む */
66       vtkTubeFilter *tubes = vtkTubeFilter::New();
67        tubes->SetInput(slines->GetOutput());
68        tubes->SetRadius(0.5);
69        tubes->SetRadiusFactor(5.0);
70        tubes->SetVaryRadiusToVaryRadiusByScalar();
71        tubes->SetNumberOfSides(8);
72
73       vtkPolyDataMapper * tubeMapper = vtkPolyDataMapper::New();
74        tubeMapper->SetInput(tubes->GetOutput());
75        tubeMapper->SetScalarRange(range[0], range[1]);
76        tubeMapper->SetLookupTable(lut);
77
78       vtkActor *Actor = vtkActor::New();
79        Actor->SetMapper(tubeMapper);
80
81       /* 球体をワイヤフレームで表示 */
82       vtkSphereSource *sphere = vtkSphereSource::New();
83        sphere->SetRadius(Bradius);
84        sphere->SetCenter(Bposition);
85
86       vtkPolyDataMapper *SMapper = vtkPolyDataMapper::New();
```

```
 87       SMapper->SetInput(sphere->GetOutput());
 88
 89      vtkActor *SActor = vtkActor::New();
 90       SActor->SetMapper(SMapper);
 91       SActor->GetProperty()->SetColor(1.0, 1.0, 1.0);
 92       SActor->GetProperty()->SetRepresentationToWireframe();
 93
 94      /* データの外枠 */
 95      vtkOutlineFilter *outline = vtkOutlineFilter::New();
 96       outline->SetInput(imgData);
 97
 98      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
 99       OLMapper->SetInput(outline->GetOutput());
100
101      vtkActor *OLActor = vtkActor::New();
102       OLActor->SetMapper(OLMapper);
103
104      vtkRenderer *vren= vtkRenderer::New();
105       vren->AddActor(Actor);
106       vren->AddActor(SActor);
107       vren->AddActor(OLActor);
108       vren->SetBackground( 0.0, 0.0, 0.0 );
109
110      vtkRenderWindow *renWin = vtkRenderWindow::New();
111       renWin->AddRenderer( vren );
112       renWin->SetSize( 500, 375 );
113
114      vtkRenderWindowInteractor *iwin
115          = vtkRenderWindowInteractor::New();
116       iwin->SetRenderWindow(renWin);
117
118      vtkInteractorStyleTrackballCamera *trackball =
119       vtkInteractorStyleTrackballCamera::New();
120
121       iwin->SetInteractorStyle(trackball);
122       iwin->Initialize();
123       iwin->Start();
124
125       reader->Delete();
126       lut->Delete();
127       pset->Delete();
128       slines->Delete();
129       integ->Delete();
130       tubes->Delete();
131       tubeMapper->Delete();
132       Actor->Delete();
133       outline->Delete();
```

```
134         OLMapper->Delete();
135         OLActor->Delete();
136         iwin->Delete();
137         trackball->Delete();
138         vren->Delete();
139         renWin->Delete();
140
141     return 0;
142   }
```

7.4.2 サンプルプログラム 2 の簡単な説明

新規な部分は，**47 - 51** 行と **65 - 71** 行である。**47 - 51** では，vtkPointSource を使い出発点をランダムに選び，**65 - 71** 行では，vtkStreamLine の出力結果 (線分) を vtkTubeFilter を使いチューブで囲んでいる。

47 - 51 : 「BPosition を中心とした半径 BRadius の球の中から，ランダムに NSP 個の点を選び出せ」，という意味。メソッドの名前がわかりやすいので，これだけで理解いただけると思う
67 : 流線のポリゴンを vtkTubeFikter に代入
68 : チューブの半径の最小値
69 : チューブの半径の最大値を最小値の何倍にするか
70 : スカラーの大きさに応じて，半径を変化させる。....ByVector() も存在する。また，SetVaryRadiusToVaryRadiusOff() で半径一定モードになる
71 : 引数が N だと，チューブが N 角形になる。この数字を大きくすると，綺麗になるが，ポリゴン数が増えるので，描画は遅くなる
81 - 92 : 点を取り出した球体をワイヤフレームで表示

表示は，Figure 7.5(左)。vtkTubeFilter の代わりに vtkRibbonFilter を使うと，Figure 7.5(右) のような可視化結果となる。この vtkTubeFilter は，いうまでもないが，入力のポリゴンデータは，

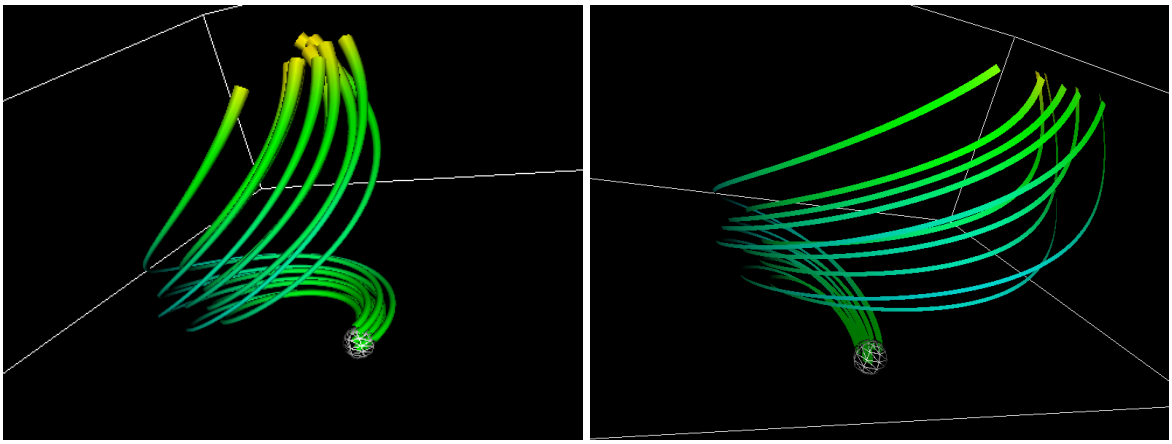

Figure 7.5: (左) チューブ，(右) リボン

流線の結果でなくてもよい。等高線をチューブで囲むこともできる。

7.5 流線 3 : 流面 (Stream Surfaces)

流線を沢山引くのも良いが，流線と流線の間に平面を入れると，少ない流線でもベクトル場の構造がわかりやすくなる。前セクションの“リボン”と違うのは，前者は単独の流線を面状にするだけであるが，こちらは，流線間に面を張るところである。

流線間に面を張るので，ある意味流線間にある流線の構造も見えている。ベクトル場全体の把握には，こちらの方が適しているかもしれない。

7.5.1 サンプルプログラム 3

指定した線分上の点をいくつか拾い上げ，そこから流線を引き，vtkRuledSurfaceFilter で流線と流線の間に面を張る。

```
 1  /* StreamLines3.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkPoints.h>
 7  #include <vtkPointSet.h>
 8  #include <vtkRungeKutta4.h>
 9  #include <vtkStreamLine.h>
10  #include <vtkLineSource.h>
11  #include <vtkRuledSurfaceFilter.h>
12  #include <vtkTubeFilter.h>
13  #include <vtkSphereSource.h>
14  #include <vtkLookupTable.h>
15  #include <vtkOutlineFilter.h>
16  #include <vtkPolyData.h>
17  #include <vtkPolyDataMapper.h>
18  #include <vtkRenderWindow.h>
19  #include <vtkActor.h>
20  #include <vtkRenderer.h>
21  #include <vtkProperty.h>
22  #include <vtkRenderWindowInteractor.h>
23  #include <vtkInteractorStyleTrackballCamera.h>
24
25  #define NSP 56 /* 解像度 */
26
27  float pos1[3] = {100.0, 140.0, 5.0};
28  float pos2[3] = {200.0, 200.0, 15.0};
29
30  int main( int argc, char *argv[] )
31  {
32      float range[2];
33      char datafile[]   = "./vector.vtk";
34
```

```
35      vtkStructuredPointsReader *reader
36         = vtkStructuredPointsReader::New();
37       reader->SetFileName(datafile);
38
39      vtkImageData *imgData;
40       imgData = reader->GetOutput();
41       imgData->Update();
42       imgData->GetScalarRange(range);
43
44      vtkLookupTable *lut = vtkLookupTable::New();
45       lut->SetHueRange(0.7, 0.0);
46       lut->Build();
47
48      /* 解像度NSPの線分を生成 */
49      vtkLineSource *pole = vtkLineSource::New();
50       pole->SetPoint1(pos1);
51       pole->SetPoint2(pos2);
52       pole->SetResolution(NSP);
53
54      /* 流線計算。上記線分の頂点が出発点となる */
55      vtkStreamLine *slines = vtkStreamLine::New();
56      vtkRungeKutta4 *integ = vtkRungeKutta4::New();
57
58       slines->SetInput(imgData);
59       slines->SetSource(pole->GetOutput());
60       slines->SetIntegrator(integ);
61       slines->SetMaximumPropagationTime(300);
62       slines->SetIntegrationStepLength(0.1);
63       slines->SetIntegrationDirectionToForward();
64       slines->SetStepLength(0.5);
65
66      /* 流線の間に面を生成する */
67      vtkRuledSurfaceFilter *ssurfaces = vtkRuledSurfaceFilter::New();
68       ssurfaces->SetInput(slines->GetOutput());
69       ssurfaces->SetOnRatio(2); /* 一つおき */
70       ssurfaces->SetOffset(0);
71       ssurfaces->SetDistanceFactor(10.0);
72       ssurfaces->SetRuledModeToResample();
73       ssurfaces->SetResolution(100, 1);
74
75      vtkPolyDataMapper * ssMapper = vtkPolyDataMapper::New();
76       ssMapper->SetInput(ssurfaces->GetOutput());
77       ssMapper->SetScalarRange(range[0], range[1]);
78       ssMapper->SetLookupTable(lut);
79
80      vtkActor *Actor = vtkActor::New();
81       Actor->SetMapper(ssMapper);
```

```
 82
 83      /* 流線そのものも表示する */
 84      vtkPolyDataMapper * slinesMapper = vtkPolyDataMapper::New();
 85       slinesMapper->SetInput(slines->GetOutput());
 86       slinesMapper->SetScalarRange(range[0], range[1]);
 87       slinesMapper->SetLookupTable(lut);
 88
 89      vtkActor *Actor2 = vtkActor::New();
 90       Actor2->SetMapper(slinesMapper);
 91
 92      /* 出発点を発生させた線分をチューブで囲み表示 */
 93      vtkTubeFilter *tubes = vtkTubeFilter::New();
 94       tubes->SetInput(pole->GetOutput());
 95       tubes->SetRadius(1.0);
 96       tubes->SetNumberOfSides(8);
 97
 98      vtkPolyDataMapper *PMapper = vtkPolyDataMapper::New();
 99       PMapper->SetInput(tubes->GetOutput());
100
101      vtkActor *PActor = vtkActor::New();
102       PActor->SetMapper(PMapper);
103
104      /* 線分の始点と終点に球体を表示 */
105      vtkSphereSource *sphere = vtkSphereSource::New();
106       sphere -> SetRadius(2.0);
107
108      vtkPolyDataMapper *sMapper = vtkPolyDataMapper::New();
109       sMapper->SetInput(sphere->GetOutput());
110
111      vtkActor *SActor1 = vtkActor::New();
112       SActor1->SetMapper(sMapper);
113       SActor1->SetPosition(pos1);
114
115      vtkActor *SActor2 = vtkActor::New();
116       SActor2->SetMapper(sMapper);
117       SActor2->SetPosition(pos2);
118
119      /* データの外枠 */
120      vtkOutlineFilter *outline = vtkOutlineFilter::New();
121       outline->SetInput(imgData);
122
123      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
124       OLMapper->SetInput(outline->GetOutput());
125
126      vtkActor *OLActor = vtkActor::New();
127       OLActor->SetMapper(OLMapper);
128
```

```
129      vtkRenderer *vren= vtkRenderer::New();
130       vren->AddActor(Actor);
131       vren->AddActor(Actor2);
132       vren->AddActor(PActor);
133       vren->AddActor(OLActor);
134       vren->AddActor(SActor1);
135       vren->AddActor(SActor2);
136       vren->SetBackground( 0.0, 0.0, 0.0 );
137
138      vtkRenderWindow *renWin = vtkRenderWindow::New();
139       renWin->AddRenderer( vren );
140       renWin->SetSize( 500, 375 );
141
142      vtkRenderWindowInteractor *iwin
143          = vtkRenderWindowInteractor::New();
144       iwin->SetRenderWindow(renWin);
145
146      vtkInteractorStyleTrackballCamera *trackball =
147       vtkInteractorStyleTrackballCamera::New();
148
149       iwin->SetInteractorStyle(trackball);
150       iwin->Initialize();
151       iwin->Start();
152
153       reader->Delete();
154       lut->Delete();
155       pole->Delete();
156       slines->Delete();
157       integ->Delete();
158       ssurfaces->Delete();
159       ssMapper->Delete();
160       Actor->Delete();
161       slinesMapper->Delete();
162       Actor2->Delete();
163       tubes->Delete();
164       PMapper->Delete();
165       PActor->Delete();
166       sphere->Delete();
167       sMapper->Delete();
168       SActor1->Delete();
169       SActor2->Delete();
170       outline->Delete();
171       OLMapper->Delete();
172       OLActor->Delete();
173       iwin->Delete();
174       trackball->Delete();
175       vren->Delete();
```

```
176        renWin->Delete();
177
178     return 0;
179   }
```

7.5.2 サンプルプログラム 3 の簡単な説明

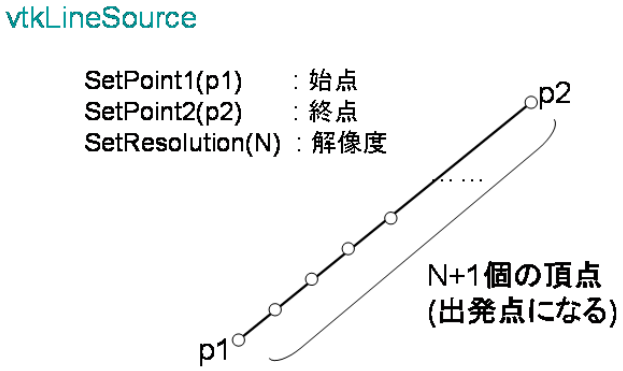

Figure 7.6: vtkLineSource で線分のポリゴンデータを生成

新規な部分は，**48 - 52** 行と **66 - 73** 行である。**48 - 52** 行では，vtkLineSource を使い，線分のポリゴンを生成している。流線計算の出発点は，線分の頂点がセットされる (Figure 7.6)。**66 - 73** 行では，vtkRuledSurfaceFilter を使い，隣り合う流線 (実際には入力された順番が「隣り合った」流線) に面を張っている。
50 - 51 : SetPoint1, 2 で，線分の始点と終点を設定
52 : SetResolution で線分の解像度を指定。引数を N とすると，上記の線分は，(N+1) 個の頂点を持つ (頂点は線分上に等間隔に配置される)
59 : vtkLineSource の出力を vtkStreamLine に代入。上記の頂点が，流線計算の出発点になる
69 : vtkRuledSurfaceFilter のメソッド SetOnRatio で，面のはり方を設定する。2 で一つおき。1 でまったく飛ばさずに面を張っていく。例えば，流線が 6 本あったとすると，引数を 1 とすると 1-2, 2-3,... の流線の間に面を張る。2 とすると，1-2, 3-4, 5-6 番目の流線の間，3 とすると 1-2, 5-6 番目の流線の間に面を張る (Figure 7.7)
71 : 引数を d とすると，流線上の点と点の距離が，出発点同士の距離の d 倍以上のところでは，面を張らなくなる
72 - 73 : 流線上の点を等間隔に 100(**73** 行の一つ目の引数) 個リサンプルして，面を作る。**73** 行の二つ目の引数は，線と線の間にいくつポリゴンを作るか。**72** 行で，SetRuledModeToPointwalk() とすると，流線上の点をそのまま用いて面を張る (ポリゴンが密なところ疎なところができる)
83 - 90 : 流線も表示させる
92 - 102 : 流線の出発点を取り出した線分を，チューブで表示
104 - 117 : 上記線分の始点終点に，球体を表示

表示は，Figure 7.8。

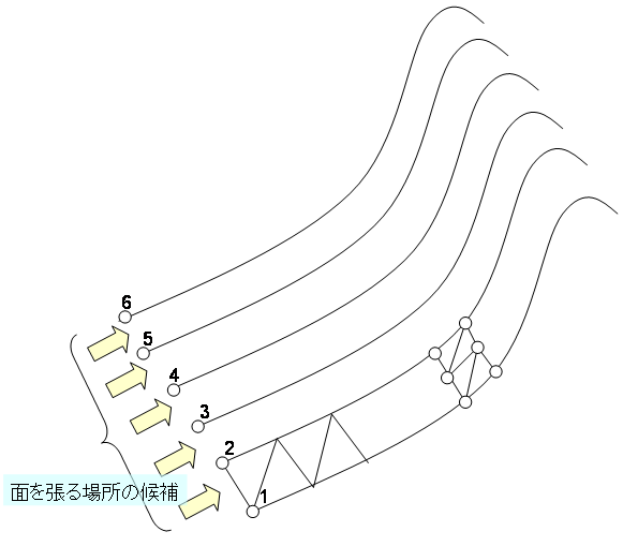

Figure 7.7: 面を張る位置

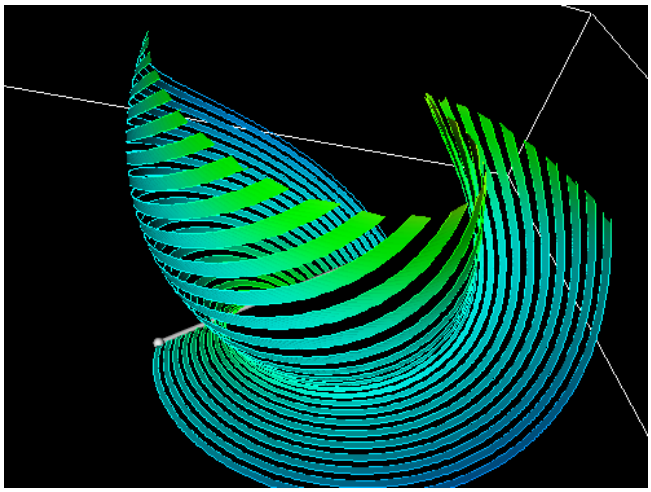

Figure 7.8: Stream Surfaces

7.6 この章のまとめ

- 流線によるベクトル場の可視化
- 点データの操作 (流線と矢印の組み合わせ)
- チューブ・リボン
- 流面 (Stream Surfaces)

本章で紹介したクラスは，Table 7.1 の通り。

Table 7.1: 本章で紹介したクラスおよびそのメソッド

点データの操作	vtkProbeFilter
点データ	vtkPoints
ポリゴンデータ	vtkPolyData
流線計算	vtkStreamLine, vtkDashedStreamLine : SetIntegrator(vtkInitialValueProblemSolver *) : SetMaximumPropagationTime(float) : SetIntegrationStepLength(float) : SetStepLength(float) : SetIntegrationDirectionTo〜(), Forward, Backward, BothDirections
流線計算 (Integrator)	vtkRungeKutta4
点データ生成	vtkPointSource : SetCenter(float [3]) : SetRadius(float) : SetNumberOfPoints(vtkIdType)
線データ生成	vtkLineSource : SetPoint1,2(float [3]) : SetResolution(int)
チューブ	vtkTubeFilter : SetRadius(float) : SetRadiusFactor(float) : SetVaryRadiusToVaryRadius〜(), ByScalar,ByVector, Off : SetNumberOfSides(int)
リボン	vtkRibbonFilter
面	vtkRuledSurfaceFilter : SetOnRatio(int) : SetOffSet(int) : SetDistanceFactor(float) : SetRuledModeTo〜(), Resample, Pointwalk : SetResolution(int, int)

Chapter 8

円柱座標・球座標

8.1 本章の概要

StructuredGrid で，円柱座標や球座標のデータを可視化する。

8.2 円柱座標

vtkStructuredGrid を使うと，円柱座標や球座標のデータも取り扱うことができる。vtkStructured-Grid で円柱や球座標のデータさえ作ってしまえば，それ以降の可視化パイプラインは，前章までの知識で組むことができる。

8.2.1 サンプルプログラム 1

ひとまず，サンプルプログラムを見ながら，円柱座標データの作り方を解説する。本章のサンプルデータは，前章 (ベクトル場の可視化) と同様のもので，スカラー場とベクトル場いずれも存在する。

```
 1  /* Cylindrical Coordinates */
 2  /* cylind1.cxx */
 3  #include <vtkStructuredGrid.h>
 4  #include <vtkFloatArray.h>
 5  #include <vtkPointData.h>
 6  #include <vtkMaskPoints.h>
 7  #include <vtkHedgeHog.h>
 8  #include <vtkContourFilter.h>
 9  #include <vtkLookupTable.h>
10  #include <vtkStructuredGridOutlineFilter.h>
11  #include <vtkPolyData.h>
12  #include <vtkPolyDataMapper.h>
13  #include <vtkRenderWindow.h>
14  #include <vtkActor.h>
15  #include <vtkRenderer.h>
16  #include <vtkProperty.h>
17  #include <vtkRenderWindowInteractor.h>
18  #include <vtkInteractorStyleTrackballCamera.h>
```

```
19
20  #define SIZE_T 300
21  #define SIZE_Z 300
22  #define SIZE_R 100
23
24  float vector[SIZE_T*SIZE_Z*SIZE_R][3];
25  float vector_abs[SIZE_T*SIZE_Z*SIZE_R];
26
27  char datafile_t[]   = "./vector_x.dat";
28  char datafile_z[]   = "./vector_y.dat";
29  char datafile_r[]   = "./vector_z.dat";
30  char datafile_abs[] = "./vector_abs.dat";
31
32  float *vector_t;
33  float *vector_z;
34  float *vector_r;
35
36  float points[SIZE_T*SIZE_Z*SIZE_R][3];
37  const double PI = 3.14159;
38  const double dt = 180.0/299.0; /* Theta の刻み幅 */
39
40  void load_data();     /* データ読み込み */
41  void cylind_coords(); /* 格子点のデータ作成 */
42
43  int main( int argc, char *argv[] )
44  {
45      int size[3] = {SIZE_T, SIZE_Z, SIZE_R};
46      float range[2];
47
48      cylind_coords();
49      load_data();
50
51      vtkFloatArray *varray = vtkFloatArray::New();
52       varray->SetNumberOfComponents(3);
53       varray->SetNumberOfTuples(SIZE_T* SIZE_Z* SIZE_R);
54       varray->SetArray(vector[0], SIZE_T* SIZE_Z* SIZE_R*3, 1);
55
56      vtkFloatArray *sarray = vtkFloatArray::New();
57       sarray->SetArray(vector_abs, SIZE_T* SIZE_Z* SIZE_R, 1);
58
59      /* 格子点の座標データ */
60      vtkFloatArray *carray = vtkFloatArray::New();
61       carray->SetNumberOfComponents(3);
62       carray->SetNumberOfTuples(SIZE_T* SIZE_Z* SIZE_R);
63       carray->SetArray(points[0], SIZE_T* SIZE_Z* SIZE_R*3, 1);
64
65      vtkPoints *coordinates = vtkPoints::New();
```

```
 66       coordinates->SetDataTypeToFloat();
 67       coordinates->SetData(carray);
 68
 69      vtkStructuredGrid *CylindData = vtkStructuredGrid::New();
 70       CylindData->SetDimensions(size);
 71
 72       /* 格子点の座標データをセット */
 73       CylindData->SetPoints(coordinates);
 74       CylindData->GetPointData()->SetVectors(varray);
 75       CylindData->GetPointData()->SetScalars(sarray);
 76       CylindData->Update();
 77       CylindData->GetScalarRange(range);
 78
 79      vtkLookupTable *lut = vtkLookupTable::New();
 80       lut->SetHueRange(0.7, 0.0);
 81       lut->Build();
 82
 83      /* 以下 HedgeHog と 等値面のパイプライン */
 84      /* HedgeHog */
 85      vtkMaskPoints *mask = vtkMaskPoints::New();
 86       mask->SetInput(CylindData);
 87       mask->SetOnRatio(1000);
 88       mask->RandomModeOn();
 89
 90      vtkHedgeHog *hedgehog = vtkHedgeHog::New();
 91       hedgehog->SetInput(mask->GetOutput());
 92       hedgehog->SetScaleFactor(5.0/range[1]);
 93
 94      vtkPolyDataMapper * vecMapper = vtkPolyDataMapper::New();
 95       vecMapper->SetInput(hedgehog->GetOutput());
 96       vecMapper->SetScalarRange(range[0], range[1]);
 97       vecMapper->SetLookupTable(lut);
 98
 99      vtkActor *vecActor = vtkActor::New();
100       vecActor->SetMapper(vecMapper);
101
102     /* 等値面 */
103      vtkContourFilter *contour = vtkContourFilter::New();
104       contour->SetInput(CylindData);
105       contour->SetValue(0, 3.0);
106       contour->ComputeNormalsOn();
107
108      vtkPolyDataMapper * isoMapper = vtkPolyDataMapper::New();
109       isoMapper->SetInput(contour->GetOutput());
110       isoMapper->ScalarVisibilityOff();
111
112      vtkActor *isoActor = vtkActor::New();
```

```
113       isoActor->SetMapper(isoMapper);
114
115     /* データの外枠 */
116       vtkStructuredGridOutlineFilter *outline
117               = vtkStructuredGridOutlineFilter::New();
118       outline->SetInput(CylindData);
119
120      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
121       OLMapper->SetInput(outline->GetOutput());
122
123      vtkActor *OLActor = vtkActor::New();
124       OLActor->SetMapper(OLMapper);
125
126      vtkRenderer *vren= vtkRenderer::New();
127       vren->AddActor(vecActor);
128       vren->AddActor(isoActor);
129       vren->AddActor(OLActor);
130       vren->SetBackground( 0.0, 0.0, 0.0 );
131
132      vtkRenderWindow *renWin = vtkRenderWindow::New();
133       renWin->AddRenderer( vren );
134       renWin->SetSize( 800, 600 );
135
136      vtkRenderWindowInteractor *iwin
137              = vtkRenderWindowInteractor::New();
138       iwin->SetRenderWindow(renWin);
139
140      vtkInteractorStyleTrackballCamera *trackball =
141       vtkInteractorStyleTrackballCamera::New();
142
143       iwin->SetInteractorStyle(trackball);
144       iwin->Initialize();
145       iwin->Start();
146
147       varray->Delete();
148       sarray->Delete();
149       coordinates->Delete();
150       CylindData->Delete();
151       lut->Delete();
152       mask->Delete();
153       hedgehog->Delete();
154       contour->Delete();
155       vecMapper->Delete();
156       vecActor->Delete();
157       isoMapper->Delete();
158       isoActor->Delete();
159       outline->Delete();
```

```
160       OLMapper->Delete();
161       OLActor->Delete();
162       iwin->Delete();
163       trackball->Delete();
164       vren->Delete();
165       renWin->Delete();
166
167     return 0;
168   }
169
170   void cylind_coords(){
171       int i,j,k;
172       double theta, r, z;
173
174     /* 格子点の位置 */
175     for(k=0; k<SIZE_R; k++){
176       for(j=0; j<SIZE_Z; j++){
177         for(i=0; i<SIZE_T; i++){
178
179           theta = PI/180.0*i*dt;
180           r = 50.0 + 0.5*k; /* 半径方向は 50, 99.5*/
181           z = j;
182
183           points[i + SIZE_T*j + k*SIZE_T*SIZE_Z][0]
184                     = r * cos(theta);
185
186           points[i + SIZE_T*j + k*SIZE_T*SIZE_Z][1]
187                     = r * sin(theta);
188
189           points[i + SIZE_T*j + k*SIZE_T*SIZE_Z][2]
190                     = z;
191
192         }
193       }
194     }
195
196   }
197
198   void load_data(){
199
200       int i, j, k;
201       double theta;
202       float temp_t, temp_z, temp_r;
203
204       FILE *fpi;
205
206       vector_t = (float *)malloc(sizeof(float)*SIZE_T*SIZE_Z*SIZE_R);
```

```
207     vector_z = (float *)malloc(sizeof(float)*SIZE_T*SIZE_Z*SIZE_R);
208     vector_r = (float *)malloc(sizeof(float)*SIZE_T*SIZE_Z*SIZE_R);
209
210
211     if ((fpi = fopen(datafile_t, "rb")) == NULL) {
212         puts("cannot open");
213         exit(1);
214     }
215
216       fread(vector_t, sizeof(float), SIZE_T*SIZE_Z*SIZE_R, fpi);
217
218     fclose(fpi);
219
220
221     if ((fpi = fopen(datafile_z, "rb")) == NULL) {
222         puts("cannot open");
223         exit(1);
224     }
225
226       fread(vector_z, sizeof(float), SIZE_T*SIZE_Z*SIZE_R, fpi);
227
228     fclose(fpi);
229
230
231     if ((fpi = fopen(datafile_r, "rb")) == NULL) {
232         puts("cannot open");
233         exit(1);
234     }
235
236       fread(vector_r, sizeof(float), SIZE_T*SIZE_Z*SIZE_R, fpi);
237
238     fclose(fpi);
239
240     if ((fpi = fopen(datafile_abs, "rb")) == NULL) {
241         puts("cannot open");
242         exit(1);
243     }
244
245       fread(vector_abs, sizeof(float), SIZE_T*SIZE_Z*SIZE_R, fpi);
246
247     fclose(fpi);
248
249     /*********************************/
250     /*  x 成分 -> theta 成分         */
251     /*  y 成分 -> z 成分             */
252     /*  z 成分 -> radial 成分        */
253     /*********************************/
```

```
254
255     for(k=0;k<SIZE_R;k++){
256      for(j=0;j<SIZE_Z;j++){
257       for(i=0;i<SIZE_T;i++){
258
259       temp_t = vector_t[i+j*SIZE_T + k*SIZE_T*SIZE_Z];
260       temp_z = vector_z[i+j*SIZE_T + k*SIZE_T*SIZE_Z];
261       temp_r = vector_r[i+j*SIZE_T + k*SIZE_T*SIZE_Z];
262
263        theta = PI/180.0*i*dt;
264
265         vector[i + j* SIZE_T + k*SIZE_T*SIZE_Z][0]
266              = -temp_t*sin(theta) + temp_r * cos(theta);
267
268         vector[i + j* SIZE_T + k*SIZE_T*SIZE_Z][1]
269              =  temp_t*cos(theta) + temp_r * sin(theta);
270
271         vector[i + j* SIZE_T + k*SIZE_T*SIZE_Z][2]
272              =  temp_z;
273
274       }
275      }
276     }
277
278     free(vector_t);
279     free(vector_z);
280     free(vector_r);
281
282  }
```

8.2.2 サンプルプログラム 1 の簡単な説明 : 格子点・データの作り方

格子点の作り方は，ベクトルデータの作り方に似ている。このサンプルプログラム 1 では，cylind_coords() 関数 (**170 - 196**) で，全格子点の座標を float points[SIZE_T*SIZT_Z*SIZE_R][3] に代入している。その後，points を carray(vtkFloatArray:**60 - 63** 行) と coordinates(vtkPoints) を介して，CylindData (vtkStructuredGrid) に代入している。この points に入っているデータは，座標値なので成分も 3 つある。それ故ベクトルデータのときと同様に，

carray->SetNumberOfComponents(3);
carray->SetNumberOfTuples(SIZE_T* SIZE_Z* SIZE_R);

としている。これは，残念ながら直接 CylindData に代入できなく，一旦 coordinates(vtkPoints:**65 - 67** 行) に代入してから

CylidData->SetPoints(coordinates);

とする。

データ (スカラーやベクトル) の CylindData への代入方法は，等間隔メッシュや Rectilinear の場合と同様である。このサンプルプログラムでは，前章と同じデータの X 成分 → Theta 成分 (0 度から 0.602 度刻み)，Y 成分 →z 成分，Z 成分 →R 成分 (50.0 から 0.5 刻み) となるように，load_data 関数 (**198 - 282** 行) で変換している。

格子点の位置やベクトルデータは，X Y Z で与えることに注意。

20 - 22 : Theta 方向 (SZIE_T), Z 方向 (SIZE_Z) の解像度はいずれも 300, R 方向 (SIZE_R) は 100
以下 cylind_coords 関数
170 - 196 :Theta は [0, π, Z は [0,300)，R は [50, 100) になるようにしている
以下 load_data 関数
198 - 282 : データを vector, vector_abs に読み込む。そのとき，sin, cos を使って，X 成分 → Theta 成分，Y 成分 → Z 成分，Z 成分 → R 成分となるように変換している (VTK とは関係ない)
59 - 63 : carray (vtkFloatArray) に，格子点の位置データ (float points[...][3]) を代入
65 - 67 : carray を coordinates (vtkPoints) に代入
72 - 73 : 円柱座標を作るために用意した箱 (CylindData) に，coordinates をセット (晴れて円柱座標の箱になる)
74 - 75 : スカラー及びベクトルデータを CylindData に代入

Figure8.1 にパイプラインを示す。

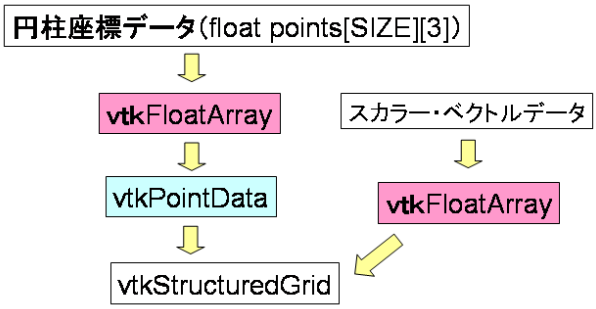

Figure 8.1: 円柱座標の作り方

8.2.3 サンプルプログラムの簡単な説明:可視化処理部

このプログラムは，ベクトル場を HedgeHog で，スカラー場を等値面で可視化する。可視化パイプラインは，前章までと同様である。唯一違うのが，外枠を作るクラスである (**115 - 118** 行)。vtkOutlineFilter ではなく，vtkStructuredGridOutlineFilter を使っている。vtkOutlineFilter を使用してもエラーや警告はでないが，表示される外枠は，円柱形の枠ではなく，データ全体が入る直方体の枠になる。表示される画像は，Figure 8.2(左)。ここではソースリストを紹介しないが，もう一つ円柱座標のサンプルプログラム (cylind2.cxx) を用意した。カラーコンターと流線で可視化している。実行結果のみ，Figure 8.2(右) に示す。

参考のため，等間隔メッシュを使い，サンプルプログラム 1 と同じ可視化パイプラインをつないだ結果を，Figure 8.3 に示す。

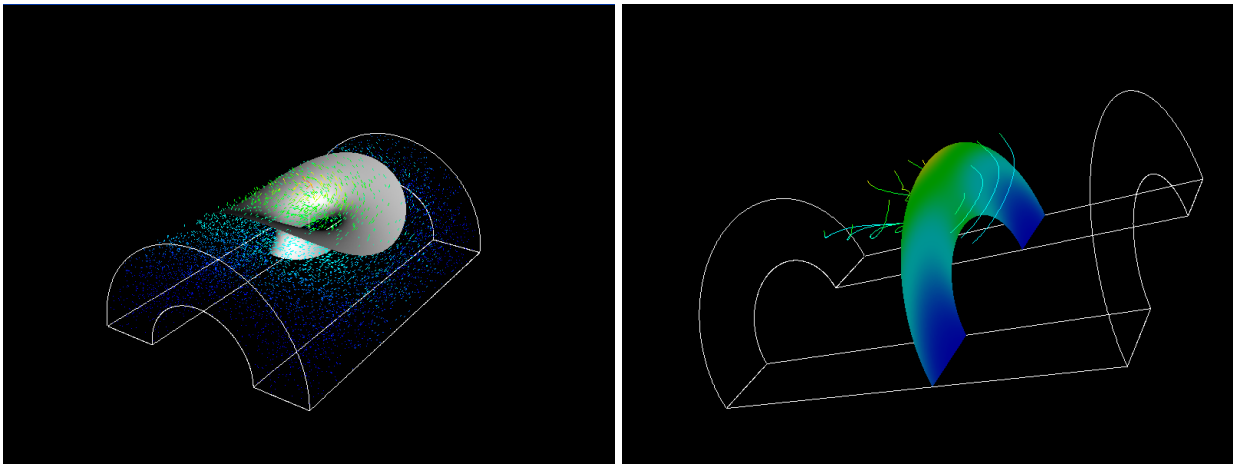

Figure 8.2: (左)HedgeHog と 等値面による可視化，(右) カラーコンターと流線による可視化

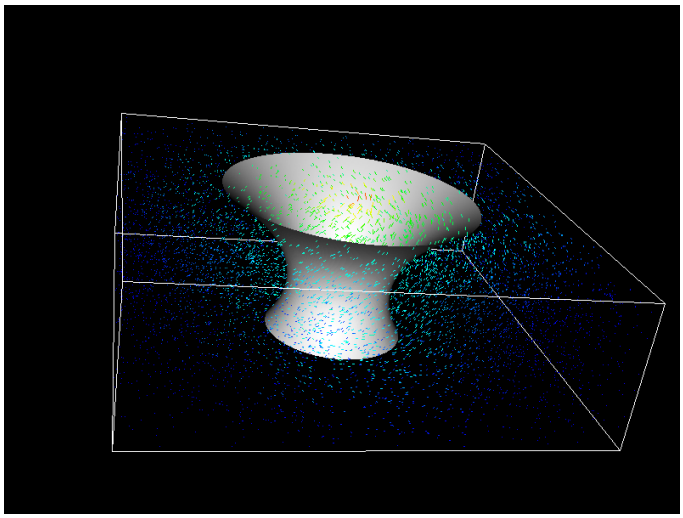

Figure 8.3: HedgeHog と 等値面による可視化 (等間隔メッシュ)

8.3 球座標

円柱座標のときと同様に，球座標の格子点のデータさえ作ってしまえば，等値面でも流線でも使えることは理解いただけていると思う。ここでは，ボリューム・レンダリングに挑戦する。(ここで紹介する方法は，StructuredGrid の他に，Rectilinear でも使えるはずである)

8.3.1 サンプルプログラム 2

座標は，X 方向 →Theta 方向 (解像度 300, [π/4, 3π/4])，Y 方向 →Phi 方向 (解像度 300, [0, π/2]), Z 方向 →R 方向 (解像度 100, [50, 75)) となるようにしている (spheri_coords 関数 **202 - 228** 行)。座標やデータの設定方法は，サンプルプログラム 1 を参考にしていただければ，理解していただけると思う。なお，半径 50.0 の球体も同時に表示するようにしている。

```
1  /* Spherical Coordinates */
2  /* spheri3.cxx */
3  #include <vtkStructuredGrid.h>
```

```
 4  #include <vtkStructuredGridGeometryFilter.h>
 5  #include <vtkFloatArray.h>
 6  #include <vtkPointData.h>
 7  #include <vtkSphereSource.h>
 8  #include <vtkDataSetMapper.h>
 9  #include <vtkPlane.h>
10  #include <vtkCutter.h>
11  #include <vtkPolyDataNormals.h>
12  #include <vtkLookupTable.h>
13  #include <vtkStructuredGridOutlineFilter.h>
14  #include <vtkPolyData.h>
15  #include <vtkPolyDataMapper.h>
16  #include <vtkRenderWindow.h>
17  #include <vtkActor.h>
18  #include <vtkRenderer.h>
19  #include <vtkProperty.h>
20  #include <vtkCamera.h>
21  #include <vtkRendererSource.h>
22  #include <vtkBMPWriter.h>
23
24  #include <unistd.h>
25
26  #define SIZE_T 300
27  #define SIZE_P 300
28  #define SIZE_R 100
29
30  float vector_abs[SIZE_T*SIZE_P*SIZE_R];
31  char datafile_abs[] = "./vector_abs.dat";
32
33  float points[SIZE_T*SIZE_P*SIZE_R][3];
34  const double PI = 3.14159;
35  const double dt = 90.0/299.0; /* Theta, Phi の刻み幅 */
36
37  void load_data();
38  void spheri_coords();
39
40  int main( int argc, char *argv[] )
41  {
42      int i;
43      int size[3] = {SIZE_T, SIZE_P, SIZE_R};
44      float range[2];
45      float theta = PI/180.0*SIZE_T/2.0*dt + PI/4.0;
46      float r = 50.0 + 0.25*SIZE_R/2.0;
47      float phi = PI/180.0*dt * SIZE_P/2.0;
48      float Color[4];
49      float center[3], normal[3];
50
```

```
51      spheri_coords();
52      load_data();
53
54      vtkFloatArray *sarray = vtkFloatArray::New();
55       sarray->SetArray(vector_abs, SIZE_T* SIZE_P* SIZE_R, 1);
56
57      /* 格子点の座標データ */
58      vtkFloatArray *carray = vtkFloatArray::New();
59       carray->SetNumberOfComponents(3);
60       carray->SetNumberOfTuples(SIZE_T* SIZE_P* SIZE_R);
61       carray->SetArray(points[0], SIZE_T* SIZE_P* SIZE_R*3, 1);
62
63      vtkPoints *coordinates = vtkPoints::New();
64       coordinates->SetDataTypeToFloat();
65       coordinates->SetData(carray);
66
67      vtkStructuredGrid *SphericalData = vtkStructuredGrid::New();
68       SphericalData->SetDimensions(size);
69
70       /* 格子点の座標データをセット */
71       SphericalData->SetPoints(coordinates);
72       SphericalData->GetPointData()->SetScalars(sarray);
73       SphericalData->Update();
74       SphericalData->GetScalarRange(range);
75
76      /* カラーテーブル作成 */
77      vtkLookupTable *lut = vtkLookupTable::New();
78       lut->SetNumberOfColors(256);
79       lut->SetHueRange(0.677, 0.0);
80       lut->Build();
81
82       for(i=0;i<256;i++){
83        lut->GetTableValue(i, Color);
84         Color[3] = i/255.0;
85
86        if(i > 100 && i < 150) Color[3] = 0.0;
87
88        lut->SetTableValue(i, Color);
89       }
90
91      /* スライス */
92       center[0] = r * sin(theta) * cos(phi);
93       center[1] = r * sin(theta) * sin(phi);
94       center[2] = r * cos(theta);
95
96      vtkPlane *plane = vtkPlane::New();
97       plane->SetOrigin(center);
```

```
 98
 99      vtkCutter *cutter = vtkCutter::New();
100       cutter->SetInput(SphericalData);
101       cutter->SetCutFunction(plane);
102       cutter->SetSortByToSortByCell();
103       cutter->GenerateCutScalarsOff();
104
105      vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
106       Mapper->SetInput(cutter->GetOutput());
107       Mapper->SetLookupTable(lut);
108       Mapper->SetScalarRange(range);
109
110      vtkActor *Actor = vtkActor::New();
111       Actor->SetMapper(Mapper);
112
113      /* 半径 50 の球体を表示する */
114      vtkSphereSource *ball = vtkSphereSource::New();
115       ball->SetRadius(50.0);
116       ball->SetPhiResolution(100);
117       ball->SetThetaResolution(100);
118
119      vtkPolyDataNormals *norm = vtkPolyDataNormals::New();
120       norm->SetInput(ball->GetOutput());
121
122      vtkPolyDataMapper *ballMapper = vtkPolyDataMapper::New();
123       ballMapper->SetInput(norm->GetOutput());
124
125      vtkActor *ballActor = vtkActor::New();
126       ballActor->SetMapper(ballMapper);
127       ballActor->GetProperty()->SetColor(1.0, 0.75, 0.75);
128
129      vtkCamera *camera = vtkCamera::New();
130       camera->SetPosition(150, 300, 0.0);
131       camera->SetFocalPoint(center[0], center[1], center[2]);
132       camera->SetClippingRange(10, 800);
133       camera->SetViewUp(0,0,1);
134
135       /* 投影面の法線ベクトルを求め，平面の法線ベクトルとする */
136       camera->ComputeViewPlaneNormal();
137       camera->GetViewPlaneNormal(normal);
138
139       plane->SetNormal(normal);
140       cutter->GenerateValues(64, -15., 15.);
141
142      /* データの外枠 */
143      vtkStructuredGridOutlineFilter *outline
144               = vtkStructuredGridOutlineFilter::New();
```

```
145       outline->SetInput(SphericalData);
146
147      vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New();
148        OLMapper->SetInput(outline->GetOutput());
149
150      vtkActor *OLActor = vtkActor::New();
151        OLActor->SetMapper(OLMapper);
152
153      vtkRenderer *vren = vtkRenderer::New();
154       vren->AddActor(ballActor);
155       vren->AddActor(OLActor);
156       vren->AddActor(Actor);
157       vren->SetActiveCamera(camera);
158       vren->SetBackground( 0.0, 0.0, 0.0 );
159
160      vtkRenderWindow *renWin = vtkRenderWindow::New();
161       renWin->AddRenderer( vren );
162       renWin->SetSize( 500, 375 );
163
164     char FileName[]="volren.bmp";
165
166      vtkRendererSource *rs = vtkRendererSource::New();
167       rs->SetInput(vren);
168       rs->WholeWindowOn();
169
170      vtkBMPWriter *bw = vtkBMPWriter::New();
171       bw->SetInput(rs->GetOutput());
172
173       renWin->Render();
174       rs->Modified();
175
176       bw->SetFileName(FileName);
177       bw->Write();
178
179       sarray->Delete();
180       coordinates->Delete();
181       SphericalData->Delete();
182       lut->Delete();
183       plane->Delete();
184       cutter->Delete();
185       Mapper->Delete();
186       Actor->Delete();
187       outline->Delete();
188       OLMapper->Delete();
189       OLActor->Delete();
190       ball->Delete();
191       ballMapper->Delete();
```

```
192       ballActor->Delete();
193       camera->Delete();
194       vren->Delete();
195       renWin->Delete();
196       rs->Delete();
197       bw->Delete();
198
199     return 0;
200   }
201
202   void spheri_coords(){
203       int i,j,k;
204       double theta, phi, r;
205
206      /* 格子点の位置 */
207      for(k=0; k<SIZE_R; k++){
208       for(j=0; j<SIZE_P; j++){
209        for(i=0; i<SIZE_T; i++){
210
211         theta = PI/180.0*i*dt + PI/4.0;
212         r = 50.0 + 0.25*k;
213         phi = PI/180.0*dt * j;
214
215         points[i + SIZE_T*j + k*SIZE_T*SIZE_P][0]
216                  = r * sin(theta) * cos(phi);
217
218         points[i + SIZE_T*j + k*SIZE_T*SIZE_P][1]
219                  = r * sin(theta) * sin(phi);
220
221         points[i + SIZE_T*j + k*SIZE_T*SIZE_P][2]
222                  = r * cos(theta);
223
224        }
225       }
226      }
227
228   }
229
230   void load_data(){
231
232       FILE *fpi;
233
234       if ((fpi = fopen(datafile_abs, "rb")) == NULL) {
235           puts("cannot open");
236           exit(1);
237       }
238
```

```
239         fread(vector_abs, sizeof(float), SIZE_T*SIZE_P*SIZE_R, fpi);
240
241       fclose(fpi);
242
243   }
```

8.3.2　サンプルプログラム 2 の簡単な説明

VTK のボリューム・レンダリングは，等間隔メッシュに限られているが，工夫次第では，様々な座標系で実行できる。テクスチャ・マッピングを利用したボリューム・レンダリングを思い出していただきたい。不透明度を含んだ画像を貼り付けたポリゴンを前後に並べてブレンドする方法である。テクスチャ機能が使えなくても，第 1 章で解説したカラーコンターの方法で代用できる。しかもデータは等間隔メッシュである必要がない。つまり，頂点に色だけでなく不透明度も割り当てたカラーコンターのスライスを前後に並べるのである。スライスの引き出しには，vtkPlane と vtkCutter が使える。これを使えば，3 次元テクスチャを利用したボリューム・レンダリングのように，投影面に平行なスライスを使ってボリューム・レンダリングできる (Figure 8.4)。

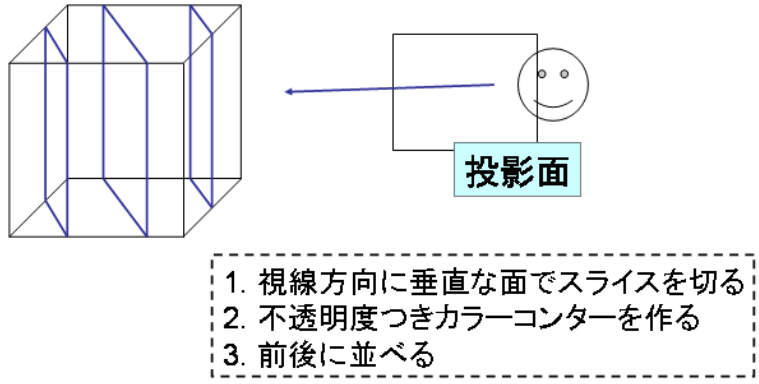

Figure 8.4: 投影面と平行な面でスライス

以下，スライス関係の説明

91 - 103 : データの中心を通る平面 (plane，原点のみ設定) と cutter を生成している
135 - 137 : 投影面の法線ベクトルを計算して，float normal[3] に代入 (規格化済み)
139 : 上で求めた法線ベクトルを plane の法線ベクトルに設定
140 : -15.0 〜 15.0 の間で，64 枚スライスを生成。意味するところは，第 3 章に登場したときは，$F(x, y, z) = 0.0$ の平面でデータを切ったが，本章では，$F(x, y, z) = A_n$(A_n は，-15.0〜15.0 の間から 64 個等間隔に値を割り当てられる) の (64 枚の) 平面で切ったスライスを表示させる

色関係は，**76 - 89** 行で設定している。カラーテーブルを一旦作って (**80** 行) から，不透明度を変えている (**82 - 89** 行)。これで，球座標のボリューム・レンダリングが実行できる。表示画像は，Figure 8.5。

ただ，この方法は，処理時間が非常に長いという欠点がある。Figure 8.5 を出すのに，venus で 20 分強費やしている。スライス数を増やすと，さらに処理時間がかかると思われる。

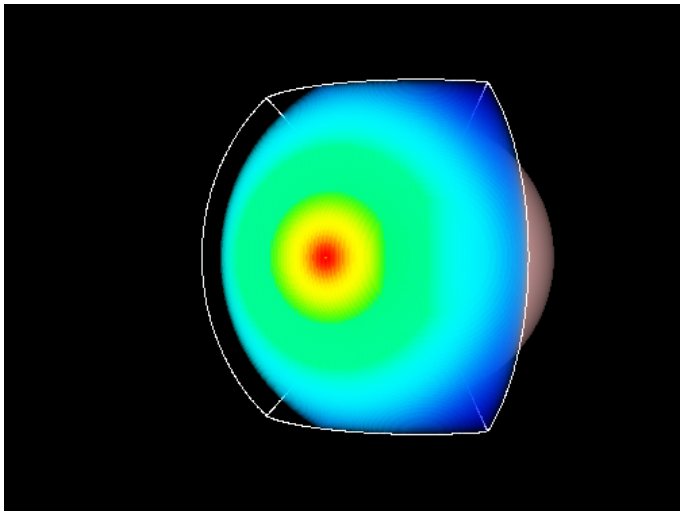

Figure 8.5: 球座標データのボリューム・レンダリング

8.3.3　その他の球座標サンプル

spheri1.cxx, spheri2.cxx という 2 つのサンプルプログラムを用意した。実行結果は，それぞれ Figure 8.6 の (左) と (右) である。

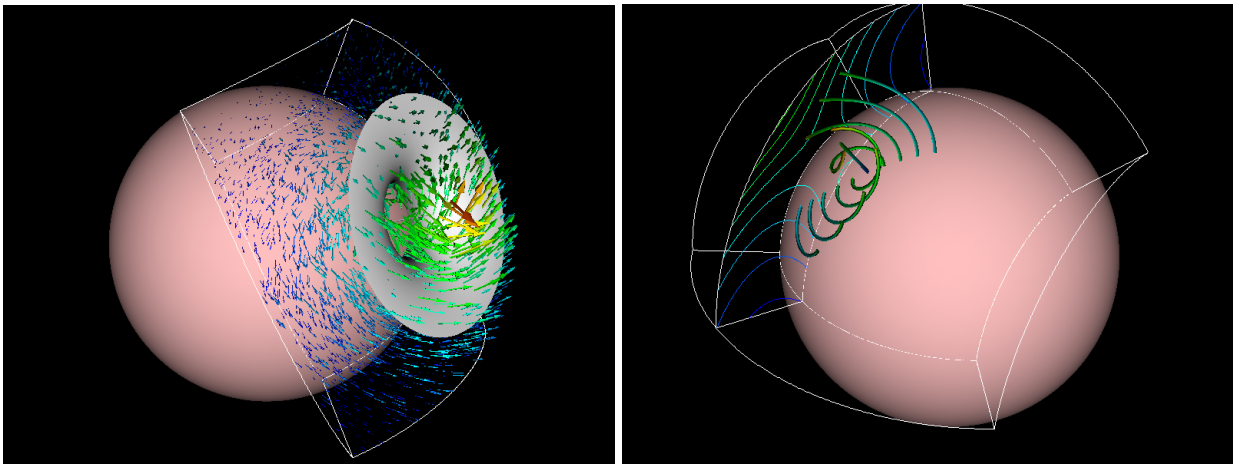

Figure 8.6: (左) 矢印 と 等値面による可視化，(右) 等高線と流線 (チューブ) による可視化

8.4　VTK 形式のデータ

Structured Grid のデータを VTK 形式にするときのヘッダを紹介する。Rectilinear Grid の座標を格子点の座標に変えるだけである。

```
1  # vtk DataFile Version 2.0
2  Structured Gridのデータ
3  BINARY
4  DATASET STRUCTURED_GRID
5  DIMENSIONS 50 100 50
```

```
 6  POINTS 250000 float
 7  格子点のバイナリデータ
 8  POINT_DATA 250000
 9  SCALARS NameOfData_S float 1
10  LOOKUP_TABLE default
11  スカラーのバイナリデータ
12  VECTORS NameOfData_V float
13  ベクトルのバイナリデータ
```

このようにして作成したデータは，vtkStructuredGridReader で読み込むことができる。

8.5 この章のまとめ

- vtkStructuredGrid を使っての円柱・球座標データの作成と可視化
- ポリゴンスライスによる球座標データのボリュームレンダリング
- vtkStructuredGrid の VTK 形式のデータ

本章で紹介したクラスは，Table 8.1 の通り。

Table 8.1: 本章で紹介したクラスおよびそのメソッド

データセット	vtkStructuredGrid : SetPoint(vtkDataArray *)
リーダー	vtkStructuredGridReader
カメラ関係	vtkCamera : ComputeViewPlaneNormal() : GetViewPlaneNormal(float [3])
断面	vtkCutter : GenerateValues(int, float, float)
外枠	vtkStructuredGridOutlineFilter

Chapter 9

その他の事項

9.1 本章の概要

可視化手法以外で，知っていると便利なことなどなどを紹介する。

9.2 コマンドの組み込み

vtkCommand クラスを使うと，インタラクティブ Window に，自分で定義したコマンドを組み込むことができます (キーボードの「U」ボタンがユーザー定義コマンド用に用意されています)。例えば,「U」ボタンを押すと，表示されている画像を BMP ファイルとして保存するなどできるようになります。また，等値面を表示するプログラムで，インタラクティブに等値面レベルの変更ができるようになると，非常に便利ではないだろうか。vtkCommand は，このようなことを可能にする。

9.2.1 サンプルプログラム 1

これは，第 3 章のサンプルプログラム “ExtractData.cxx”を改造したものである。

```
 1  /* ExtractData_k.cxx */
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkExtractVOI.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkContourFilter.h>
 9  #include <vtkOutlineFilter.h>
10  #include <vtkAppendPolyData.h>
11  #include <vtkPolyDataMapper.h>
12  #include <vtkRenderWindow.h>
13  #include <vtkActor.h>
14  #include <vtkRenderer.h>
15  #include <vtkProperty.h>
16  #include <vtkCommand.h>
17  #include <vtkRendererSource.h>
```

```
18  #include <vtkBMPWriter.h>
19  #include <vtkRenderWindowInteractor.h>
20  #include <vtkInteractorStyleTrackballCamera.h>
21
22  vtkExtractVOI *voi;
23  vtkContourFilter *contour;
24  vtkRenderer *vren;
25  vtkRenderWindowInteractor *iwin;
26
27  int extent_vol[6];
28  float range[2];
29  float iso_value;
30
31  class OurCommand:public vtkCommand
32  {
33  public:
34    static OurCommand *New ()
35    {
36      return new OurCommand;
37    }
38    virtual void Execute (vtkObject * caller, unsigned long, void *)
39    {
40      /* 「U」が押されると実行される */
41      static int num = 0;
42      char FileName[128];
43      char c;
44
45      vtkRendererSource *rs = vtkRendererSource::New ();
46      vtkBMPWriter *bw = vtkBMPWriter::New ();
47
48      printf ("Command\n");
49      printf ("[s]napshot\n");
50      printf ("e[x]tract data\n");
51      printf ("[i]sosurface value :: ");
52
53      scanf ("%c", &c);
54      fflush (stdin);
55
56      switch (c)
57        {
58        case 's':
59          /* 表示されている画像を BMP ファイルとして保存 */
60          rs->SetInput (vren);
61          rs->WholeWindowOn ();
62          rs->Modified ();
63
64          bw->SetInput (rs->GetOutput ());
```

```
 65
 66          sprintf (FileName, "shot%d.bmp", num);
 67          bw->SetFileName (FileName);
 68          bw->Write ();
 69
 70          num++;
 71          break;
 72
 73        case 'x':
 74          /* データの一部を切り出す */
 75          puts ("Input i1 i2 j1 j2 k1 k2");
 76          scanf ("%d %d %d %d %d %d", &extent_vol[0], &extent_vol[1],
 77                 &extent_vol[2], &extent_vol[3], &extent_vol[4],
 78                 &extent_vol[5]);
 79
 80          printf ("[%d - %d] [%d - %d] [%d - %d]\n", extent_vol[0],
 81                  extent_vol[1], extent_vol[2], extent_vol[3],
 82                  extent_vol[4], extent_vol[5]);
 83
 84          voi->SetVOI (extent_vol);
 85          fflush (stdin);
 86          break;
 87
 88        case 'i':
 89          /* 等値面・等高線のレベルを変える */
 90          printf ("Input Isosurface value (%f - %f) :: ",
 91                  range[0], range[1]);
 92          scanf ("%f", &iso_value);
 93
 94          printf ("value = %f\n", iso_value);
 95
 96          contour->SetValue (0, iso_value);
 97          fflush (stdin);
 98          break;
 99
100        default:
101          puts ("No Command");
102          break;
103        }
104
105      iwin->Render();
106      rs->Delete ();
107      bw->Delete ();
108
109    }
110
111  };
```

```
112
113  int main (int argc, char *argv[])
114  {
115     char datafile[] = "./plasma_data.vtk";
116
117      vtkStructuredPointsReader *reader
118         = vtkStructuredPointsReader::New();
119       reader->SetFileName(datafile);
120
121      vtkImageData *imgData;
122       imgData = reader->GetOutput();
123       imgData->Update ();
124       imgData->GetScalarRange (range);
125
126      vtkLookupTable *lut = vtkLookupTable::New ();
127       lut->SetHueRange (0.7, 0.0);
128       lut->Build ();
129
130       voi = vtkExtractVOI::New ();
131       voi->SetInput (imgData);
132       voi->SetVOI (0, 255, 0, 127, 0, 127);
133
134      contour = vtkContourFilter::New ();
135      contour->SetInput (voi->GetOutput ());
136      contour->SetValue (0, 10);
137      contour->ComputeNormalsOn ();
138
139     vtkPolyDataMapper *CMapper = vtkPolyDataMapper::New ();
140      CMapper->SetInput (contour->GetOutput ());
141      CMapper->SetLookupTable (lut);
142      CMapper->SetColorModeToMapScalars ();
143      CMapper->SetScalarRange (range[0], range[1]);
144
145     vtkActor *CActor = vtkActor::New ();
146      CActor->SetMapper (CMapper);
147
148     vtkOutlineFilter *outline = vtkOutlineFilter::New ();
149      outline->SetInput (imgData);
150
151     vtkOutlineFilter *outline_voi = vtkOutlineFilter::New ();
152      outline_voi->SetInput (voi->GetOutput ());
153
154     vtkAppendPolyData *outlines = vtkAppendPolyData::New ();
155      outlines->AddInput (outline->GetOutput ());
156      outlines->AddInput (outline_voi->GetOutput ());
157
158     vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New ();
```

```
159      OLMapper->SetInput (outlines->GetOutput ());
160
161     vtkActor *OLActor = vtkActor::New ();
162      OLActor->SetMapper (OLMapper);
163
164      vren = vtkRenderer::New ();
165      vren->AddActor (CActor);
166      vren->AddActor (OLActor);
167      vren->SetBackground (0.0, 0.0, 0.0);
168
169     vtkRenderWindow *renWin = vtkRenderWindow::New ();
170      renWin->AddRenderer (vren);
171      renWin->SetSize (500, 375);
172      renWin->LineSmoothingOn();
173
174      iwin = vtkRenderWindowInteractor::New ();
175      iwin->SetRenderWindow (renWin);
176
177     /* 定義したコマンドのクラスを生成して，iwin に登録 */
178     OurCommand *oc = OurCommand::New ();
179      iwin->AddObserver (vtkCommand::UserEvent, oc);
180     oc->Delete ();
181
182     vtkInteractorStyleTrackballCamera *trackball =
183       vtkInteractorStyleTrackballCamera::New ();
184
185     iwin->SetInteractorStyle (trackball);
186     iwin->Initialize ();
187     iwin->Start ();
188
189     reader->Delete ();
190     lut->Delete ();
191     outline->Delete ();
192     CActor->Delete ();
193     OLActor->Delete ();
194     CMapper->Delete ();
195     OLMapper->Delete ();
196     voi->Delete ();
197     contour->Delete ();
198     outlines->Delete ();
199     vren->Delete ();
200     renWin->Delete ();
201     trackball->Delete ();
202     iwin->Delete ();
203
204    return 0;
```

```
205  }
```

9.2.2 コマンド組み込みの方法

コマンドを組み込むには，vtkCommand を継承したクラスを自作することが必要である。といっても，難しいことはなく，下記の **10** 行のところに,「U」が押されたら実行させたい処理を書くだけである。もちろん，クラス名は OurCommand でなくても構わない。

```
 1  class OurCommand : public vtkCommand
 2  {
 3  public:
 4    static OurCommand *New ()
 5    {
 6      return new OurCommand;
 7    }
 8    virtual void Execute (vtkObject * caller, unsigned long, void *)
 9    {
10     /* ここに，実行させたい処理を C/C++で書く */
11    }
12  };
```

このようにして作ったクラスをサンプルプログラムの **177 - 180** 行のようにして，vtkRedner-WindowInteractor に登録すると,「U」ボタンを押されたときに，上記 **10** 行のところに書いた処理が実行される。

サンプルプログラム 1 では,「U」ボタンが押されると，ターミナルで 3 種類のコマンド (“s”, “x”, “i”) の入力待ち (scanf を使ってます) になり，“s”でスナップショット，“x”でデータ切り出し (さらに切り出す範囲を入力する)，“i”で等値面のレベル変更 (レベルをさらに入力する) ができるようにしてある (Figure 9.1)。

これを使えば，かなり高機能なインタラクティブ可視化ソフトを自作できる (はず)。(bf 105 行の iwin->Render() は，再描画の指示を Window に送る。これをいれないとマウスで Window をなぞるまで等値面の再計算などは行われない)

9.3 時間発展のデータ

いままでは，一つのデータだけを使い可視化してきた (アニメーションを作るときも)。何も工夫しなくても，時間発展のデータを扱うことができる。

9.3.1 サンプルプログラム 2:Tiny MovieMaker

このサンプルプログラムは，第 4 章のテクスチャ・マッピングを利用したボリューム・レンダリングのプログラム (VolumeRendering_2d.cxx) を改造したものである。実行すると，カメラアングル (Azimuth 方向に 3 度ずつ) を変えつつデータも変えていく (時間発展させる)。同時に，データ 1 つにつき，3 枚の画像を保存するようにしてある。サンプルデータは，同じく，地球磁気圏の温度で，本章では 10 個用意した。

```
1  /* tinymm.cxx */
```

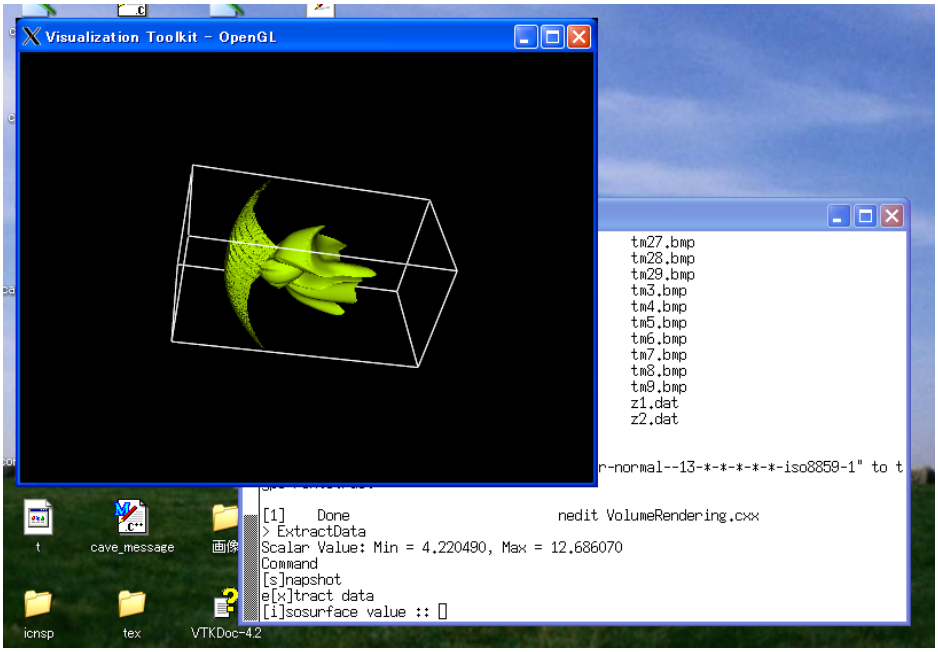

Figure 9.1: コマンド待ちの様子

```
 2  #include <vtkImageData.h>
 3  #include <vtkStructuredPoints.h>
 4  #include <vtkStructuredPointsReader.h>
 5  #include <vtkPointData.h>
 6  #include <vtkSphereSource.h>
 7  #include <vtkLookupTable.h>
 8  #include <vtkOutlineFilter.h>
 9  #include <vtkImageShiftScale.h>
10  #include <vtkPiecewiseFunction.h>
11  #include <vtkColorTransferFunction.h>
12  #include <vtkVolumeProperty.h>
13  #include <vtkVolumeTextureMapper2D.h>
14  #include <vtkVolume.h>
15  #include <vtkPolyData.h>
16  #include <vtkPolyDataMapper.h>
17  #include <vtkRenderWindow.h>
18  #include <vtkActor.h>
19  #include <vtkRenderer.h>
20  #include <vtkProperty.h>
21  #include <vtkCamera.h>
22  #include <vtkScalarBarActor.h>
23  #include <vtkVectorText.h>
24  #include <vtkFollower.h>
25  #include <vtkRendererSource.h>
26  #include <vtkBMPWriter.h>
27
28  #define CAMERA_STEP 30
```

```
29  #define DATA_NUM    10
30
31  float range[2] = { 4.2, 13.3 };
32
33  int
34  main (int argc, char *argv[])
35  {
36     int i, file_num = 0, pic_num = 0;
37     char datafile[128];
38     float shift, scale;
39     float opacity_table[256];
40     float color_table[3 * 256], temp_color[4];
41     float earth_position[3] = { 100.5, 63.5, 63.5 };
42     char FileName[128];
43
44    sprintf (datafile, "./tempe%d.vtk", 200);
45
46     vtkStructuredPointsReader *reader
47         = vtkStructuredPointsReader::New();
48      reader->SetFileName(datafile);
49
50     vtkImageData *imgData;
51      imgData = reader->GetOutput();
52      imgData->SetSpacing (1.0, 1.0, 1.0);
53
54      shift = -range[0];
55      scale = 255.0 / (range[1] - range[0]);
56
57     vtkImageShiftScale *f2uc = vtkImageShiftScale::New ();
58      f2uc->SetShift (shift);
59      f2uc->SetScale (scale);
60      f2uc->SetOutputScalarTypeToUnsignedChar ();
61      f2uc->SetInput (imgData);
62
63     vtkLookupTable *lut = vtkLookupTable::New ();
64      lut->SetHueRange (0.7, 0.0);
65      lut->SetNumberOfTableValues (256);
66      lut->SetRange (range);
67      lut->Build ();
68
69     for (i = 0; i < 256; i++)
70       {
71         lut->GetTableValue (i, temp_color);
72         color_table[i * 3] = temp_color[0];
73         color_table[i * 3 + 1] = temp_color[1];
74         color_table[i * 3 + 2] = temp_color[2];
75       }
```

```
 76
 77     vtkColorTransferFunction *tf4color
 78         = vtkColorTransferFunction::New ();
 79      tf4color->SetColorSpaceToRGB ();
 80      tf4color->BuildFunctionFromTable (
 81                0, 255, 256, color_table);
 82
 83     for (i = 0; i < 256; i++)
 84       {
 85         if (i > 187)
 86           opacity_table[i] = i / 255.0 * 0.025;
 87         else
 88           opacity_table[i] = 0.0;
 89       }
 90
 91     vtkPiecewiseFunction *tf4opacity
 92             = vtkPiecewiseFunction::New ();
 93      tf4opacity->BuildFunctionFromTable (
 94                 0, 255, 256, opacity_table, 1);
 95
 96     vtkVolumeProperty *vp = vtkVolumeProperty::New ();
 97      vp->SetColor (tf4color);
 98      vp->SetScalarOpacity (tf4opacity);
 99      vp->SetInterpolationTypeToLinear ();
100
101     vtkVolumeTextureMapper2D *vMapper
102          = vtkVolumeTextureMapper2D::New ();
103      vMapper->SetInput (f2uc->GetOutput ());
104      vMapper->SetMaximumNumberOfPlanes (256);
105      vMapper->SetTargetTextureSize (256, 128);
106
107     vtkVolume *Volume = vtkVolume::New ();
108      Volume->SetMapper (vMapper);
109      Volume->SetProperty (vp);
110
111     vtkOutlineFilter *outline = vtkOutlineFilter::New ();
112      outline->SetInput (imgData);
113
114     vtkPolyDataMapper *OLMapper = vtkPolyDataMapper::New ();
115      OLMapper->SetInput (outline->GetOutput ());
116
117     vtkActor *OLActor = vtkActor::New ();
118      OLActor->SetMapper (OLMapper);
119
120     vtkSphereSource *earth = vtkSphereSource::New ();
121      earth->SetRadius (3.3);
122
```

```
123     vtkPolyDataMapper *EMapper = vtkPolyDataMapper::New ();
124      EMapper->SetInput (earth->GetOutput ());
125
126     vtkActor *EActor = vtkActor::New ();
127      EActor->SetMapper (EMapper);
128      EActor->SetPosition (earth_position);
129      EActor->GetProperty ()->SetColor (0.0, 0.0, 1.0);
130
131     vtkVectorText *EText = vtkVectorText::New ();
132      EText->SetText ("   EARTH");
133
134     vtkPolyDataMapper *TMapper = vtkPolyDataMapper::New ();
135      TMapper->SetInput (EText->GetOutput ());
136
137     vtkFollower *TActor = vtkFollower::New ();
138      TActor->SetMapper (TMapper);
139      TActor->SetScale (5.0, 5.0, 5.0);
140      TActor->AddPosition (earth_position);
141
142     vtkScalarBarActor *scalarbar = vtkScalarBarActor::New ();
143      scalarbar->SetTitle ("Temperature");
144      scalarbar->SetLookupTable (lut);
145      scalarbar->SetOrientationToVertical ();
146      scalarbar->SetWidth (0.075);
147      scalarbar->SetHeight (0.75);
148
149     vtkCamera *camera = vtkCamera::New ();
150      camera->SetPosition (-200.0, -200.0, 237.5);
151      camera->SetFocalPoint (127.5, 63.5, 63.5);
152      camera->SetViewUp (0.0, 0.0, 1.0);
153      camera->OrthogonalizeViewUp ();
154      camera->SetClippingRange (30.0, 2000.0);
155
156     vtkRenderer *vren = vtkRenderer::New ();
157      vren->AddVolume (Volume);
158      vren->AddActor (EActor);
159      vren->AddActor (OLActor);
160      vren->AddActor (TActor);
161      vren->AddActor (scalarbar);
162      vren->SetActiveCamera (camera);
163      vren->SetBackground (0.0, 0.0, 0.0);
164
165      TActor->SetCamera (vren->GetActiveCamera ());
166
167     vtkRenderWindow *renWin = vtkRenderWindow::New ();
168      renWin->AddRenderer (vren);
169      renWin->SetSize (500, 375);
```

```
170
171     vtkRendererSource *rs = vtkRendererSource::New ();
172     vtkBMPWriter *bw = vtkBMPWriter::New ();
173
174     for (i = 0; i < CAMERA_STEP; i++)
175       {
176
177         if (i % (CAMERA_STEP / DATA_NUM) == 0 && i != 0)
178           {
179            /* 次のデータを読み込む */
180            file_num++;
181
182            sprintf (datafile, "./tempe%d.vtk", 200 + file_num * 5);
183
184            reader->SetFileName(datafile);
185            reader->Update();
186
187           printf ("Data Read %d\n", 200 + file_num * 5);
188
189           }
190
191      renWin->Render ();
192      rs->SetInput (vren);
193      rs->WholeWindowOn ();
194      rs->Modified ();
195
196     sprintf (FileName, "tm%d.bmp", pic_num);
197      bw->SetInput (rs->GetOutput ());
198      bw->SetFileName (FileName);
199      bw->Write ();
200
201      pic_num++;
202
203      camera->Azimuth (3.0); /* カメラ位置を3度移動 */
204
205     }
206
207     reader->Delete ();
208     f2uc->Delete ();
209     lut->Delete ();
210     tf4color->Delete ();
211     tf4opacity->Delete ();
212     vp->Delete ();
213     vMapper->Delete ();
214     Volume->Delete ();
215     outline->Delete ();
216     OLMapper->Delete ();
```

```
217     OLActor->Delete ();
218     earth->Delete ();
219     EMapper->Delete ();
220     EActor->Delete ();
221     EText->Delete ();
222     TMapper->Delete ();
223     TActor->Delete ();
224     scalarbar->Delete ();
225     vren->Delete ();
226     camera->Delete ();
227     renWin->Delete ();
228     rs->Delete ();
229     bw->Delete ();
230
231    return 0;
232  }
```

9.3.2 サンプルプログラム 2 の簡単な解説

177 - 189 行で，データを更新しているだけで，なにも新しいことはない (その他，(時系列すべてを通した) スカラー値の最大値･最小値も予め与えてある)。これだけのプログラムで，MovieMaker[3] の真似事をさせることができた。表示は，Figure 9.2 (**185** 行を忘れずに。手動でデータを作る

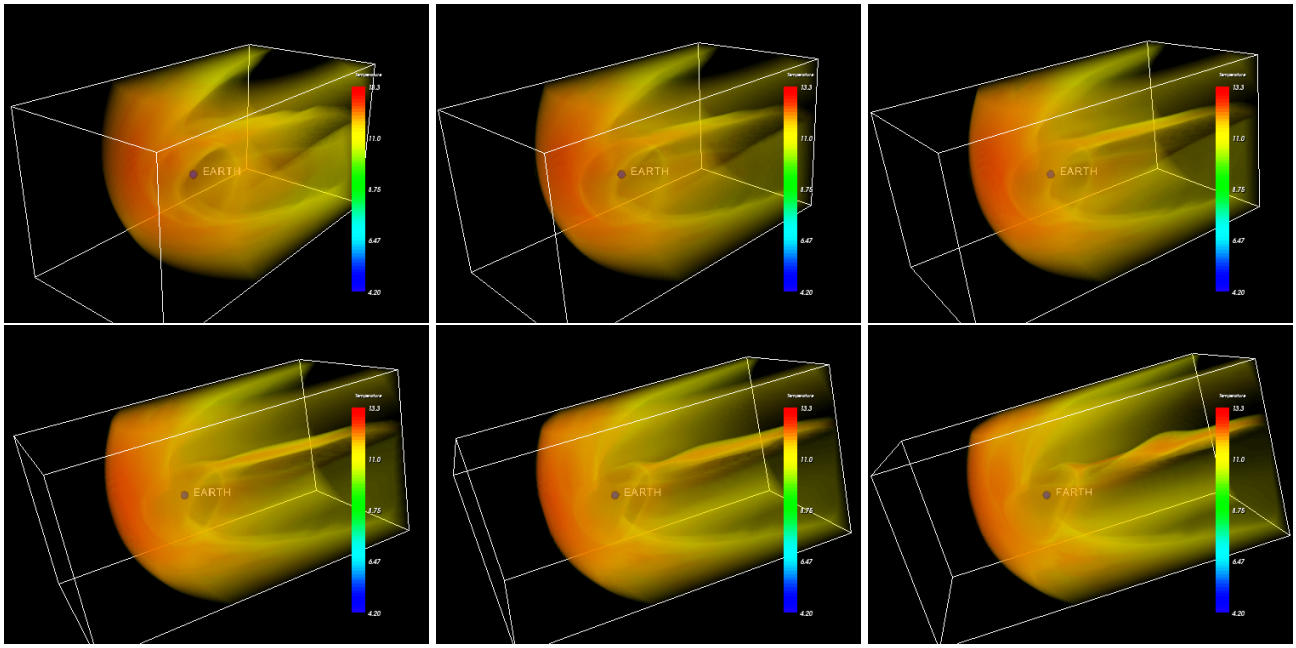

Figure 9.2: 時間発展するデータの可視化

ときは，例えば farray(vtkFloatArray) にデータを入れているのであれば，データを読み込んだあとに，farray->Modified() を実行して，VTK にデータが変化したことを知らせなければならない)

9.4 ポリゴンデータ

可視化像と同時に線などを表示したい場合がある (例えば，気圧の等高線と同時に，低気圧の (これまでの) 進路などを表示させる)。矢印や円柱など，もともと VTK に用意されているものについては，それを使えばよいが，用意されていない場合，vtkPolyData を利用し自作することもできる。

9.4.1 ポリゴンデータの作成

点・連結した線分・ポリゴン・連結した三角形のデータ (Figure 9.3) を格納できる。これらは，OpenGL の GL_POINTS, GL_LINE_STRIP, GL_POLYGON, GL_TRIANGLE_STRIP に相当する。

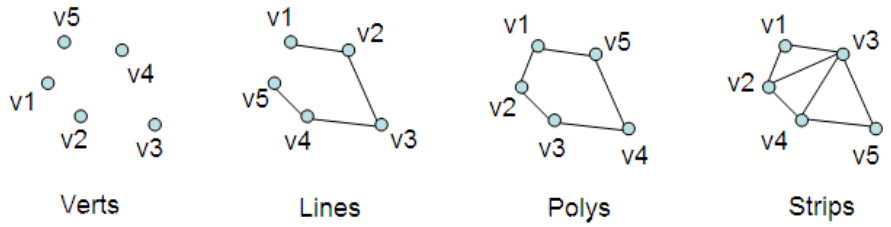

Figure 9.3: ポリゴンの種類

次のサンプルプログラムは，ポリゴンデータ (vtkPolyData) をマニュアル作成する例である。

```
 1  /* pyramid.cxx */
 2  #include <vtkCellArray.h>
 3  #include <vtkFloatArray.h>
 4  #include <vtkPolyData.h>
 5  #include <vtkLookupTable.h>
 6  #include <vtkPoints.h>
 7  #include <vtkPointData.h>
 8  #include <vtkPolyDataMapper.h>
 9  #include <vtkActor.h>
10  #include <vtkRenderer.h>
11  #include <vtkRenderWindow.h>
12  #include <vtkRenderWindowInteractor.h>
13  #include <vtkInteractorStyleTrackballCamera.h>
14
15  const int n_points = 9;
16  const int n_polys  = 5;
17  const int n_lines  = 6;
18  const int dp = 8; /* dummy point id */
19
20  int main( int argc, char *argv[] )
21  {
22    int i;
23
24    float vpoints[n_points][3]={
```

```
25          {-2.5, -2.5, -2.5}, { 2.5, -2.5, -2.5},
26          { 2.5,  2.5, -2.5}, {-2.5,  2.5, -2.5},
27          {-2.5, -2.5,  2.5}, { 2.5, -2.5,  2.5},
28          { 2.5,  2.5,  2.5}, {-2.5,  2.5,  2.5},
29          { 0.0,  0.0,  0.0}};
30
31      int poly_nverts[n_polys] = {3, 3, 3, 3, 4};
32      vtkIdType  poly_pts[n_polys][4] = {
33          {8, 0, 1, dp}, {8, 1, 2, dp},
34          {8, 2, 3, dp}, {8, 3, 0, dp},
35          {0, 3, 2, 1}};
36
37      int line_nverts[n_lines] = {5, 5, 2, 2, 2, 2};
38      vtkIdType  line_pts[n_lines][5]={
39          {0, 1,  2,  3,  0}, {4, 5,  6,  7,  4},
40          {0, 4, dp, dp, dp}, {1, 5, dp, dp, dp},
41          {2, 6, dp, dp, dp}, {3, 7, dp, dp, dp}};
42
43
44      float scalars[n_points] = {
45       5.0, 0.0, 5.0, 0.0, 7.5, 7.5, 7.5, 7.5, 10.0
46      };
47
48
49      vtkPoints *points = vtkPoints::New();
50       for (i=0;i<n_points;i++)
51         points->InsertPoint(i, vpoints[i]);
52
53      vtkCellArray *polys = vtkCellArray::New();
54       for (i=0;i<n_polys;i++)
55         polys->InsertNextCell(poly_nverts[i], poly_pts[i]);
56
57      vtkCellArray *lines = vtkCellArray::New();
58       for (i=0; i<n_lines;i++)
59        lines->InsertNextCell(line_nverts[i], line_pts[i]);
60
61      vtkFloatArray *farray = vtkFloatArray::New();
62       farray->SetArray(scalars, n_points, 1);
63
64      vtkPolyData *pyramid = vtkPolyData::New();
65       pyramid->SetPoints(points);
66       pyramid->SetPolys(polys);
67       pyramid->SetLines(lines);
68       pyramid->GetPointData()->SetScalars(farray);
69
70      vtkLookupTable *lut = vtkLookupTable::New();
71       lut->SetHueRange(0.7, 0.0);
```

```
 72     lut->Build();
 73
 74    vtkPolyDataMapper *Mapper = vtkPolyDataMapper::New();
 75     Mapper->SetInput(pyramid);
 76     Mapper->SetLookupTable(lut);
 77     Mapper->SetScalarRange(0.0, 10.0);
 78
 79    vtkActor *Actor = vtkActor::New();
 80     Actor->SetMapper(Mapper);
 81
 82    vtkRenderer *ren = vtkRenderer::New();
 83     ren->AddActor(Actor);
 84     ren->SetBackground(0,0,0);
 85
 86    vtkRenderWindow *renWin = vtkRenderWindow::New();
 87     renWin->AddRenderer(ren);
 88     renWin->SetSize(500, 500);
 89
 90    vtkInteractorStyleTrackballCamera *trackball =
 91     vtkInteractorStyleTrackballCamera::New();
 92
 93    vtkRenderWindowInteractor *iwin
 94      = vtkRenderWindowInteractor::New();
 95     iwin->SetRenderWindow(renWin);
 96     iwin->SetInteractorStyle(trackball);
 97
 98     iwin->Initialize();
 99     iwin->Start();
100
101     points->Delete();
102     polys->Delete();
103     lines->Delete();
104     farray->Delete();
105     pyramid->Delete();
106     lut->Delete();
107     Mapper->Delete();
108     Actor->Delete();
109     ren->Delete();
110     renWin->Delete();
111     trackball->Delete();
112     iwin->Delete();
113
114    return 0;
115  }
```

このサンプルプログラムの実行結果は，Figure 9.4 である
ポリゴンデータを作成するには，以下 (の作業) が必要である。

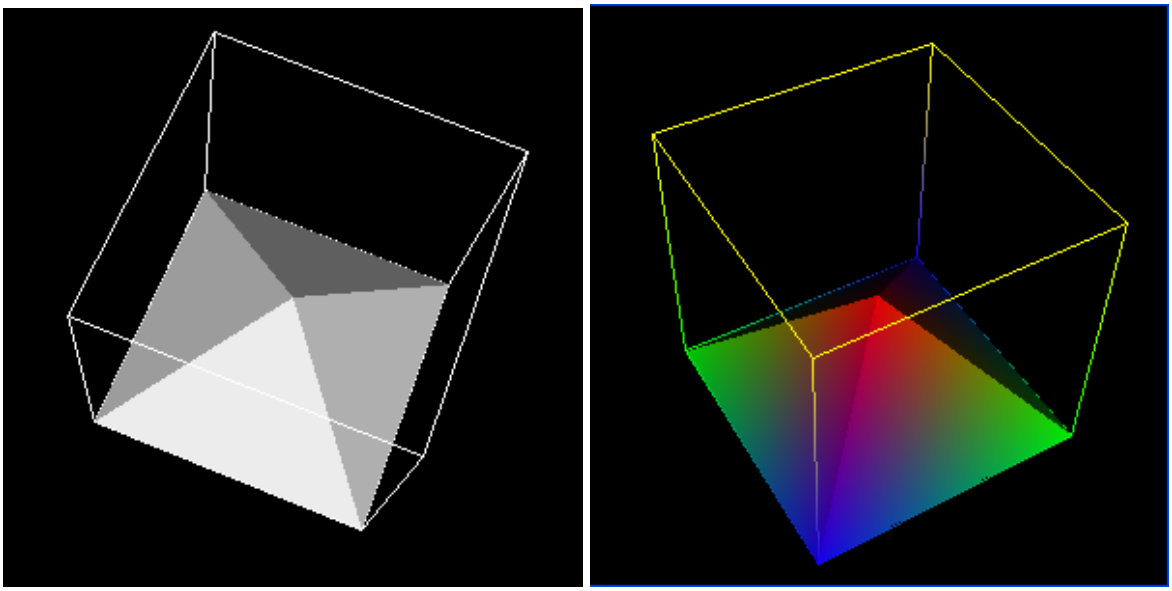

Figure 9.4: マニュアル作成したデータ ((左)ScalarVisibilityOff())

1. 頂点のデータ
2. ポリゴンを構成する頂点の結び方の指定 (4 種類ある:Figure 9.3 参照)
3. 頂点に付随するスカラー，ベクトル，法線ベクトル (なくとも良い)

以下，順に見ていく。
1 : 本サンプルプログラムでは，頂点のデータの設定は，**24 - 29**, **49 - 51**, **65** 行でおこなっている。vtkPoints を介して，vtkPolyData に代入している。これは，流線の出発点の設定のところでもおこなっている。こうして格納した頂点のデータは，Figure 9.5 のように，代入した順番に番号がふられる。
2 : 上で格納した点をどのように結びポリゴンを作るのか，を指定する。例えば，0 番と 4 番を結び線を引く，0 番･1 番･8 番の頂点を結び 3 角形のポリゴンを作る，など。サンプルプログラムでの関係箇所は，**31- 41**, **53 - 59**, **66 - 67** 行。ポリゴンのデータ (頂点の結び方) を作るには，vtkCellArray を使用する。vtkCellArray のデータ構造は，$[n_0, p_{id_{00}}, p_{id_{01}}, p_{id_{02}}..., n_1, p_{id_{10}}, p_{id_{11}}, p_{id_{12}}, ..]$ のような整数の羅列である (n_x, p_{id_y}... は，すべて整数)。n はセルの配列の数 (ポリゴンの頂点の数)，p_{id} は対応する頂点の id。上の例で，「0 番と 4 番を結び線を引き」たい場合は $[2, 0, 4]$，「0 番・1 番・8 番の頂点を結び 3 角形のポリゴンを作」りたい場合は，$[3, 0, 1, 8]$ という整数が入った vtkCellArray を作ればよい。**53 - 59** 行で，この作業を行っている。InsertNextCell の一つ目の引数が n であり，二つ目の引数は id が入った配列のポインタである。

このようにして作成した vtkCellArray を **66 - 67** 行で，vtkPolyData に代入している。代入するときに，ポリゴンの種類によって，SetVerts, SetLines, SetPolys, SetStrips を使い分ける。

サンプルプログラムでは，8・0・1，8・1・2，8・2・3，8・3・0 を結び三角形を，0・1・2・3(Figure 9.5) を結び四角形のポリゴンをを作っている。また，このポリゴンを囲むように線を引いている。
3 : 頂点に付随するスカラーやベクトル，法線ベクトルもセットできる。サンプルプログラムでは，スカラーをセットしている。**44 - 46**, **61 - 62**, **68** 行でこの作業を行っている。方法は，vtkImageData や vtkRectilinearGrid などの場合と，まったく同様である [1]。

[1]法線ベクトルの場合は，ベクトルの大きさを 1.0 にしなければならない。

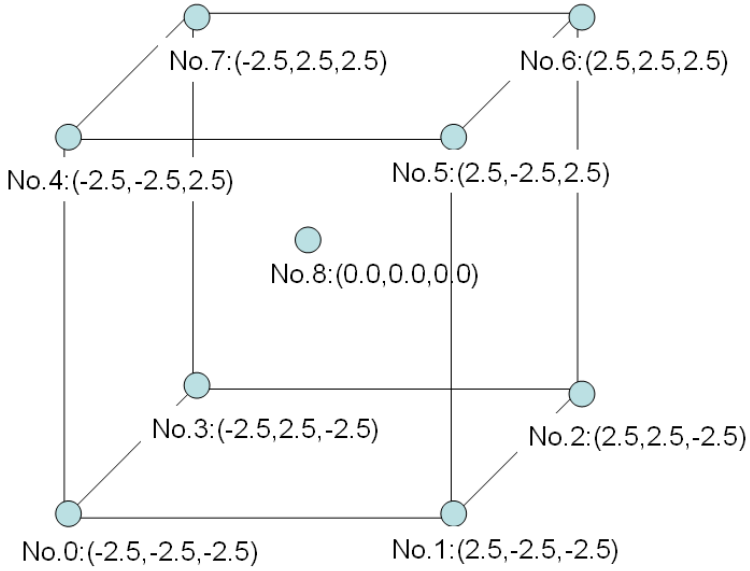

Figure 9.5: 頂点と頂点の番号

このようにして作成したポリゴンデータは，可視化のためのフィルタからの出力のポリゴンデータと同様に vtkPolyDataMapper を通して表示できる。

9.5 アンチ・エイリアジング

サンプルプログラム 1 の **172** 行目のように，...SmoothingOn() とするとアンチ・エイリアジングがかかり，画像が少し綺麗になる。点・線・ポリゴンについて，それぞれ下記のメソッドが vtkRenderWindow に用意されている。

- PointSmoothingOn();
- LineSmoothingOn();
- PolygonSmoothingOn();

あとがき

VTK の可視化ライブラリを使えば，可視化アルゴリズムも OpenGL もほとんど知らなくとも，かなり良くできた 3 次元可視化ツールを開発することができます (実際，本章では MovieMaker の真似事をやりました)。これまでの内容を基に，自分だけの可視化ツールを開発してください。健闘を祈ります。なお，本勉強会で使用したサンプルプログラムの作成に当たり，Vtisualization Toolkit (3rd Edition)， VTK User's Guide や附属 CD に収録されているサンプルプログラム (主に Tcl) を参考にしました。

長い間，おつき合いありがとうございました。

Acknowledgments

サンプルデータとして使用した球状トカマクの圧力データは，核融合科学研究所 理論・データ解析センター 水口直紀博士から提供していただいたシミュレーションデータを加工したものです。また，地球磁気圏の温度データは，名古屋大学 太陽環境研究所 荻野龍樹教授から提供していただいたシミュレーションデータを加工したものです。

参考文献

(1) Will Schroeder, Ken Martin and Bill Lorensen: Visualization Toolkit(3rd Edition), Kitware (2002)
(2) VTK User's Guide, Kitware (2003)
(3) H. UEHARA, S. KAWAHARA, N. OHNO, M. FURUICHI, F. ARAKI and A. KAGEYAMA: "MovieMaker: A Parallel Movie-Making Software for Large Scale Simulations", J. Plasma Physics Vol.72, part 6, pp.841-844 (2006)