POV-Ray によるボリュームレンダリング表示と GUI フロントエンド Oosawa

○武田隆顕 [1],

[1] 国立天文台 天文シミュレーションプロジェクト

概要

シミュレーションデータの可視化には、データの形式とその目的に応じてさまざまな手段や方法がある。ここでは主に「見せる」ための可視化について記述する。前半で、フリーウェアのレイトレーサーである POV-Ray を用いて陰影などを含めてボリュームデータを可視化する方法について簡単に記述する。後半では現在開発中のフロントエンドツール「Oosawa」について記述する。Oosawa は、GUI を用いてアニメーションの設定といった操作を行い、POV-Ray でボリュームデータのレンダリングを行なって、綺麗な可視化映像を作成をすることを目的としたツールになっている。

1 序

シミュレーションデータの可視化には一般的に大きく分けて２つの目的が存在する。一つは研究者自身が直感的に把握しやすい形で計算結果を見るために行われる、研究の補助ツールとしての可視化である。もう一つは、研究の結果を単なる点や線による図やグラフではなく、専門外の人間に対しても理解しやすく印象的な画像、映像を作成するための可視化、つまり「見せる」ための可視化である。非専門家に興味を喚起するという意味での「見せる」ための可視化には、映像や画像としてある程度凝ったものであることが望ましい。そうした映像の作成には、カメラワークの設定や影の導入、レンズフレアやモーションブラーといった効果の付加などが考えられる。ここで述べる可視化方法の記述は、スパコンなどで作られた巨大なデータなどを、デスクトップ環境で扱えるサイズまでリダクションして持ち帰り、自分のデスクトップ環境でカメラワークなどのついた「見せる」可視化映像を作る、といった作業を想定している。

POV-Ray[1] はレイトレースの技法を使って写実的な CG を作成するためのツールとして古典的に知られているツールである。POV-Ray は完全なフリーウェアであり、Windows/MacOS/Linux/UNIX など、多くのプラットフォームで利用できる。POV-Ray はテキストファイルによってシーンを記述するという特徴があるため、幾何学的形状など計算で記述することのできるシーンをスクリプトで簡単に作成することができ、そうした用途には広く使われているソフトウェアである。あまり知られていないが、Pov-Ray にはボリュームデータを読み込んでレンダリングをする機能があり、レイトレースされたボリュームデータの表示を行なうことができる。第２節では、POV-Ray でのシーン設定の概念や、ボリュームデータのフォーマットと、その表示法について簡単に述べる。

POV-Ray を用いてボリュームデータを表示することはできるが、テキストファイルでシーンを記述するという特性上、カメラワークの作成といった映像作成のための操作は通常は困難を伴う。そうした困難を解消するため、GUI によってカメラの操作や、等値面の閾値、断面図の位置といった各種パラメータの時間変化を設定することのできるフロントエンドツール「Oosawa」の開発を継続的に行なっている。第３節では Oosawa の設計の概念とその基本的な機能を記述する。

2　POV-Ray のシーン設定の概念と、ボリュームデータの表示

POV-Ray は、テキストファイルによってシーンを設計し、描画を行なう。POV-Ray におけるシーン設定の概念をつかむために、ボリュームデータを表示するために必要な最低限の設定について記述する。各パラメータの詳細などは、POV-Ray のマニュアルなどを参考にしてほしい。

まず、どこからの視線で絵を作るのか、カメラ設定を決める必要がある。また、物を見るために必要な光源の設定も必要になる。そして、ボリュームデータを表示するための箱が必要になる。それらを記述した POV-Ray のシーンファイル (.pov) の例としては次のようになる。

`camera {`	カメラの設定
`  location <3, 3, 3>`	位置
`  sky      <0, 0, 1>`	カメラを振るときの軸になる方向
`  look_at   <0, 0, 0>`	どこを見るか
`  angle    45`	カメラの視野角
`}`	
`light_source{`	光源の設定
`  <2, 3, 5>`	光源位置
`  color rgb 1.5 * <1, 1, 1>`	色の設定 (rgb)
`  parallel`	平行光線
`  point_at <0, 0, 0>`	原点向きに照射
`}`	
`box {`	箱を置く
`  <-1, -1, -1> <1, 1, 1>`	箱の両端の座標
`  pigment {`	箱の色設定
`    rgbt <1, 1, 1, 0>`	色の設定 (rgb+透明度)
`  }`	
`}`	

ここで、ファイルから密度情報を読み込むことになるが、読み取りとそれに伴う設定の部分を書き連ねると、少々冗長なので、密度ファイルを読み込みパラメータを設定する部分を `DensityProfile` という名前のマクロとして作成する。

`#macro DensityProfile ( VTranslate, VScale)`	#マクロ定義開始 (引数...)
`density {`	密度定義
`    density_file df3 "densitydata.df3"`	読み込みファイル名
`    interpolate 1`	補間方法 1: linear
`    scale VScale`	スケール設定
`    translate VTranslate`	位置のオフセット設定
`    color_map {`	カラーマップ
`      [0.0 rgb <0.2, 0.0, 0.0>]`	密度 0 で赤
`      [0.5 rgb <0.5, 0.5, 0.0>]`	密度 0.5 で黄
`      [1.0 rgb <1.0, 1.0, 1.0>]`	密度 1 で白
`    }`	
`}`	
`#end`	#マクロ定義終了

密度分布は、デフォルトでは `<0, 0, 0> - <1, 1, 1>` に配置される。今回は `<-1, -1, -1>, <1, 1, 1>` の箱に結果を入れたいので、スケールを 2 倍 `<2,2,2>` して、オフセット `<-1, -1, -1>` することでぴったりとはまることになる。こうした設定を、マクロへの引数で与えられるように作ってある。今作ったマクロを利用して `box` の内部を変更し、密度分布をもった立方体を作成する。

box {
  <-0.99, -0.99, -0.99> <0.99, 0.99, 0.99>
  hollow
  pigment {rgbt <1,1,1,0.95>}
  interior {
  media {
  DensityProfile (<-1, -1, -1>, <2, 2, 2>)
  intervals 1
  samples 32,64
  method 3
  emission rgb 0.3 * <1, 1, 1>
  absorption rgb 1.0 * <1, 1, 1>
  scattering { 1, rgb <1, 1, 1> }
  }
  }
}
	箱を置く
	箱の両端の座標
	中空の箱
	箱の表面の色（ほぼ透明）
	箱の中身設定
	中身のガスの設定
	マクロで密度読み込み
	レンダリング精度設定
	レンダリング精度設定
	レンダリング手法
	発光の設定
	吸収係数の設定
	散乱係数の設定（非常に重い）

こういったシーンファイルを用いてボリュームデータをファイルから読み込み、ボリュームレンダリングを行なうことができる。

2.1 データフォーマット

POV-Ray で読み込めるボリュームデータは、.df3 フォーマットである。このフォーマットは POV-Ray でボリュームデータをファイルから読み込むために作られた非常にシンプルなもので、ボリュームの大きさ記述したヘッダと、３重ループで記述されたボリューム本体部分からなる。

ヘッダ部分は、3 つの 16bit 整数からなる 6byte であり、xyz 方向のボクセル数を記述する。本体部分は、8bit、16bit 又は 32 bit の符号なし整数の羅列である。データの bit 数は、ファイルサイズから推定されるので、明示的に記述する必要は無い。

非常にシンプルなフォーマットであるが、注意点として整数のバイナリフォーマットが big-endian であることがあげられる。x86 系のシステムで.df3 ファイルを作る場合には、バイト順序の入れ替え用のルーチンが必要になる。

2.2 POV-Ray でのボリュームレンダリング

ボリュームデータを用いたレイトレーシングの例をあげる。充分に複雑で、見た目にも特徴的なボリュームデータとして、頭部の CT スキャンデータからつくられたボリュームデータを用いた。前節では、立方体の中にボリュームデータを配置する例のみを記述したが、POV-Ray では、複数の形状を組み合わせた表示も行なえる。図 1 にはデータの一部を切り取って中の露出したデータを表示した例を示す。

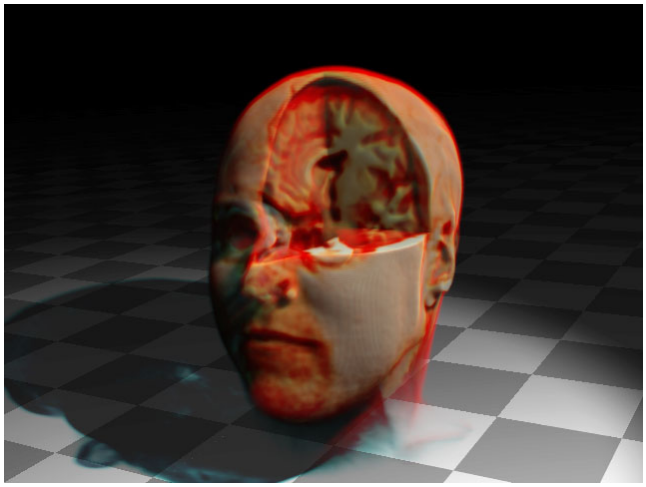

図 1: POV-Ray によるボリュームデータのレンダリング例。

この例では、ボリューム内で光の散乱を計算し、自分に落ちる影なども計算しているために、レンダリング時間はかなり長い。特に映像の作成を考えると、短時間でプレビューや試行錯誤を繰り返すことのできる仕組みがあることが非常に望ましい。

3　Oosawa による GUI での設定

POV-Ray は、テキストファイルでシーンを記述するという特性上、カメラワークの作成といった映像作成のための操作は通常は困難を伴う。そのためにグラフィックボードを用いてボリュームデータを表示して、GUI による設定などを行なうためのフロントエンドツール「Oosawa」の開発を行なっている。Oosawa の実行画面の例を図 2 に示す。Oosawa は時系列に従った連番ボリュームデータに対して、データの表示、等値面や断面図、流線の表示、何種類かの背景の表示、といった表示機能と、それを POV-Ray 用のシーンファイルとして出力する機能、そしてレベル補正やガウス暈しといった簡単なフィルター処理を実装している。Oosawa の実行ファイルとソースコードは、現在個人のホームページ [3](http://th.nao.ac.jp/ takedatk/COMPUTER/OOSAWA/oosawa.html) 上での公開を行なっている。

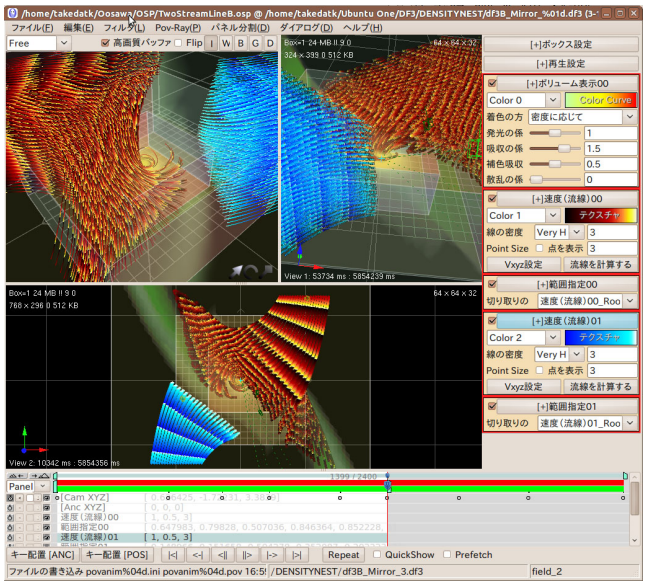

図 2: Oosawa 実行画面。ボリューム表示と多量の流線による速度場のアニメーション表示を行なっている。

Oosawa の基本コンセプトは、映像作成用のツールに近い操作性を持った可視化ツールであり、タイムライン（図 3）とその上でのキーフレーム操作によって、カメラの位置、等値面の閾値といったパラメータの時間依存性を制御している。

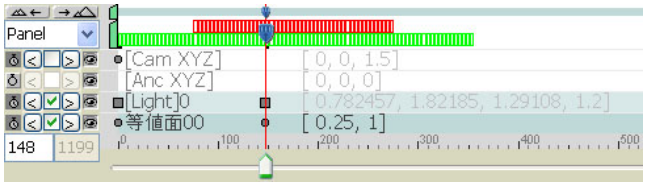

図 3: Oosawa のタイムライン。タイムライン上にキーフレームを配置していくことで、各種のパラメータの時間変化の制御をすることができる。こうしたタイムラインによる操作は、映像編集などでよく用いられる操作法である。

Oosawa では時系列方向の操作の応答を良くするために、使用したデータをメモリ上に保持し、メ

モリ使用量に応じて古いデータから除去するという処理を行なっている。タイムライン上に色で表示されている領域は、既にメモリ上にデータがある時間フレームを示している。

また、描画前に簡単なデータ処理を行なうためのフィルタ処理を実装している。フィルタは、上から順番にデータを処理して下に受け渡すという比較的シンプルなものである（図 4）。複数のデータを組み合わせて複雑なデータ処理をするといったことはできないが、単純で直感的な理解はしやすいものである。こうした 1 次元的なフィルタ処理の流れも、映像編集ソフトなどでよく使われる手法である。

図 4: Oosawa のフィルタ処理。上から下に順に処理をするシンプルな処理になっている。

図 5 は、図 4 のフィルタを使った表示の例である。比較的ノイジーなデータをレベル補正した後にボリュームで表示している。その後、ガウス暈しによって勾配のゆるい滑らかなデータに変換した後に等値面を作成している。

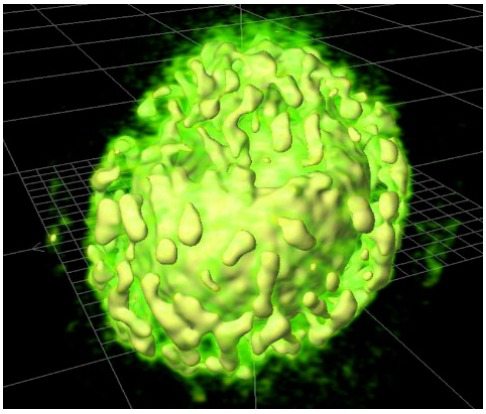

図 5: フィルタ処理をした描画の例。図 4 のフィルタ処理をすることにより、等値面はもとのデータに比べて滑らかな表示になっている。

3.1 大規模データへの対応

非常に大規模なシミュレーションデータは、デスクトップ環境などに持ち帰っての可視化作業にはあまり向いていない。しかし、8 分木ツリーやネスティッドグリッドといった階層化されたデータに一部対応することで、大規模データへのある程度の対応を可能にしている。8 分木ツリーに分割されたデータの場合には、図 6 のようにカメラに近い位置ではより細かい子供のデータを、カメラ

から遠い場所では親のデータを用いるシーンファイルを作成することで POV-Ray がレンダリング時に使用するデータ量を削減する。

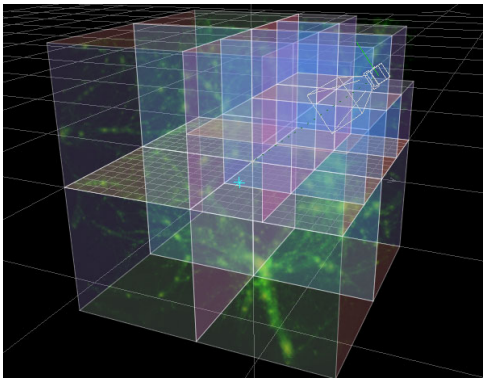

図 6: ８分木ツリー構造のデータに対し、カメラの位置に応じて使うデータを決定する。

3.2 コード開発

Oosawa は、C++言語で開発している。マルチプラットフォームを実現するために、GUI 部分には wxWidgets[2] を採用している。wxWidgets は、OS 間の違いを吸収するためのライブラリの一つで、LGPL ライセンスに近いライセンスで公開されている。現時点で Oosawa は、Windows および Linux 環境でコンパイルして動作するように開発をしている。

GUI ツールにおいてもレスポンスの向上のための高速化は重要な要素である。Oosawa のコード開発に当たっても、幾つかの高速化のために心がけている点があり、主だったものは以下のようになる。

1. 前述したメモリ管理によって、データの読み込みの待機時間をできるだけ減らす。
2. 等値面の作成やガウス暈し処理のような重いフィルター処理は、マルチスレッド処理で高速化を図る。
3. 操作の待機時間中に、次のステップのデータの読み込みを進めるなどのプリフェッチ機能で読み込みの待機時間を減らす。
4. データ本体の描画と、それ以外の補助線などの描画を分けて描画し、データ本体の再描画が不要な場合には再利用して操作のレスポンスをよくする。
5. 全体的な高速化としてループ内のメンバ変数へのアクセス（比較的遅い）を出来るだけ減らす。

このほか必要に応じて SSE の利用や GPU を計算部分に利用するといった高速化も考えられるが、現時点では自分が不慣れなことがあり、それらの高速化はほとんど行なっていない。

3.3 まとめと今後

ボリュームデータの「見せる」ための可視化に関して、POV-Ray を用いたボリュームレンダリング方法の基礎について第２節で記述した。POV-Ray 自体は、テキストファイルによってシーンを記述するために、カメラワークの設定やトライアンドエラーによる映像の作成の手順には向いていない。GUI を用いて可視化映像を作成するためのフロントエンドツール「Oosawa」の開発を行い、順

次公開を行なっている（第３節）。

現在、ボリュームデータからの映像製作に不可欠な機能の多くは、まがりなりにも実装することができた。今後の操作性向上のためには、マウスによる操作の強化は重要な課題である。現在のスライダーによって断面図の位置を調整するといった操作に加え、直接画面上でマウスを使って操作することを可能にする、といった細かな操作性の充実を図ることが重要だろう。

また、データの操作に用いるフィルタ類の充実や、座標変換による直行座標系以外のデータへの対応といった機能の追加など、より充実したツールへの開発を継続して行なっていく予定である。

Oosawa は現在ソースコードと実行ファイルあわせて公開中なので、興味があれば弄って見てほしい。

Reference

1)Persistence of Vision Pty. Ltd., Williamstown, Victoria, Australia. http://www.povray.org/

2)http://www.wxwidgets.org/

3)http://th.nao.ac.jp/ takedatk/COMPUTER/OOSAWA/oosawa.html