遠隔可視化ツール PBVR フィルタ分離型
(v.2.2) マニュアル

2023 年 11 月

国立研究開発法人日本原子力研究開発機構

システム計算科学センター

版数	改定日	章番号	改版内容
1.04	2015.3.31	-	新規
1.05	2015.5.18	5.2	サーバの可視化パラメータファイル読み込み機能と粒子データ出力機能を追加
		5.4.2	メインパネルに File ボタンを追加
		5.4.6	粒子データ保存機能を追加
1.06a	2015.10.30	5.4.2	Mac 版の特徴領域抽出機能を追加
		5.4.3	伝達関数エディタにヒストグラムを追加
		5.4.5	画像ファイル作成機能を拡張（ファイル出力ディレクトリ、キーフレーム機能）
1.07	2016.2.1	1.2	動作環境に ICEX を追加。ライブラリに VTK を追加。
		2	逐次版を含めてパッケージ更新。VTK 用フィルタを追加。
		3.4	入力データ形式に STL、PLOT3D、VTK を追加し、パラメータファイルを拡張。
		4.2	コマンドラインにプションに-pd を追加し、-plimit の定義を修正。これに伴い、-sl オプションを廃止。
		5.2	コマンドラインにプションに-pd を追加し、-plimit の定義を修正。これに伴い、-sl オプションを廃止。-pin オプションを-pin10 まで拡張。
		5.4.5	粒子統合エディタを追加。
		6.4	粒子統合エディタの使用事例を追加。
1.08	2016.4.25	2	ソースコードパッケージを更新。コンパイル・インストール方法を変更
		5.4.2	伝達関数エディタ・粒子統合エディタ・動画作成用パネルをサブパネル化し、メインパネルにボタンを追加。
1.09	2017.3.2	5.1	コマンドラインオプションに-Bs、-Be、 -Bd を追加。
		5.1.1	-vin での複数 pfi ファイルの指定方法を追加。
		4.2.2	分散ファイルの処理方法について追記。
		6.1	・-vin での複数 pfi ファイルの指定方法を追加。 ・クライアント起動時に開くパネルの数を変更。

		6.3.2	・メインパネルに以下３つのボタンを追加。 Transfer Function Panel、Particle Panel、 Animation Control Panel ・メインパネルに no-repeat sampling until Transfer Functions be edited.チェックボックスを追加。 ・メインパネルに以下３つのボタンを追加。 Legend Panel、Coordinate Panel 、 Viewer Control Panel. ・メインパネルに表示されている粒子数を表示する Display Particle Number を追加。
		6.3.3	伝達関数エディタを有効化する方法を変更し、Close ボタンを追加。
		6.3.3.3	関数エディタの演算処理で NaN が現れた場合の処理を追加。
		6.3.5	粒子統合エディタを有効化する方法を変更し、Close ボタンを追加。
		6.3.6	動画作成用パネルを有効化する方法を変更し、Close ボタンを追加。
		6.3.7	レジェンドパネルを追加。
		6.3.8	座標エディタによる座標軸変換機能を追加。
		6.3.9	ビューワ制御パネルを追加。
1.10	2017.3.15	3	フィルタ削除 フィルタプログラムとサーバプログラム結合による削除
		3	入出力ファイル追加
		5.4.2.	メインパネル変更
		5.4.3.	伝達関数エディタ変更 TFS の機能拡張作業による機能変更、GUI 変更
		4.2.	起動方法：プログラム引数：pin 削除、fin 追加 フィルタプログラムとサーバプログラム結合による変更
		5.2.	起動方法：プログラム引数：pin 削除、fin 追加 フィルタプログラムとサーバプログラム結合による変更
		6.1.	プログラム起動：プログラム引数：pin 削除、fin 追加 フィルタプログラムとサーバプログラム結合による変更
		6.5.	バッチモード処理：プログラム引数：pin 削除、fin 追加 フィルタプログラムとサーバプログラム結合による変更
1.11	2019.3.30	3.	フィルタ有りプログラムに関する記述を追加
1.12	2020.6.01	-	クライアントの GUI を変更し、関連する画像・記述を更新

1.13	2020.9.17	2.3	内蔵した可視化ライブラリの替わりに、ユーザがインストールする可視化ライブラリ KVS を利用するように機能を変更
1.13.1	2020.10.16	5.3.1.1	伝達関数およびヒストグラムにおける最大最小値の表示を変更
1.14	2020.11.05	1.3 2.1.3 2.3	Microsoft Visual Studio 2019 における 2017 環境を利用した Windows 上のビルド手順をアップデート
1.15	2020.5.1	-	軽微な不具合の修正
1.17	2022.1.30		ポリゴン合成機能を追加
1.171	2022.2.1		時系列ポリゴンの合成機能を追加
1.172	2022.5.23		描画時のバグに関する修正 CPU レンダリングモードを追加
2.0	2023.01,01		クライアントのソースコードを CS-PBVR と統合 伝達関数で利用できるカラーマップを変更
2.1	2023.03.10		Mac 版を Apple シリコンに対応
2.2	2023.11.01	3	テクスチャ付きポリゴンと３次元点群データに対応

目次

1　はじめに

1.1. 概要

本書は、日本原子力研究開発機構システム計算科学センターで開発した遠隔可視化システム PBVR のマニュアルである。本システムは、京都大学小山田研究室で開発された PBVR（Particle Based Volume Rnderring）法、および、可視化ライブラリ KVS (https://github.com/CCSEPBVR/KVS) をベースにして開発されたものであり、遠隔地にあるサーバ上の大規模ボリュームデータの高速な遠隔可視化を実現している。システムは以下の３つのコンポーネントで構成されている。

（1）フィルタ

フィルタは PBVR の前処理プログラムである。これは PBVR で大規模データを効率よく並列処理するために、ボリュームデータをサブボリュームデータに領域分割する。独自の KVSML 形式のボリュームデータを処理可能である。

（2）PBVR サーバ

ボリュームデータを入力して PBVR 法による並列可視化を行い、粒子データを出力する。独自の KVSML 形式のボリュームデータを処理可能である。

（3）PBVR クライアント

クライアントは CPU または GPU を用いて OpenGL で粒子データをレンダリングする。

上記のコンポーネントだけでなく、AVS、VTK、Ensight Gold、CGNS などの形式のボリュームデータを KVSML 形式に変換するためのライブラリ KVSML コンバータが別途用意されている。

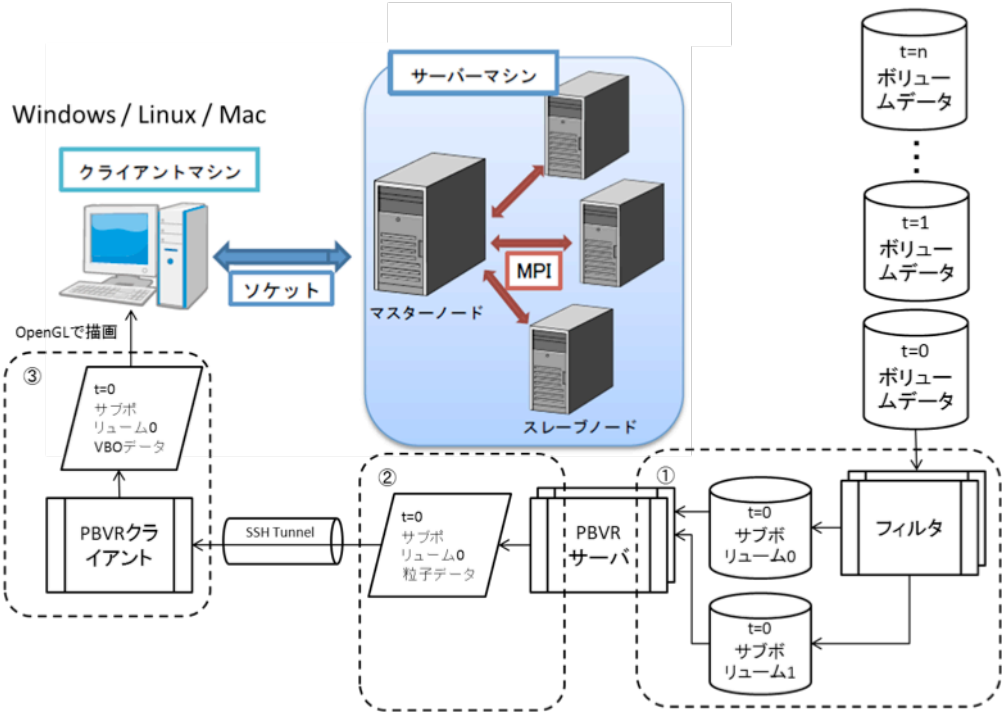


図 1-1　可視化アプリケーションシステムの全体構成

粒子 Client には、様々なシミュレーションに汎用的に対応可能な多変量可視化機能が実装されている。３次元データの可視化ではボリュームデータに色・不透明度を割り当てる。そして、その関係を記述する関数を伝達関数と呼ぶ。従来の可視化では一つの物理値に対して、色・不透明度を割り当てる１次元伝達関数が利用されていた。しかし従来の可視化手法では多変量向けの多次元伝達関数を設計することが困難であった。粒子 Client は代数式により多次元伝達関数を設計できる伝達関数エディタを提供している。伝達関数エディタでは各変量の１次元伝達関数を編集し、それらを変数として任意の代数式による多次元伝達関数を記述できる。この代数式では基本的な演算子や初等関数、微分演算子が利用可能である。

３次元シミュレーションの可視化では、境界条件のポリゴンデータと計算結果のボリュームデータを合成して表示することで複雑な形状や位置関係の把握に役立つ。粒子 Client は粒子ベースのボリュームレンダリングと手元のポリゴンデータを合成して表示できる。従来のレンダリング手法では、アルファブレンディングに伴う順序計算がボトルネックとなり、ポリゴンとボリュームの合成レンダリングは困難であった。しかし本アプリでは、ポリゴンのレンダリングを粒子ベースのレンダリングと同じ枠組みで処理するSthochastic Rendering Compositor により、効率的なポリゴンとボリュームの合成レンダリングが可能である。

１.２. 動作環境

遠隔可視化システムPBVRはクロスプラットフォームのプログラムであり、Linux、Mac、Windowsで動作する。また超並列環境に関して、フィルタ/サーバプログラムは富岳や機構所有のスーパーコンピュータ SGI8600 等、様々な環境で動作する。本プログラムは C++コンパイラに加えてライブラリとして OpenGL と Qt を利用する。本システムの動作確認を行った環境を示す。

●スーパーコンピュータ

プラットフォーム	CPU（アキテクチャ）	コンパイラ
SGI8600	Intel Xeon Gold	Intel コンパイラ
富岳	A64FX（ARM）	富士通コンパイラ

●フィルタ/サーバ/クライアント

プラットフォーム	OS	コンパイラ	ライブラリ
Linux 64bit	Kubuntu18.04	g++ 5.3.1	OpenGL、 Qt6.4
Mac 64bit	OSX12(Apple シリコン)	clang 10.0.0	OpenGL、 Qt6.4
Windows 64bit	Windows10	MSVC2019	OpenGL、 Qt6.4

1.3. Windows 上の環境構築

Windows 環境下では C++コンパイラとして Microsoft Visual Studio を使用する。ここでは Visual Studio 2019 Community のインストール手順を示す。

Visual Studio Installer を起動し、「詳細▼」→「変更」を選択し、下記の項目をインストールする。

A) 「ワークロード」タブ

(1) C++ によるデスクトップ開発（図 1-2）

B) 「個別のコンポーネント」タブで選択

(1) Windows 10 SDK (10.0.19041.0)（図 1-3）

(2) MSVC v142 - VS2019 C++ x64/x86 ビルドツール(最新版)（図 1-4）

(3) CMake の Visual C++ ツール(図 1-5)

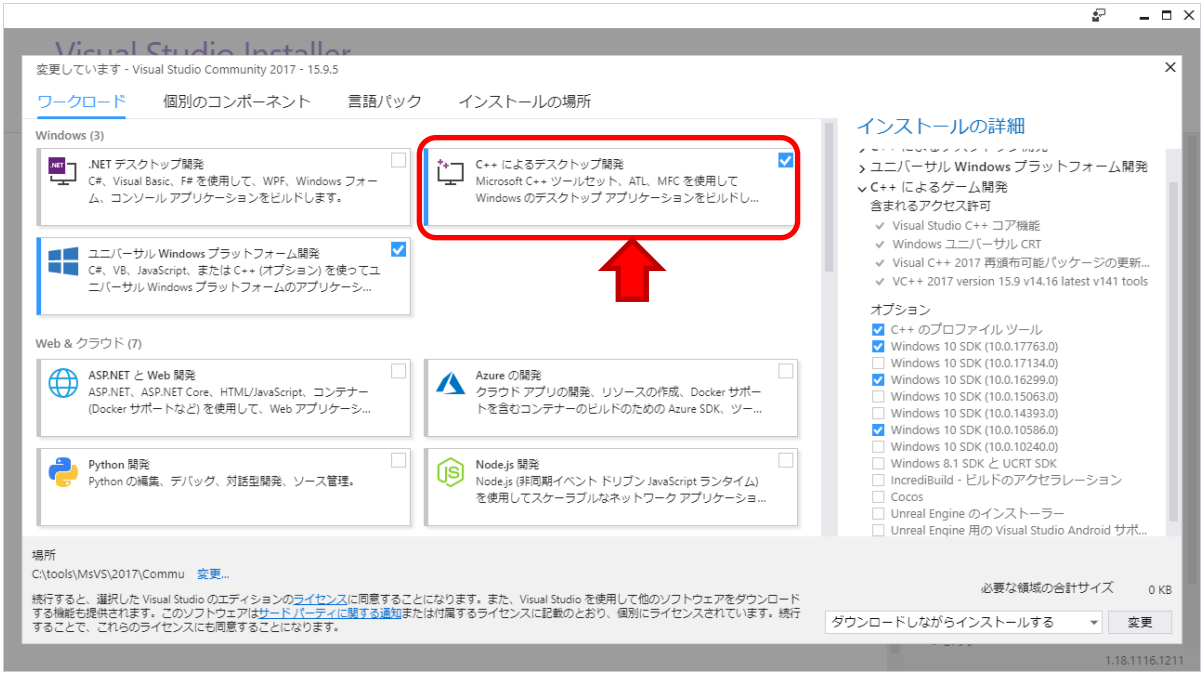


図 1-2 Visual Studio Insteller の画面 コンポーネント選択 - C++によるデスクトップ開発

図 1-3 Visual Studio Installer の画面 コンポーネント選択 - Windows SDK

図 1-4 Visual Studio Installer の画面 コンポーネント選択 - ビルドツール

図 1-5 Visual Studio Installer の画面 コンポーネント選択 - CMake

1.3.1. Qt の利用

Qt の公式ページ（https://www.qt.io/download-qt-installer）からダウンロードできる Qt Online Installer は、各プラットフォームに対して Qt のパッケージを自動でインストールする。Qt Creator IDE はこのインストーラーから提供される。Qt のインストーラーにはオンラインインストーラーとオフラインインストーラーが存在し、本プログラムのビルドにはオフラインインストーラーから提供される Qt6 が使用される。

インストーラーでは、「インストールフォルダー」の画面で「カスタムインストール」を選択し、クライアントプログラムの構築に利用される以下の項目を選択する必要がある。

- Qt/Qt 6.2.4
- Qt/Developer and Design Tools/Qt Creator 10.0.1
- Qt/Developer and Design Tools/Qt Creator 10.0.1 CDB Debuffer Support
- Qt/Developer and Design Tools/CMake 3.24.2
- Qt/Developer and Design Tools/Ninja 1.10.2

図 1-6 Qt インストーラー – コンポーネントの選択

２ インストール

PBVR のフィルタ、サーバ、そしてクライアントプログラムは Linux、Mac、Windows 上で動作し、逐次処理および OpenMP で並列化されたバイナリを提供する。またスーパーコンピュータ SGI8600 および富岳むけに MPI+OpenMP により超並列化されたバイナリが利用可能である。

２.１. フィルタ

フィルタプログラムのロードモジュールは、逐次処理版、スレッド並列処理用の OpenMP 版、および、超並列処理用の MPI+OpenMP 版が用意されている。各環境のロードモジュールをパスの通った適当なディレクトリにコピーすることによりインストールが完了する。ロードモジュールの構成は以下の通りである。

表 2.1-1　フィルタプログラムのロードモジュール一覧

機種	並列化	ロードモジュール名
Linux 64bit	逐次	pbvr_filter_linux
	OpenMP	pbvr_filter_linux_omp
Mac 64bit	逐次	pbvr_filter_mac
	OpenMP	pbvr_filter_mac_omp
Windows 64bit	逐次	pbvr_filter_win
	OpenMP	pbvr_filter_omp_win
SGI8600 ※1	MPI+OpenMP	pbvr_filter_s86_mpi_omp
富岳	MPI+OpenMP	pbvr_filter_fugaku_mpi_omp

※1. スーパーコンピュータ用のロードモジュールは計算ノード専用となるので、Linux サーバによって構成されるログインノードやポスト処理ノードでは Linux 用ロードモジュールを用いる。

2.2. サーバ

サーバプログラムは C++で実装され、逐次処理版、スレッド並列処理用の OpenMP 版、および、超並列処理用の MPI+OpenMP 版のロードモジュールが利用できる。各環境のロードモジュールをパスの通った適当なディレクトリにコピーすることによりインストールが完了する。ロードモジュールの構成は以下の通りである。

表 2.2-1　サーバプログラムのロードモジュール一覧

機種	並列化	ロードモジュール名
Linux 64bit	逐次	pbvr_server_linux
	OpenMP	pbvr_server_linux_omp
Mac 64bit	逐次	pbvr_server_mac
	OpenMP	pbvr_server_mac_omp
Windows 64bit	逐次	pbvr_server_win
	OpenMP	pbvr_server_omp_win
SGI8600※1	MPI+OpenMP	pbvr_server_s86_mpi_omp
富岳	MPI+OpenMP	pbvr_server_fugaku_mpi_omp

※1. スーパーコンピュータ用のロードモジュールは計算ノード専用となるので、Linux サーバによって構成されるログインノードやポスト処理ノードでは Linux 用ロードモジュールを用いる。

2.3. クライアント

クライアントプログラムのロードモジュールは pthread により並列化されている。各環境のロードモジュールをパスの通った適当なディレクトリにコピーすることによりインストールが完了する。ロードモジュールの構成は以下の通りである。

表 2.3-1　クライアントプログラムのロードモジュール一覧

機種	並列化	ロードモジュール名
Linux 64bit	pthread	pbvr_client_linux
Mac 64bit	pthread	pbvr_client_mac
Windows 64bit※1	pthread	pbvr_client_win

※1. Windows 版のロードモジュールを動作させるには、別途 GLUT の動的ライブラリをパスの通ったディレクトリかあるいはロードモジュールと同じディレクトリに配置する必要がある。GLUT の動的ライブラリである glut32.dll は OpenGL のサイト（3.1.2）から利用できる。

３ ビルド

フィルタプログラムおよびサーバプログラムは C++で実装されており、コンフィグファイルで設定することで逐次処理版、スレッド並列処理用の OpenMP 版、および、超並列処理用の MPI+OpenMP 版を切り替えてビルドできる。クライアントプログラムは C++、Qt と OpenGL によって実装されており、コンフィグファイルで設定することで CPU レンダラーと GPU レンダラーを切り替えてビルドすることができる。粒子統合エディタを使用したポリゴンデータと粒子データの合成表示(6.3.5)は GPU レンダラーでのみ可能である。

３.１. フィルタ・サーバプログラムのビルド

フィルタおよびサーバプログラムは任意のディレクトリに展開されたソースコードパッケージの中で、PBVR/ディレクトリ配下の pbvr.conf、Makefile によりコンパイルされる。pbvr.conf ファイルにより make を設定し、フィルタ、サーバプログラムをコンパイルする。PBVR/ディレクトリの構成は以下のようになっている。

表 3.1-1　ソースコードパッケージのディレクトリ構成

ディレクトリ・ファイル	説明
PBVR/	
KMATH/	並列乱数生成ライブラリ KMATH
KVS/	KVS（サーバプログラム用）※2
glui/	GUI 用ウィジェットライブラリ
Client/	OpenGL 版のクライアントプログラム
FunctionParser/	関数エディタライブラリ
Common/	プロトコル、通信、共通のライブラリ
Filter/	フィルタプログラム
Server/	サーバプログラム
arch/	各コンパイルの設定ファイル
pbvr.conf ※1	make 設定ファイル
Makefile ※1	PBVR/配下のソースコードをコンパイルする

※1. Windows 環境では、Makefile と pbvr.conf の代わりに VisualStudio 用のソリューションファイル pbvr.sln を用いる。

※2. この KVS はクライアントプログラムでは使用されない。この KVS はサーバ上の並列可視化処理に特化した独自のバージョンである。

pbvr.conf において変数の入力値を変更することで、インストールする機能を指定する。pbvr.conf の変数の一覧を以下に示す。

表 3.1-2 pbvr.conf の変数と入力値の一覧

変数	入力値	説明
PBVR_MACHINE	文字列	arch/配下のコンパイル設定ファイル
PBVR_MAKE_CLIENT	0 or 1	OpenGL 版クライアントのサポートの有無
PBVR_MAKE_FILTER	0 or 1	フィルタのサポートの有無
PBVR_MAKE_SERVER	0 or 1	サーバのサポートの有無
PBVR_SUPPORT_KMATH	0 or 1	KMATH サポートの有無（サーバのみ）※1
PBVR_SUPPORT_VTK	0 or 1	VTK サポートの有無（フィルタのみ）

※1. Mac 版、windows 版サーバでは並列乱数生成ライブラリ KMATH を使用できないため、逐次乱数生成ライブラリ TinyMT を用いる。

変数 PBVR_MACHINE ではコンパイル環境に応じて arc/ディレクトリ配下に格納されている Makefile の設定ファイルを指定する。

表 3.1-3 コンパイル設定ファイルの一覧

ファイル名	説明
Makefile_machine_gcc	gcc による逐次版コンパイルの設定
Makefile_machine_gcc_omp	gcc による OpenMP 版コンパイルの設定
Makefile_machine_gcc_mpi_omp	gcc による MPI+OpenMP 版コンパイルの設定
Makefile_machine_intel	intel による逐次版コンパイルの設定
Makefile_machine_intel_omp	intel による OpenMP 版コンパイルの設定
Makefile_machine_intel_mpi_omp	intel による MPI+OpenMP 版コンパイルの設定
Makefile_machine_s86_mpi_omp	JAEA 大型計算機上でコンパイルの設定 Intel コンパイラと mpt ライブラリを使用
Makefile_machine_fugaku_clang	富岳（ARM）clang モードでのコンパイルの設定
Makefile_machine_fugaku_trad	富岳（ARM）trad モードでのコンパイルの設定

3.1.1. Linux/Mac

以下に Linux、 Mac 上でビルドする手順を示す。

（1） PBVR/ディレクトリ配下の pbvr.conf を、各環境で使用する機能に合わせて編集する。
gcc コンパイラを使用して OpenMP により並列化されたフィルタ、サーバプログラムをビルドする例を示す。

■pbvr.conf の例

```
PBVR_MACHINE=Makefile_machine_gcc_omp
PBVR_MAKE_CLIENT=0
PBVR_MAKE_FILTER=1
PBVR_MAKE_SERVER=1
PBVR_SUPPORT_KMATH=0
```

Qt 版クライアントを利用する場合、PBVR_MAKE_CLIENT =0 と記述して OpenGL 版クライアントのサポートを無効化する。

（2） PBVR/配下で下記のようにビルドを行う。
$make
PBVR 配下で下記のように実行モジュールが作成される。
フィルタ： Filter/pbvr_filter
サーバ： Server/pbvr_server

（3） 生成されたロードモジュールをパスの通った適当なディレクトリにコピーする

3.1.2. Windows

以下に Windows 上でのビルドする手順を示す。

（1） glut（64bit）のインストールについて

① 下記のホームページより、glut-3.7.6-bin_x64.zip(64bit)をダウンロードする。
https://user.xmission.com/~nate/glut.html

② ①を解凍し、上記ホームページのインストール手順を参考に適切な場所に置く。

- glut.h
- glut32.lib
- glut32.dll

尚、pbvr.sln（後述）の設定では、上記ファイルは以下のディレクトリに配置される設定となっている。
C:¥pbvr¥glut-3.7.6¥include¥GL¥glut.h
C: ¥pbvr¥glut-3.7.6¥lib¥ glut32.lib
また、glut の DLL ファイルである glut32.dll はクライアントの実行モジュール（pbvr_client.exe）と同じディレクトリに配置される必要がある（後述）。

（2） MSVC 環境のある端末上で、PBVR 環境を展開する。

（3） 展開したディレクトリ配下の pbvr.sln を実行し、MSVC を起動する。

（4） 各環境で使用する機能を選択するため、MSVC のソリューションエクスプローラーから“ソリューションのプロパティ”を選択する。そして“構成プロパティ”で各プロジェクトのサポートの有無を選択する。Qt 版クライアントを利用する場合、Client のチェックボックスを外して OpenGL 版クライアントのサポートを無効化する。

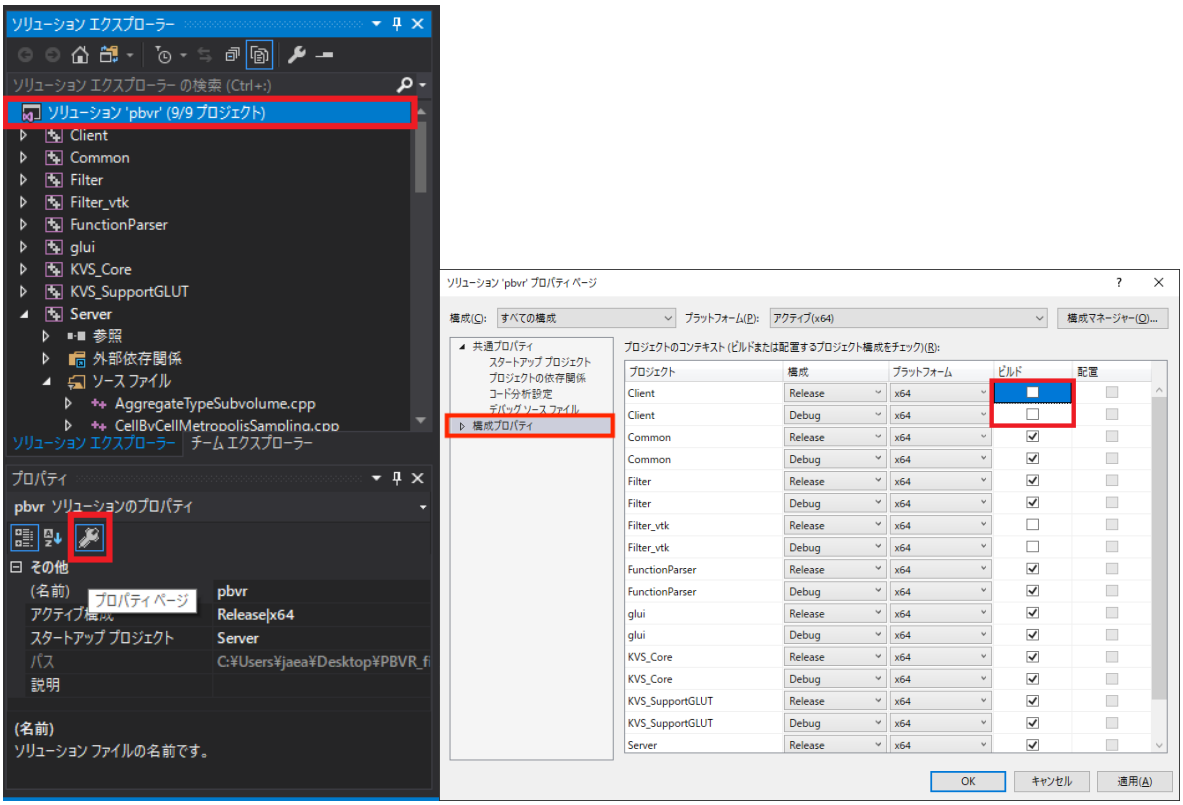


図 3-1　OpenGL 版クライアントのサポートの無効化

（5） OpenMP により並列化されたプログラムを生成する場合は、下記プロジェクトファイルの“OpenMP のサポート”を有効化する。

- Client
- Common
- Filter
- FunctionParser
- glui
- KVS_Core
- KVS_SupportGLUT
- Server

下図は Server プロジェクトファイルの OpenMP サポートを有効化する例を示す。

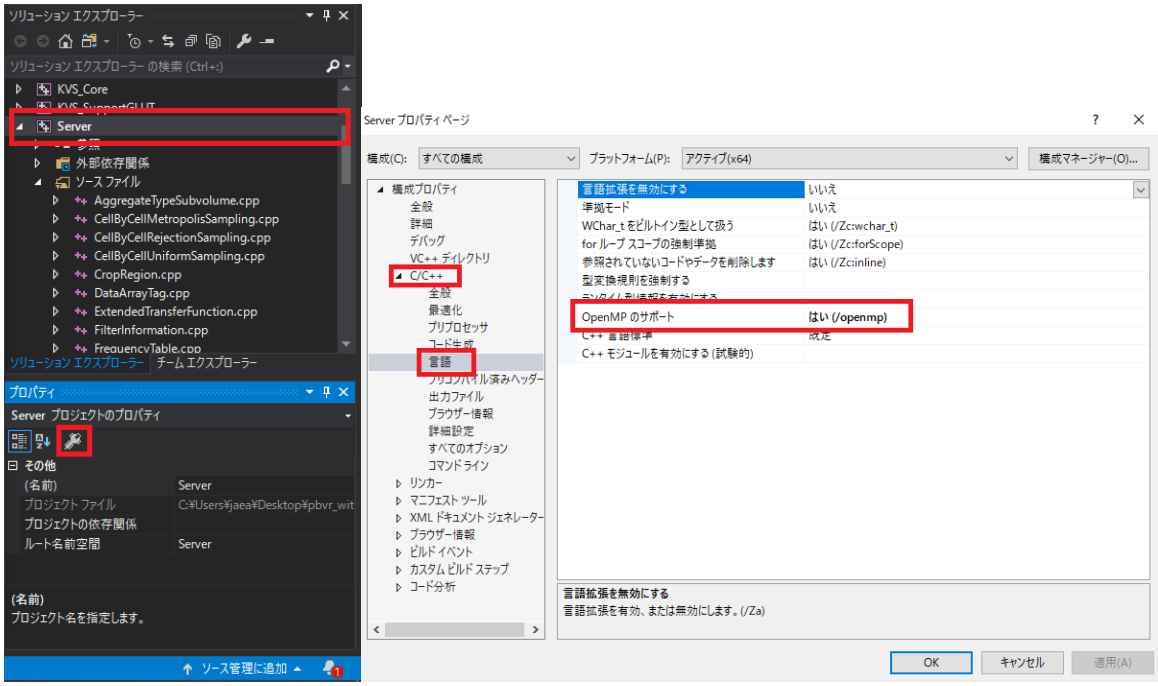


図 3-2　プロジェクトの OpenMP サポート

（6） ビルドのために pbvr.sln の構成（下図参照）を Release x64 に変更する。

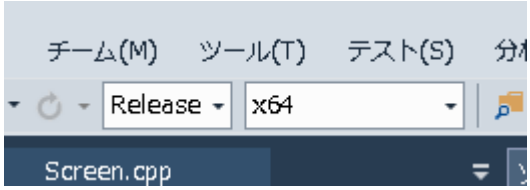


図 3-3　MSVC の構成例

（7） MSVC メニューの[ビルド] ⇒[ソリューションのビルド]を実行する。

¥¥x64¥Release 配下に pbvr_filter.exe（フィルタ）、pbvr_server.exe（サーバ）、pbvr_client.exe（クライアント）それぞれの実行モジュールが作成される。

3.1.3. VTK 用フィルタのビルド

VTK 用フィルタのビルドには VTK6.0 以上が必要である。VTK ライブラリのインストールについては VTK ウェブサイト（http://www.vtk.org/）を参照すること。また、インストールにあたっては CMake-gui において以下の設定を行うこと。

① BUILD_SHARED_LIBS オプションをオンにする。

② CMAKE_BUILD_TYPE オプションを “Release” にする。

③ CMAKE_INSTALL_PREFIX を VTK をインストールする場所に設定する。

VTK 用フィルタに必要な設定を以下に示す。

Linux、 Mac

以下のように環境変数を設定する。

```
% export VTK_VERSION=n.n
% export VTK_LIB_PATH=/usr/local/lib
% export VTK_INCLUDE_PATH=/usr/local/include/vtk-n.n
```

ここで、n.n には使用する VTK のバージョンを入力し、パスは VTK ライブラリのインストール環境に合わせて書き換えること。

Windows

コントロールパネル->システム->詳細設定->環境変数　に移動し、以下の環境変数を設定する

変数	値
VTK_VERSION	n.n
VTK_LIB_PATH	d:¥environments¥VTK¥lib
VTK_INCLUDE_PATH	d:¥environments¥VTK¥include¥vtk-n.n

ここで、n.n には使用する

3.2. クライアントプログラムのビルド

クライアントプログラムは、可視化ライブラリ KVS (https://github.com/naohisas/KVS)と統合開発環境 Qt Creator を利用してビルドされる。

3.2.1. KVS の設定とビルド

KVS のインストール方法はダウンロードページの Wiki に詳細が記されているため、参照されたし。ただし、本クライアントプログラムは、オリジナルの KVS ではなく、フレームバッファの取り扱い等について改良した独自バージョンの KVS (https://github.com/CCSEPBVR/KVS)を利用してビルドされる。

インストールのための環境変数として、KVS_DIR に KVS のインストールパスを指定し、PATH に KVS 用のコマンドを追加する。

Windows

システム環境設定から下記の環境変数を作成し値を設定する。

Variable	Value
KVS_DIR	任意のインストール場所。例 C:¥Program Files¥kvs
PATH	%PATH%;%KVS_DIR%¥bin

Linux/Mac

ターミナルから下記のコマンドで環境変数を設定する。

```
export KVS_DIR=~/local/kvs
export PATH=$KVS_DIR/bin:$PATH
```

また、Apple シリコン搭載 Mac であえて x86-64 アーキテクチャを使用したい場合、Rosetta をインストールし、以下のコマンドでアーキテクチャを切り替えて下さい。

```
arch -x86_64 bash
```

---

kvs.conf を以下のように指定し、コンフィギュレーションを設定する。

```
KVS_ENABLE_OPENGL     = 1
KVS_ENABLE_GLU        = 0
KVS_ENABLE_GLEW       = 0    # Linux/Windows のみ 1 とする
KVS_ENABLE_OPENMP     = 0
KVS_ENABLE_DEPRECATED = 0

KVS_SUPPORT_CUDA      = 0
KVS_SUPPORT_GLUT      = 0    # Linux/Windows のみ 1 とする
KVS_SUPPORT_OPENCV    = 0
KVS_SUPPORT_QT        = 0
KVS_SUPPORT_PYTHON    = 0
KVS_SUPPORT_EGL       = 0
KVS_SUPPORT_OSMESA    = 0
```

Mac 環境では KVS のインストールに GLUT および GLEW を利用していない。しかし、Linux/Windows 環境では GLUT および GLEW を利用しており、適宜インストールしパスを設定する必要がある。

Linux

GLUT のインストール

```
sudo apt-get install freeglut3-dev libglut3-dev
```

GLEW のインストール

```
sudo apt-get install libglew1.5-dev
```

環境変数 KVS_GLUT_DIR と KVS_GLEW_DIR にパスを設定する。

```
export KVS_GLUT_DIR=GLUT のインストールパス
export KVS_GLEW_DIR=GLEW のインストールパス
```

Windows

1. 下記の URL の"ダウンロード（構築用ファイル）"より 64 ビットモードで構成された glut をダウンロードする。
   http://coskx.webcrow.jp/mrr/for_students/LectGLCG/distribution/index.html
   ダウンロードファイルに含まれる glut32.dll、glut32.lib は名称が 32bit になっているが、64bit 向けに構築されている。
2. GLEW のダウンロードページ（http://glew.sourceforge.net）から 64bit 版 GLEW をダウンロードする。

3. ダウンロードファイルを解凍し、そこに含まれる以下のファイルを指定するフォルダにコピーする。以下の例では GLUT と GLEW のインストール先を C:¥PBVR_Dev¥OpenGL とする。

ファイル名	コピー先
glut32.dll glew32.dll	C:¥PBVR_Dev¥OpenGL¥bin
glut32.lib glew32.lib glew32s.lib	C:¥PBVR_Dev¥OpenGL¥lib
glut.h glew.h wglew.h	C:¥PBVR_Dev¥OpenGL¥include¥GL l

システム環境設定から環境変数 KVS_GLUT_DIR と KVS_GLEW_DIR を作成し、GLUT と GLEW のインストール先（上記の例ならば）C:¥PBVR_Dev¥OpenGL を設定する。

ターミナル上で KVS をコンパイルしインストールする。

Windows

Visual Studio の開発者コマンドプロンプトを起動し、nmake コマンドをタイプして MSVC により KVS をビルドする。下記の例では KVS のソースコードのディレクトリを C:SRC¥KVS とする。

```
cd C:¥SRC¥KVS
nmake
nmake install
```

Linux/Mac

ターミナル上から make を利用して KVS をビルドする。下記の例では KVS のソースコードのディレクトリを/SRC/KVS とする。

```
cd /SRC/KVS
make
make install
```

コンパイルに関するその他のオプションはダウンロードページの Wiki を参照すること。

3.2.2. クライアントプログラムの設定とビルド

クライアントプログラムのコンフィギュレーションは qtpbvr.conf で設定される。qtpbvr.conf を以下のように編集して PBVR_MODE=CS を有効にする。

```
#PBVR_MODE - Either CS (ClientServer), or IS (Insitu) - Needed on all platforms
PBVR_MODE = CS
#PBVR_MODE = IS
```

ソースコードに含まれるプロジェクトファイル QtClient.pro を Qt Creator で開き、クライアントプログラムの設定とビルドを行う。

1. Qt Creator を起動し、File ⇒ **Open File or Project** を選択する。
2. **QtClient.pro** ファイルを **QTPBVR/QtClient** ディレクトリから選択する。
   PBVRClient/QtClient/QtClient.pro
3. **Active Project** “**QtClient**” を選択する。
4. “**Build Settings**”を選択し、 “**Shadow Build**”にチェックを入れる。
5. “**Build Directory**” にビルド後の実行ファイルを格納するディレクトリを指定する。例えば、 “**../build**”を設定する。

図 3-4 クライアントプログラムの設定

次に、プロジェクトに KVS の環境変数を設定する。“**Build Settings**”における“**Build Environment**”で“Detail”を押下してセクションを展開し、環境変数として KVS_DIR（KVS がインストールされた場所）を追加する。

Linux/Windows の場合、環境変数として GLEW_DIR（GLEW がインストールされた場所）および環境変数 PATH に%GLEW_DIR%¥bin を追加する。

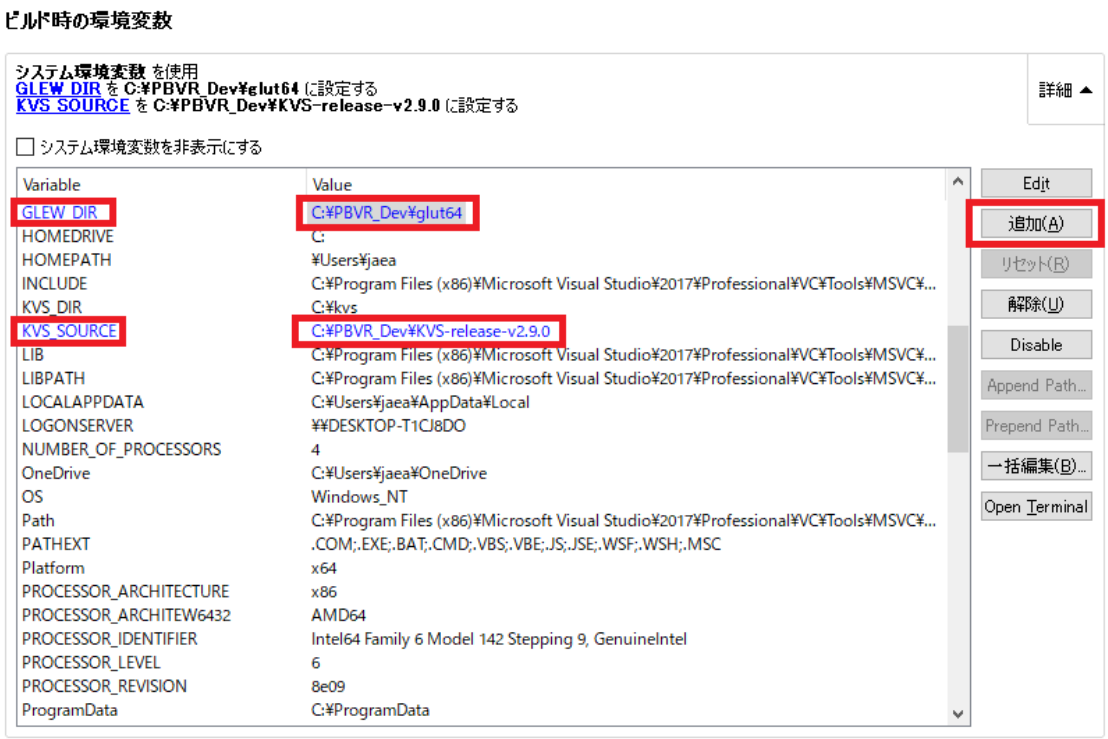


図 3-5　KVS の環境変数の設定

次にクライアントプログラムをビルドする。

1. 左のツールバーで Edit を選ぶ。
2. QtClient プロジェクトを右クリックして、”Run qmake”を選択する。
3. QtClient プロジェクトを右クリックして、”Build”を選択する。

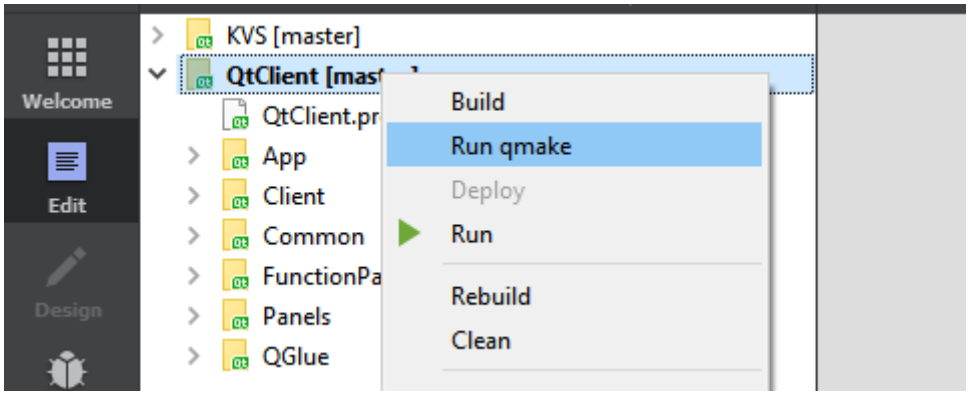


図 3-6　クライアントプログラムのビルド

3.2.3. Windows 環境におけるクライアントプログラムのデプロイ

この手順は Windows 環境下で、ビルドした PC から配布可能なアプリケーションをデプロイする場合にのみ必要とされます。

```
C:> cd C:¥PBVR_Dev¥QTPBVR¥build¥App
C:> windeployqt -release pbvr_client.exe
C:> cp C:¥PBVR_Dev¥OpenGL¥bin¥*.dll .
```

次に、インストール済みの KVS からシェーダプログラムを App フォルダにコピーします。

コピー元： /include/Core/Visualization/Shader/

コピー先： /App/include/Core/Visualization/Shader/

作成した App フォルダは他の Windows マシンで実行することができます。

3.2.4. CPU/GPU レンダラー

本プロジェクトはデフォルトで GPU レンダラーが生成される。しかしスパコンのような描画用の GPU を搭載していない環境のために、CPU レンダラーによる PBVR をビルドできる。
CPU レンダラーをビルドするには、pbvr_client -> QtClient -> qtpbvr.conf を開き、ビルドモードを指定する変数を以下のように変更する。

REND_MODE = GPU
を
REND_MODE = CPU
に変更する。

３.２.５. CGFormatExt4KVS のビルド

CG データ表示機能に必要となる CGFormatExt4KVS および依存ライブラリのインストール手順について説明する。

３.２.５.１ Assimp

assimp-5.0.0.zip を展開するとできる「assimp-5.0.0」フォルダを任意のフォルダに移動する。以降、このフォルダを「<LIB_DIR>」として説明する。

３.２.５.１.１. Windows

Tools コマンドプロンプトで「<LIB_DIR>/assimp-5.0.0」フォルダに移動し、以下のコマンドを実行する。

```
cd assimp-5.0.0
SET SOURCE_DIR=.
SET GENERATOR=Visual Studio 15 2017
SET BINARIES_DIR="./BINARIES/x64"
SET CMAKE_GENERATOR=Visual Studio 15 2017
SET CMAKE_GENERATOR_INSTANCE=C:¥Program Files (x86)¥Microsoft Visual Studio¥2017¥Community
%CMAKE_BIN% CMakeLists.txt -G "%GENERATOR%" -A x64 -D CMAKE_GENERATOR_INSTANCE="%CMAKE_GENERATOR_INSTANCE%" -D CMAKE_GENERATOR="%CMAKE_GENERATOR%" -S %SOURCE_DIR% -B %BINARIES_DIR%
%CMAKE_BIN% --build %BINARIES_DIR% --config debug
%CMAKE_BIN% --build %BINARIES_DIR% --config release
```

コマンド実行後に、

<LIB_DIR>¥assimp-5.0.0¥BINARIES¥x64¥include¥assimp¥config.h

を

<LIB_DIR>¥assimp-5.0.0¥include¥assimp¥config.h

にコピーする。

ビルドが成功すると「assimp-5.0.0¥BINARIES¥x64¥code」フォルダ下の「Release」、「Debug」フォルダに、以下の assimp のライブラリファイルが作成される。

- Release
  - assimp-vc141-mt.lib
  - assimp-vc141-mt.dll
- Debug
  - assimp-vc141-mtd.lib
  - assimp-vc141-mtd.dll
  - assimp-vc141-mtd.pdb

以下の２つのユーザ環境変数を設定する。

変数名	変数値
ASSIMP_INC_DIR	<LIB_DIR>¥assimp-5.0.0¥include
ASSIMP_LIB_DIR	<LIB_DIR>¥assimp-5.0.0¥BINARIES¥x64¥code¥Release

3.2.5.1.2. macOS

Homebrew などで cmake をインストールした後、ターミナルで「<LIB_DIR>/assimp-5.0.0」フォルダに移動し、以下のコマンドを実行する。

```
cd assimp-5.0.0
cmake -DCMAKE_OSX_ARCHITECTURES=”x86_64” -DCMAKE_OSX_DEPLOYMENT_TARET=”10.15” -DCMAKE_BUILD_TYPE=Release -DASSIMP_BUILD_ZLIB=on -DBUILD_SHARED_LIBS=off CmakeList.txt
make
```

ビルドが成功すると「assimp-5.0.0/lib/」フォルダに、以下の assimp のライブラリファイルが作成される。

- libassimp.a
- liblrrXML.a
- libzlibstatic.a

以下の２つのユーザ環境変数を設定する。

変数名	変数値
ASSIMP_INC_DIR	<LIB_DIR>/assimp-5.0.0/include
ASSIMP_LIB_DIR	<LIB_DIR>/assimp-5.0.0/lib

3.2.5.1.3. Linux

ディストリビューションのパッケージ管理システム(yum, apt など)で cmake をインストールした後、ターミナルで「<LIB_DIR>/assimp-5.0.0」フォルダに移動し、以下のコマンドを実行する。

```
cd assimp-5.0.0
cmake -DCMAKE_BUILD_TYPE=Release -DASSIMP_BUILD_ZLIB=on -DBUILD_SHARED_LIBS=off CmakeList.txt
make
```

ビルドが成功すると「assimp-5.0.0/lib/」フォルダに、以下の assimp のライブラリファイルが作成される。

- libassimp.a
- liblrrXML.a
- libzlibstatic.a

以下の２つのユーザ環境変数を設定する。

変数名	変数値
ASSIMP_INC_DIR	<LIB_DIR>/assimp-5.0.0/include
ASSIMP_LIB_DIR	<LIB_DIR>/assimp-5.0.0/lib

３.２.５.２ Autodesk FBX SDK

Autodesk 社の Web サイト(https://aps.autodesk.com/developer/overview/fbx-sdk)から、OS に合わせたインストーラーをダウンロードし、以下の手順でインストールする。

３.２.５.２.１. Windows

Autodesk_FBX_Review_Win_64bit.exe を実行し、インストーラーの指示に従いインストールする。インストール先フォルダは、通常は初期設定である「C:¥Program Files¥Autodesk¥FBX¥FBX SDK¥2019.5」で問題ないが、変更する事も可能である。以後、このインストール先フォルダを<FBX_SDK_DIR>と呼ぶ。

以下の２つのユーザ環境変数を設定する。いずれも、<FBX_SDK_DIR>の部分は前述のインストール先フォルダを示す。

変数名	変数値
FBX_SDK_INC_DIR	<FBX_SDK_DIR>¥include
FBX_SDK_LIB_DIR	<FBX_SDK_DIR>¥lib¥vs2017¥x64¥release

３.２.５.２.２. macOS

fbx202034_fbxsdk_clang_mac.pkg.tgz (safari でダウンロードした場合、自動展開されて fbx202034_fbxsdk_clang_mac.pkg.tar となっていることがある)を tar コマンドで展開すると fbx202034_fbxsdk_clang_macos.pkg ができる。このファイルを finder でダブルクリックするとインストールが始まる。

「"fbx202034_fbxsdk_clang_macos.pkg"が悪質なソフトウェアかどうかを Apple は確認できないため、このソフトウェアは開けません。」という警告ダイアログが表示されるが、「開く」をクリックするとインストーラーが起動する。以降、インストーラーに表示された手順通りに操作するとインストールが完了する。

/Applications/Autodesk/FBX SDK/2020.3.4/ にインストールされる。

以下の２つのユーザ環境変数を設定する。

変数名	変数値
FBX_SDK_INC_DIR	/Applications/Autodesk/FBX SDK/2020.3.4/include
FBX_SDK_LIB_DIR	/Applications/Autodesk/FBX SDK/2020.3.4/lib/clang/release

3.2.5.2.3. Linux

fbx202034_fbxsdk_linux.tar.gz を tar コマンドで展開すると fbx202034_fbxsdk_linux と Install_Fbxsdk.txt の 2 ファイルができる。前者がインストーラー、後者がインストールマニュアルである。ターミナルから以下のように実行するとインストーラーが起動する。インストール先は任意のディレクトリで構わないが、ここでは<FBXSDK_DIR>として表記する。

```
./fbx202234_fbxsdk_linux　<FBXSDK_DIR>
```

インストーラーに表示された手順通りに操作するとインストールが完了する。

以下の 2 つのユーザ環境変数を設定する。

変数名	変数値
FBX_SDK_INC_DIR	<FBXSDK_DIR>/include
FBX_SDK_LIB_DIR	<FBXSDK_DIR>/lib/gcc/release

3.2.5.3 CGFormatExt4KVS

CGFormatExt4KVS.zip を展開するとできる「CGFormatExt4KVS」フォルダを任意のフォルダに移動する。以降、このフォルダを「<LIB_DIR>」として説明する。

3.2.5.3.1. ビルド設定

「CGFormatExt4KVS」フォルダにある「kvsmake_libs.vc.conf」(Windows) もしくは「kvsmake_libs.conf」(macOS/Linux)をテキストエディタで開き、以下の項目を設定する。

これらの値は半角スペースを含むことがあるため、「”　”」で括って設定する。

- CGFORMATEXT4KVS_SUPPORT_FBXSDK
  FBX 形式の CG データ読み込みに対応させる場合は 1、させない場合は 0 を設定する。
- FBX_SDK_DIR
  “CGFORMATEXT4KVS_SUPPORT_FBXSDK = 1“の時のみ有効。
  FBX SDK のインストールディレクトリを設定する。
- FBX_SDK_LIB_PATH
  “CGFORMATEXT4KVS_SUPPORT_FBXSDK = 1“の時のみ有効。
  FBX SDK に含まれるライブラリファイルのディレクトリを設定する。
- CGFORMATEXT4KVS_SUPPORT_ASSIMP
  3DS 形式の CG データ読み込みに対応させる場合は 1、させない場合は 0 を設定する。
- ASSIMP_DIR
  “CGFORMATEXT4KVS_SUPPORT_ASSIMP = 1“の時のみ有効。
  assimp のインストールディレクトリを設定する。

- ASSIMP_LIB_PATH
  “CGFORMATEXT4KVS_SUPPORT_ASSIMP = 1 “の時のみ有効。
  ASSIMP に含まれるライブラリファイルのディレクトリを設定する。

3.2.5.3.2. Windows でのビルド

Tools コマンドプロンプトで「<LIB_DIR>¥CGFormatExt4KVS¥Lib」フォルダに移動し、以下のコマンドを実行する。

```
kvsmake -g LibCGFormatExt4KVS
kvsmake lib
```

コマンド実行が成功すると、「<LIB_DIR>¥CGFormatExt4KVS¥Lib」フォルダに「LibCGFormatExt4KVS.lib」が作成される。

以下のユーザ環境変数を設定する。

変数名	変数値
CGFORMAT_EXT4KVS_SHADER_DIR	<LIB_DIR>¥CGFormatExt4KVS¥Lib

3.2.5.3.3. macOS/Linux

ターミナルで「<LIB_DIR>/CGFormatExt4KVS/Lib」フォルダに移動し、以下のコマンドを実行する。

```
kvsmake -g LibCGFormatExt4KVS
kvsmake lib
```

コマンド実行が成功すると、「<LIB_DIR>¥CGFormatExt4KVS¥Lib」フォルダに「libLibCGFormatExt4KVS.a」が作成される。

以下のユーザ環境変数を設定する。

変数名	変数値
CGFORMAT_EXT4KVS_SHADER_DIR	<LIB_DIR>/CGFormatExt4KVS/Lib

3.2.6. VR および MR 用の依存ライブラリ

クライアントプログラムを VR モードでビルドするために必要となるライブラリのインストール手順について説明する。なお、これらのライブラリの動作環境が Windows に限定されるため、VR モードは Windows でのみ動作する。

3.2.6.1 Oculus SDK

Oculus SDK は VR モードで必須のライブラリである。

適当なフォルダ（以降の説明では、このフォルダを<LIB_DIR>とする）内に「ovr_sdk_win_1.30.0_public」フォルダを作成する。次に、ovr_sdk_win_1.30.0_public.zip を展開して、その中にあるファイルとフォルダを全て「<LIB_DIR>¥ovr_sdk_win_1.30.0_public」フォル

ダにコピーする。これらのファイルには最初からビルド済みの「LibOVR.lib」ファイルが含まれているので、ビルドする必要はない。

以下の２つのユーザ環境変数を設定する。

変数名	変数値
OCULUS_INC_DIR	<LIB_DIR>¥ovr_sdk_win_1.30.0_public¥LibOVR¥Include
OCULUS_LIB_DIR	<LIB_DIR>¥ovr_sdk_win_1.30.0_public¥LibOVR¥Lb¥Windows¥x64¥Release¥VS2017

３.２.６.２ **Dear ImGui**

Dear ImGui は OpenGL ベースの GUI ライブラリである。VR 空間内にダイアログを表示するために使用する。

imgui-1.79.zip を展開してできる「imgui-1.79」フォルダを任意のフォルダに移動する。以降、このフォルダを「<LIB_DIR>」として説明する。imgui-1.79_cmakefiles.zip を展開してできる「imgui-1.79」フォルダ内のファイル及びフォルダを「<LIB_DIR>¥imgui-1.79」フォルダに移動する。

Tools コマンドプロンプトで「<LIB_DIR>¥imgui-1.79」フォルダに移動し、以下のコマンドを実行する。

```
cmake -G "Visual Studio 15 2017" -A x64 .
cmake --build . --config release
copy Release¥libimgui.lib  .¥
copy examples¥Release¥libimgui_impl_opengl3.lib  .¥
```

3.2.7. クライアントプログラムの設定とビルド

3.2.7.1 モード設定

クライアントプログラムのコンフィギュレーションは qtpbvr.conf で設定される。qtpbvr.conf を以下のように編集して適切な PBVR_MODE を有効にする。

モード	設定値
CS モード	PBVR_MODE = CS
IS モード	PBVR_MODE = IS
VR モード	PBVR_MODE = VR

3.2.7.2 CG ファイルフォーマット対応設定

初期状態では、FBX 形式および 3DS 形式の CG データ読み込みに対応するように設定されているが、これらのフォーマットに対応しないようにする場合は QtClient/SETTINGS.pri をテキストエディタ等で開き、以下の項目の設定値を変更する。

(1) 3DS 形式に対応しない場合
以下の 2 項目をコメントアウトする。
- DEFINES += CGFORMATEXT4KVS_SUPPORT_ASSIMP
- DEFINES += ENABLE_ASSIMP

(2) FBX 形式に対応しない場合
以下の 2 項目をコメントアウトする。
- DEFINES += CGFORMATEXT4KVS_SUPPORT_FBXSDK
- DEFINES += ENABLE_FBXSDK

なお、これらの項目を有効にしていても、CGFormatExt4KVS のビルド時にこれらの項目を無効にしていた場合はビルドエラーになるため、両者の設定を一致させておく必要がある。

3.2.7.3 ビルド設定(共通)

ソースコードに含まれるプロジェクトファイル QtClient.pro を Qt Creator で開き、クライアントプログラムの設定を行う。

1. Qt Creator を起動し、File ⇒ **Open File or Project** を選択する。
2. **QtClient.pro** ファイルを **QTPBVR/QtClient** ディレクトリから選択する。
   PBVRClient/QtClient/QtClient.pro
3. **Active Project** “**QtClient**” を選択する。
4. **ビルドと実行**から OS に応じた設定項目の「**ビルド**」を選択する。
   a. Windows：Desktop Qt 6.2.4 MSVC2019 64bit
   b. macOS：Qt 6.2.4 for macOS (x86-64
   c. Linux：Desktop Qt 6.2.4 GCC 64bit

5. “Build Settings”を選択し、“Shadow Build”にチェックを入れる。
6. “Build Directory” にビルド後の実行ファイルを格納するディレクトリを指定する。例えば、“../build”を設定する。

図 3-7 クライアントプログラムの設定

3.2.7.4 ビルド設定(VR モードのみ)

VR モードの時は、プロジェクトの環境変数への追加設定が必要となる。

”Build Settings”における“Build Environment”で“Detail”を押下してセクションを展開し、環境変数として IMGUI_DIR（Dear ImGui がインストールされた場所）を追加する。

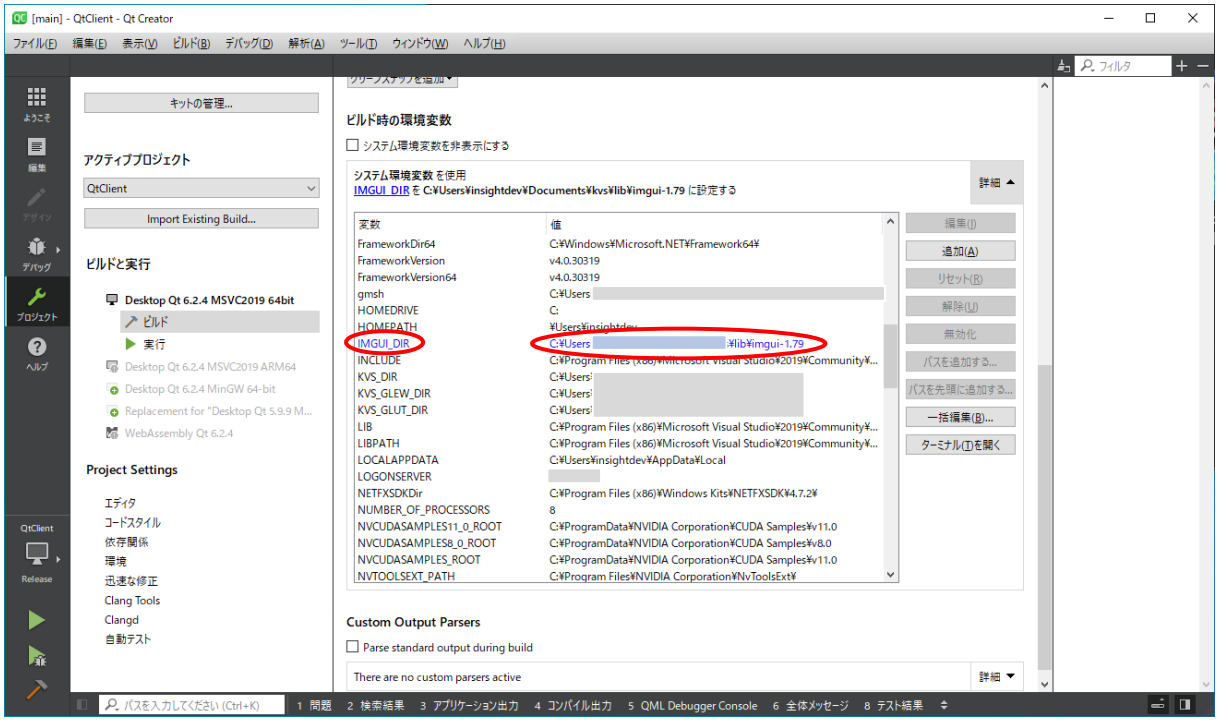


図 3-8　IMGUI_DIR 環境変数の設定

3.2.7.5 ビルド

クライアントプログラムをビルドする。

1. 左のツールバーで Edit を選ぶ。
2. QtClient プロジェクトを右クリックして、”Run qmake”を選択する。
3. QtClient プロジェクトを右クリックして、”Build”を選択する。

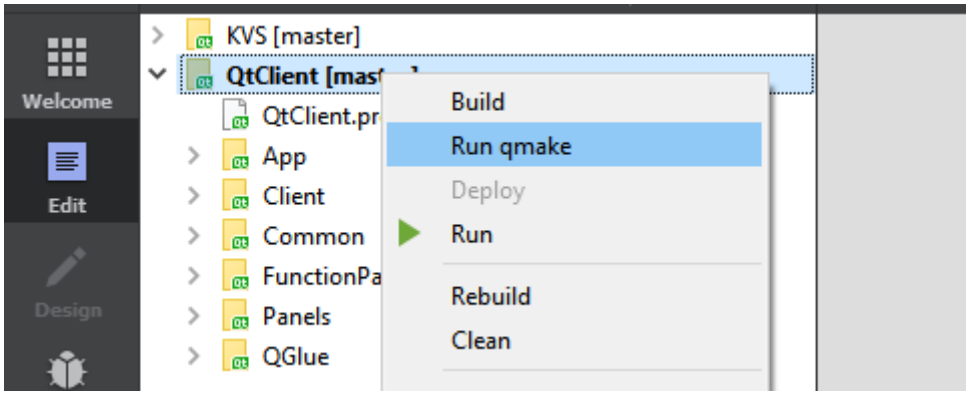


図 3-9　クライアントプログラムのビルド

３.２.８. Windows 環境におけるクライアントプログラムのデプロイ

この手順は Windows 環境下で、ビルドした PC から配布可能なアプリケーションをデプロイする場合にのみ必要とされる。

```
C:> cd C:¥PBVR_Dev¥QTPBVR¥build¥App
C:> windeployqt -release pbvr_client.exe
C:> cp C:¥PBVR_Dev¥OpenGL¥bin¥*.dll .
```

次に、インストール済みの KVS からシェーダプログラムを App フォルダにコピーする。

コピー元： /include/Core/Visualization/Shader/

コピー先： /App/include/Core/Visualization/Shader/

作成した App フォルダは他の Windows マシンで実行することができる。

３.２.９. CPU/GPU レンダラー

本プロジェクトはデフォルトで GPU レンダラーが生成される。しかしスパコンのような描画用の GPU を搭載していない環境のために、CPU レンダラーによる PBVR をビルドできる。
CPU レンダラーをビルドするには、pbvr_client -> QtClient -> qtpbvr.conf を開き、ビルドモードを指定する変数を以下のように変更する。

REND_MODE = GPU
を
REND_MODE = CPU
に変更する。

４ フィルタプログラム

フィルタプログラムはストレージ上の １ ヶ所にまとめられた時系列ボリュームデータを、PBVR サーバの並列処理を目的として分割し、可視化用のサブボリュームデータを生成するプログラムであり、PBVR とは独立な前処理プログラムである。データ分割には８分木構造モデルを適用し、構造格子および非構造格子データに対し、任意の８分木領域で分割を行い、PBVR サーバの並列処理に有効な入力となるファイル出力を行う。

４.１. 分割方式

八分木構造の領域分割は図 ４-1 に示すように、直方体の各辺を２分割し、１つに直方体を８個の直方体に分割する。この分割を再帰的に繰り返すことにより領域の細分化を行う。１つの直方体には８個の子となる直方体が存在し、子の直方体には１つの親となる直方体が存在することとなる。

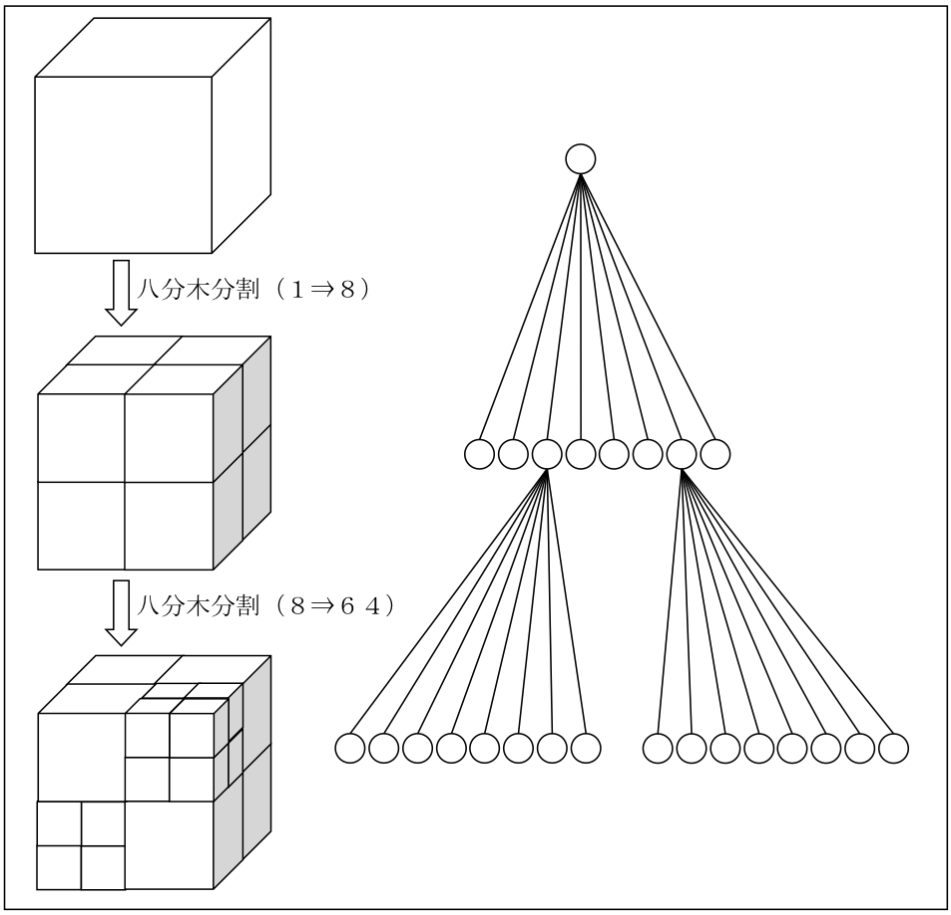


図 ４-1　八分木構造の領域分割

８ 分木の領域分割の座標値は、図 ４-2 に示すとおり、親となる直方体の頂点座標に対して、各軸の最小値と最大値および最小値と最大値の和を２で割ることによる中点で求める事ができる。各軸方

向の座標を各軸の中点との大小関係を比較することにより、ある点がどの領域（子の直方体）内の点となるかが判断可能となる。

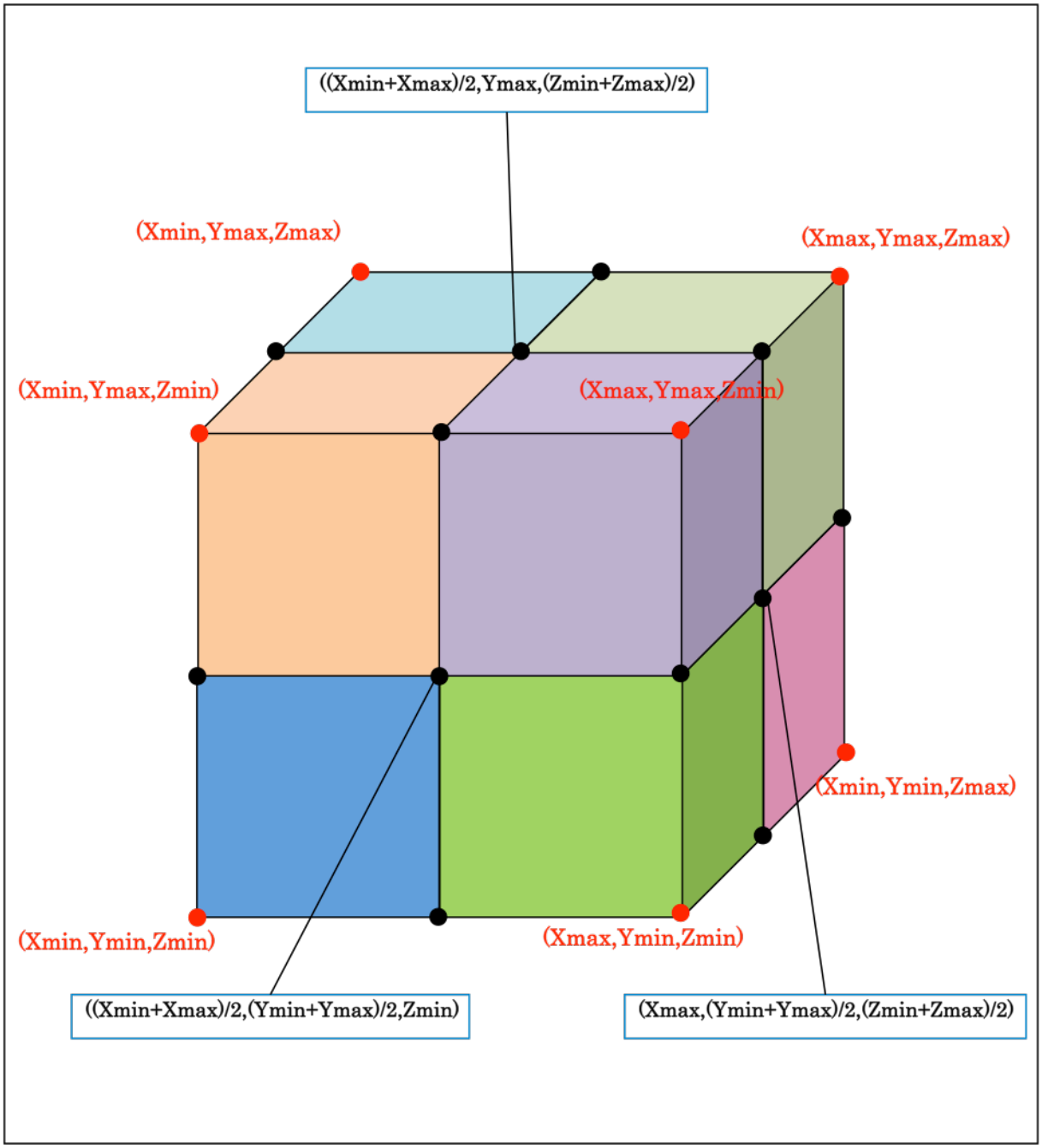


図 4-2　八分木構造の領域分割での分割点座標値

４.２. 起動方法

実行コマンド（filter）直後にパラメータを記述したパラメータファイル名（任意）を指定することによりパラメータが読みとられ実行される。パラメータファイル名が指定されない場合や、存在しないファイル名を指定した場合はエラーとなり正しく起動されない。

（MPI+OpenMP での起動方法、N にプロセス並列数を入力する）
$ mpiexec -n N filter param.txt

（OpenMP のみでの起動方法）
$ filter param.txt

※1. どちらの場合も OpenMP スレッド数は環境変数 OMP_NUM_THREADS で指定。

４.２.１. VTK データ用フィルタの起動

ロードモジュールパッケージに含まれる VTK データ用フィルタを起動するには、各環境に合わせた環境変数の設定が必要になる。

Linux

以下のように環境変数を設定する。

% export LD_LIBRARY_PATH=${VTK_LIB_PATH}:$LD_LIBRARY_PATH

Mac

以下のように環境変数を設定する。

% export DYLD_LIBRARY_PATH=${VTK_LIB_PATH}:$DYLD_LIBRARY_PATH

Windows

コントロールパネル->システム->詳細設定->環境変数　に移動し、以下の環境変数を設定する。

変数	値
Path	d:¥Program Files¥VTK 6.3.0¥bin

値は vtk のインストールディレクトリ配下の bin ディレクトリを指定する。

4.3. ファイル形式

フィルタの入出力ファイル形式に関して以降に示す。なお、出力されるファイルの中で、バイナリ形式のファイルは全てヘッダ、フッタなしの単精度であり、リトルエンディアンに統一されている。ファイル形式としては図 4-3 に示す SPLIT 形式（別名 KVSML 形式）、サブボリューム集約形式、ステップ集約形式の３種類が利用可能である。SPLIT 形式では全てのステップ・サブボリューム毎に独立なファイルを生成する。しかしながら、この方式では８分木の階層が増大するとファイル数が爆発的に増大するためサブボリューム毎に時間方向にファイルを集約するサブボリューム集約形式、および、ステップ毎に空間方向にファイルを集約するステップ集約形式を利用可能とした。ファイル形式の詳細を以下で説明する。

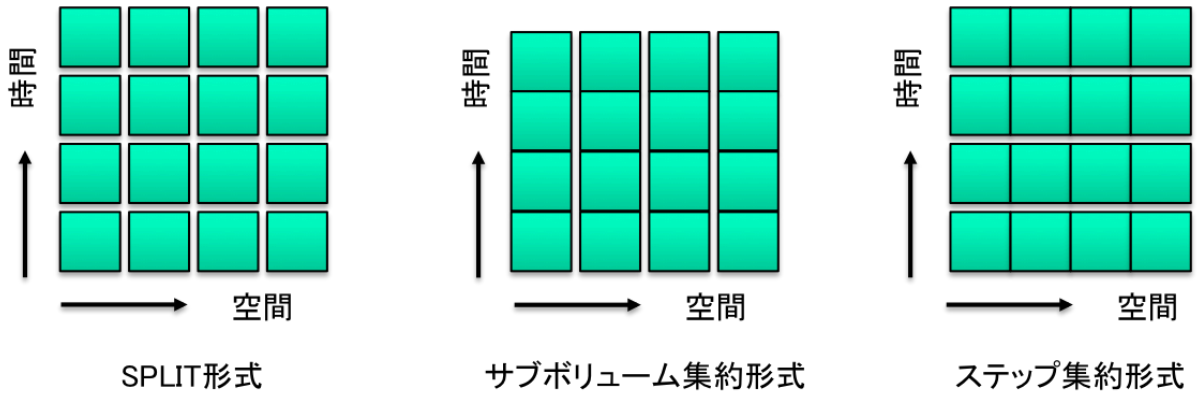


図 4-3　三つのファイル形式の模式図

4.3.1. 入力データ形式

フィルタプログラムで処理できる入力データは以下の通り。

（1）　AVSFLD バイナリデータ※1
（2）　AVSUCD アスキー・バイナリデータ※1
（3）　STL バイナリデータ※２
（4）　PLOT3D バイナリデータ※3
（5）　VTK Legacy バイナリデータ※4

※1.　AVS データフォーマットの詳細については http://www.cybernet.co.jp/avs/products/avsexpress/dataformat.html を参照。AVSUCD データは data 形式のみで、geom 形式、data_geom 形式は未対応。要素タイプは表 4.7-1 に示す２次元要素、３次元要素に対応しており、混合要素も利用可能。

※2.　STL データフォーマットの詳細については https://en.wikipedia.org/wiki/STL_(file_format)等を参照。

※3.　PLOT3D データフォーマットの詳細については http://ntrs.nasa.gov/archive/nasa

/casi.ntrs.nasa.gov/19900013774.pdf 等を参照。

※4. VTK データフォーマットの詳細については http://www.vtk.org/等を参照。VTK Structured Points、 VTK Structured Grid、 VTK Rectilinear Grid、 VTK UnstructuredGrid、および、 VTKPolygonalData を利用可能。

4.3.2. エンディアン

フィルタプログラムで扱うバイナリファイルは、リトルエンディアンに統一されている。リトルエンディアンのマシン上ではエンディアンに関して特別な処理は必要ないが、ビッグエンディアンのマシンでは可視化データをリトルエンディアンで出力する、あるいは、リトルエンディアンに変換した上でフィルタプログラムを実行する必要がある。

４.３.３. フィルタ出力情報ファイル(.pfi)

　このファイルはバイナリ形式ファイルである。サブボリューム分割前のモデル全体の情報を１つのファイルにまとめて出力する。

全ノード数(int)
全要素数(int)
要素タイプ(int)※1
ファイルタイプ(int)※2
ファイル数(int)※3
成分数(int)
開始ステップ(int)
最終ステップ(int)
サブボリューム数(int)※4
全体３次元空間Ｘ軸最小値(float)
全体３次元空間Ｙ軸最小値(float)
全体３次元空間Ｚ軸最小値(float)
全体３次元空間Ｘ軸最大値(float)
全体３次元空間Ｙ軸最大値(float)
全体３次元空間Ｚ軸最大値(float)
サブボリューム１ノード数(int)
サブボリューム２ノード数(int)
サブボリューム３ノード数(int)
:
サブボリュームｎノード数(int)
サブボリューム１要素数(int)
サブボリューム２要素数(int)
サブボリューム３要素数(int)
:
サブボリュームｎ要素数(int)
サブボリューム１　Ｘ軸最小値(float)
サブボリューム１　Ｙ軸最小値(float)
サブボリューム１　Ｚ軸最小値(float)
サブボリューム１　Ｘ軸最大値(float)
サブボリューム１　Ｙ軸最大値(float)
サブボリューム１　Ｚ軸最大値(float)
サブボリューム２　Ｘ軸最小値(float)
サブボリューム２　Ｙ軸最小値(float)
サブボリューム２　Ｚ軸最小値(float)
サブボリューム２　Ｘ軸最大値(float)
サブボリューム２　Ｙ軸最大値(float)
サブボリューム２　Ｚ軸最大値(float)
:
サブボリュームｎ　Ｘ軸最小値(float)
サブボリュームｎ　Ｙ軸最小値(float)
サブボリュームｎ　Ｚ軸最小値(float)
サブボリュームｎ　Ｘ軸最大値(float)
サブボリュームｎ　Ｙ軸最大値(float)
サブボリュームｎ　Ｚ軸最大値(float)
ステップ１　成分１　最小値
ステップ１　成分１　最大値
ステップ１　成分２　最小値

ステップ１　成分２ 最大値
：
ステップ１　成分Ｎ 最小値
ステップ１　成分Ｎ最大値
：
ステップｍ　成分１ 最小値
ステップｍ　成分１ 最大値
ステップｍ　成分２ 最小値
ステップｍ　成分２ 最大値
：
ステップｍ　成分Ｎ 最小値
ステップｍ　成分Ｎ最大値

※1. 要素タイプの定義は表 ４.７-1 参照。

※2. ファイルタイプは以下の３つの形式。

- ０　：SPLIT 形式
- １　：サブボリューム集約形式
- ２　・ステップ集約形式

※3. ファイルタイプが「サブボリューム集約形式」の場合のファイル総数を示す。

※4. サブボリューム数は 8^{n_layer} となる。

- n_layer=0　：１個
- n_layer=1　：８個
- n_layer=2　：64 個
- n_layer=3　：512 個
- n_layer=4　：4、096 個
- n_layer=5　：32、768 個
- n_layer=6　：262、144 個
- n_layer=7　：2、097、152 個

4.3.4. SPLIT ファイル形式

要素構成ファイルおよびノード座標ファイルは、サブボリューム単位に分割される。kvsml ファイルおよび成分ファイルは、サブボリューム単位、ステップ単位に分割される。全ファイル数は、

サブボリューム数×2+サブボリューム数×ステップ数×2

となる。（例：n_layer=7&step=100：423、624、704 個）

4.3.4.1 ファイル構成

prefix_XXXXX_YYYYYYY_ZZZZZZZ.kvsml ……kvsml ファイル（アスキー形式）
prefix_YYYYYYY_ZZZZZZZ_connect.dat ……要素構成ファイル（バイナリ形式）
prefix_YYYYYYY_ZZZZZZZ_coord.dat ……ノード座標ファイル（バイナリ形式）
prefix_XXXXX_YYYYYYY_ZZZZZZZ_value.dat ……成分ファイル（バイナリ形式）

XXXXX ：ステップ数（5 桁数値）
YYYYYYY ：サブボリューム番号（7 桁数値）
ZZZZZZZ ：全サブボリューム数（7 桁数値）

4.3.4.2 kvsml ファイル仕様

```
<?xml version="1.0" ?>
<KVSML>
    <Object type="UnstructuredVolumeObject">
        <UnstructuredVolumeObject cell_type="要素タイプ">
            <Node nnodes="サブボリューム内ノード数">
              <Value veclen="成分数">
                <DataArray type="float" file="prefix_XXXXX_YYYYYYY_ZZZZZZZ_value.dat" format="binary" />
              </Value>
              <Coord>
                  <DataArray type="float" file=" prefix_YYYYYYY_ZZZZZZZ_coord.dat" format="binary" />
              </Coord>
            </Node>
            <Cell ncells="サブボリューム内要素数">
              <Connection>
                  <DataArray type="uint"   file=" prefix_YYYYYYY_ZZZZZZZ_connect.dat" format="binary" />
              </Connection>
            </Cell>
        </UnstructuredVolumeObject>
    </Object>
</KVSML>
```

４.３.４.３ 要素構成ファイル仕様

要素１ 構成ノード１
要素１ 構成ノード２
：
要素１ 構成ノードｎ
要素２ 構成ノード１
要素２ 構成ノード２
：
要素２ 構成ノードｎ
要素３ 構成ノード１
要素３ 構成ノード２
：
要素３構成ノードｎ
：
要素Ｎ 構成ノード１
要素Ｎ 構成ノード２
：
要素Ｎ構成ノードｎ

４.３.４.４ ノード座標ファイル仕様

ノード１　Ｘ座標
ノード１　Ｙ座標
ノード１　Ｚ座標
ノード２　Ｘ座標
ノード２　Ｙ座標
ノード２　Ｚ座標
ノード３　Ｘ座標
ノード３　Ｙ座標
ノード３　Ｚ座標
：
：
ノードｍ　Ｘ座標
ノードｍ　Ｙ座標
ノードｍ　Ｚ座標

４.３.４.５ 成分ファイルの仕様

ノード１　成分１
ノード２　成分１
ノード３　成分１
：
ノードｎ　成分１
ノード１　成分２
ノード２　成分２
ノード３　成分２
：
ノードｎ　成分２
ノード１　成分ｍ
ノード２　成分ｍ
ノード３　成分ｍ
：
ノードｎ　成分ｍ

4.3.5. サブボリューム集約ファイル形式

要素構成、ノード座標および全ステップの成分が、サブボリューム単位に分割される。指定により複数サブボリューム情報を１ファイルに集約することも可能である。全ファイル数は、最大でサブボリューム数となる。（n_layer=7：2、097、152 個）

4.3.5.1 ファイル構成

prefix_YYYYYYY_ZZZZZZZ.dat ……（バイナリ形式）

prefix	：接頭辞
YYYYYYY	：ファイル番号（7 桁数値）
ZZZZZZZ	：全ファイル総数（7 桁数値）

4.3.5.2 ファイル仕様

（サブボリュームが１つの場合）

開始サブボリューム番号	サブボリューム情報
終了サブボリューム番号	
要素１ 構成ノード１	要素単位
要素１ 構成ノード２	
：	
要素１ 構成ノードｎ	
要素２ 構成ノード１	
要素２ 構成ノード２	
：	
要素２ 構成ノードｎ	
要素３ 構成ノード１	
要素３ 構成ノード２	
：	
要素３構成ノードｎ	
要素４ 構成ノード１	
要素４ 構成ノード２	
：	
要素４構成ノードｎ	
：	
：	
要素Ｎ 構成ノード１	
要素Ｎ 構成ノード２	
：	
要素Ｎ構成ノードｎ	
ノード１　Ｘ座標	ノード単位
ノード１　Ｙ座標	
ノード１　Ｚ座標	
ノード２　Ｘ座標	
ノード２　Ｙ座標	
ノード２　Ｚ座標	
ノード３　Ｘ座標	
ノード３　Ｙ座標	
ノード３　Ｚ座標	
ノード４　Ｘ座標	
ノード４　Ｙ座標	
ノード４　Ｚ座標	
：	
：	
ノードｍ　Ｘ座標	
ノードｍ　Ｙ座標	
ノードｍ　Ｚ座標	

要素構成

ノード座標

ステップ１　ノード１　成分１	成分単位
ステップ１　ノード２　成分１	
ステップ１　ノード３　成分１	
：	
ステップ１　ノードｎ　成分１	
ステップ１　ノード１　成分２	
ステップ１　ノード２　成分２	
ステップ１　ノード３　成分２	
：	
ステップ１　ノードｎ　成分２	
ステップ１　ノード１　成分ｍ	
ステップ１　ノード２　成分ｍ	
ステップ１　ノード３　成分ｍ	
：	
ステップ１　ノードｎ　成分ｍ	
：	
：	
：	
：	
ステップＭ　ノード１　成分１	
ステップＭ　ノード２　成分１	
ステップＭ　ノード３　成分１	
：	
ステップＭ　ノードｎ　成分１	
ステップＭ　ノード１　成分２	
ステップＭ　ノード２　成分２	
ステップＭ　ノード３　成分２	
：	
ステップＭ　ノードｎ　成分２	
ステップＭ　ノード１　成分ｍ	
ステップＭ　ノード２　成分ｍ	
ステップＭ　ノード３　成分ｍ	
：	
ステップＭ　ノードｎ　成分ｍ	

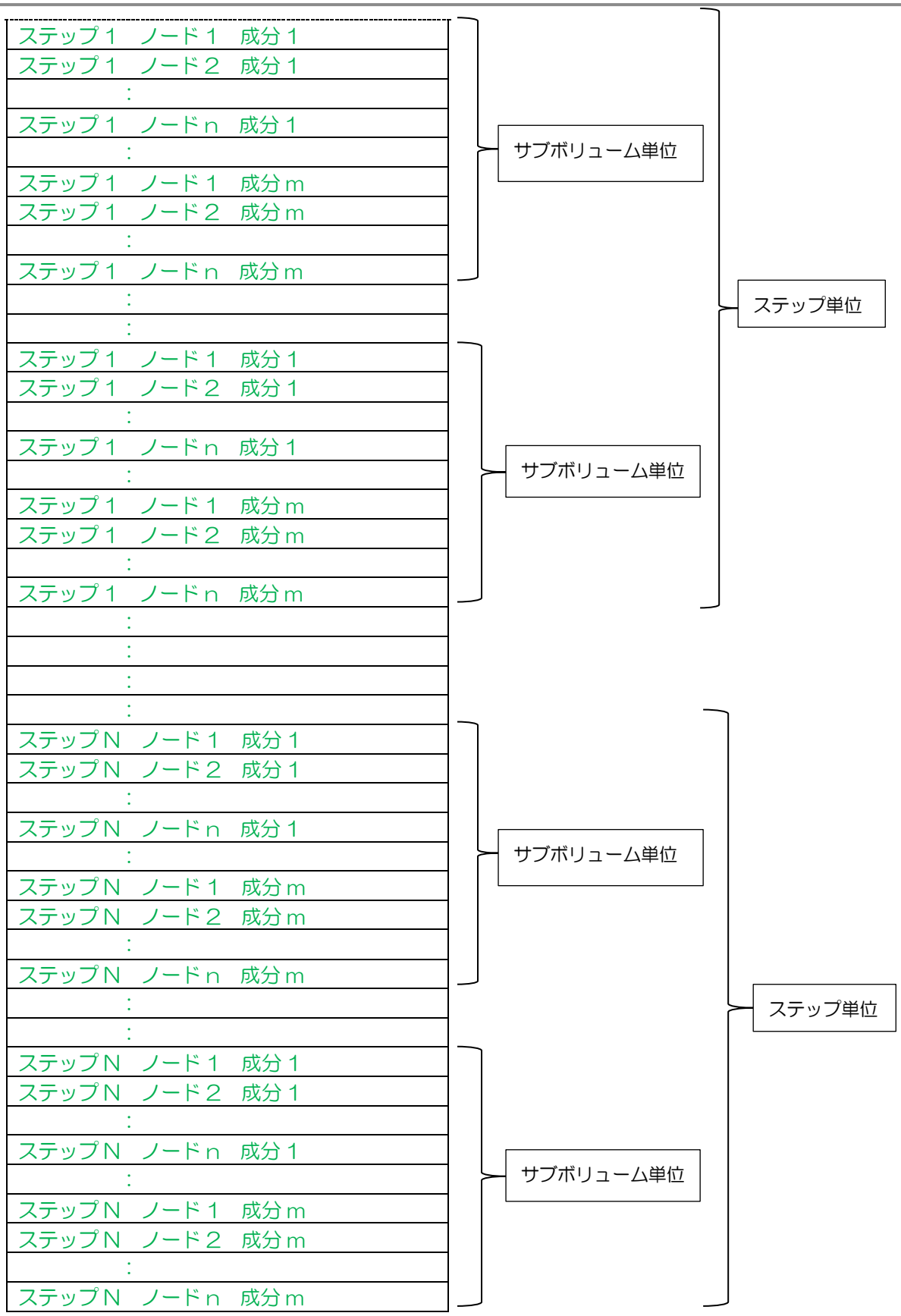
ステップ単位

（サブボリュームが複数の場合）

開始サブボリューム番号	サブボリューム情報
終了サブボリューム番号	
要素１ 構成ノード１	要素構成
要素１ 構成ノード２	
：	
要素１ 構成ノードｎ	
：	
要素Ｎ 構成ノード１	
要素Ｎ 構成ノード２	
：	
要素Ｎ構成ノードｎ	
ノード１　Ｘ座標	ノード座標
ノード１　Ｙ座標	
ノード１　Ｚ座標	
：	
ノードｍ　Ｘ座標	
ノードｍ　Ｙ座標	
ノードｍ　Ｚ座標	
：	
：	
：	
：	
：	
：	
：	
要素１ 構成ノード１	要素構成
要素１ 構成ノード２	
：	
要素１ 構成ノードｎ	
：	
要素Ｎ 構成ノード１	
要素Ｎ 構成ノード２	
：	
要素Ｎ構成ノードｎ	
ノード１　Ｘ座標	ノード座標
ノード１　Ｙ座標	
ノード１　Ｚ座標	
：	
ノードｍ　Ｘ座標	
ノードｍ　Ｙ座標	
ノードｍ　Ｚ座標	
ノードｍ　Ｘ座標	
ノードｍ　Ｙ座標	
ノードｍ　Ｚ座標	

サブボリューム１

サブボリュームｎ


ステップ１　ノード１　成分１
ステップ１　ノード２　成分１
：
ステップ１　ノードｎ　成分１
：
ステップ１　ノード１　成分ｍ
ステップ１　ノード２　成分ｍ
：
ステップ１　ノードｎ　成分ｍ
：
：
ステップ１　ノード１　成分１
ステップ１　ノード２　成分１
：
ステップ１　ノードｎ　成分１
：
ステップ１　ノード１　成分ｍ
ステップ１　ノード２　成分ｍ
：
ステップ１　ノードｎ　成分ｍ
：
：
：
：
ステップＮ　ノード１　成分１
ステップＮ　ノード２　成分１
：
ステップＮ　ノードｎ　成分１
：
ステップＮ　ノード１　成分ｍ
ステップＮ　ノード２　成分ｍ
：
ステップＮ　ノードｎ　成分ｍ
：
：
ステップＮ　ノード１　成分１
ステップＮ　ノード２　成分１
：
ステップＮ　ノードｎ　成分１
：
ステップＮ　ノード１　成分ｍ
ステップＮ　ノード２　成分ｍ
：
ステップＮ　ノードｎ　成分ｍ
サブボリューム単位
サブボリューム単位
サブボリューム単位
サブボリューム単位
ステップ単位
ステップ単位

4.3.6. ステップ集約ファイル形式

要素構成ファイルおよびノード座標ファイルは、それぞれが 1 つのファイルとなり（全サブボリュームの情報を１つのファイルに集約）、成分ファイルは、ステップ単位に分割される。全ファイル数は、ステップ数＋2 となる。

4.3.6.1 ファイル構成

prefix_connect.dat　　……要素構成ファイル（バイナリ形式）
prefix_coord.dat　　……ノード座標ファイル（バイナリ形式）
prefix_XXXXX_value.dat……成分ファイル（バイナリ形式）

prefix　：接頭辞
XXXXX ：ステップ数（5 桁数値）

4.3.6.2 要素構成ファイル仕様

サブボリューム１　要素１ 構成ノード１	要素単位
サブボリューム１　要素１ 構成ノード２	
：	
サブボリューム１　要素１ 構成ノードｎ	
サブボリューム１　要素２ 構成ノード１	サブボリューム単位
サブボリューム１　要素２ 構成ノード２	
：	
サブボリューム１　要素２ 構成ノードｎ	
サブボリューム１　要素３ 構成ノード１	
サブボリューム１　要素３ 構成ノード２	
：	
サブボリューム１　要素３構成ノードｎ	
：	
サブボリューム１　要素Ｎ 構成ノード１	
サブボリューム１　要素Ｎ 構成ノード２	
：	
サブボリューム１　要素Ｎ構成ノードｎ	
：	
サブボリュームＭ　要素１ 構成ノード１	
サブボリュームＭ　要素１ 構成ノード２	
：	
サブボリュームＭ　要素１ 構成ノードｎ	
サブボリュームＭ　要素２ 構成ノード１	
サブボリュームＭ　要素２ 構成ノード２	
：	
サブボリュームＭ　要素２ 構成ノードｎ	
サブボリュームＭ　要素３ 構成ノード１	
サブボリュームＭ　要素３ 構成ノード２	
：	
サブボリュームＭ　要素３構成ノードｎ	
：	
サブボリュームＭ　要素Ｎ 構成ノード１	
サブボリュームＭ　要素Ｎ 構成ノード２	
：	
サブボリュームＭ　要素Ｎ構成ノードｎ	

4.3.6.3 ノード座標ファイル仕様

サブボリューム１　ノード１　Ｘ座標
サブボリューム１　ノード１　Ｙ座標
サブボリューム１　ノード１　Ｚ座標
サブボリューム１　ノード２　Ｘ座標
サブボリューム１　ノード２　Ｙ座標
サブボリューム１　ノード２　Ｚ座標
サブボリューム１　ノード３　Ｘ座標
サブボリューム１　ノード３　Ｙ座標
サブボリューム１　ノード３　Ｚ座標
：
：
サブボリューム１　ノードｍ　Ｘ座標
サブボリューム１　ノードｍ　Ｙ座標
サブボリューム１　ノードｍ　Ｚ座標
：
：
：
サブボリュームＭ　ノード１　Ｘ座標
サブボリュームＭ　ノード１　Ｙ座標
サブボリュームＭ　ノード１　Ｚ座標
サブボリュームＭ　ノード２　Ｘ座標
サブボリュームＭ　ノード２　Ｙ座標
サブボリュームＭ　ノード２　Ｚ座標
サブボリュームＭ　ノード３　Ｘ座標
サブボリュームＭ　ノード３　Ｙ座標
サブボリュームＭ　ノード３　Ｚ座標
：
：
サブボリュームＭ　ノードｍ　Ｘ座標
サブボリュームＭ　ノードｍ　Ｙ座標
サブボリュームＭ　ノードｍ　Ｚ座標

ノード単位

サブボリューム単位

4.3.6.4 成分ファイル仕様

サブボリューム１　ノード１　成分１	成分単位
サブボリューム１　ノード２　成分１	
サブボリューム１　ノード３　成分１	
：	
サブボリューム１　ノードｎ　成分１	
サブボリューム１　ノード１　成分２	サブボリューム単位
サブボリューム１　ノード２　成分２	
サブボリューム１　ノード３　成分２	
：	
サブボリューム１　ノードｎ　成分２	
サブボリューム１　ノード１　成分ｍ	
サブボリューム１　ノード２　成分ｍ	
サブボリューム１　ノード３　成分ｍ	
：	
サブボリューム１　ノードｎ　成分ｍ	
：	
：	
：	
サブボリュームＭ　ノード１　成分１	
サブボリュームＭ　ノード２　成分１	
サブボリュームＭ　ノード３　成分１	
：	
サブボリュームＭ　ノードｎ　成分１	
サブボリュームＭ　ノード１　成分２	
サブボリュームＭ　ノード２　成分２	
サブボリュームＭ　ノード３　成分２	
：	
サブボリュームＭ　ノードｎ　成分２	
サブボリュームＭ　ノード１　成分ｍ	
サブボリュームＭ　ノード２　成分ｍ	
サブボリュームＭ　ノード３　成分ｍ	
：	
サブボリュームＭ　ノードｎ　成分ｍ	

4.4. パラメータファイル

パラメータファイルは（AVSFLD/UCD、PLOT3D、STL 用）フィルタ、および、VTK 用フィルタで共通に使用されるアスキー形式のファイルである。フィルタ実行時の引数としてファイル名を指定することにより、そのファイルが入力パラメータとして解釈され、パラメータがセットされる。

表 4.4-1　フィルタ入力パラメーター覧表

パラメータ名	パラメータ内容	デフォルト値	備考
in_dir	入力ファイルディレクトリ	“.”	入力ファイルのディレクトリパス※1
field_file	AVSFLD ファイル名	-	※2、※3、※4
stl_binary_file	STL ファイル名	-	※2
plot3d_config_file	PLOT3D 設定ファイル名	-	※2、※3
vtk_file	VTK ファイル名	-	※2、※3、※5
vtk_in_prefix	VTK 時系列ファイル名の接頭辞	-	※2、※3、※5
vtk_in_suffix	VTK 時系列ファイル名の接尾辞	-	※2、※3、※5
ucd_inp	AVSUCD ファイル名	-	アスキー形式※2
in_prefix	AVSUCD 時系列ファイル名の接頭辞	-	バイナリ形式※2
in_suffix	AVSUCD 時系列ファイル名の接尾辞	-	バイナリ形式※2
format	VTK、AVSUCD 時系列ステップ数形式	“%05d”	
out_dir	出力ファイルディレクトリ	“.”	出力ファイルのディレクトリパス※1
out_prefix	出力ファイル名の接頭辞	“output.”	
start_step	開始ステップ数	1	※6
end_step	終了ステップ数	1	※6
n_layer	八分木分割数	0	0～7 までの整数
output_type	ファイル形式	0	0:SPLIT 形式 1:サブボリューム集約 2:ステップ集約 上記以外出力なし
file_number	出力ファイル数	0	0 以上の整数で、0 の場合はサブボリューム数となる サブボリューム集約ファイルの場合のみ有効
mpi_volume_div	サブボリューム分割数	1	サブボリュームの分割数※7
mpi_step_div	ステップ分割数	1	ステップの分割数※7

mpi_div	MPI 並列における分割軸	2	0:ステップ分割数、サブボリューム分割数により決定 1:サブボリューム分割軸を優先 2:ステップ分割軸を優先 mpi_volume_div と mpi_step_div が指定された場合は無効
multi_elem_type	非構造格子の混合要素データフラグ	0	0:単一タイプの要素のみで構成 1:複数タイプの要素で構成
temp_delete	混合要素データ実行時の一時ファイル削除指示	1	0:一時ファイルを残す 1:一時ファイルを消す

※1. ディレクトリ指定は、絶対・相対パスでの指定が可能であるが、~(チルダ)での指定は不可。

※2. field_file、stl_binary_file、vtk_file、vtk_in_prefilx（suffix）、ucd_inp、in_prefix（suffix）のどれか一つを指定すること。

※3. 入力データが構造格子の場合は、3 次元では 6 面体一次要素（非構造格子）、2 次元では 4 角形一次要素（非構造格子）として出力される。

※4. nstep、ndim、dim1、dim2、dim3、veclen、coord [123]、variable に関するパラメータのみ参照。

※5. VTK Legacy 形式については、5 つのデータ形式（VTK Structured Points、VTK Structured Grid、 VTK Rectilinear Grid、 VTK UnstructuredGrid、および、VTKPolygonalData）をプログラムが自動判別する。

※6. 時系列データのみで指定すること。

※7. mpi_volume_div と mpi_step_div が指定された場合、その積とプロセス数が一致しない場合はエラーとなる。

4.4.1. PLOT3D 設定ファイル

PLOT3D 設定ファイルによって PLOT3D データのファイル形式を記述する。ここで、usebytecount は Fortran バイナリで 1、C バイナリで 0 を設定する。

表 4.4-2　PLOT3D 設定ファイルパラメーター覧

パラメータ名	パラメータ内容	デフォルト値
coordinate_file_prefix	座標ファイル名の接頭辞	-
coordinate_file_suffix	座標ファイル名の接尾辞	-
coordinate_mode_precision	精度（float｜double）	double
coordinate_mode_usebytecount	1 for true、 0 for false	true
coordinate_mode_endian	エンディアン（little｜big）	little
coordinate_mode_iblanks	1 for true、 0 for false	false
solution_file_prefix	ソリューションファイル名の接頭辞	-
solution_file_suffix	ソリューションファイル名の接尾辞	-
solution_mode_precision	精度（float｜double）	double
solution_mode_usebytecount	1 for true、 0 for false	true
solution_mode_endian	エンディアン（little｜big）	little
function_file_prefix	ファンクションファイル名の接頭辞	-
function_file_suffix	ファンクションファイル名の接尾辞	-
function_mode_precision	精度（float｜double）	double
function_mode_usebytecount	1 for true、 0 for false	true
function_mode_endian	エンディアン（little｜big）	little

4.5. MPI 並列処理

MPI 並列処理における分割数の決定方法を説明する。以下では 50 ステップ×8 サブボリュームのデータ処理を考える。

（1）　ステップ分割軸優先

- プロセス数がステップ数以下の時は、全ステップをプロセス数で分割する。
  例）8 プロセスで起動する場合の各プロセスの担当領域は 6 ステップ×8 サブボリューム、もしくは、7 ステップ×8 サブボリュームとなる。
- プロセス数がステップ数より大きい時は、フィルタ処理を行うプロセス数は、ステップ数の整数倍のプロセス数とし、サブボリューム数側の分割も行う。
  例）128 プロセスで起動する場合、50×2=100 プロセス（余り 28 プロセス）の処理となり、各プロセスの担当領域は 1 ステップ×4 サブボリュームとなる。

（2）　サブボリューム分割軸優先

- プロセス数がサブボリューム数以下の時は、全サブボリュームをプロセス数で分割する。
  例）8 プロセスで起動する場合の各プロセスの担当領域は 50 ステップ×1 サブボリュームとなる。
- プロセス数がサブボリューム数より大きい時は、フィルタ処理を行うプロセス数は、サブボ

リューム数の整数倍のプロセス数とし、ステップ数側の分割も行う。
例）128 プロセスで起動する場合、8×16=128 プロセス（余り０プロセス）の処理となり、各プロセスの担当領域は６ステップ×1 サブボリューム、もしくは、７ステップ×1 サブボリュームとなる。

（３） ユーザ指定分割

- パラメータファイルによる指定では、分割数の積とプロセス数が一致しない場合はエラーとする。

４.６. ステージング環境での実行方法

スーパーコンピュータにおけるステージング環境での実行方法に関して記述する。パラメータファイルの指定とファイル転送の指定に関しては整合性を図り、フィルタプログラムを起動する必要がある。また、フィルタプログラムの出力形式によっては、複数プロセスから１つのファイルに出力を行う場合がある。そのような場合には出力先を複数プロセスからアクセス可能な共有ディレクトリを指定する必要がある。

4.6.1.実行シェルとパラメータファイル

```
#!/bin/bash -x
#
#PJM --rsc-list "elapse=01:00:00"
#PJM --rsc-list "node=64"
#PJM --rsc-list "rscgrp=small"
#PJM --stg-transfiles all
#PJM --mpi "proc=64"
#PJM --mpi "use-rankdir"
#PJM --stgin  "rank=* ./filter             %r:./"          ……①
#PJM --stgin  "rank=* ./param.txt          %r:./"          ……②
#PJM --stgin  "rank=0 /data/ucd/ucd*.dat    0:../"         ……③
#PJM --stgout "rank=* %r:../output*.dat        ./"         ……④
#PJM --stgout "rank=* %r:./pbvr_filter.*      ./LOG/"      ……⑤
#PJM -S

. /work/system/Env_base

export PARALLEL=8
export OMP_NUM_THREADS=8

mpiexec -n 64 lpgparm -p 4MB -s 4MB -d 4MB -h 4MB -t 4MB filter param.txt    ……⑥
```

① 実行モジュールを各プロセスのランクディレクトリに転送
② パラメータファイルを各プロセスのランクディレクトリに転送
③ 入力データを共有ディレクトリに転送（パラメータファイル内で指定するファイル）
④ 出力データを共有ディレクトリからローカルディレクトリに転送
⑤ ログおよびエラーファイルをランクディレクトリからローカルディレクトリに転送
⑥ 各プロセスのランクディレクトリ内のパラメータファイルを引数として各プロセスのランクディレクトリ内の実行モジュールを起動。

```
#
in_dir=../                          ……⑦
field_file=pd3d.fld
out_prefix=case0
out_dir=./                          ……⑧
file_type=0                         ……⑨
n_layer=3
start_step=0
end_step=511
```

⑦ 入力データのパスを指定（ステージング環境では相対パスによる指定、上記例では、共有ディレクトリから入力データを読み込む）

⑧ 出力データのパスを指定（ステージング環境では相対パスによる指定。上記例では、各プロセス内のランクディレクトリへ出力）

⑨ ファイル出力形式の指定（上記例では、SPLIT 形式）

4.6.2. 入出力ファイルとディレクトリ

ステージング環境におけるフィルタプログラムの入出力ファイルとディレクトリの関係を以下に示す。入出力先を共有ディレクトリに指定した場合は問題ない（ログ＆エラーファイルを除く）。出力データに関しては、SPLIT 形式のみランクディレクトリに出力可能である。他のファイル形式ではデータ集約処理のため共有ディレクトリの使用が必要となる。

表 4.6-1　京における入出力ファイルとディレクトリの対応表

IO	種　別		ランクディレクトリ	共有ディレクトリ
入力	パラメータファイル		○※1	○
	入力データ		○※2	○
出力	出力データ	SPLIT 形式	○	○
		ステップ集約形式	×	○
		サブボリューム集約形式	×	○
	ログ＆エラーファイル		○※3	×

※1. パラメータファイルはランク０のみが入力

※2. 全プロセスのランクディレクトリに全入力ファイルの転送が可能な場合。

※3. 出力先はランクディレクトリに固定

4.7. 要素タイプの混合した非構造格子の実行

複数要素タイプを含んだ非構造格子のボリュームデータの場合、フィルタプログラムでは、まず単一要素タイプ毎に分割した UCD バイナリデータを生成し、その後、その単一要素タイプの UCD バイナリデータをフィルタプログラムでサブボリューム分割をし、PBVR サーバの入力となるファイル出力を行う。

次に示すようにパラメータファイル内のパラメータ名「multi_element_type」に 1 を設定することにより、フィルタプログラムは要素タイプ毎に分割したサブボリューム分割ファイルの出力を行う。

```
#
in_dir=.
in_prefix=MULTI
in_suffix=.dat
out_dir=.
out_prefix=div
out_prefix=.dat
format=%03
start_step=1
end_step=20
multi_element_type=1
```

要素タイプ毎に出力されるファイルは、ファイル名の先頭に要素タイプの 2 桁のコードが付加されたファイル名となる。以下に要素名と要素タイプコードの一覧を示す。

表 4.7-1　要素タイプ一覧表

要素名	要素タイプコード
３角形	2
４角形	3
４面体	4
ピラミッド	5
プリズム	6
６面体	7
３角形２次	9
４角形２次	10
４面体２次	11
６面体２次	14

上記パラメータファイルで、４面体一次要素と４面体二次要素から構成されるボリュームデータの場合には以下のような出力ファイル名となる。

表 4.7-2　ファイル分割でのファイル名

オリジナル混合データ		４面体一次データ	４面体二次データ
MULTI001.dat		04-div001_～	11-div001_～
MULTI002.dat		04-div002_～	11-div002_～
MULTI003.dat	⇒	04-div003_～	11-div003_～
MULTI004.dat	分割	04-div004_～	11-div004_～
MULTI005.dat		04-div005_～	11-div005_～
：		：	：
MULTI020.dat		04-div020_～	11-div020_～

５ サーバ

サーバはフィルタの出力データであるサブボリュームファイルを読み込み、PBVR 法で利用する可視化用粒子データを生成する。可視化用粒子データはソケット通信を介してクライアントプログラムに送信される。

５.１. 起動方法

サーバプログラムはスーパーコンピュータ上で粒子データのみをバッチ処理で生成するバッチモード、および、ソケット通信可能なサーバとクライアントを接続して対話処理により粒子データを生成するクライアント・サーバモード（クラサバモード）による処理が可能である。PC やワークステーション上のスタンドアロン処理はクラサバモードでクライアントプログラムとサーバプログラムを同一マシン上で起動することにより実現する。サーバの起動方法を以下に示す。

（MPI+OpenMP での起動方法、N にプロセス並列数を入力する）
$ mpiexec -n N pbvr_server　　※1、２

（OpenMP のみでの起動方法）
$ pbvr_server

※1. MPI+OpenMP 版サーバはマスター・スレーブ型で動作するので、並列数 N はスレーブの並列数+１で指定。
※2. どちらの方法の場合も OpenMP スレッド数は環境変数 OMP_NUM_THREADS で指定。

Windows 環境でのサーバは MSVC の x64 Native Tools コマンドプロンプトからのコマンドを実行により起動される。

（サーバ起動）
Windows> pbvr_server.exe

遠隔 PC とユーザ PC のソケット通信の方法は（５.２）に、クライアントの起動方法は（６.１）に記載する。

表 5.1-1　サーバのコマンドラインオプション一覧 ※1

オプション	適用先	指定値	デフォルト	機能
-h	CS、B	-	-	オプションおよび指定可能パラメータの一覧を表示
-B	B	-	-	バッチモードを指定する
-pa	B	ファイル名	-	可視化パラメータファイルを指定する ※2
-pd	B	実数値	1.0	画像の濃さを指定する
-S	B	u、 m、 r	u	粒子サンプリング方法 u: uniform sampling、 m: metropolis sampling r: rejection sampling
-plimit	B	1 ～ 99999999	1000000	粒子数制限値
-vin	B	ファイル名	-	入力ボリュームデータファイル※3
-pout	B	ファイル名	-	出力粒子データファイル名※4
-p	CS	ポート番号	60000	ソケット通信ポート番号
-viewer	B	100～9999 × 100 ～ 9999	620×620	ビューワ解像度
-Bd	B	-	-	タイムステップ毎に粒子ファイルを1つにまとめず、サブボリューム毎に出力する
-Bs	B	0 以上の整数値	指定した PFI ファイル群の先頭ステップ	処理対象とするタイムステップ範囲の開始ステップを指定する
-Be	B	0 以上の整数値	指定した PFI ファイル群の最終ステップ	処理対象とするタイムステップ範囲の終了ステップを指定する

※1. 適用先の凡例は、CS：クラサバモード、B：バッチモードである。

※2. 可視化パラメータファイルの詳細は6.3.2を参照。

※3. フィタ処理後のボリュームデータから生成される.pfi ファイル、もしくは分散処理用の.pfl ファイル(5.1.1.1 を参照)を絶対パス、もしくは、相対パスで指定する。拡張子を省略しないこと。このオプションは、-pa で指定するパラメータファイル内のオプションよりも優先される。

※4. 指定されたファイル名をプレフィックスとして
“ファイル名_時刻番号_サブボリューム数_サブボリューム番号.kvsml”
という形式の粒子ファイル群を生成する。

5.1.1. バッチモードでの起動

サーバ起動時のコマンドラインオプションとして -B を指定すると、バッチモードで起動する。バッチモードでの起動例（MPI+OpenMP 版）を以下に示す。

```
$ mpiexec -n 5 pbvr_server -B -vin ./data/case.pfi -pout ./output/case -pa ./param.in
```

上記の例では、入力ボリュームデータファイル./data/case.pfi を可視化パラメータファイル./param.in に記述されたパラメータによって処理を行い、以下の粒子データを出力する。

```
./output/case_XXXXX_YYYYYYY_ZZZZZZZ.kvsml
./output/case _XXXXX_YYYYYYY_ZZZZZZZ_colors.dat
./output/case _XXXXX_YYYYYYY_ZZZZZZZ_coords.dat
./output/case _XXXXX_YYYYYYY_ZZZZZZZ_normals.dat
```

ここで、XXXXX は時刻、YYYYYYY はサブボリューム番号、ZZZZZZZ は全サブボリューム数を示し、colors、coords、normals はそれぞれ色、座標、法線ベクトルのデータを示す。

通常は時刻ごとに全てのサブボリュームが統合されるので YYYYYYY、ZZZZZZZ 共に 1 のみだが、統合を行わずサブボリューム毎に粒子データを出力したい場合は、バッチモードのサーバ起動時のコマンドラインオプション -Bd を指定する

バッチモードのサーバ起動時のコマンドラインオプション -pa で指定する可視化パラメータファイルはクラサバモードの対話処理によって生成する。このファイルをそのまま利用、あるいは、パラメータを編集し、バッチ処理で大量データの処理を行う。

5.1.1.1 分散ファイルの処理

ストレージに分散して保存された複数のボリュームデータを統合して可視化可能である。複数のボリュームデータは一つ一つフィルタ処理し、それぞれに pfi ファイルを作成する。そして pfi ファイルを列挙した pfl ファイルを作成し、-vin オプションで pfl ファイルを指定する。

pfl ファイルの先頭には「#PBVR PFI FILES」の記述が必要で、以降、1 行毎に pfi ファイルを絶対パス、もしくは、pfl ファイルのあるディレクトリからの相対パスで記述する。

pfl ファイルの記述例を示す。

```
#PBVR PFI FILES
hex_filter_out/hex.pfi
hex2_filter_out/hex2.pfi
```

5.1.2. クライアント・サーバモードでの起動

サーバ起動時のコマンドラインオプションとして -B を指定しなかった場合は、クライアント・サーバモードで起動する。以下に起動例を示す。

```
$ mpiexec -n 5 pbvr_server
first reading time[ms]:0
Server initialize done
Server bind done
Server listen done
Waiting for connection ...
```

上記のようにクライアントとのソケット通信の接続待ちになったら、別の端末からクライアントを起動する。なお、入力ボリュームデータ名はクライアント側で指定する。

サーバ起動時のポート番号の省略値は 60000 である。ポート番号は、以下のように起動時のコマンドラインオプション-p で変更できる。

```
$mpiexec -n 5 pbvr_server -p 55555
```

5.2. ソケット通信によるクライアント・サーバの接続

クライアントプログラムとサーバプログラムは、ポートフォワードで接続したポートを利用してソケット通信を行い、粒子データや可視化パラメータを送受信する。以下では ssh を用いたポートフォワードの方法を記述する。

5.2.1. ローカル接続

この節では 1 台のマシン（machineA）上でクライアントプログラムとサーバプログラムをスタンドアロンで起動する例を示す。この例ではクライアントプログラムとサーバプログラムの両方が machineA におけるデフォルトのポート番号 60000 を用いて連携する。

手順 1[サーバ起動]

ターミナル上で以下のコマンドを実行

```
machineA> mpiexec -n 5 pbvr_server
```

手順 2[クライアント起動]

サーバと別のターミナルを起動し以下のコマンドを実行

```
machineA> pbvr_client -vin filename
```

5.2.2. リモート接続

この節では手元のマシン（machineA）と遠隔地のマシン（machineB）を ssh ポートフォワードで接続し、machineA 上でクライアントプログラムを、machineB 上でサーバプログラムを起動する例を示す。基本的には ssh ポートフォワードが確立した後の起動方法はスタンドアロンと同じである。

ssh ポートフォワードとは、ssh によって疎通した通信経路（ssh トンネル）を利用してネットワーク上のポートを転送する仕組みである。ポートフォワードには、ローカルフォワード、リモートフォ

ワード、ダイナミックフォワードの 3 種類あるが、本システムではクライアント PC のポートを ssh サーバ経由でターゲットへ接続するローカルフォワードを利用する。以下に ssh ローカルフォワードの概要とコマンドを示す。

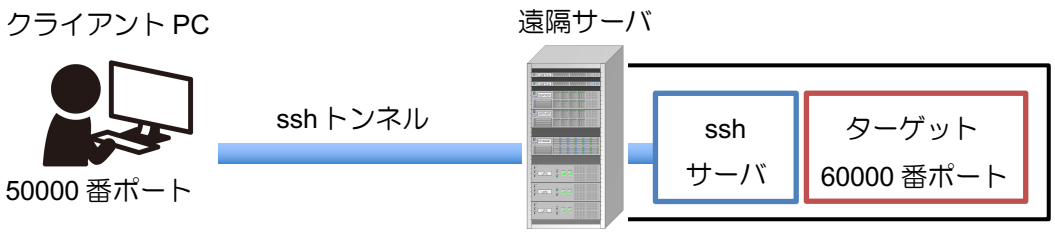


図 5-1　ローカルフォワードの概要

【ssh ローカルフォワードのコマンド】

クライアント PC>　ssh -L 50000:ターゲットのアドレス:60000 遠隔サーバのアドレス

ssh ローカルフォワードのコマンドにおいて、遠隔サーバとターゲットが一致している場合、ターゲットのアドレスは“localhost”となる。また、この例ではポート番号として 50000 番と 60000 番が利用されているが、TCP や UDP に割り当てられているポート番号をのぞいて任意のポート番号が利用可能である。

ssh ポートフォワードを用いて machineA の 50000 番ポートを machineB の 60000 番ポートに接続し、各マシンでクライアントプログラムとサーバプログラムを起動する例を示す。

手順 1[ssh ポートフォワード]

machineA> ssh -L 50000:localhost:60000 username@machineB

（machineA の 50000 番ポートを machineB の 60000 番ポートに接続。machintB 自身がターゲットであるので、ターゲットのアドレスは localhost となる。）

手順 2[サーバ起動]

machineB> mpiexec -n 5 pbvr_server

手順 3[クライアント起動]

machineA> pbvr_client -vin filename

machineB のログインノードから対話ノード interactB に入り、ssh ポートフォワードを用いて machineA の 50000 番ポートを interactB の 60000 番ポートに接続し、各マシンでクライアントプログラムとサーバプログラムを起動する例を示す。

手順 1[ssh ポートフォワード]

machineA> ssh -L 50000:interactB:60000 username@machineB

（machineA の 50000 番ポートを、machineB を経由して、interactB の 60000 番ポートに接続）

手順 2[サーバ起動]

machineB> mpiexec -n 5 pbvr_server

手順 3[クライアント起動]

machineA> pbvr_client -vin filename

5.2.3. ssh ポートフォワーディングの接続確認方法

ssh ポートフォワーディングによる接続ができているかどうかを確認するにあたり、クライアント側からタイプインした文字列をサーバ側へエコー出力するだけの単純なソケット通信の動作確認用プログラム（サーバ側プログラム名：server、クライアント側プログラム名：client）を利用すると便利である。このプログラムは以下から入手できる。

「C for Linux 2」小俣光之著（株)秀和システム 2005 年 9 月　サポートページ
http://www.ncad.co.jp/~komata/c4linux2/

サーバ側起動方法
server ポート番号

クライアント側起動方法
client サーバホスト名　ポート番号

5.2.4. ssh クライアントによるリモート接続

Windows 環境において SSH クライアントソフト Tera Term を用いたポートフォワード（ローカルフォワード）の方法を示す。

（1） まず Tera Term を起動する。“新しい接続”ウインドウが立ち上がるが、“キャンセル”を選択する。

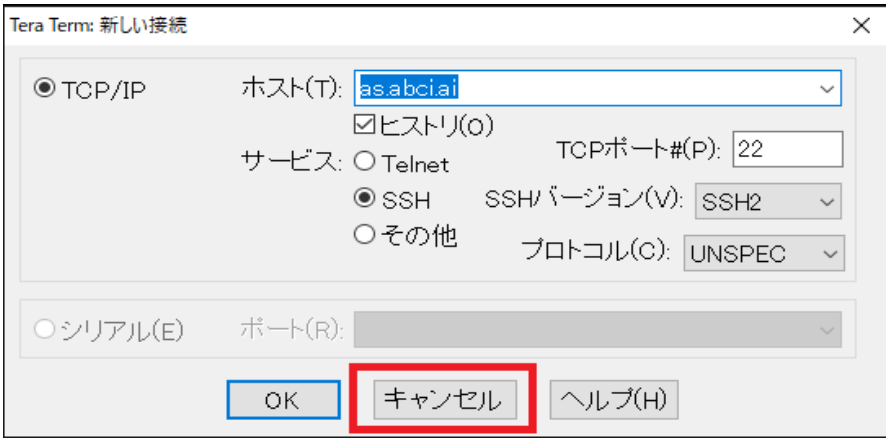


図 5-2 “新しい接続”ウインドウ

ツールバーから“設定”の“SSH 転送”を選択する。

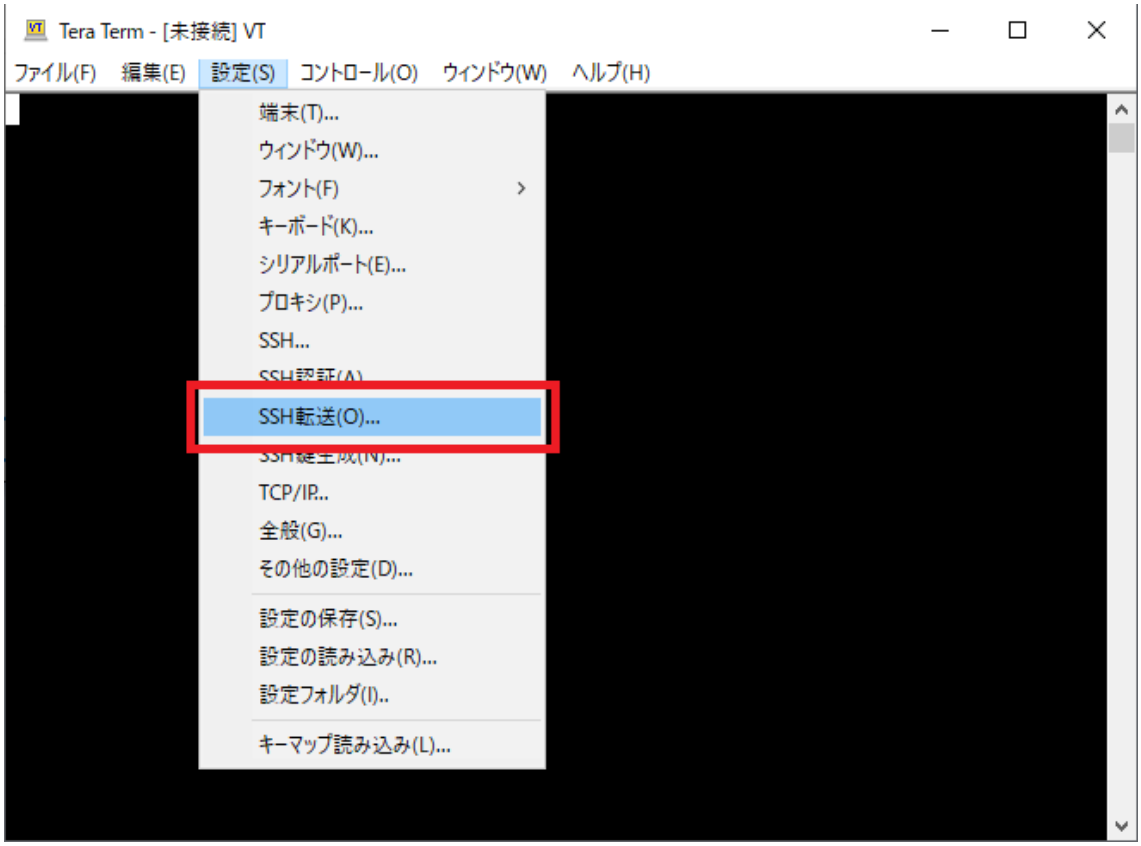


図 5-3　ツールバーの設定タブ

（2）　“SSH ポート転送”ウィンドウが立ち上がる。“追加”を選択する。ここで、「リモートの X アプリケーションをローカルの X サーバに表示する」のチェックボックスをチェックする。

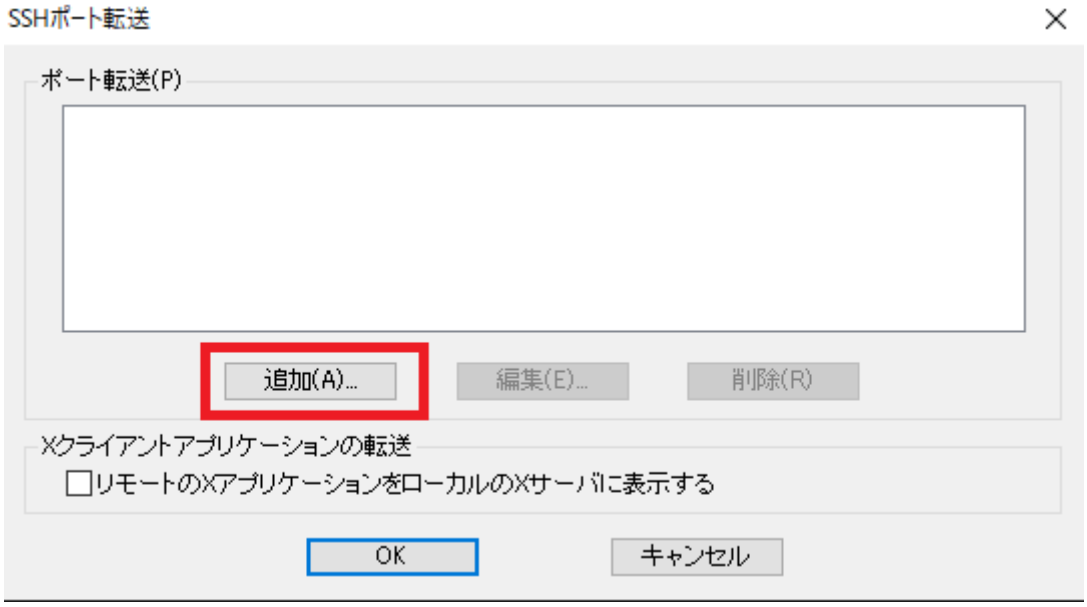


図 5-4　“SSH ポート転送”ウィンドウ

（3） ローカルフォワードを行うために、“ポート転送を行う向きの選択”で“ローカルのポート”を選択し、そこにクライアント側で使用するポート番号（図 5-1 におけるクライアント PC のポート番号 50000 番に対応）を入力する。“リモート側ホスト”にサーバ側のドメイン名あるいは IP アドレス（図 5-1 におけるターゲットのアドレスに対応）を入力する。“ポート”にサーバ側で使用するポート番号（図 5-1 におけるターゲットのポート番号 60000 番に対応）を入力する。“OK”を選択する。

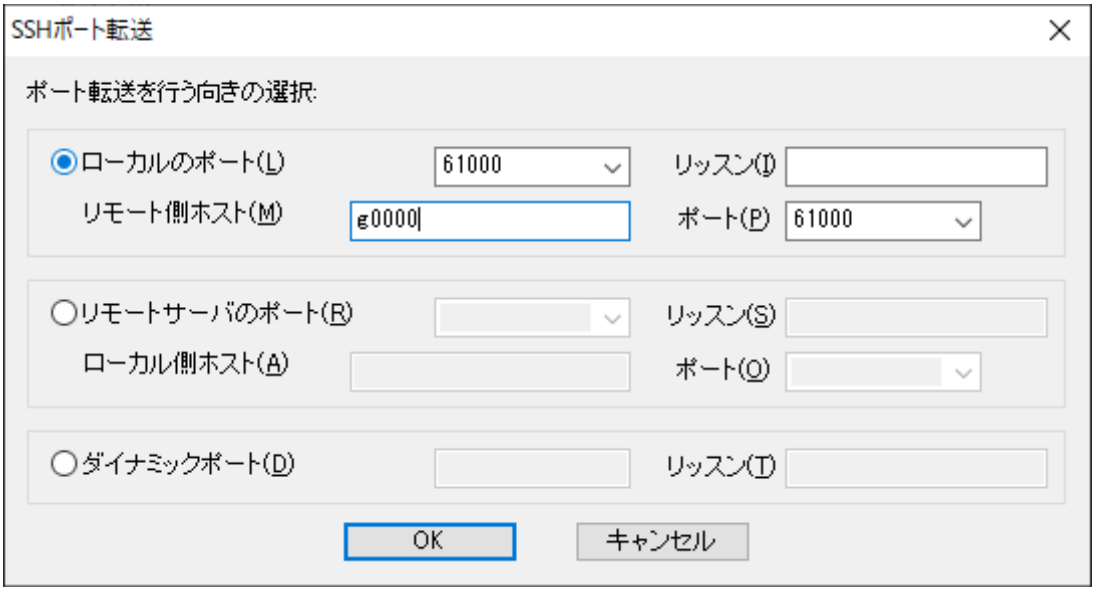


図 5-5 ポート転送を行う向きの選択

下図の表示を確認し、“OK”を選択する。これでポートフォワードの設定は完了となる。

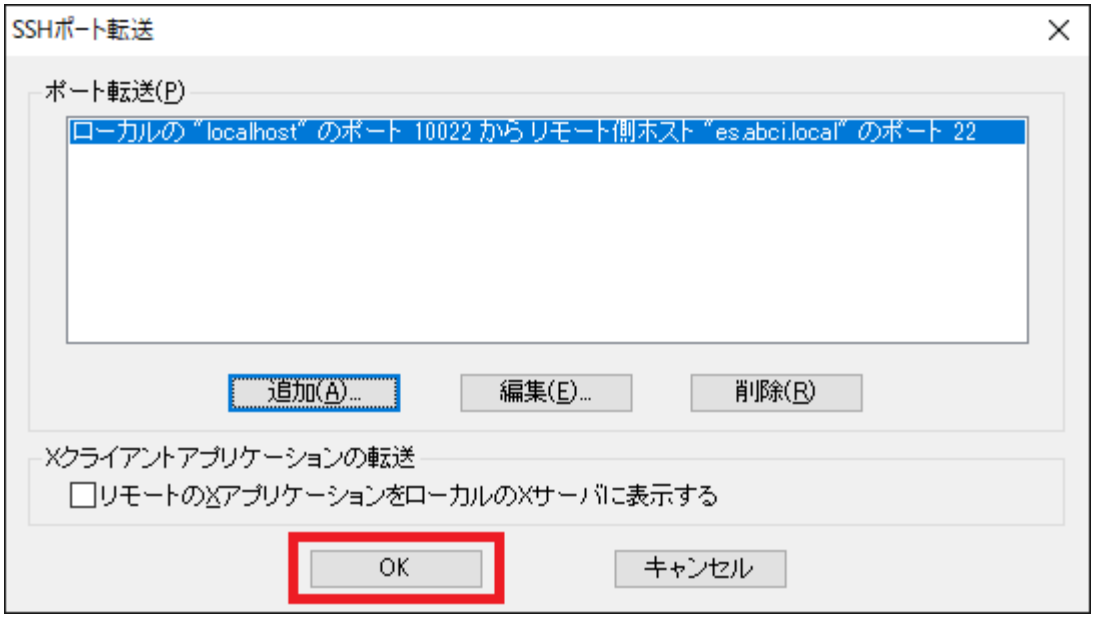


図 5-6 ポート転送の表示の確認

（4） 次にポートフォワードされたサーバと接続する。メニューバーから ファイル→新しい接続を選択する。

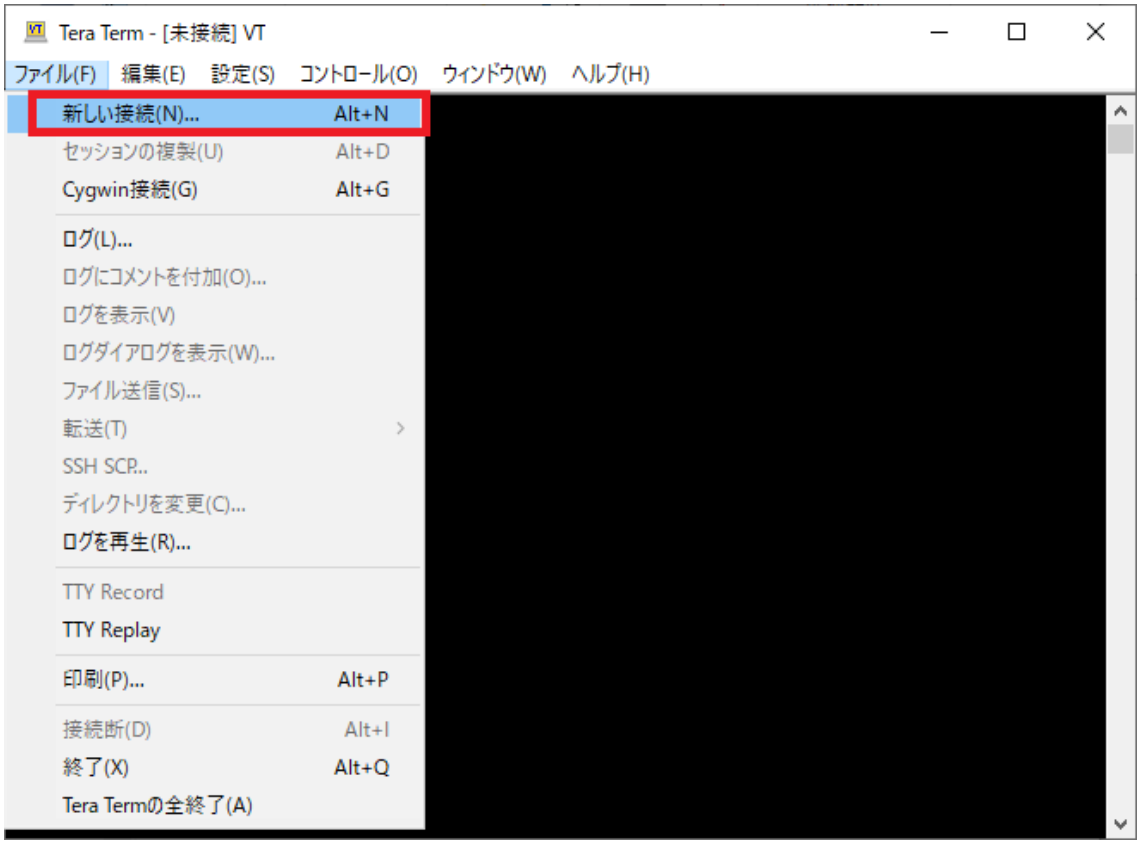


図 5-7　“ツールバー”の“ファイル”タブ

“新しい接続”ウインドウが立ち上がるのでサーバ側のホスト名（図 5-1 における遠隔サーバのアドレスに対応）を入力し“OK”を選択する。

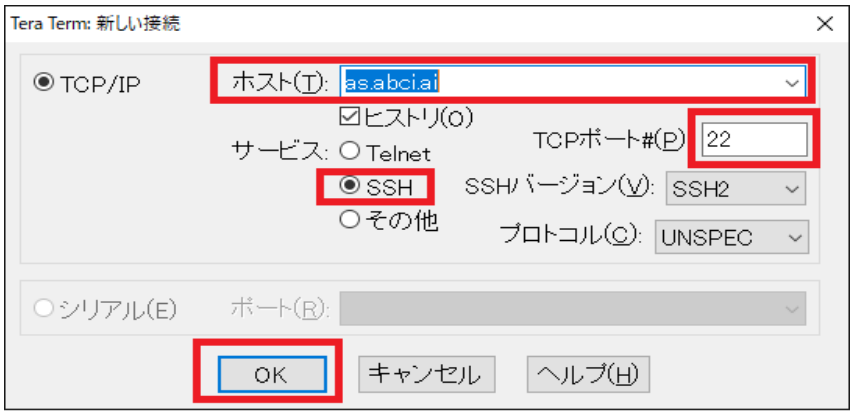


図 5-8　“新しい接続”ウインドウへの入力

“SSH 認証”ウインドウが立ち上がるので、ユーザ名とパスフレーズ、あるいは秘密鍵のディレクトリを指定し“OK”を選択する。これでサーバに接続されたターミナルが起動する。

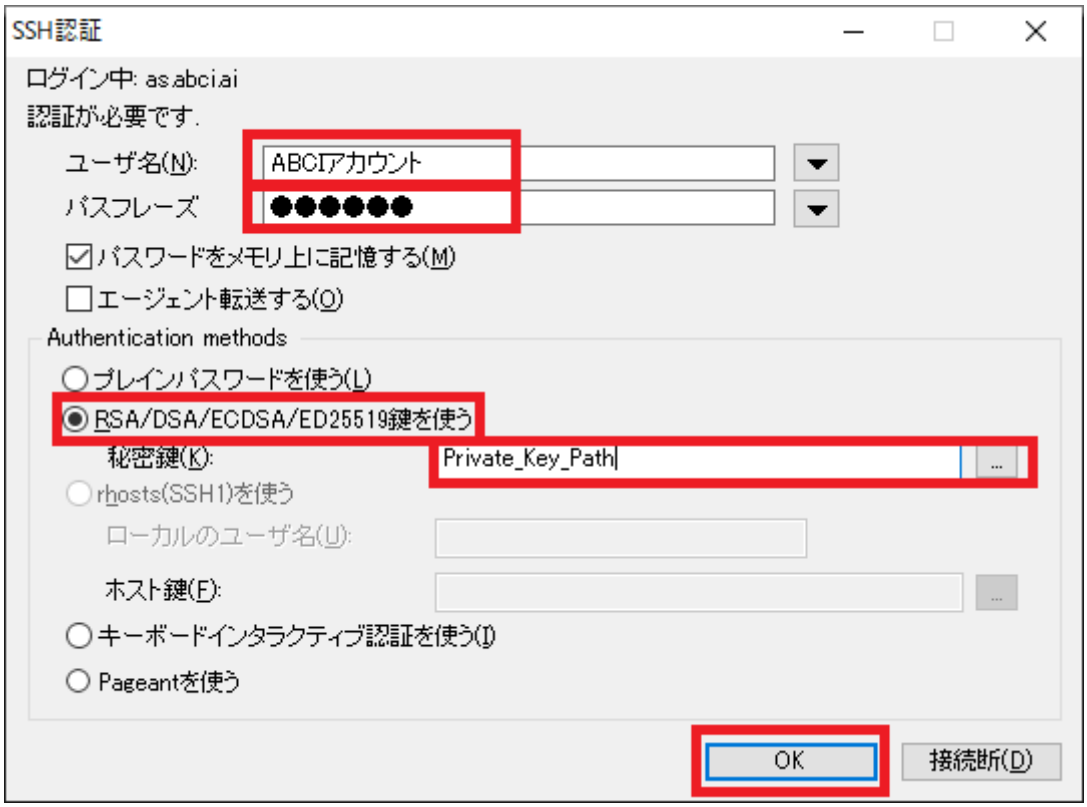


図 5-9 “SSH 認証”ウインドウ

Teraterm の上部メニューから「設定(S)」 →「設定の保存(S)」を選択し、作成した SSH 接続の設定を保存します。Teraterm は次回起動時にこの設定を参照します。

5.3. フロントエンドサーバ上での可視化

PBVR はソケット通信を介したクライアント/サーバ型可視化だけでなく、VNC や X Window System によりフロントエンドサーバ上での可視化が可能です。フロントエンドサーバ上でクライアントプログラムとサーバプログラムの両方を起動してローカル接続（5.2.1）することで、X Window System を利用して GUI が、あるいは VNC を利用して GUI を含むデスクトップ画面がユーザ PC に転送されます。

VNC

フロントエンドサーバに OSMesa と VNC サーバがインストールされていれば、PBVR をフロントエンドサーバ上で動作させ、VNC ビューア上で可視化することができます。VNC サーバの起動/接続方法はフロントエンドサーバの運用に依存しますので、管理者に問い合わせてください。

X Window System（Linux）

Linux はデフォルトで X Window System をサポートしているため、PBVR をローカル接続で起動するだけでリモート利用できる。２つのターミナルでフロントエンドサーバ（machineA ）にログインしてクライアントプログラムとサーバプログラムをそれぞれ起動してください。

［手順１］

ターミナル１でサーバプログラムを起動する。

UserPC:$ ssh usename@machineA

machineA:$ mpiexec -n 5 pbvr_server

［手順２］

ターミナル２でクライアントプログラムを起動する。GUI を X Window System で転送するために、XY オプションが必要である。

UserPC:$ ssh -XY usename@machineA

machineA:$ pbvr_client -vin filename

X Window System（Windows）

Windows 端末から X Window System を利用するには Xming と TreraTerm のインストールが必要です。

［手順１］

Xming および Xming-fonts をダウンロードしインストールする。

［手順２］

Teraterm の X11 フォワード設定（5.2.4）を行う。

［手順３］

Xming に付属する Xlaunch で Xming を起動する。

［手順４］

Teraterm を起動しフロントエンドサーバに接続する。

［手順５］

以降の手順は Linux の場合と同様です。

X Window System（Mac）

Mac には X Window System をサポートするアプリケーションとして Xquartz が存在するが、PBVR は Xquartz に対応していない。

６ クライアント

クライアントは、遠隔 PC とユーザ PC で通信して可視化を分散処理するクライアント・サーバモード（以降、クラサバモードと表記）と、生成済みの粒子データを表示するスタンドアロンモードで動作する。

クラサバモードでは、サーバの可視化結果として送信される粒子データを受信し、OpenGL を用いて描画する。また、サーバがボリュームレンダリングを行う際の可視化パラメータを入力し、サーバへ送信する。クライアントとサーバ間のデータの送受信は、任意のポート番号によるソケット通信で行う。

スタンドアロンモードでは、サーバがバッチモードで出力した粒子データファイルを読み込んで表示する。

サーバプログラムの起動方法は（５.１）に、遠隔 PC とユーザ PC のソケット通信の方法は（５.２）に記載する。

６.１. 起動方法

クラサバモードおよびスタンドアロンモードでのクライアントプログラムの起動例を以下に示す。

（クラサバモードでの起動方法）※1
$ pbvr_client -vin [入力ボリュームデータファイル※2] [コマンドラインオプション]

（スタンドアロンモードでの起動方法）
$ pbvr_client -pin1 [粒子データファイル名] [コマンドラインオプション]

※1. クラサバモードはサーバを起動した状態で実行すること。
※2. 入力ボリュームデータファイルは.pfi ファイルを絶対パス、もしくは、相対パスで指定する。
複数の pfi ファイルを指定したい場合は、pfi ファイルを列挙した pfl ファイルを作成し、-vin オプションで pfl ファイルを指定する。

表 6.1-1　クライアントのコマンドラインオプション一覧

オプション	指定値	デフォルト	機能
-h	-	-	オプションおよび指定可能パラメータの一覧を表示
-pd	実数値	1.0	画像の濃さを指定する
-S	u、 m、 r	u	粒子サンプリング方法 u: uniform sampling、 m: metropolis sampling r: rejection sampling
-plimit	1～99999999	1000000	粒子数制限値
-vin	ファイル名	-	入力ボリュームデータファイル※1
-tf	ファイル名	-	伝達関数※2
-p	ポート番号	60000	ソケット通信ポート番号
-shading	{L/P/B}、ka、kd、ks、n	-	シェーディング方法 ※3
-polygon_model		-	3DS および FBX 形式のポリゴンデータを入力する ※4
-las_model		-	LAS 形式の３次元点群データを入力する ※5

※1. フィタ処理後のボリュームデータから生成される.pfi ファイル、もしくは分散処理用の.pfl ファイルを絶対パス、もしくは、相対パスで指定する。拡張子を省略しないこと。このオプションは、-pa で指定するパラメータファイル内のオプションよりも優先される。

※2. 伝達関数ファイルは伝達関数エディタの Export File ボタンで出力される。起動後に伝達関数エディタの Apply ボタンを押すことで伝達関数が反映される。伝達関数エディタの Import File ボタンでも同様の伝達関数ファイルを読み込むことができる。

※3. シェーディング方法の指定は以下の通り。

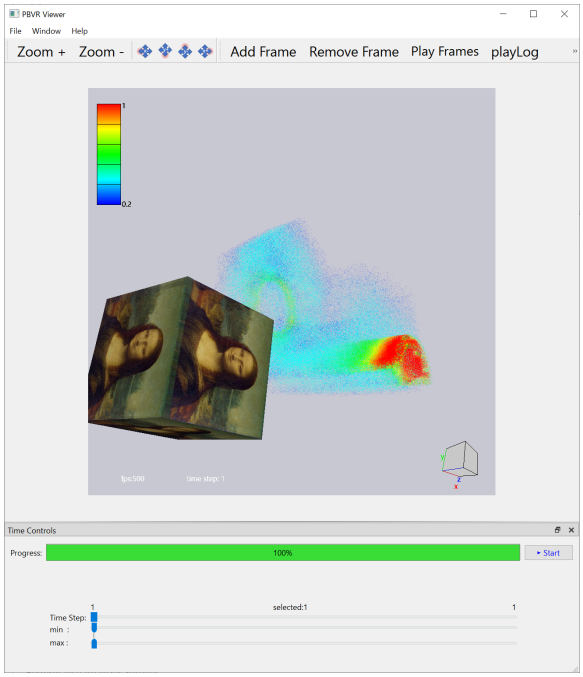
L:ランバートシェーディング
効果：拡散反射を考慮したシェーディングを与える。
使用パラメータ：ka（物体の明るさに掛かる係数。0～1 の実数）、kd（法線方向と光線方向から計算される拡散反射成分に掛かる係数。0～1 の実数）

P：フォンシェーディング
効果：ランバートシェーディングにハイライト効果を追加したもの。
使用パラメータ：ka、kd、ks（視線方向、法線方向、光線方向から計算される鏡面反射成分に掛かる係数。0～1 の実数）、n（ハイライトの強さ。0～１００の実数）

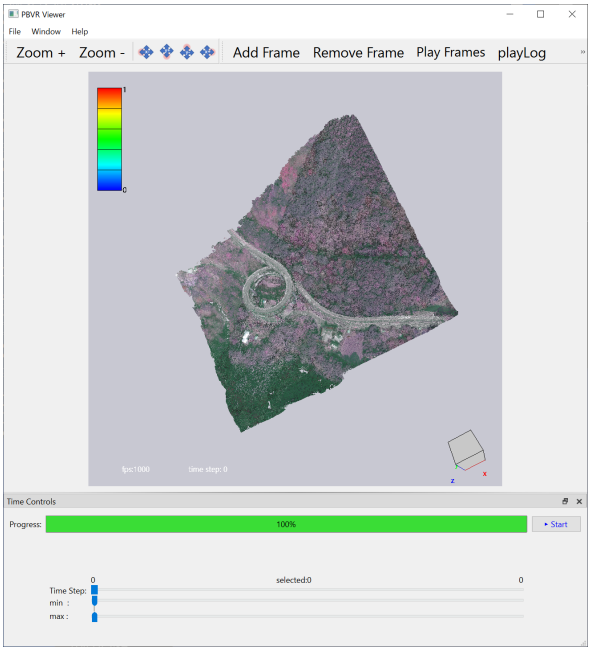
B：ブリン-フォンシェーディング

効果：フォンシェーディングの簡略化モデル
使用パラメータ：ka、kd、ks、n

※4. FBX 形式のサンプルデータ（六面体に絵画のテクスチャを貼ったもの）とボリュームデータの合成表示の例



※5. ３次元点群データ（G 空間情報センターの VIRTUAL SHIZUOKA プロジェクト提供データを使用 https://www.geospatial.jp/ckan/dataset/virtual-shizuoka-mw）の表示例



6.2. 終了方法

6.2.1. 通常終了

クライアントは時系列データの先頭時刻から描画を開始し、最終時刻の描画後は再度先頭時刻の描画に戻るというループ動作をする。このためクライアントの終了はクライアントプログラムを起動したコンソールにおいて Ctrl+c キーを押して行う。

クラサバモードでの動作中に Ctrl+c キーを押すと、クライアントとサーバの時刻更新のタイミングで同期してサーバも終了する。ただし、タイムステップ制御パネルの Stop ボタンでクラサバ通信を中断させた状態だと Ctrl+c キーの入力は無視される。

6.2.2. 強制終了

サーバを Ctrl+c キーで終了させてしまうと、クライアントを Ctrl+c キーで終了させることができなくなる。また、クライアントを Ctrl+c キーで終了させても、サーバでの粒子生成が長時間に及ぶ等で時刻更新がなされない場合は、クライアントもサーバも応答しない状態に陥ることがある。このような場合は以下に示すように、ps コマンドでクライアントとサーバのプロセス番号を取得し、kill コマンドで強制終了させる。

【クライアントプロセスの強制終了】

```
$ ps -C pbvr_client
  PID TTY          TIME CMD
19582 pts/6    00:00:00 pbvr_client
$ kill -9 19582
```

【サーバプロセスの強制終了】

```
$ ps -C pbvr_server
  PID TTY          TIME CMD
19539 pts/5    00:00:00 pbvr_server
$ kill -9 19539
```

6.3. クライアントプログラムの GUI

起動後のクライアントプログラムは可視化結果を表示するだけでなく、可視化パラメータを対話的に制御できる GUI を提供する。

6.3.1. ビューワ

図 6-1 に示すビューワには、粒子データの描画結果が表示される。

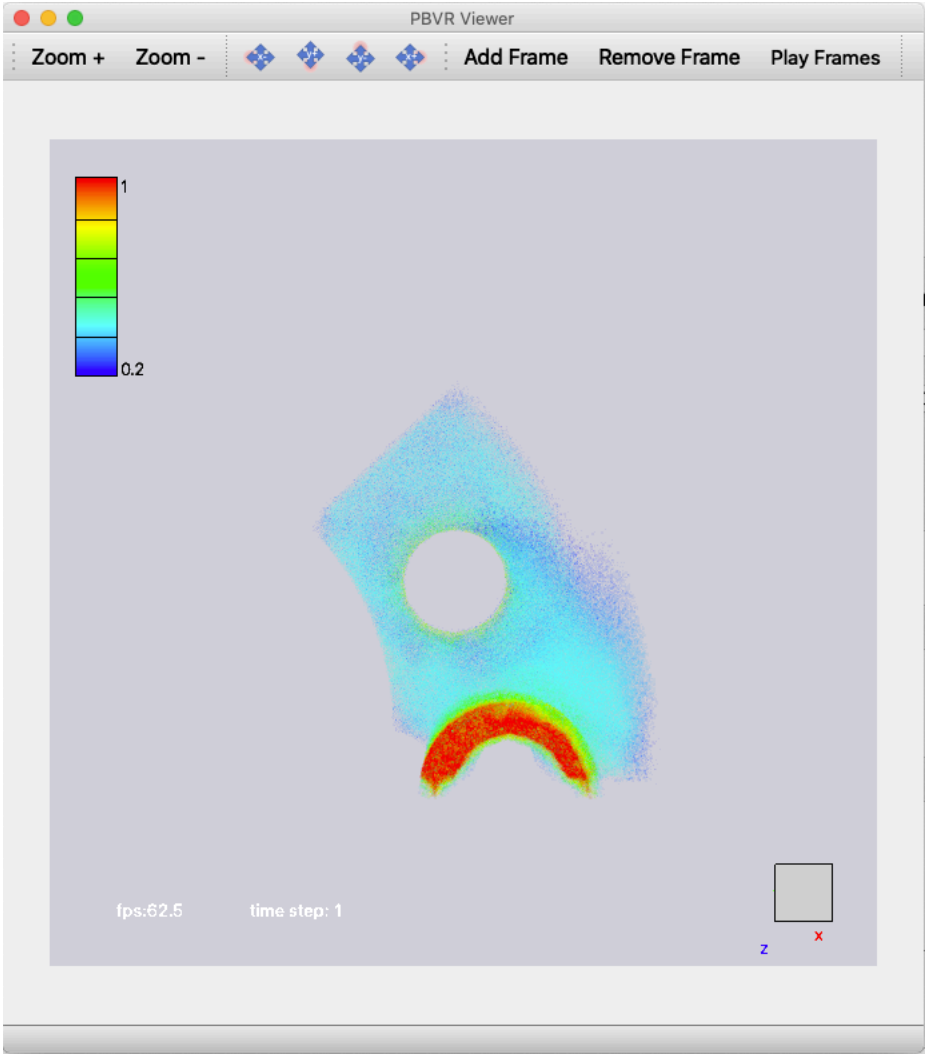


図 6-1　ビューワ

【操作方法】

回転：マウスで左ドラッグ

移動：マウスで右ドラッグ

拡大・縮小：マウスで Shift+左ドラッグ　もしくは　マウスホイール押下+ドラッグ

リセット：home ボタン（Mac では fn + left arrow）

【表示内容】

time step：表示しているデータのタイムステップ

fps：フレームレート（frame/sec）

【ツールバー】

拡大、縮小

平行移動

キーフレームの取得/削除/アニメーションの再生

6.3.2. ツールバー

クライアントプログラムの様々な機能はツールバーから選択される。“File”タブは可視化パラメータファイルや伝達関数の入出力を制御する。

可視化パラメータとは、ビューワの解像度、粒子密度、粒子数制限値、サーバ上の入力ボリュームデータファイル名（pfi ファイルまたは pfl ファイル）、伝達関数等、クライアントで設定可能なパラメータ郡のことを意味し、可視化パラメータファイルとはそれらがタグ形式で記述されたファイルのことである。

以下に“File”タブとその機能を示す。

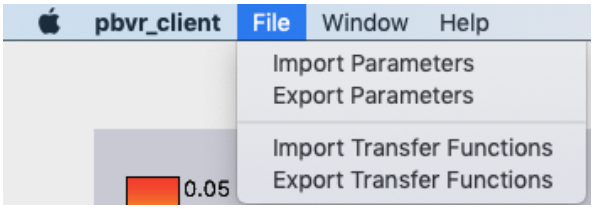


図 6-2 “File”タブ

- Import Parameters
  入力する可視化パラメータファイルを指定する。
- Export Parameters
  可視化パラメータファイルを出力する。
- Import Transfer Functions
  入力する伝達関数ファイルを指定する。
- Export Transfer Functions
  伝達関数ファイルを出力する。

以下に“Window”タブとその機能を示す。

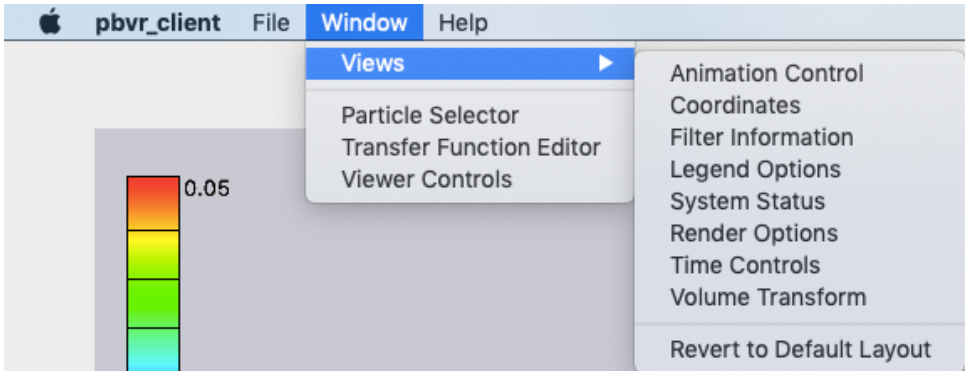


図 6-3 “Window”タブ

- Animation Control
  動画作成用パネルを表示する。詳細については後述。

- Coordinates
  座標エディタを表示する。詳細については後述。
- Filter Information
  入力ボリュームデータの情報を表示する。詳細については後述。
- Legend Options
  レジェンドパネルを表示する。詳細については後述。
- System Status
  システム情報を表示する。詳細については後述。
- Render Options
  レンダリングのオプションを制御する。詳細については後述。
- Time Controls
  タイムステップを制御する。詳細については後述。
- Volume Transform
  描画対象の幾何変換を指定する。詳細については後述。
- Revert to Default Layout
  GUI のレイアウトをクライアントプログラム起動時に戻す。
- Transfer Function Editor
  伝達関数エディタを表示する。詳細については後述。
- Viewer Controls
  ビューワ制御パネルを表示する。詳細については後述。

6.3.2.1 **FilterInfo**

FilterInfo パネルでは入力されたフィルタ処理済みのボリュームデータの情報を表示する。各項目について以下に説明する。

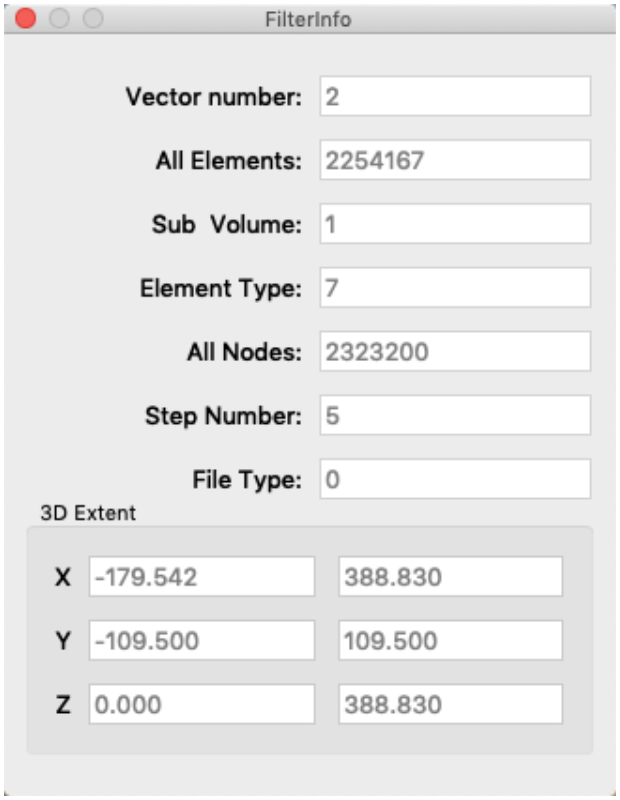


図 6-4 FilterInfo

- Vector number
  ボリュームデータに含まれる変数の数
- All Elements
  ボリュームデータの要素数
- Sub Volume
  ボリュームデータの分割数
- Element Type
  ボリュームデータを構成する要素タイプ。詳細は表 4.7-1 を参照
- All Nodes
  ボリュームデータの頂点数
- Step Number
  ボリュームデータのタイムステップ数
- FileType
  フィルタの分割形式。詳細は4.3を参照
- 3D Extent
  ボリュームデータの X-Y-Z 座標の最大最小値

6.3.2.2 System Status

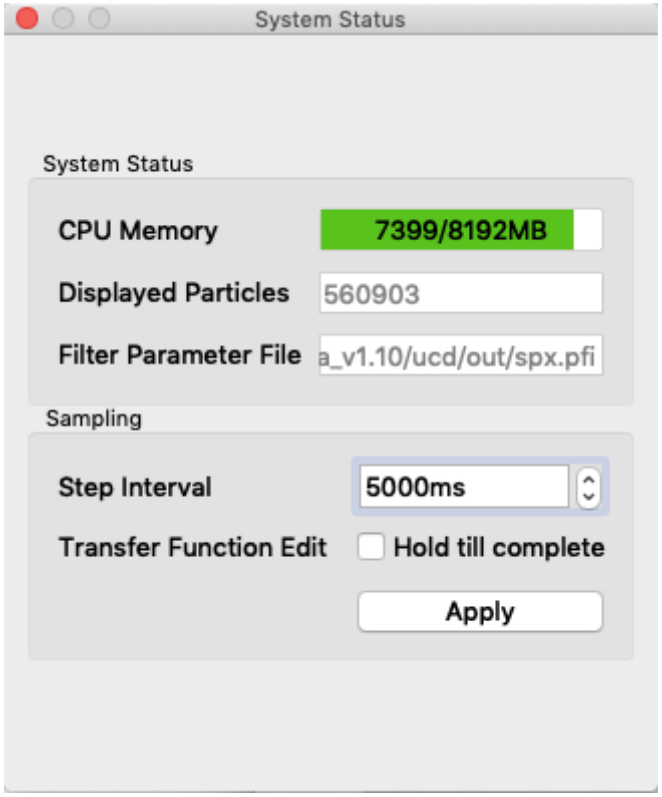


図 6-5　システム情報

- CPU Memory
  ステムメモリの使用状況（単位：MB）を表示する。
- Displayed Particles
  ビューワに表示されている粒子の数を表示する。
- Filter Parameter File
  サーバ上の入力ボリュームデータファイル名（pfi ファイルまたは pfl ファイル）を表示する。
- Step Interval
  タイムステップの更新間隔の最低値を指定する（単位 msec）。タイムステップの更新間隔が短すぎる場合の調整に利用する。
- Transfer Function Edit の Hold till complete チェックボックス
  伝達関数の更新をタイムステップ更新が完了するまで留める。

6.3.2.3 Render Options

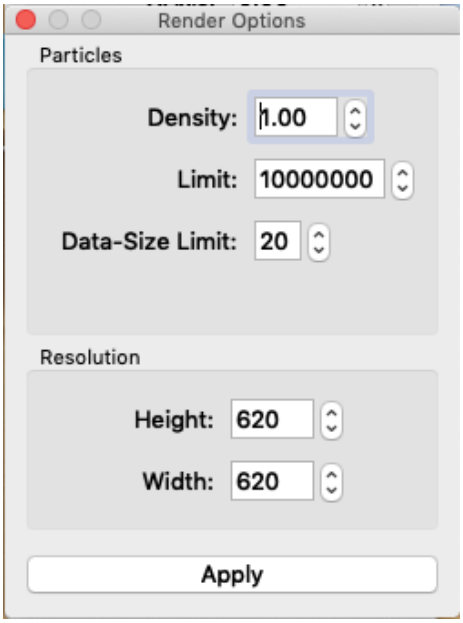


図 6-6　レンダリング用のオプション

- Density
  画像の濃さに影響する粒子密度を指定する。
- Limit
  伝達関数誤指定等による粒子数の爆発を避けるため、サーバプログラム側で生成する粒子数の上限値を指定する。粒子数がこの上限を超える場合、サーバプログラムは自動的に画質を下げて粒子数がこの上限に収まるように調整する。
- Data-Size Limit
  伝達関数誤指定等による粒子数の爆発を避けるため、サーバプログラム側で生成する粒子データサイズの上限値を指定する。単位は[GB]。粒子データサイズがこの上限を超える場合、粒子生成は強制的に停止する。
- Resolution
  レンダリング解像度を指定する。

6.3.2.4 **Volume Transform**

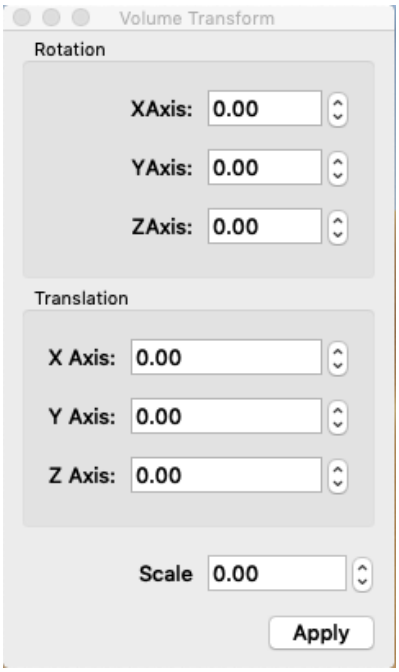


図 6-7　描画対象の幾何変換

・Rotation

物体のｘ軸、ｙ軸、ｚ軸それぞれに関する回転角（度数）を指定する。

・Translation

物体のｘ、ｙ、ｚ方向の平行移動を指定する。

・Scale

物体の拡大率を指定する。

6.3.3. 伝達関数エディタ

伝達関数エディタでは、物理値に割り当てられる色および不透明度を定義する伝達関数を作成できる。通常のボリュームレンダリングでは伝達関数は一つの物理量のみによって定義されるが、PBVRでは

（1） 色と不透明度に独立な変量を割り当てる。
（2） 各変量を座標 X、Y、Z、変量 q1、q2、q3…の任意の関数式で定義。
（3） １次元伝達関数の色関数を C1、C2…、不透明度関数を O1、O2、….で定義し、さらにそれら任意の関数式で合成して多次元伝達関数を定義。

という、新たな伝達関数設計を可能としたことで、極めて自由度の高い可視化処理を実現した。図 6-8 に伝達関数エディタパネルを示す。パネル内の各項目について以下に説明する。

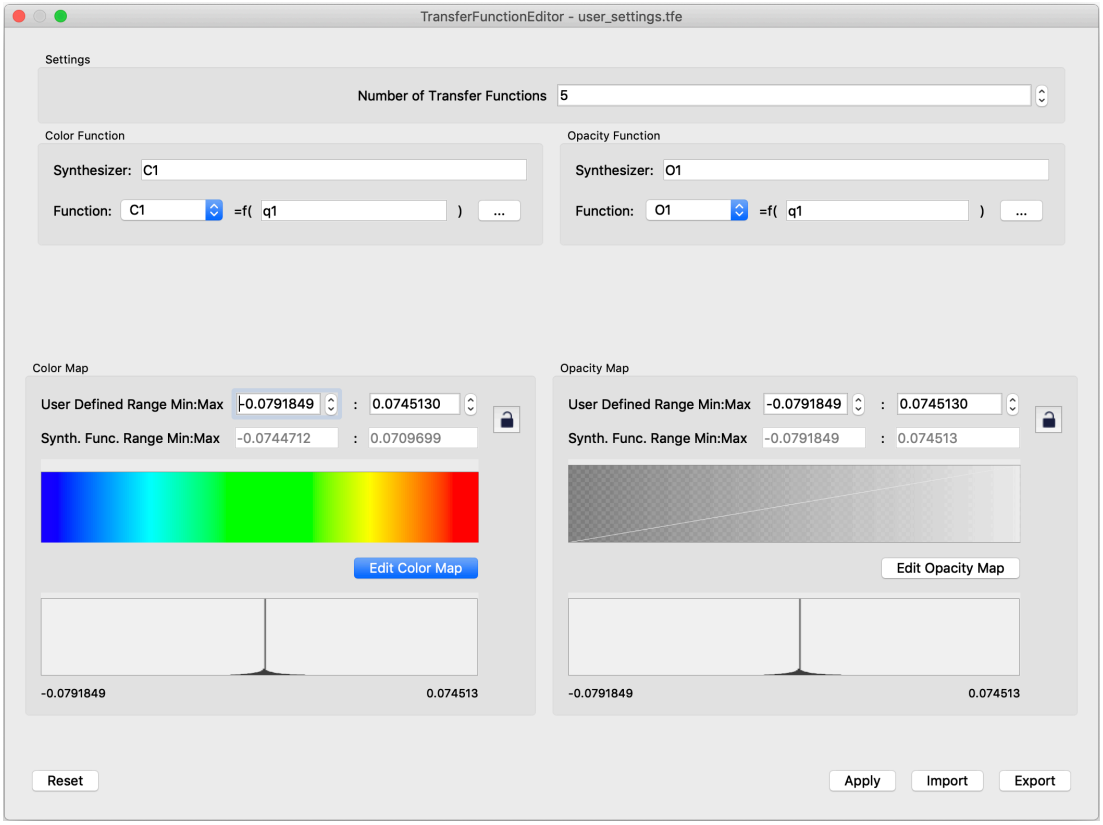


図 6-8 伝達関数エディタパネル

【操作方法】

- histogram スケール変更
  Histogram 上でマウスを上下にドラッグ
- Number of Transfer Fuctions
  作成可能な伝達関数の制限数を指定する
- Color Fuction カテゴリ
  色関数とその引数となる変量の合成式を指定する。
- Opacity Fuction カテゴリ

不透明度関数とその引数となる変量の合成式を指定する。

- Color Map カテゴリ

  色関数を編集する。

- Opacity Map カテゴリ

  不透明度関数を編集する。

- Reset ボタン

  本パネルを初期状態にリセットする。

- Apply ボタン

  本パネルで作成した伝達関数をサーバへ送信する。編集された伝達関数はタイムステップ更新のタイミングでサーバに読み込まれるため、可視化結果に反映されるまで１ステップ分のタイムラグがある。

- Export ボタン

  本パネルで作成した伝達関数を、-pa オプションで指定するパラメータファイルと同じ書式で、ファイルへ保存する。

- Import ボタン

  ファイルに保存されている伝達関数を読み込んで本パネルへ反映する。

【代数式で使用可能な演算子】

+, -, *, /, ^, sin(), cos(), tan(), sqrt(), log(), exp()

6.3.3.1 カラーマップ指定機能

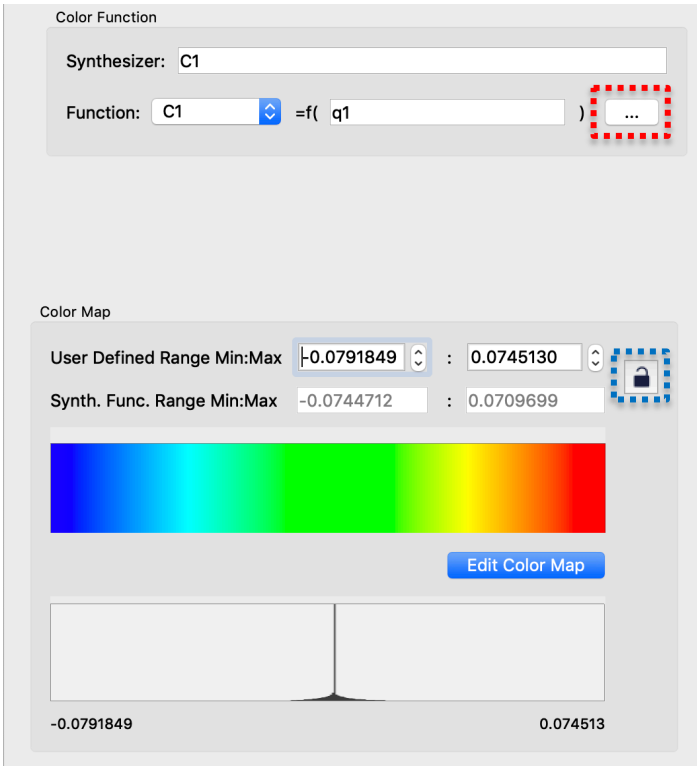


図 6-9　上部：Color Function カテゴリ、下部：Color Map カテゴリ

【Color Function カテゴリ】

- Synthesizer

  色関数 C1～C[N] ※1 による合成式を指定する。

- Function スピンボタン

  編集する色関数 C1～C[N]を選択する。

- Color Function Editor ボタン

  上図 6-9 中の赤い点線で囲まれたボタン。Color Function Editor 画面を表示して、選択された色関数 C1～C[N]の引数となる（合成）変量を作成する。

【Color Map カテゴリ】

- User Defined Range Min:Max

  Color Function Editor で指定した（合成）変量に対して、色関数を割り当てる最大最小値を指定する。

- Synth. Func. Range Min:Max

  Color Function Editor で指定した（合成）変量の最大最小値を表示する。

- Edit Color Map ボタン

  選択した色関数名に対応するカラーマップを作成するための Color Map Editor パネルを開く。

- Histogram

  User Defined Range Min:Max で指定した最大最小値の範囲のヒストグラムが表示される。

・鍵ボタン

上図 6-9 中の青い点線で囲まれたボタン。このボタンを押してロック状態にすると、User Defined Range Min:Max への入力が禁止され、強制的に Synth. Func. Range Min:Max の値が採用される。

※1：[N]は Number Transfer Fuctions にて指定した伝達関数の制限数の値である。

6.3.3.1.1. Color Function Editor

Color Function Editor では、ユーザーが指定した代数式を使用して物理値を合成できる。合成した物理値は色関数 C1～C[N]の引数となる。代数式で使用できる変量の名称は以下に示される。

・物理量：q1、 q2、 q3、・・・qn

・座標値：X、 Y、 Z

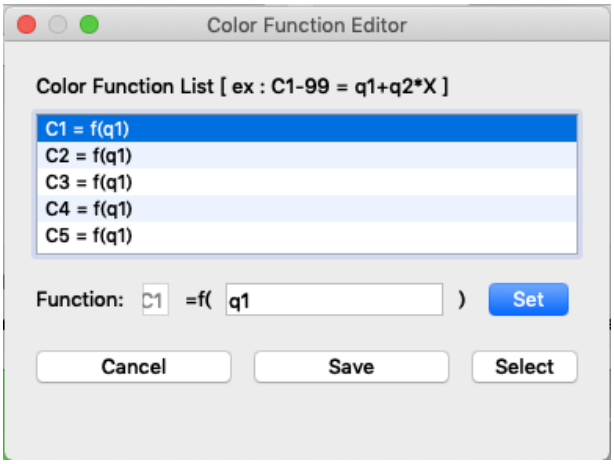


図 6-10 Color Function Editor 画面

・Color Function List

作成されている伝達関数を表示する。

・Function

伝達関数 C[N] = f(変数) を入力する

・Set ボタン

Color Function List に反映する

・Cancel ボタン

本パネルを閉じる

・Save ボタン

Color Function List の伝達関数を適用する。

・Select ボタン

Color Function List の伝達関数の適用、リスト選択の伝達関数を選択する。

6.3.3.1.2. Color Map Editor：Freeform Curve Edit タブ

Color Map Editor の Freeform Curve Edit タブでは、C1～C[N]に対応する色関数をマウスによる自由曲線入力で作成できる。

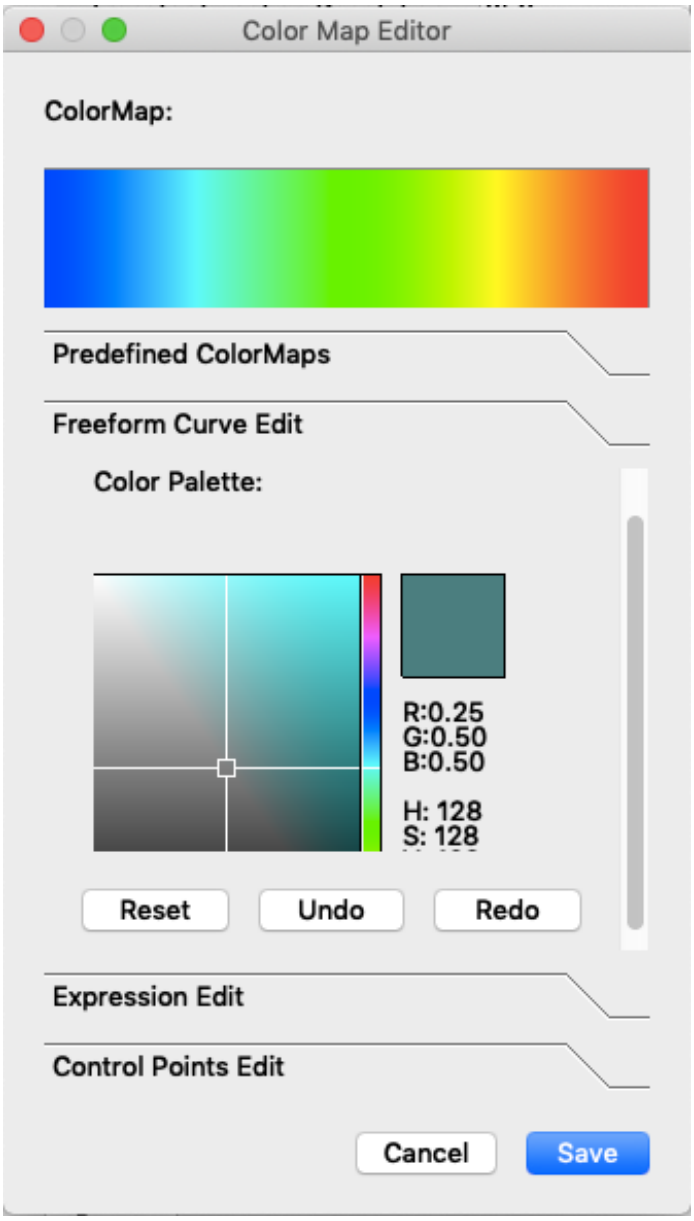


図 6-11　Color Map Editor の Freeform Curve Edit タブ

- Color palette

  横軸が彩度、縦軸が明度で、マウスの位置によりこれらを指定する。
- RGB 指定バー

  色相をマウスの位置により指定する。Color palette には選択した色相が反映される。
- Reset ボタン

  本パネルを初期状態に戻す。

・Undo ボタン

１マウスアクションを取り消す。

・Redo ボタン

取り消したマウスアクションを再実行する。

・Save ボタン

本パネルで作成した伝達関数を保存する。

・Cancel ボタン

本パネルを閉じる。

ColorMap を左クリックでなぞることによって、Color palette と RGB 指定バーで作成した色でバーを上塗りする。ここで、既存の色との合成比率はバーの縦軸方向のマウス位置によって決定される。例えば、カラーバー上辺を左右になぞれば、なぞった範囲を指定色のみで塗りつぶし、カラーバー上下中央位置を左右になぞれば、なぞった範囲を既存色と 50%の割合で合成する。

6.3.3.1.3. Color Map Editor：Expression Edit タブ

Color Map Editor の Expression Edit タブでは、C1～C[N]に対応する色関数を数式記述で作成できる。

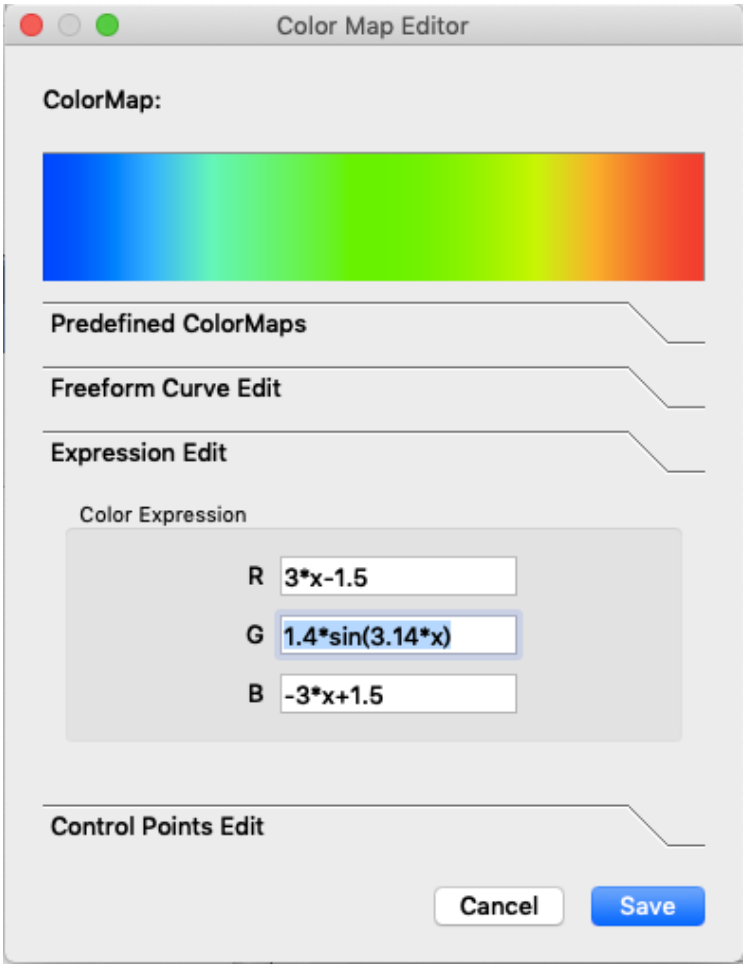


図 6-12　Color Map Editor の Expression Edit タブ

- R

  色の R 成分の色関数式を代数式で記述する。

- G

  色の G 成分の色関数式を代数式で記述する。

- B

  色の B 成分の色関数式を代数式で記述する。

色関数の変数は x であり、色関数の定義域と値域は 0 から 1 である。

6.3.3.1.4. Color Map Editor：Contlol Points Edit タブ

Color Map Editor の Control Points Edit タブでは、C1～C[N]に対応する色関数を制御点指定で作成できる。

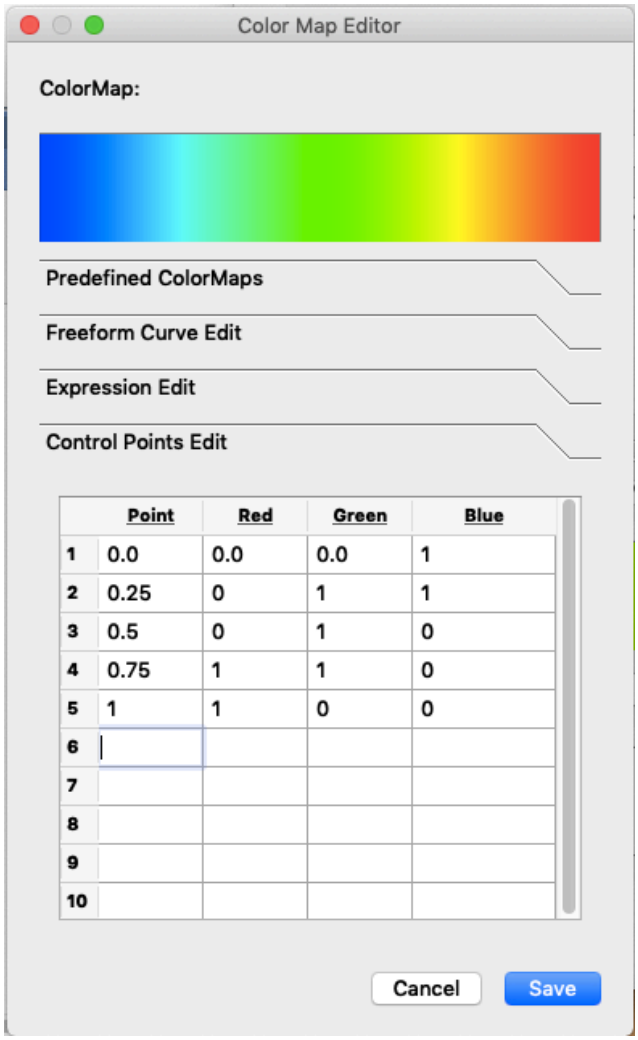


図 6-13　Color Map Editor の Control Points Edit タブ

- Point

  制御点（最大 10 個）の値を指定する。定義域は0から１の範囲である。

- Red

  制御点の値に対応する色の R 成分値を指定する。値域は0から１の範囲である。

- Green

  制御点の値に対応する色の G 成分値を指定する。値域は0から１の範囲である。

- Blue

  制御点の値に対応する色の B 成分値を指定する。値域は0から１の範囲である。

各制御点は区分線形関数で補間される。

6.3.3.1.5. Color Map Editor：Predefined ColorMaps タブ

Color Map Editor の Predefined ColorMaps タブでは、C1～C[N]に対応する色関数をあらかじめ用意されている色関数から選択できる。

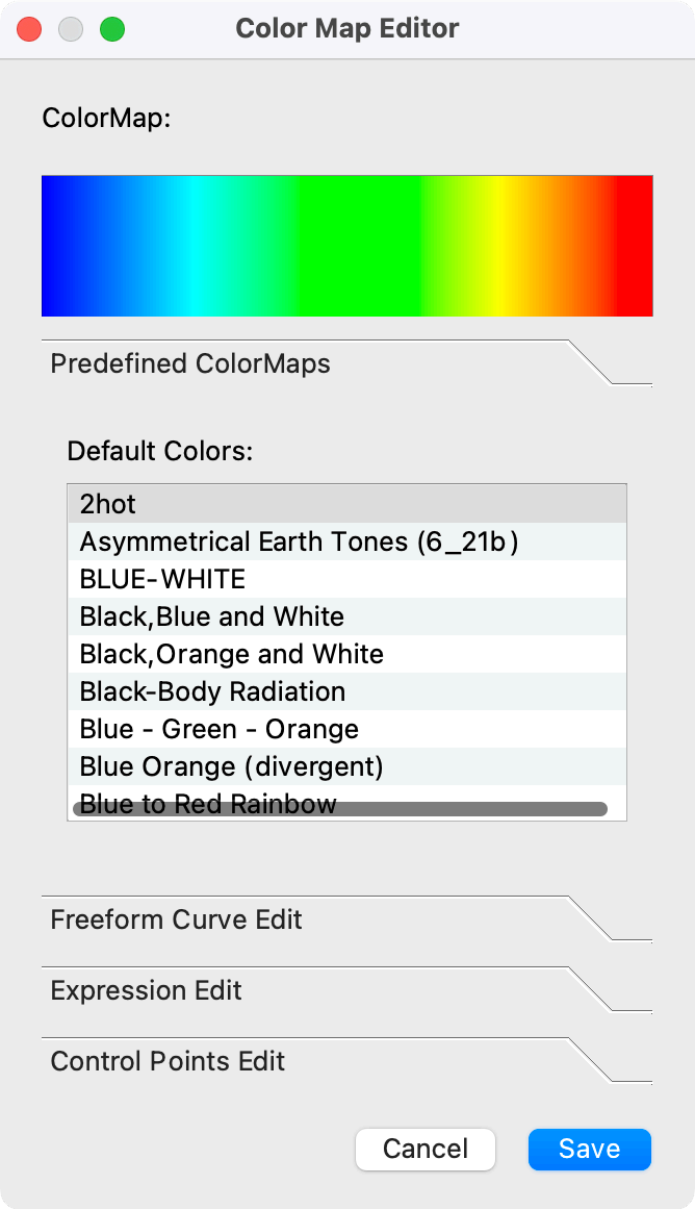


図 6-14　Color Map Editor の Predefined ColorMaps タブ

- Color

  本パネルで作成した伝達関数のカラーバーが表示される。
- Default Color プルダウンメニュー

  伝達関数として設定するカラーバーを選択する。選択肢は以下。
  - RainBow
  - Blue-white-red
  - Black-red-yellow-white

- Black-blue-violet--yellow-white
- Black- yellow-white
- Blue-green-red
- Green-red-violet
- Green- blue--white
- HSV model
- Gray-scale
- Black
- White

6.3.3.2 不透明度指定機能

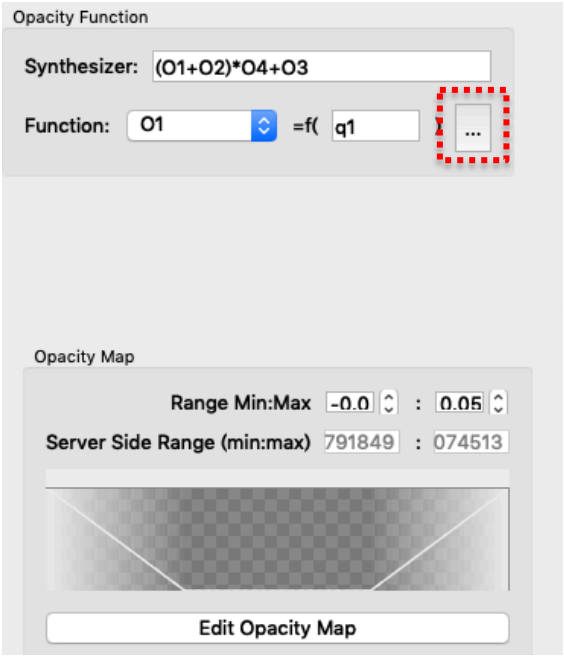


図 6-15　上部：Opacity Function カテゴリ、下部：Opacity Map カテゴリ

【Opacity Function カテゴリ】

- Synthesizer

  不透明度関数 O1～O[N] ※1 による合成式を指定する。

- Function スピンボタン

  編集する不透明度関数 O1～O[N]を選択する。

- Color Function Editor ボタン

  上図 6-15 中の赤い点線で囲まれたボタン。Opacity Function Editor 画面を表示して、選択された不透明度関数 O1～O[N]の引数となる（合成）変量を作成する。

【Opacity Map カテゴリ】

- Range Min:Max

  選択した不透明度関数名に対する（合成）変量の最小値：最大値を指定する。

- Server side range (min:max)

  Opacity Function Editor で指定した（合成）変量について、サーバ側で取得した最小値：最大値を表示する。

- Edit Opacity Map ボタン

  選択した不透明度関数名に対応する不透明度関数を作成するための Opacity Map Editor パネルを開く。

※1：[N]は Number Transfer Fuctions にて指定した伝達関数の制限数の値である。

6.3.3.2.1. Opacity Function Editor

Opacity Function Editor 上では、不透明度関数 O1～O[N]の引数となる（合成）変量の定義式が指定される。定義式で使用できる変量の名称は以下に示される。

・物理量：q1、 q2、 q3、・・・qn

・座標値：X、 Y、 Z

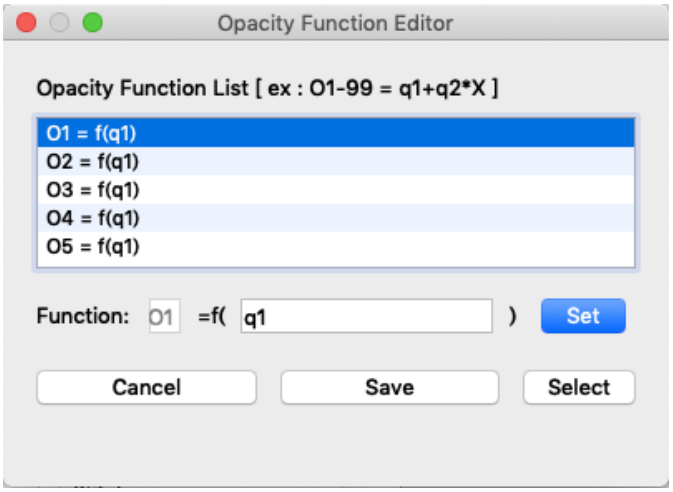


図 6-16 Opacity Function Editor 画面

・Opacity Function List

作成されている伝達関数を表示する。

・Function

伝達関数 O[N] = f(変数)を入力する

・Set ボタン

Opacity Function List に反映する

・Cancel ボタン

本パネルを閉じる

・Save ボタン

Opacity Function List の伝達関数を適用する。

・Select ボタン

Opacity Function List の伝達関数の適用、リスト選択の伝達関数を選択する。

6.3.3.2.2. OpacityMap Editor：Freeform Curve Editor タブ

Opacity Map Editor の Freeform Curve Edit タブは、O1～O[N]に対応する不透明度関数をマウスによる自由曲線入力で作成する機能を提供する。

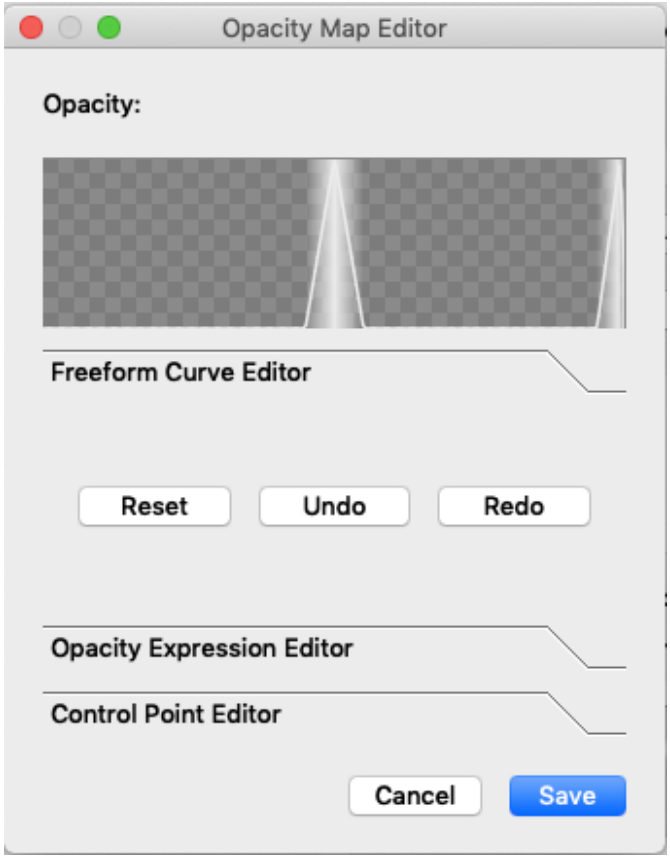


図 6-17　Opacity Map Editor の Freeform Curve Editor タブ

- Reset ボタン
  本パネルを初期状態に戻す。
- Undo ボタン
  １マウスアクションを取り消す。
- Redo ボタン
  取り消したマウスアクションを再実行する。
- Save ボタン
  本パネルで作成した伝達関数を保持する。
- Cancel ボタン
  本パネルを閉じる。

マウス操作により不透明度の伝達関数が作成される。左ドラッグで自由曲線が、右クリックで２点間の線形補間直線が描画される。

6.3.3.2.3. Opacity Map Editor：Opacity Expression Editor タブ

Opacity Map Editor の Opacity Expression Editor タブでは、O1～O[N]に対応する不透明度関数を数式記述で作成するパネルを開くことができる。

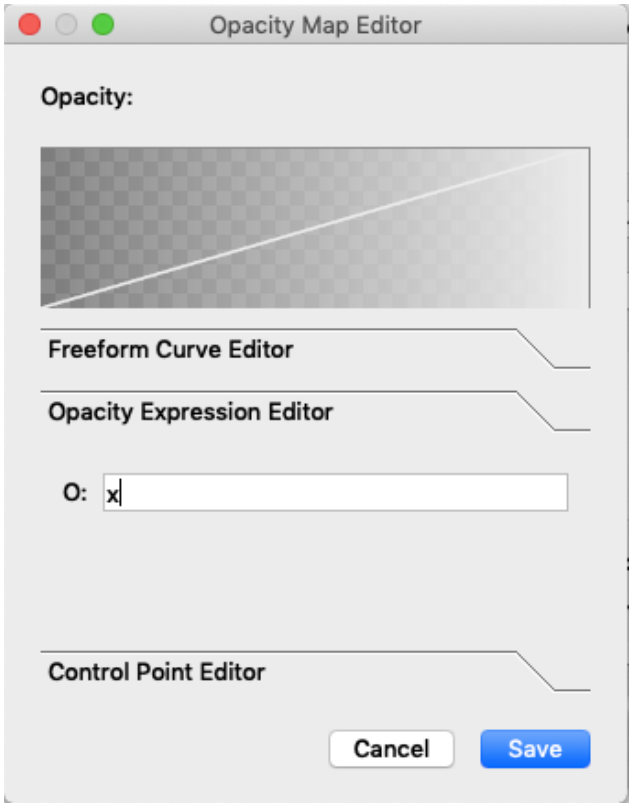


図 6-18　Opacity Map Editor の Opacity Expression Editor タブ

・O

不透明度関数をユーザ指定の代数式で記述できる。不透明度関数の変数は x であり、不透明度関数の定義域および値域は0から１である。

6.3.3.2.4. Opacity Map Editor：Control Point Editor タブ

Opacity Map Editor の Control Points Editor タブでは、O1～O[N]に対応する不透明度関数を制御点指定で作成できる。

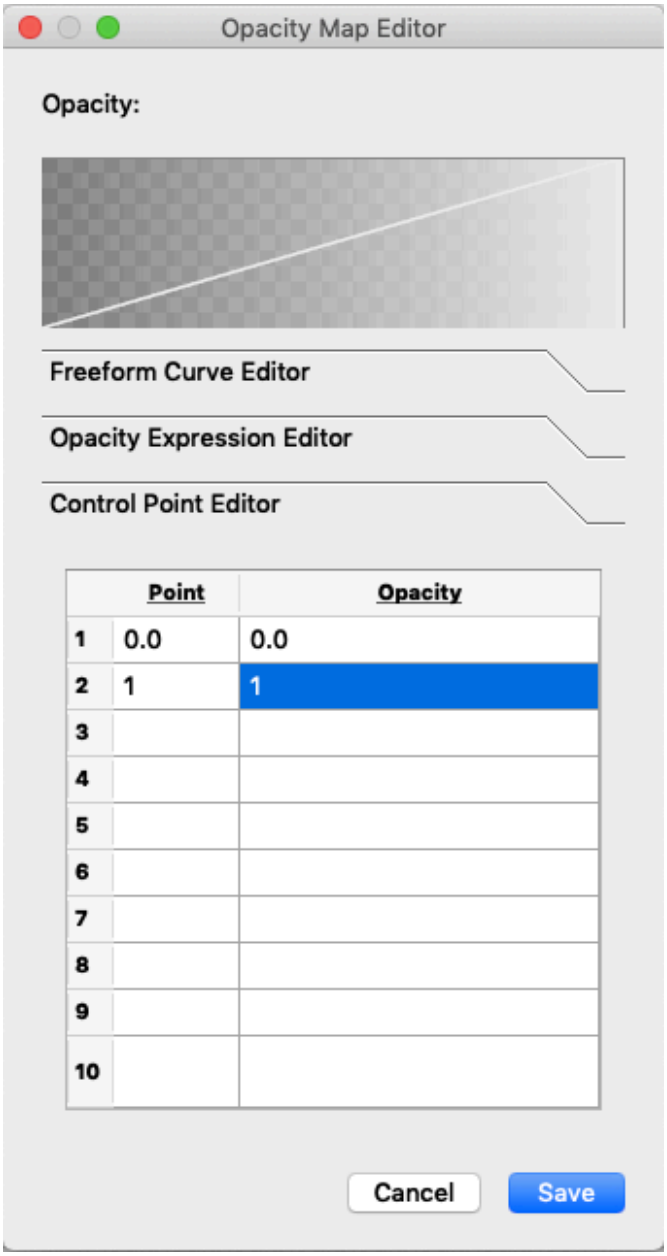


図 6-19　Opacity Map Editor の Control Point Editor タブ

- Control Point

  制御点（最大 10 個）の値を CP1)～CP10)に指定する。値の範囲は0から１の実数である。

- Opacity（中段右側）

  制御点の値に対応する不透明度を指定する。値の範囲は0から１の実数である。

6.3.3.3 関数エディタ

伝達関数エディタにおける伝達関数合成、変量合成、カラーマップ曲線、不透明度曲線の入力に使用される関数エディタで使用できる組み込み関数は以下の通り。

表 6.3-1　関数エディタで利用可能な演算

演算	書式
+	+
-	-
×	*
/	/
Sin	sin(x)
Cos	cos(x)
Tan	tan(x)
Log	log(x)
Exp	exp(x)
平方根	sqrt(x)
冪乗	x^y

関数エディタの演算処理で NaN が現れた場合には PBVR はエラーメッセージを出力して描画処理を停止する。

6.3.4. タイムステップ制御パネル

タイムステップ制御パネルを図 6-20 に示す。各ウィジットの動作は以下のとおり。

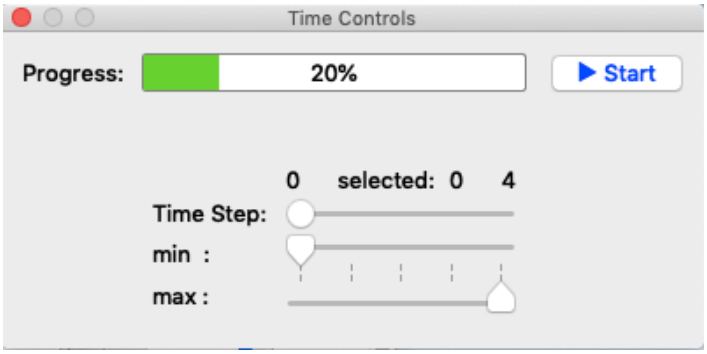


図 6-20　タイムステップ制御パネル

- Progress
  タイムステップの進捗状況をパーセント表示する。
- Time step
  レンダリングするタイムステップを指定する。
- Min Time
  レンダリングするタイムステップの区間（最小タイムステップ）を指定する。
- Max Time
  レンダリングするタイムステップの区間（最大タイムステップ）を指定する。
- Start/Stop
  サーバ/クライアント間の通信を開始または停止する。

6.3.5. 粒子・ポリゴンの統合表示

CS-PBVR（GPU レンダラー）は、ボリュームレンダリングとポリゴンの合成表示をサポートしている。CS-PBVR（CPU レンダラー）は粒子データとの合成表示のみ可能である。複数の粒子データおよびポリゴンを統合して表示する粒子統合エディタを図 6-21 に示す。粒子統合エディタはメインパネルの Particle Panel ボタンを押下すると開く。粒子統合エディタの操作方法は以下の通り。

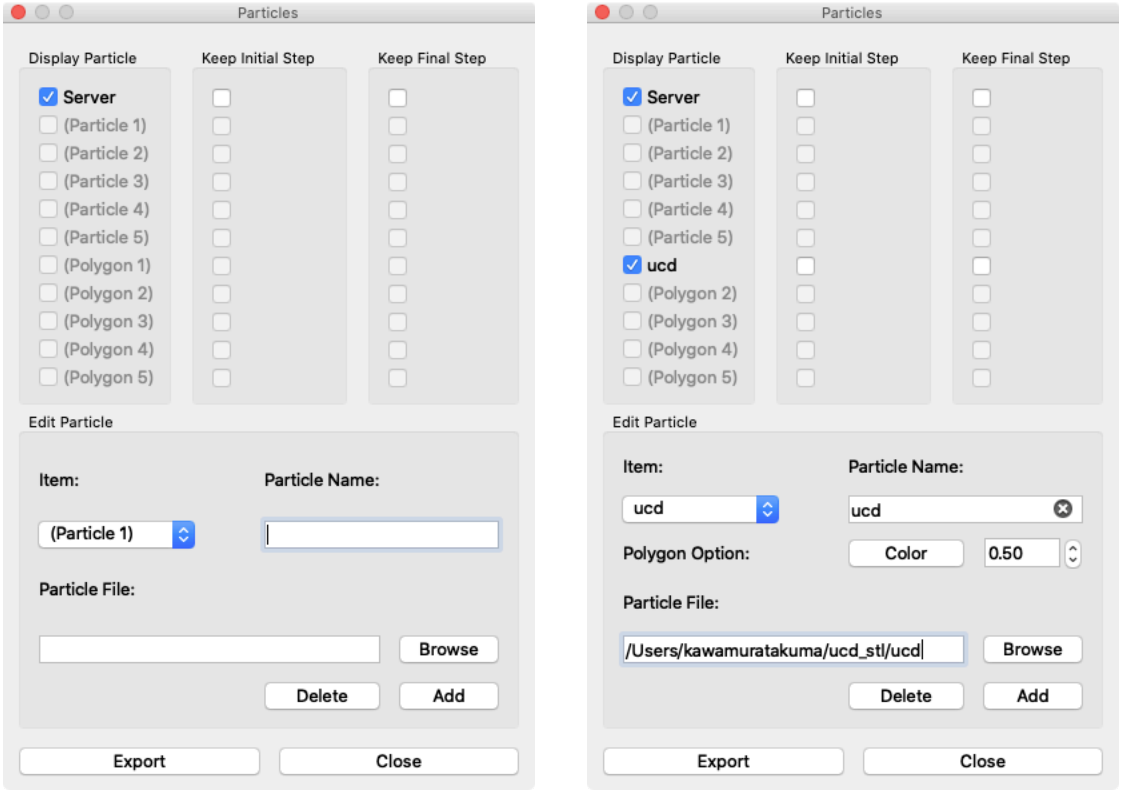


図 6-21　粒子統合エディタ。左と右はそれぞれ、起動時の状態とポリゴンを選択した状態のエディタである。

Display Particle カテゴリは統合表示の対象となる粒子データおよびポリゴンの一覧を示す。粒子データはサーバから送られてきたデータ、もしくは、クライアントのローカルファイルから読み込んだデータ（最大 5 個まで）から構成され、ポリゴンはクライアントのローカルファイルから読み込んだデータ（最大 5 個まで）から構成される。

粒子統合エディタは STL 形式のポリゴンをサポートしている。本機能は以下の命名規則に従う 1 タイムステップにつき 1 ファイルに分割された STL ファイルを処理可能である。

prefix_*****.stl　（*****は 5 桁表示のタイムステップ数）

サーバから送信されてくる粒子データと、ローカルの粒子データのサブピクセルレベルが異なる場合、サーバのサブピクセルレベルが優先される。スタンドアロンモードにおいて、複数のローカルの粒子データを表示する場合で、かつ各粒子データのサブピクセルレベルが異なる場合、後に読み込んだ粒子データのサブピクセルレベルが優先される。

Keep Initial Step カテゴリは開始タイムステップが異なる粒子データ/ポリゴンを統合表示するときに、時系列の開始前に先頭のタイムステップのデータを表示する粒子データ/ポリゴンの一覧を示

す。Keep Final Step カテゴリは終了タイムステップが異なる粒子データ/ポリゴンを統合表示するときに、時系列の終了後に最終のタイムステップのデータを表示する粒子データ/ポリゴンの一覧を示す。

【Display Particle/ Keep Initial Step/ Keep Final Step カテゴリ】

- Server チェックボックス
  クラサバモードで動作中にサーバから送られてきた粒子データを統合表示の対象とする場合にチェックする。スタンドアロンモードで動作中のときはチェックできない。
- (Particle1)～(Particle5)チェックボックス
  クライアントのローカルファイルから読み込んだ粒子データを統合表示の対象とする場合にチェックする。粒子データが読み込まれていない状態ではチェックできない。後述する Particle file パネルで粒子データを読み込む。
- (Polygon1)～(Polygon5)チェックボックス
  クライアントのローカルファイルから読み込んだポリゴンを統合表示の対象とする場合にチェックする。ポリゴンが読み込まれていない状態ではチェックできない。後述する Particle file パネルでポリゴンを読み込む。

【Edit Particle カテゴリ】

Edit Particle category では、クライアント PC のローカルファイルから粒子統合エディタの一覧へ粒子データやポリゴンを追加したり、統合して表示した粒子データを保存したりできる。

- Item スピンボタン
  (Particle1)～(Particl5), (Polygon1)～(Polygon5)の一覧から、粒子データを追加する項目を選択する。
- Particle Name
  Item スピンボタンで選択した粒子データまたはポリゴンに名前をつける。この記述を省略すると、Particle File 欄に表示されているファイル名が採用される。
- Browse ボタン
  粒子データまたはポリゴンを読み込むためのファイルダイアログを開く。読み込んだ粒子データまたはポリゴンのファイルのパスは Particle File 欄に表示される。全角文字列を含むパスを指定することはできない。
- Add ボタン
  Particle File 欄に表示されている粒子データを、Item スピンボタンで選択した項目へ追加する。既に追加済みの項目を選択した場合は、後から読み込んだ粒子データまたはポリゴンで上書きする。
- Delete ボタン
  Item スピンボタンで選択中の項目に追加されている粒子データまたはポリゴンを削除する。
- Export ボタン
  メモリ上にある粒子データを統合して Particle File 欄で指定したファイルへ出力する。
- Close ボタン

粒子統合エディタをクローズする。

・Polygon Option

Item スピンボタンで Polygon を選択した場合に出現する。ポリゴンの色と不透明度を指定できる

6.3.5.1 ボリューム・ポリゴンの合成例

サンプルデータに含まれる非構造格子データ spx.inp と、その境界形状のポリゴン spx.stl を用いた合成表示の例を示す。サーバ起動時に-vin オプションでフィルタした spx.inp のパスを指定し、クライアントでボリュームレンダリングを実行する（図 6-22 左）。次に粒子統合パネルの Edit Particle カテゴリで Item スピンボックスから Polygon を選択し、Particle File で spx.stl のパスを指定する。次に Particle Name でデータ名“spx.stl”を指定し、色と不透明度を設定する。そして Display Particle カテゴリで spx.stl をチェックする（図 6-22 右）と、境界形状が合成される（図 6-22 中央）。

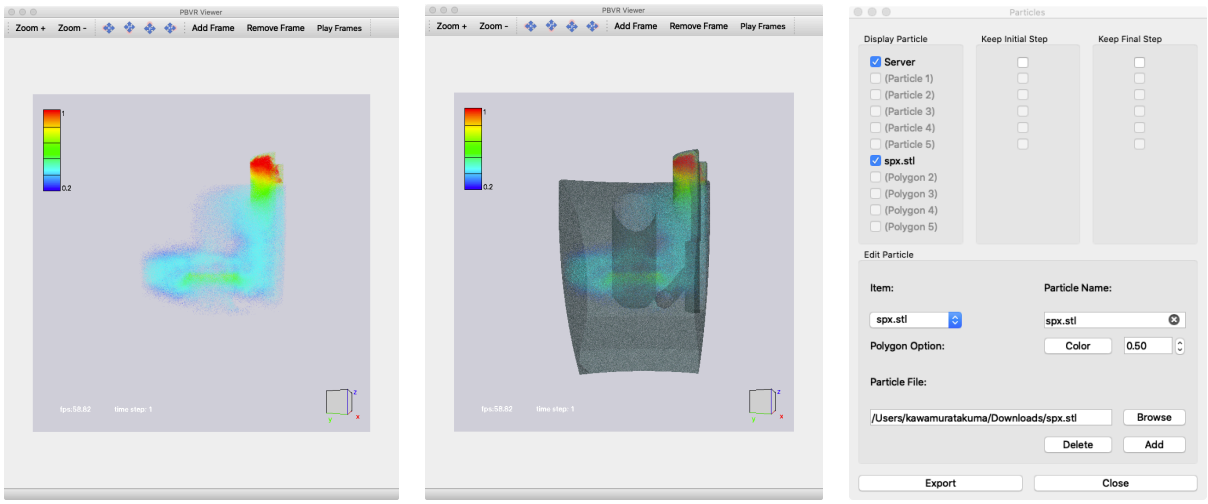


図 6-22　左：spx.inp のボリュームレンダリング結果、中央：spx.inp のボリュームと境界形状の合成表示、右：合成表示における粒子統合パネルの設定

6.3.5.2 複数タイムステップ統合時の動作例

タイムステップが異なるデータを統合した場合の動作を、図 6-23（サーバ側：1～4タイムステップのデータ、クライアント側：0～3タイムステップの粒子データがある場合）で説明する。

Server チェックボックスと Particle チェックボックスの両方がチェックされている場合、全てのステップが表示対象となる。表 6.3-2 に表示されるタイムステップを示す。

クライアントの Keep Initial Step がチェックされている場合、はじめのタイムステップが表示され続ける。表 6.3-3 に表示されるタイムステップを示す。

クライアントの Keep Final Step がチェックされている場合、最後のタイムステップが表示され続ける。表 6.3-4 に表示されるタイムステップを示す。

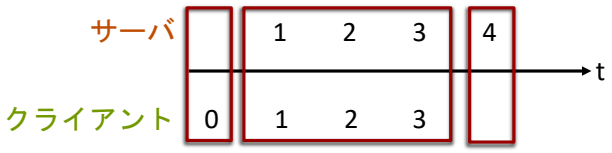


図 6-23　タイムステップの異なる粒子統合の動作

表 6.3-2　デフォルトで表示されるタイムステップ

	ステップ0	ステップ1	ステップ2	ステップ3	ステップ4
サーバ	なし	1	2	3	4
クライアント	0	1	2	3	なし

表 6.3-3　サーバの Keep Initial Step で表示されるタイムステップ

	ステップ0	ステップ1	ステップ2	ステップ3	ステップ4
サーバ	1	1	2	3	4
クライアント	0	1	2	3	なし

表 6.3-4　クライアントの Keep Final Step で表示されるタイムステップ

	ステップ0	ステップ1	ステップ2	ステップ3	ステップ4
サーバ	なし	1	2	3	4
クライアント	0	1	2	3	3

6.3.6. 画像ファイル作成

動画作成用パネルで、ビューワに表示した内容をキャプチャして保存し、動画として再生できる。動画作成用パネルはメインパネルの Animation Control Panel ボタンを押すと現れる。以下の２つのモードでのキャプチャができる。

- イメージキャプチャ

ビューワに表示した内容を時刻毎の連番画像ファイルとして保存する。画像ファイルの形式は BMP である。保存した一連の画像ファイルは、ImageMagic の convert や ffmpeg 等の外部コマンドを用いて、mpeg 等の動画ファイルとして圧縮できる。

- キーフレームアニメーション

ビューワに表示した内容に対応するビュー情報を、任意のタイミングでキーフレームとして保存する。保存した一連のキーフレームは動画として再生できる。

動画作成用パネルの内容を図 6-24 に示す。パネル内の各項目について以下に説明する。

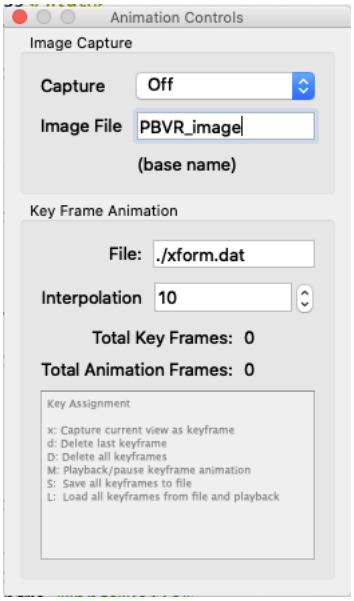


図 6-24　動画作成用パネル

・Capture プルダウンメニュー

イメージキャプチャの開始／終了を指定する。選択肢は off または on。

・Image File

イメージキャプチャした内容を連番画像ファイルとして保存するときのファイル名の、ベース名の部分を指定する。省略値は「PBVR_image」。

・File

キーフレームアニメーションでキャプチャしたキーフレームを保存するファイル名を指定する。省略値は「./xform.dat」。

・Interpolation

キーフレームアニメーション再生時にキーフレーム間のビューを補間するときのフレーム数を指定する。省略値は 10。補間は線形・等間隔で行う。

・Total Key Frames

キーフレームアニメーションで現在キャプチャされているキーフレームの数を表示する。初期値は 0。x キーの操作が成功する度に現在値+1、d キーの操作が成功する度に現在値-1、D キーの操作が成功すれば 0 という表示結果になる。

・Total Animation Frames

キーフレームアニメーション再生時のフレーム数を表示する。フレーム数は、

（Total Key Frames の値 － 1）＊ Interpolation の値

である。

6.3.6.1 イメージキャプチャ

イメージキャプチャを行うときの操作方法を説明する。

【操作方法】

① image file で、連番画像として保存するファイル名のベース名の部分を指定する。
② capture プルダウンメニューで on を選択する。
③ 時刻の更新毎にビューワの表示内容が連番画像として保存される。
④ capture プルダウンメニューで off を選択すると、連番画像の保存を停止する。

連番画像は、クライアント起動時に-iout オプションで指定したディレクトリの下に保存される。-iout を省略した場合、クライアントを起動したディレクトリの直下に連番画像が保存される。

file で指定したベース名が PBVR_image の場合、保存される連番画像の名前は以下のようになる。

PBVR_image.00001.bmp
PBVR_image.00002.bmp
:
:

後述するキーフレームアニメーションの再生をイメージキャプチャした場合は、以下に示す例のように、連番画像として保存するファイル名のベース名の最後に ”_k” が補われる。

PBVR_image_k.00001.bmp
PBVR_image_k.00002.bmp
:
:

6.3.6.2 静止画のキーフレームアニメーション

タイムステップ制御パネルの stop ボタンを押した状態の静止画からキーフレームアニメーションを作成するときの操作方法を説明する。

【キーフレームをキャプチャし、ファイルに保存する】

① file で、キーフレームを保存するファイル名を指定する。
② ビューワウィンドウをマウスカーソルでクリックしてアクティブにする
③ キーフレームとしてキャプチャしたい描画内容に対して、キーボードの x キーを押す。これにより x キーを押した時点の描画内容に対応するビュー情報をキーフレームとしてメモリ上に保存する。
④ ③の操作を必要な数だけ繰り返す。
⑤ キーボードの M（Shift+m）キーを押す。これによりメモリ上に保存したキーフレームをアニメーションとして再生する。
⑥ 再生内容に問題が無ければ、キーボードの S（Shift+s）キーを押して、キーフレームをファイルに保存する。

【ファイルに保存したキーフレームを再生する】

① file で、キーフレームが保存されているファイル名を指定する。
② ビューワウィンドウをマウスカーソルでクリックしてアクティブにする
③ キーボードの L（Shift+f）キーを押す。これによりファイルに保存したキーフレームをアニメーションとして再生する。

F キーを押した後に x キーを押すと、ファイルからメモリ上に読み込んだキーフレームに対して新たなキーフレームを追記する。

キーフレームアニメーションで使うキーの機能を表 6.3-5 に示す。

表 6.3-5　キーフレームアニメーションの操作キー

キー	機能
x	現在のビューワの表示に対応するビュー情報をキーフレームとしてメモリ上に保存する
d	メモリ上の最終キーフレームを削除する
D	メモリ上の全てのキーフレームを削除する
M	メモリ上のキーフレームをアニメーション表示／一時停止する
S	メモリ上のキーフレームをファイルへ保存する
L	ファイルからメモリ上に読み込んだキーフレームをアニメーション表示する

6.3.6.3 時系列データのキーフレームアニメーション

時系列データのキーフレームアニメーションを作成する場合の動作について説明する。

① 時系列データを描画中にｘキーを押すと、キーフレーム情報として、ビューと共に、現在表示している時系列オブジェクトのタイムステップ番号を保存する。ただしオブジェクトのファイル名は保存しない（これにより、オブジェクトＡで作成したキーフレーム情報をオブジェクトＢに対して適用することができる）。

② Ｓキーを押したときに、ビューと共に、タイムステップ番号もキーフレーム情報としてファイルへ保存する。

③ Ｆキーを押したときに、ビューと共に、タイムステップ番号もメモリに読み込んでキーフレームアニメーションを開始する。これにあたり、パネルの interpolation で指定したコマ数でタイムステップ番号を補間しながら、対応するタイムステップ番号のオブジェクトをキーフレームとして読み直す。

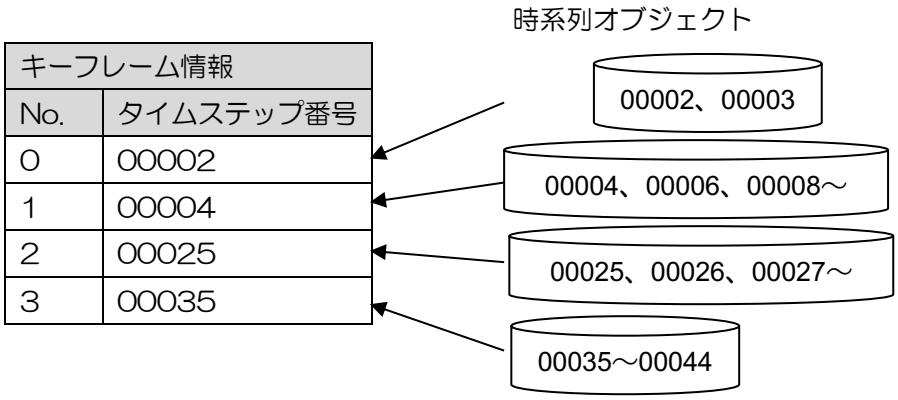


図 6-25　時系列データのキーフレームアニメーション構成

図 6-24　動画作成用パネルの例で interpolation を 10 フレームに指定した場合、キーフレーム No.0 と No.1 の間のビュー情報を補間した 10 個のビューをタイムステップ番号 00002 と 00003 のデータに割り当てて５フレームずつ表示する。次に、キーフレーム No.1 と No.2 の間の 10 フレームに関しては 10 個のビューを時間方向に等間隔に割り当てて、タイムステップ番号 00004、00006、…000024 のデータを表示する。

6.3.6.4 キーフレーム情報ファイルのフォーマット

キーフレーム情報を保存するファイル（省略値は「./xform.dat」）はバイナリファイルである。そのファイルフォーマットを図 6-26 に示す。

ファイルフォーマット

キーフレームデータ 1
キーフレームデータ 2
:

型	サイズ (byte)	用途
int	4	タイムステップ
float	4	rotation[0].x
float	4	rotation[0].y
float	4	rotation[0].z
float	4	rotation[1].x
float	4	rotation[1].y
float	4	rotation[1].z
float	4	rotation[2].x
float	4	rotation[2].y
float	4	rotation[2].z
float	4	translation.x
float	4	translation.y
float	4	translation.z
float	4	scaling.x
float	4	scaling.y
float	4	scaling.z

図 6-26　キーフレーム情報ファイルのフォーマット

6.3.7. レジェンドパネル

物理量と色との対応を示すカラーバーを表示するレジェンドパネルを図 6-27 に示す。レジェンドパネルはメインパネルの Legend Panel ボタンを押すと現れる。パネル内の各項目について以下に説明する。

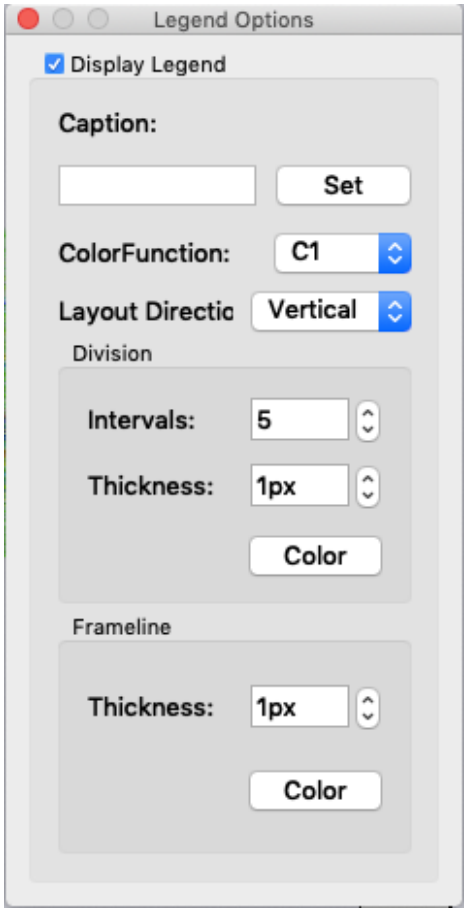


図 6-27　レジェンドパネル

- Display Legend チェックボックス
  レジェンド表示の On/Off を指定する。
- Caption テキストボックス
  レジェンドに対するキャプション文字列を入力する。Set ボタンで内容が反映される。
- Color Function：スピンボタン
  レジェンドのカラーマップと値域を定義する伝達関数を選択する。
- Layout Direction スピンボタン
  レジェンドの向き（縦／横）を選択する。
- Division
  レジェンドの目盛線に関する属性を指定する。
  Intervals：目盛線の数
  Thickness：目盛線の太さ
  Color：目盛線の色

・Frameline

レジェンドの枠線に関する属性を指定する。

Thickness：枠線の太さ

Color：枠線の色

レジェンドの表示例を図 6-28　レジェンドの表示例図 6-28 に示す。

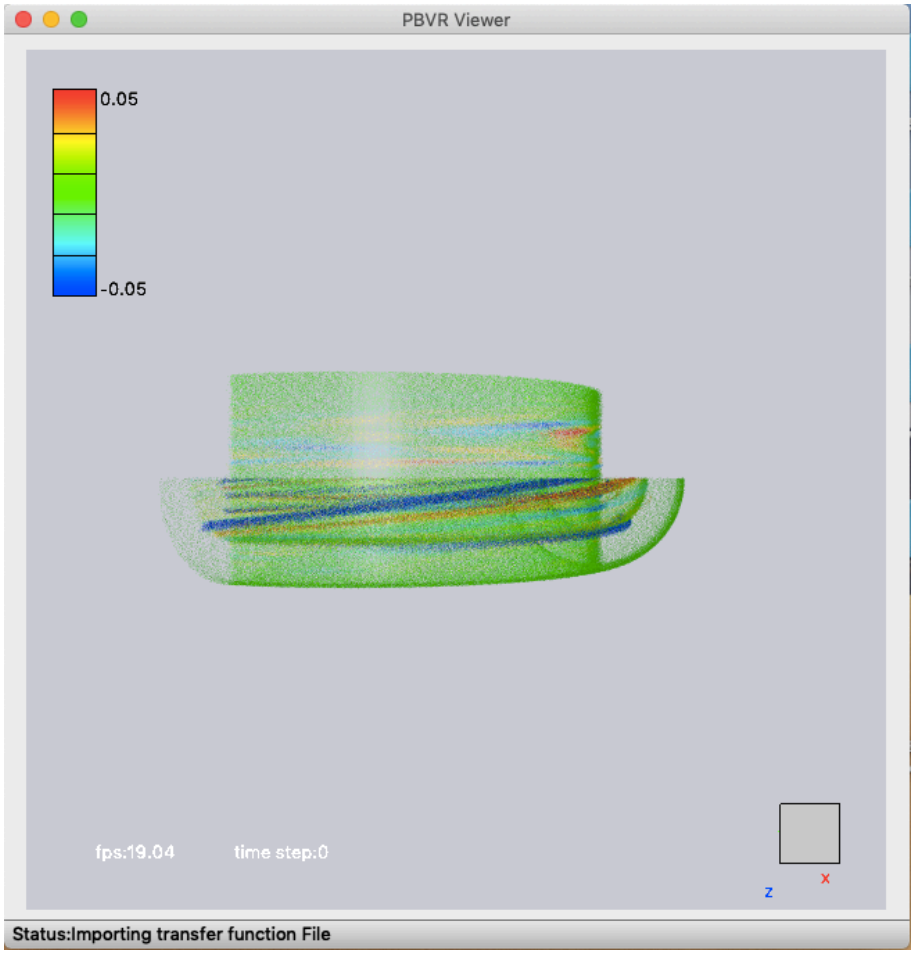


図 6-28　レジェンドの表示例

6.3.8. 座標エディタ

サーバサイドで作成される粒子の座標軸に対して、関数エディタを利用した座標軸変換機能を利用することが出来る。これにより、直交座標系から円筒座標系へ変換して可視化を行うなどのことが可能になる。座標エディタパネルを図 6-29 に示す。

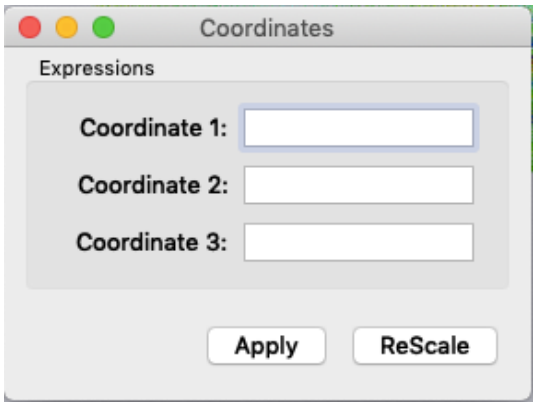


図 6-29　座標エディタパネル

- Coordinate 1 テキストボックス
  新たな X 軸を決定する式を入力する。空白は「X」を意味する。
- Coordinate 2 テキストボックス
  新たな Y 軸を決定する式を入力する。空白は「Y」を意味する。
- Coordinate 3 テキストボックス
  新たな Z 軸を決定する式を入力する。空白は「Z」を意味する。
- ReScale ボタン
  ボタンを押すことでそれまでに表示されたデータが持つ座標値の最大値と最小値を用いてビューワーの表示をリスケールする。
- Apply ボタン
  本パネルで作成した伝達関数をサーバへ送信する。

Coordinate 1～3 のテキストボックスには、座標軸の変換式を設定できる。式中に記述できる変数は、元の座標値(X、 Y、 Z)、物理量(q1、q2、…、q9)および時刻(T)である。X、 Y、 Z、 T については、大文字/小文字を区別しない。また、式中に記述できる演算は伝達関数エディタ（6.3.3.3 を参照）と同じである。指定した物理量がデータ中に存在しない場合は 0 として評価される。

Coordinate 1～3 のテキストボックスは、変換式を記述後、Apply ボタンを押すと変換が反映される。

6.3.9. ビューワ制御パネル

ビューワの属性を制御するビューワ制御パネルを図 6-30 に示す。ビューワ制御パネルはメインパネルの Viewer Control Panel ボタンを押すと現れる。パネル内の各項目について以下に説明する。

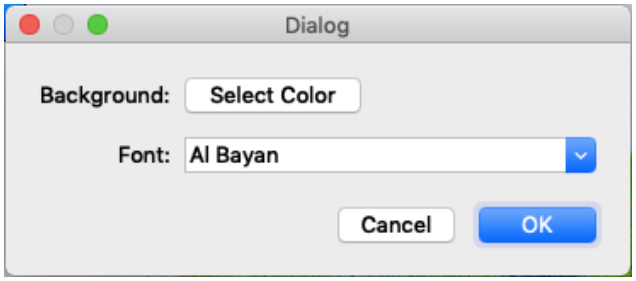


図 6-30　ビューワ制御パネル

• Background

ビューワの背景色を指定する。

• Font

ビューワに表示される文字の Font の種類とサイズを選択する。

7 サンプルデータを用いた可視化事例

サンプルデータ gt5d.tgz を用いた可視化処理によって PBVR の機能を説明する。

7.1. フィルタ処理

gt5d.tgz を展開すると./gt5d 以下に下記のファイルが生成される。

gt5d.fld	AVS フィールドファイル
co3d.dat	座標データ
pd3d.dat	変量１
psid.dat	変量２
param.txt	フィルタプログラム入力パラメータ
demo.tf	デモ用伝達関数ファイル

フィルタプログラムを実行する。（OpenMP 版）

```
$ filter ./gt5d/param.txt
```

param.txt の内容は以下の通り。

```
#
in_dir=./gt5d
field_file=gt5d.fld
out_dir=./gt5d
out_prefix=case
start_step=0
end_step=4
```

上記の例ではデフォルトの SPLIT ファイル形式、領域分割数１とし、入力と出力に同じディレクトリを指定している。この処理によって指定ディレクトリに以下のファイルが生成される。

case.pfi	pfi ファイル
case_YYYYYYY_ZZZZZZZ_connect.dat	要素構成ファイル
case_YYYYYYY_ZZZZZZZ_coord.dat	ノード座標ファイル
case_XXXXX_YYYYYYY_ZZZZZZZ.kvsml	kvsml ファイル
case_XXXXX_YYYYYYY_ZZZZZZZ_value.dat	成分ファイル

7.2. プログラム起動

ssh ポートフォワードを用いて machineA の 50000 番ポートを machineB の 60000 番ポートに接続し、各マシンでクライアントプログラムとサーバプログラムを起動する例を示す。

手順 1[ssh ポートフォワード]
machineA> ssh -L 50000:localhost:60000 username@machineB

手順 2[サーバ起動]
machineB> mpiexec -n pbvr_server
first reading time[ms]:0
Server initialize done
Server bind done
Server listen done
Waiting for connection ...

手順 3[クライアント起動] この起動例では伝達関数ファイル demo.tf を起動時に読み込み、粒子生成処理にメトロポリスサンプリングを使用、シェーディングパラメータにフォンシェーディングを設定している。
machineA> pbvr_client -S m -vin ./gt5d/case.pfi -tf demo.tf -shading P,0.6,0.6,0.6,30

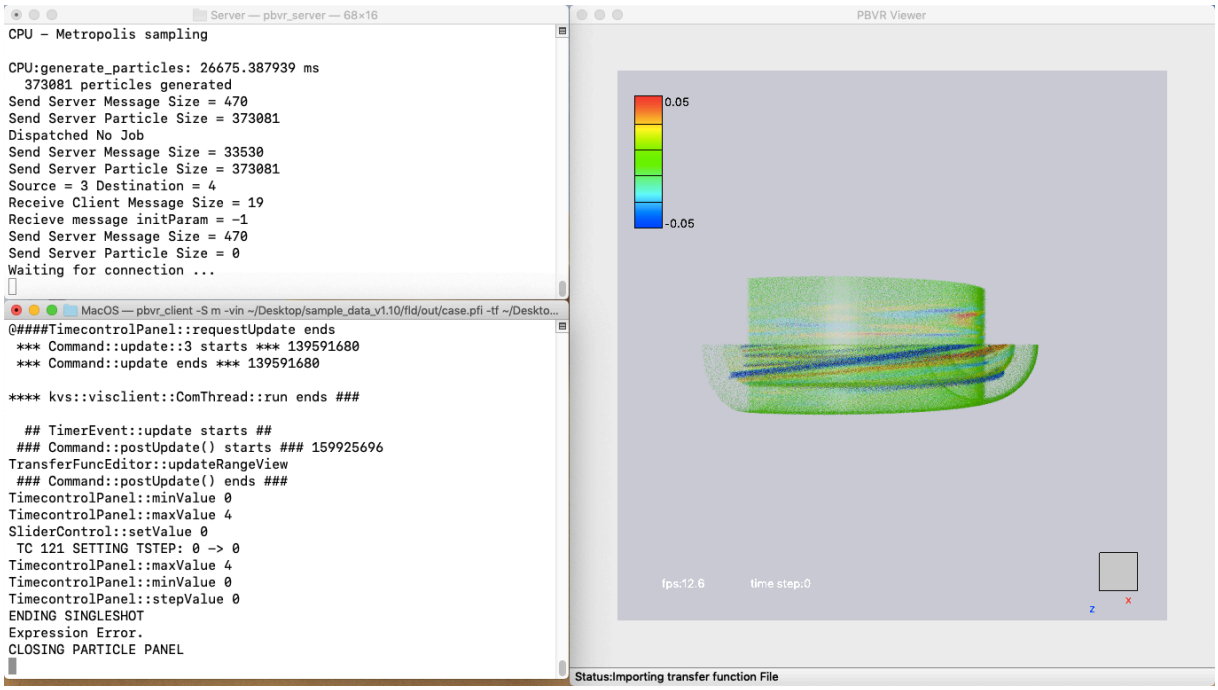


図 7-1 PBVR 起動直後の状態

7.3. 伝達関数の設計

以下では 2 変数の構造格子ボリュームデータである gt5d.fld に対して PBVR の高度な伝達関数設計機能を適用した可視化事例を紹介する。

7.3.1. 単変量のボリュームレンダリング

まずは伝達関数 t1 を図 7-2 のように設定して変数 q1 の概要を確認する。この例では、パネル左側の色の設計とパネル右側の不透明度の設計の両方に変数 q1 を使用して単変量のボリュームレンダリングを行っている。

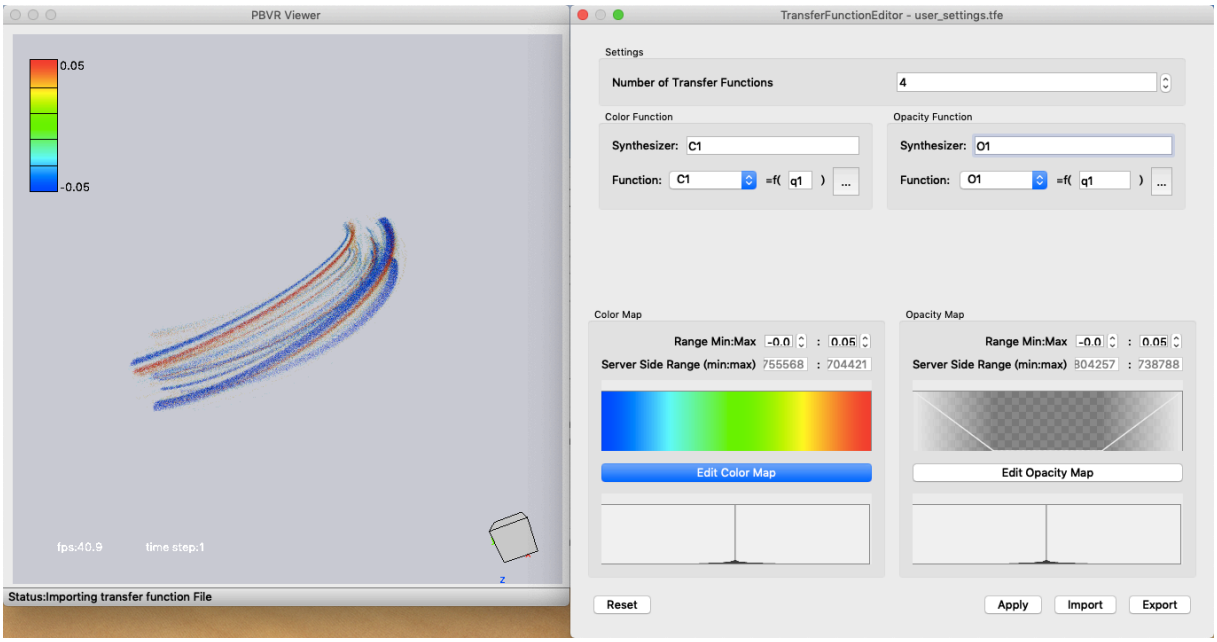


図 7-2 変量 q1 に対する単変量ボリュームレンダリング

Color Function SYNTHESIZER ：C1
Opacity Function SYNTHESIZER ：O1
Color Map Function：C1 = f(q1)
Opacity Map Function：O1 = f(q1)

7.3.2. 多変量のボリュームレンダリング

次に変量 q1 と変量 q2 を合成する多変量のボリュームレンダリングの事例を示す。この例では、色の設計に変量 q1 を使用し、不透明度の設計に変量 q2 を使用している。不透明度の伝達関数では変量 q2 のトーラス状の等値面 2 枚を抽出し、そこでの変量 q1 の変化を色で示している。

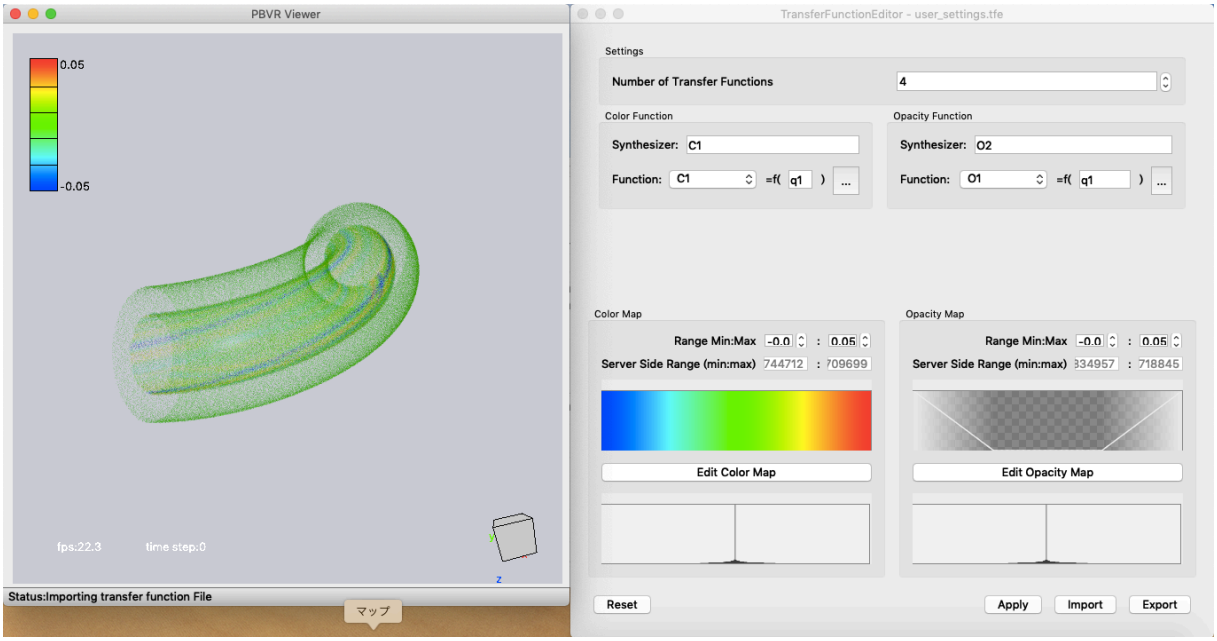


図 7-3　変量 q2 の等値面に変量 q1 の色をマップする多変量ボリュームレンダリング

Color Function SYNTHESIZER ：C1
Opacity Function SYNTHESIZER ：O2
Color Map Function：C1 = f(q1)
Opacity Map Function：O2 = f(q2)

7.3.3. 断面表示

多変量ボリュームレンダリングの応用例として図 7-4 に断面表示例を示す。この例では不透明度の設計に座標を使用している。PBVR では伝達関数設計に任意の関数を利用可能であり、この例では X^2+Z^2=const.という円筒面を抽出して、そこでの変量 q1 の変化を示している。

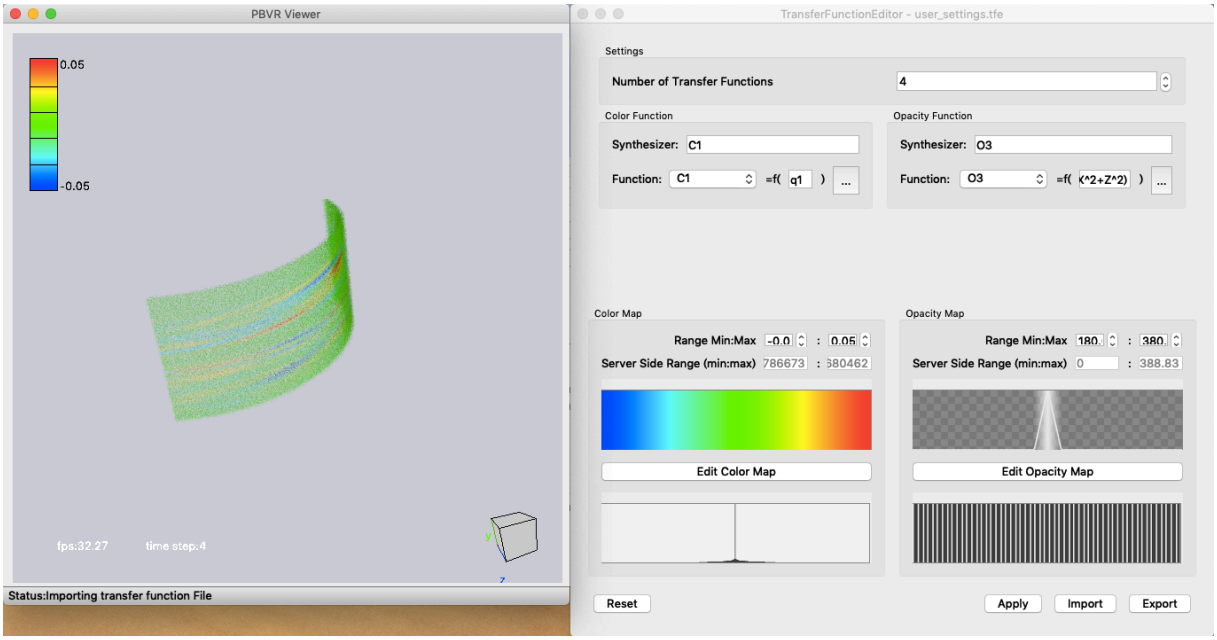


図 7-4　多変量ボリュームレンダリング機能を応用した断面表示

Color Function SYNTHESIZER ：C1

Opacity Function SYNTHESIZER ：O3

Color Map Function：C1 = f(q1)

Opacity Map Function：O3 = f(sqrt(X^2+Z^2))

Opacity Range Min：180

Opacity Range Max：380

7.3.4. 伝達関数の合成

PBVR における伝達関数の合成機能を説明する。図 7-5 では不透明度を座標 Y で与え、Y>0 の領域を透明にする伝達関数 O4 を定義し、伝達関数 O4 と先に定義した 3 つの伝達関数 O1、O2、O3 を(O1+O2)*O4+O3 と合成することにより部分領域を抽出する可視化例を示している。伝達関数の合成にあたり、C2 と C3 の色は（R、G、B）＝（0、0、0）と設定し、C4 の色は（R、G、B）＝（1、1、1）と設定している。これに上記合成式を適用すると、色の設計は C1 で使用されている q1 の rainbow によって定義される。一方、不透明度の設計は O1 と O2 を合成したものに O4 をかけて Y<0 の領域を抽出し、それと O3 の円筒面を合成したものが定義される。PBVR では通常の領域抽出機能を Crop パネルで実装しているが、伝達関数による領域抽出機能は着目する等値面やボリュームレンダリングのみに適用できるので極めて自由度が高い。

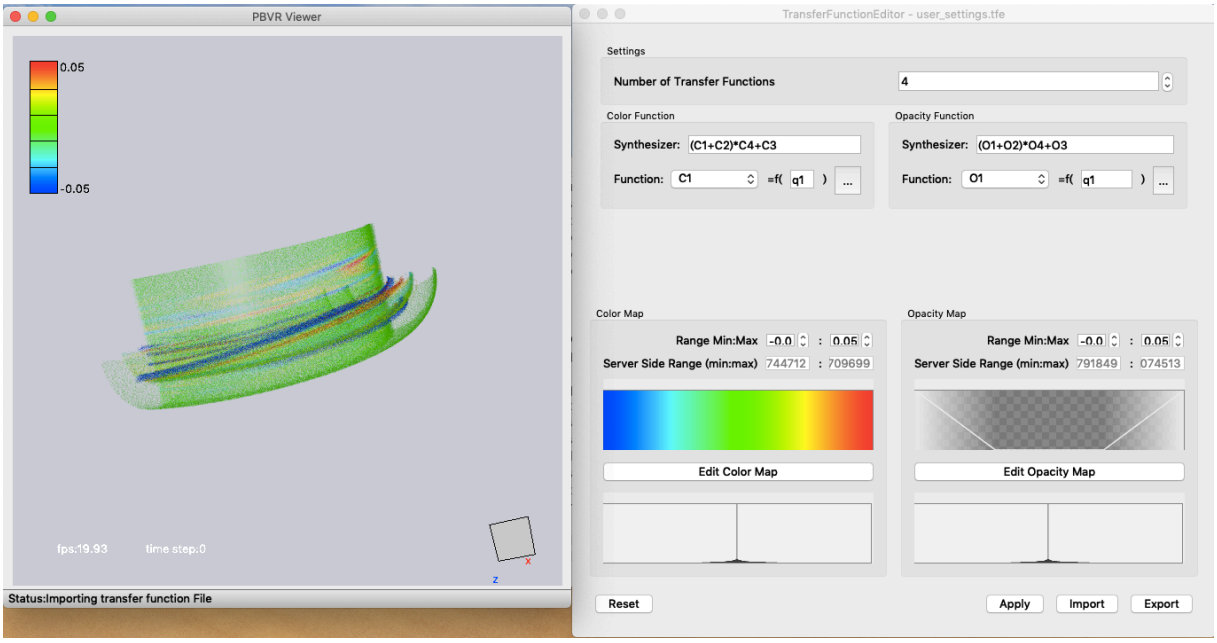


図 7-5　伝達関数の合成機能

Color Function SYNTHESIZER ：(C1+C2)*C4+C3
Opacity Function SYNTHESIZER ：(O1+O2)*O4+O3

７.４. 粒子データの統合

前節では多次元伝達関数を用いたボリュームレンダリング表示、等値面表示、断面表示の統合例を示したが、粒子統合機能を用いても同様の複雑な可視化処理を実現できる。以下にその手順を説明する。

７.４.１. 粒子データの保存

粒子データの保存は粒子統合エディタにおける Particle File パネルを使用して行う。図 ７-6 に粒子データファイル名の指定例を示す。この場合にはプレフィックスが p1 となり、以下のファイルが生成される。

./particle/p1_XXXXX_YYYYYYY_ZZZZZZZ.kvsml
./particle/p1_XXXXX_YYYYYYY_ZZZZZZZ_colors.dat
./particle/p1_XXXXX_YYYYYYY_ZZZZZZZ_coords.dat
./particle/p1_XXXXX_YYYYYYY_ZZZZZZZ_normals.dat

ここで、XXXXX は時刻、YYYYYYY はサブボリューム番号、ZZZZZZZ は全サブボリューム数を示し、colors、coords、normals はそれぞれ色、座標、法線ベクトルのデータを示す。粒子データファイル名を入力し、Export ボタンを押すと粒子保存が開始され、粒子データの保存処理中は図 ７-7 に示すように Export ボタンが非アクティブ状態となり、全時系列の粒子データを保存後に粒子保存が終了し Export ボタンがアクティブ状態に復帰する。

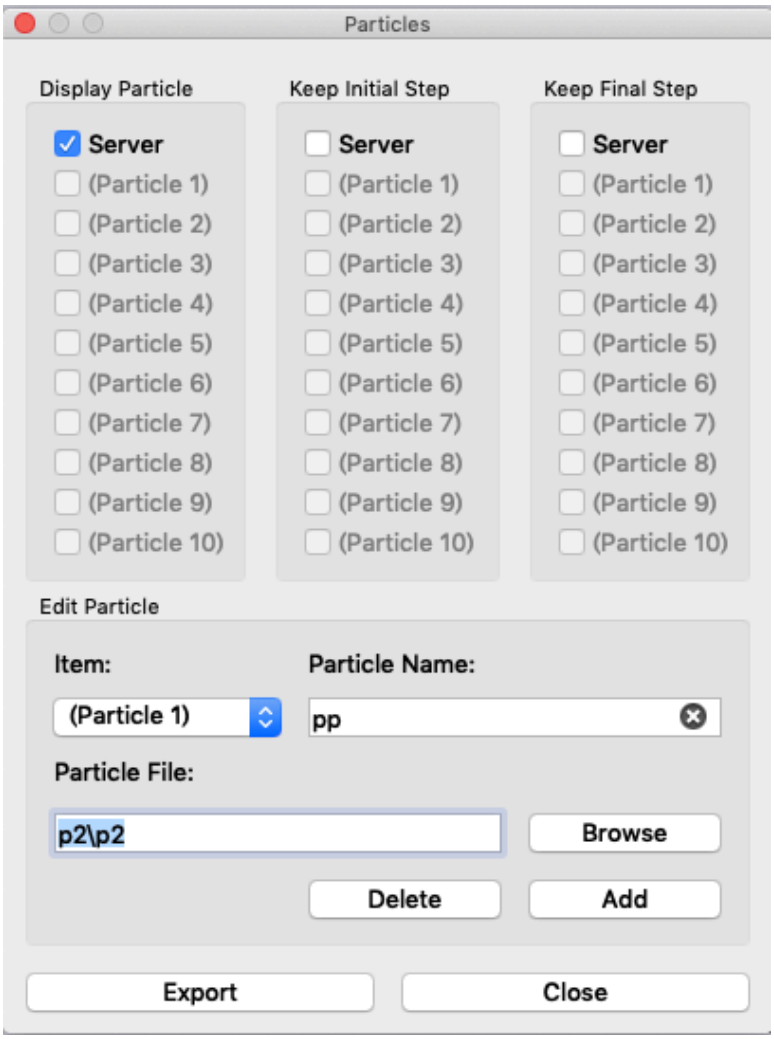


図 7-6　Particle File パネル（粒子データ保存前後）

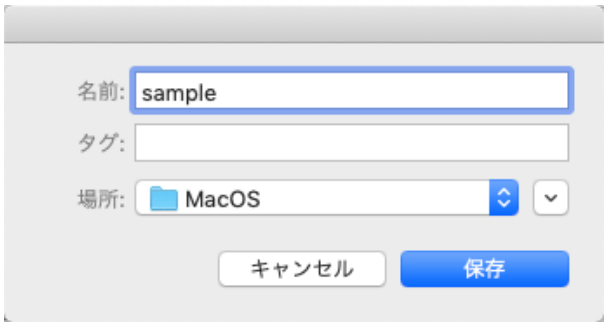


図 7-7　Particle File パネル（粒子データ保存中）

7.4.2. 粒子データの読み込み

以下の例では図 7-2～図 7-4 に示す可視化結果を保存した粒子データ p1～p3 を読み込んで合成する例を示す。ここでは以下のコマンドでクライアントプログラムをスタンドアロンモードで起動

し、コマンドラインオプションで粒子データを指定して読み込む。（クラサバモードの場合にはParticle panel で粒子データを指定して読み込む。）

$ pbvr_client -shading P、0.6、0.6、0.6、30 -pin1 ./particle/p1 -pin2 ./particle/p2 -pin3 ./particle/p3

起動後に Particle panel を開くと p1～p3 が読み込まれている。ここで、p1 の Display Particle チェックボックスをオンにすると図 7-8 のようにボリュームレンダリングの粒子データが表示される。次に、p2、および、p3 のチェックボックスをオンにすることで、図 7-9 のように３つの粒子データが統合される。Particle File パネルを使用して統合された粒子データを１つの粒子データとして保存することも可能である。粒子統合機能を使用する際には、全ての粒子データが同じ particle density、および、particle limit パラメータ（コマンドラインオプション”-pd”、および、”-plimit”、もしくは、Main panel で指定）を用いて処理されたデータでないと正しい合成画像とならないことに注意すること。

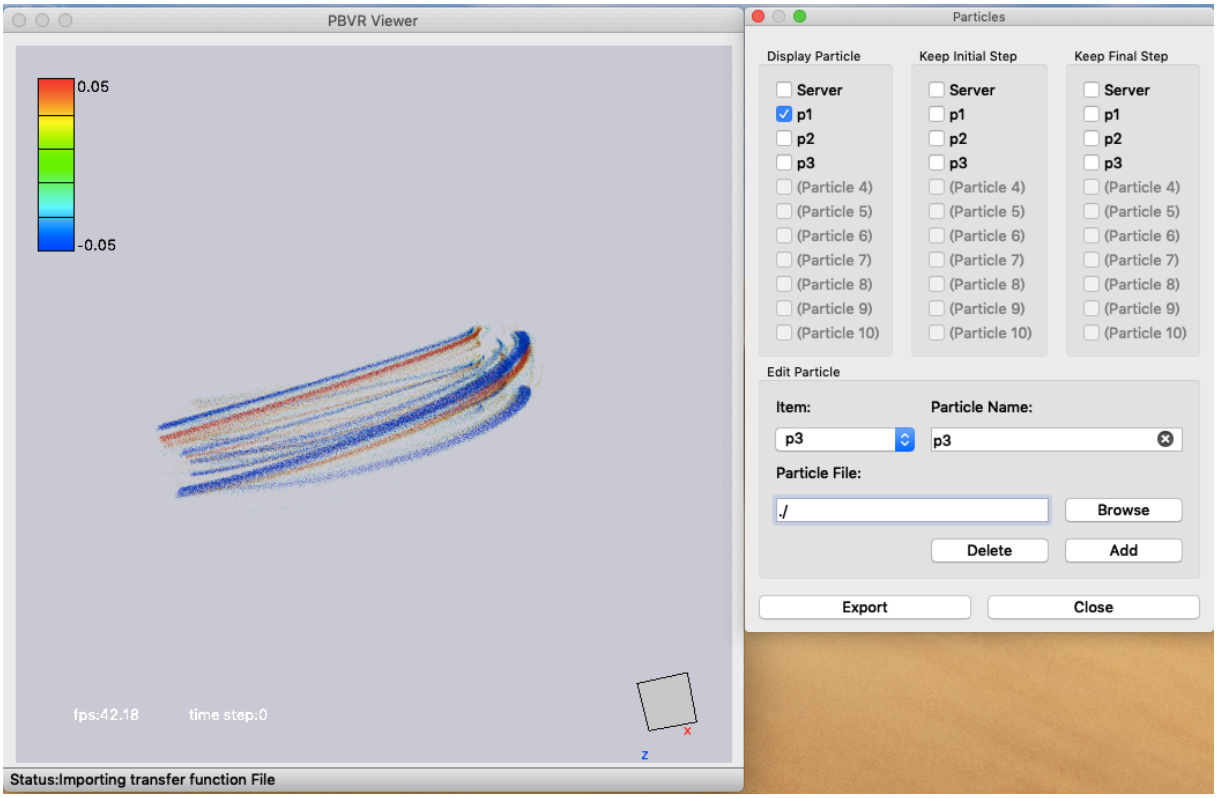


図 7-8　p1 の表示結果

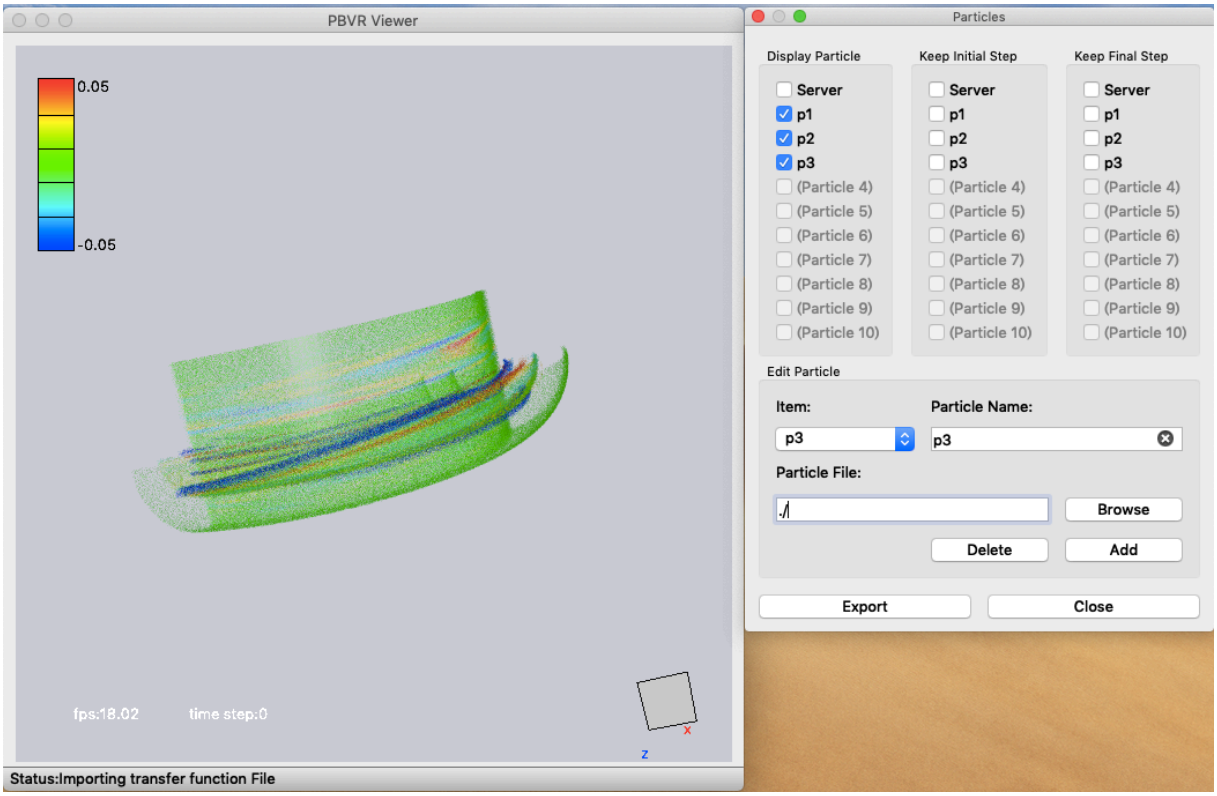


図 7-9　p1、p2、および、p3 の統合表示

7.5. 可視化結果の保存

伝達関数の設計が完了し、必要な可視化処理が完了したら可視化結果を保存する。可視化結果を保存するには以下の３つの方法が利用できる。

（1） 画像出力（6.3.6）
可視化結果を画像ファイルとして出力するには画像出力パネルの capture を on にする。これにより、ビットマップ画像ファイルが生成される。

（2） 伝達関数ファイル（6.3.3）
可視化に用いた伝達関数を保存するには伝達関数エディタの File Path に伝達関数ファイル名を入力し、Export File ボタンを押す。これにより生成される伝達関数ファイルは Import File ボタンで読み込み可能である。

（3） 可視化パラメータファイル（6.3.2）
バッチモードのサーバ処理用に伝達関数を含めた全ての可視化パラメータファイルを保存できる。メインパネルの File サブパネルでファイル名を指定し、Export File ボタンを押して可視化パラメータファイルを保存する。

7.6. バッチモード処理

前節で保存した可視化パラメータファイルを用いるバッチモードの処理例を説明する。バッチモードはスーパーコンピュータ上の超並列処理を想定して開発された機能であるが、スタンドアロンモードの描画処理では粒子生成処理の待ち時間がなくなるため、時系列データの高速な描画にも適用が可能である。

［手順１］ サーバプログラムをバッチモードで実行する（OpenMP 版）。

```
$ pbvr_server -B -vin ./gt5d/case.pfi -pout ./output/case -pa ./param.in
```

［手順２］ クライアントプログラムをスタンドモードで実行する。

```
$ pbvr_client -pin1 ./output/case -shading P、0.6、0.6、0.6、30
```