In-Situ PBVR(v1.00)マニュアル

2019年3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

改訂履歴

版数	改定日	章番号	改版内容
	2019.03.28	-	新規

目次

1　はじめに

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 In-Situ による遠隔可視化システム In-Situ PBVR のマニュアルである。In-Situ 可視化は、シミュレーションと同時に計算結果を可視化することで、大規模なデータ I/O を回避し、大規模シミュレーションを確実に可視化する。In-Situ PBVR は可視化用粒子データを用いることで大規模データを可視化用に圧縮し、手元の PC で対話的な視点移動が可能な In-Situ 可視化システムである。本システムは C++で開発され，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 KV を用いて In-Situ 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）粒子サンプラ

粒子サンプラはシミュレーションコードに結合される可視化用ライブラリである。粒子サンプラはシミュレーションの各タイムステップで計算結果を可視化用粒子に変換する。可視化用粒子は各計算ノードにつき１ファイルがストレージ上に出力される。粒子サンプラはストレージ上の可視化パラメータファイルを参照して可視化用粒子を生成する。

（２）デーモン

デーモンは対話ノード上で動作する、In-Situ PBVR のファイルベース制御プログラムである。デーモンはストレージ上の粒子ファイルを集約し、ネットワークを介してユーザ PC に転送する。またデーモンは PBVR クライアントが送信する可視化パラメータを受信し、ファイルとして出力する。

（３）PBVR クライアント

PBVR クライアントはユーザ PC 上で動作し、粒子データを画面に投影して可視化画像をビューワ上に表示する。ユーザは可視化画像を観察し、色、不透明度等の可視化パラメータを調整し、デーモンに送信する。

In-Situ PBVR 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。従来の可視化では一つの物理値に対して、色・不透明度関数（合わせて伝達関数）をマッピングすることで物理値の空間分布を描画していた。In-Situ PBVR は、代数式によって多変量向け伝達関数の設計を可能にする、伝達関数合成機を提供している。

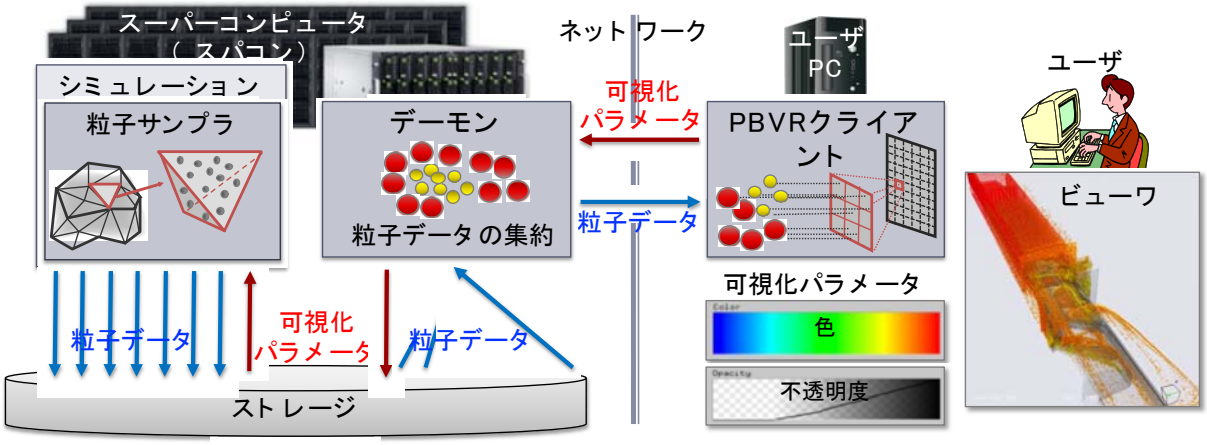


図　1-1　In-Situ PBVR フレームワークの全体構成

1.1. 粒子サンプラの概要

粒子サンプラは、MPI/OMP のハイブリッドプログラミングモデルで並列化され、更に SIMD 演算を利用することで可視化用粒子の生成を加速させている。粒子サンプラは、シミュレーションで用いる領域分割を変えることなく MPI 並列化され、各部分領域で OpenMP による要素並列で粒子を生成する。多変量可視化で必要となる物理値の合成や伝達関数の合成、そして物理値の補間計算が SIMD 演算の利用によりベクトル化されている。

粒子サンプラはシミュレーションで利用する格子の種類に合わせて、構造格子向け、非構造格子向け、そして階層格子向けの３種類が利用可能である。粒子サンプラは C／C++そして、FORTRAN で記述されたシミュレーションコードをサポートしている。In-Situ 可視化するために、粒子サンプラはシミュレーションのタイムステップループに挿入される。この時、シミュレーション結果の多変量データ、格子データ、そして領域の全体座標が粒子サンプラの引数として入力される。また、多変量データのメモリレイアウトは配列構造体（SOA）を仮定している。

1.2. デーモンの概要

デーモンはスーパーコンピュータの対話ノードあるいは対話ジョブで実行され、バッチ処理実行時の対話的可視化を実現するための要となる。デーモンはストレージ上のファイル情報を常に監視しており、粒子サンプラが出力した粒子ファイルを収集する。この収集作業は粒子サンプラ及びシミュレーションと非同期であるために、シミュレーションのパフォーマンスを阻害しない。粒子ファイルの収集は OpenMP で並列化されていて、デーモンは読み込んだ粒子データを一つの粒子データに集約して PBVR クライアントに送信する。また PBVR クライアントから送信された可視化パラメータを受信し、粒子サンプラが使用するストレージ上の可視化パラメータファイルを更新する。

In-Situ PBVR ではデーモンと PBVR クライアントはインターネットを介したソケット通信でデータを送受信するため、対話ノードとユーザ PC がポートフォワードで接続される必要がある。そのため、ポートフォワードが許可されない運用のスーパーコンピュータではデーモンが動作せず、対話的可視化が利用できない。またバッチ処理終了時までファイル出力が行われないステージングによる I/O を採用したスーパーコンピュータでもまた対話的可視化が利用できない。

1.3. PBVR クライアントの概要

PBVR クライアントはユーザ PC 上で起動され、可視化結果を表示するための画面、そして多変量可視化を実現するための機能である“伝達関数合成器”（TFS）から構成されている。TFS は、結果データに含まれる変量を組み合わせ新しいボリュームデータを生成するボリュームデータ合成機能と、複数の伝達関数を組み合わせる伝達関数合成機能を備えている。ユーザは TFS 上で代数式により合成関数を指定する。代数式は可視化パラメータとして粒子計算モジュールに転送され、数式処理機能によりリアルタイムにボリュームデータ・伝達関数の合成が行われる。ユーザは TFS 上の代数式として、初頭的な数学関数や、物理変数の空間微分が利用可能であり、それらを組み合わせて自在に数式を構築可能である。

2 パッケージ構成

In-Situ PBVR はソースコードパッケージおよびロードモジュールパッケージとして提供される。In-Situ PBVR を構成するデーモンと PBVR クライアントは単体で動作するプログラムである。対して粒子サンプラはシミュレーションコードに結合するライブラリとして提供され、以下の３つで構成される。

（１） 粒子生成機能を提供する粒子生成ライブラリ
（２） 可視化機能を提供する KVS ライブラリ
（３） 代数式処理機能を提供する数式処理ライブラリ

2.1. ロードモジュールパッケージ

In-Situ PBVR のロードモジュールのうち、粒子サンプラおよびデーモンは機構所有のスーパーコンピュータ ICEX および KNL クラスタ（JKNL）で生成したものが提供されている。また PBVR クライアントは Mac、Windows、Linux に対して生成したものが提供されている。これ以外の環境で利用する場合は、ユーザがソースコードからビルドする必要がある。以下の表 2-1~表 2-7 にロードモジュールの一覧を示す。本コードは MPI ライブラリとして、Intel MPI での動作をサポートしている。

表 2-1 構造格子向け粒子生成ライブラリ（粒子サンプラの一部）

機種	並列化	ロードモジュール名
ICEX	MPI+OpenMP	InSituLib_struct/libInSituPBVR_icex.a
JKNL	MPI+OpenMP	InSituLib_struct/libInSituPBVR_jknl.a

表 2-2 非構造格子向け粒子生成ライブラリ（粒子サンプラの一部）

機種	並列化	ロードモジュール名
ICEX	MPI+OpenMP	InSituLib_unstruct/libInSituPBVR_icex.a
JKNL	MPI+OpenMP	InSituLib_unstruct/libInSituPBVR_jknl.a

表 2-3 階層格子向け粒子生成ライブラリ（粒子サンプラの一部）

機種	並列化	ロードモジュール名
ICEX	MPI+OpenMP	InSituLib_AMR/libInSituPBVR_icex_a
JKNL	MPI+OpenMP	InSituLib_AMR/libInSituPBVR_jknl.a

表 2-4 KVS ライブラリ（粒子サンプラの一部）

機種	並列化	ロードモジュール名
ICEX		libkvsCore_icex.a

JKNL		libkvsCore_jknl.a

表 2-5　数式処理ライブラリ（粒子サンプラの一部）

機種	並列化	ロードモジュール名
ICEX		libpbvrFunc_icexa
JKNL		libpbvrFunc_jknla

表 2-6　デーモン

機種	並列化	ロードモジュール名
ICEX	OpenMP	pbvr_daemon_icex
JKNL	OpenMP	pbvr_daemon_jknl

表 2-7　PBVR クライアント

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

※1. Windows 版については glut32.dll も同じディレクトリにコピーする。

2.2. ソースコードパッケージ

ソースコードパッケージを所望の環境でコンパイルすることで、In-Situ PBVR の粒子サンプラ、デーモン、そして PBVR クライアントが生成される。ソースコードパッケージを構成するソースツリーを表 2-8 に示す。ソースコードパッケージには粒子サンプラ結合済みのテストシミュレーションコードが含まれている。

表 2-8 In-Situ PBVR のソースツリー

ディレクトリ・ファイル	説明
pbvr_inSitu_1.00/	1.00 はバージョン番号
|-pbvr.conf	make の設定ファイル
|-Makefile	サンプラ、デーモン、クライアントのコンパイルする
|-arch/	各環境のコンパイラの設定ファイル
|-Client/	PBVR クライアントプログラム
|-Common/	プロトコル、通信、共通ライブラリ
|-Daemon/	デーモンプログラム
|-Example/	テストシミュレーションコードのサンプル
|C/	C 版
|-Hydrogen_struct/	構造格子テストシミュレーションプログラム
|-Hydrogen_AMR/	階層構造格子テストシミュレーションプログラム
|-Hydrogen_unstruct/	非構造格子テストシミュレーションプログラム
|-Fortran/	Fortran 版
|-Hydrogen_struct/	構造格子テストシミュレーションプログラム
|-Hydrogen_AMR/	階層構造格子テストシミュレーションプログラム
|-Hydrogen_unstruct	非構造格子テストシミュレーションプログラム
|-FunctionParser/	数式処理ライブラリ
|-glui/	GUI ウィジットライブラリ
|-InSituLib/	粒子生成ライブラリ
|-struct/	構造格子向け粒子生成ライブラリ
|-AMR/	階層構造格子向け粒子生成ライブラリ
|-unstruct/	非構造格子向け粒子生成ライブラリ
|-KVS/	可視化ライブラリ KVS
|-shell/	スパコン用の実行シェルのサンプル

3 ビルド方法

ソースパッケージのビルドは、コンフィグファイルである pbvr.conf を環境に合わせて編集することで制御される。pbvr.conf 内で指定している変数の概要を表 3-1 に、PBVR_MACHINE の値として利用できるコンパイル設定ファイルの一覧を表 3-2 に示す。

表 3-1　pbvr.conf の変数一覧

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を指定
PBVR_MAKE_CLIENT	0 or 1	0 ⇒ デーモン、粒子サンプラがコンパイル対象 1 ⇒ PBVR クライアントがコンパイル対象

表 3-2　コンパイル設定ファイル一覧

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_fujitsu	富士通コンパイラによる逐次版コンパイルの設定
Makefile_machine_fujitsu_omp	富士通コンパイラによる OpenMP 版コンパイルの設定
Makefile_machine_fujitsu_mpi_omp	富士通コンパイラによる MPI+OpenMP 版コンパイルの設定
Makefile_machine_icex	当機構大型計算機上での逐次版コンパイルの設定
Makefile_machine_icex_omp	当機構大型計算機上での OpenMP 版コンパイルの設定
Makefile_machine_icex_mpi_omp	当機構大型計算機上での MPI+OpenMP 版コンパイルの設定
Makefile_machine_jknl_omp	当機構 KNL クラスタでの OpenMP 版コンパイルの設定
Makefile_machine_jknl_mpi_omp	当機構 KNL クラスタでの MPI+OpenMP 版コンパイルの設定

3.1. デーモン、粒子サンプラ

ソースパッケージからデーモンおよび粒子サンプラのライブラリをビルドする手順を示す。以下の手順では、ファイルのダウンロード先を$HOME/JAEA とする。また粒子サンプラはライブラリのビルド後、ソースパッケージに添付するテストシミュレーションコード（表 2-8）に対してリンクされる。

① pbvr_inSitu_1.00.tar.gz を解凍する。

```
$ cd $HOME/JAEA
$ tar xvfz pbvr_inSitu_1.00.tar.gz
```

② pbvr_inSitu_1.00/pbvr.conf（表 3-1）を編集し、ビルドする内容を指定する。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00
$ cat pbvr.conf
PBVR_MACHINE=Makefile_machine_gcc_mpi_omp
PBVR_MAKE_CLIENT=0
```

PBVR_MACHINE ⇒ pbvr_inSitu_1.00/arch 以下のコンパイル設定ファイル（表 3-2）を指定。
PBVR_MAKE_CLIENT ⇒ 0 を指定。（0：デーモン、粒子サンプラ）

③ KVS ライブラリは PBVR_MAKE_CLIENT=0 の場合に OpenGL および GLUT の機能が無効化されて、PBVR_MAKE_CLIENT=1 の場合に OpenGL および GLUT の機能が有効化されてビルドされる。そのため、直前にクライアントをビルドしていた場合、以下のコマンドを実行し、KVS をリビルドする必要がある。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00/KVS
$ make all_clean
$ make
```

④ ソースツリーのルートから全体をビルドする。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00
$ make
```

このビルドはデーモンおよび粒子サンプラに関連して、下表 3-3 デーモン・粒子サンプラのロードモジュールのロードモジュールを生成する。

表 3-3 デーモン・粒子サンプラのロードモジュール

ディレクトリ名	ロードモジュール名	説明
KVS	libkvsCore.a	KVS ライブラリ。粒子データフォーマットや可視化機能を提供する。
Common	libpbvrCommon.a	通信ライブラリ。ソケット通信用のプロトコルを提供する。
Daemon	pbvr_daemon	デーモン
FuctionParser	libpbvrFunc.a	数式処理ライブラリ。数式処理の機能を提供する。
InsituLib/struct	libInSituPBVR.a	粒子サンプラ
InsituLib/unstruct	libInSituPBVR.a	粒子サンプラ
InsituLib/AMR	libInSituPBVR.a	粒子サンプラ

⑤ シミュレーションコードは粒子サンプラが組み込まれ（5.4章）、上記で生成したライブラリがリンクされてビルドされる。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00/Example/C/Hydrogen_struct
$ make
```

ユーザのシミュレーションコードのビルドには、KVS ライブラリ、数式処理ライブラリ、粒子サンプラの参照・リンクが必要とされる。以下に Makefile の例を示す。

```
PBVR_DIR = ../../..
include ${PBVR_DIR}/pbvr.conf
include ${PBVR_DIR}/arch/${PBVR_MACHINE}
KVS_SOURCE_DIR = ${PBVR_DIR}/KVS/Source
FUNC_DIR = ${PBVR_DIR}/FunctionParser
INSITU_DIR = ${PBVR_DIR}/InSituLib/struct
CXXFLAGS += -I${INSITU_DIR} ¥
            -I${FUNC_DIR} ¥
            -I${KVS_SOURCE_DIR}
LDFLAGS  += -L${INSITU_DIR} -lInSituPBVR ¥
            -L${FUNC_DIR} -lpbvrFunc ¥
            -L${KVS_SOURCE_DIR}/Core/Release -lkvsCore
all: $(TARGET)
$(TARGET): $(TEST_OBJS)
      $(LD) -o $@ $^ $(LDFLAGS)
.cpp.o:
      $(CXX) $(CXXFLAGS) -o $@ -c $<
```

3.2. PBVR クライアント

ソースパッケージから PBVR クライアントをビルドする手順を示す。PBVR クライアントのビルドには OpenGL および GLUT が利用され、これらがインストールされた環境が必要である。ビルドする手順は環境（Linux/Mac, Windows）によって異なる。PBVR クライアントのビルドは下表 3-4 のロードモジュールを生成する。

表 3-4 PBVR クライアントのロードモジュール

ディレクトリ名	ロードモジュール名	説明
KVS	libkvsCore.a	KVS ライブラリ。粒子データフォーマットや可視化機能を提供する。
Common	libpbvrCommon.a	通信ライブラリ。ソケット通信用のプロトコルを提供する。
FuctionParser	libpbvrFunc.a	数式処理ライブラリ。数式処理の機能を提供する。
glui	libglui.a	GLUI ライブラリ。GUI ウィジェットの機能を提供。
Client	pbvr_client	PBVR クライアント

3.2.1. Linux・Mac

Linux および Mac 環境で PBVR クライアントをビルドする手順を示す。以下の手順では、ファイルのダウンロード先を$HOME/JAEA とする。

① pbvr_inSitu_1.00.tar.gz を解凍する。

```
$ cd $HOME/JAEA
$ tar xvfz pbvr_inSitu_1.00.tar.gz
```

② pbvr_inSitu_1.00/pbvr.conf（表 3-1）を編集し、ビルドする内容を指定する。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00
$ cat pbvr.conf
PBVR_MACHINE=Makefile_machine_gcc_mpi_omp
PBVR_MAKE_CLIENT=1
```

PBVR_MACHINE ⇒ pbvr_inSitu_1.00/arch 以下のコンパイル設定ファイル（表 3-2）を指定。
PBVR_MAKE_CLIENT ⇒ 1 を指定。（1：PBVR クライアント）

③ KVS ライブラリは PBVR_MAKE_CLIENT=0 の場合に OpenGL および GLUT の機能が無効化されて、PBVR_MAKE_CLIENT=1 の場合に OpenGL および GLUT の機能が有効化されてビルドされる。そのため、直前にデーモン・粒子サンプラをビルドしていた場合、以下のコマンドを実行し、KVS をリビルドする必要がある。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00/KVS
$ make all_clean
$ make
```

④ ソースツリーのルートから全体をビルドする。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00
$ make
```

Client ディレクトリ内部に PBVR クライアントのロードモジュールが生成される。

3.2.2. Windows

Windows 環境で PBVR クライアントをビルドする手順を示す。以下の手順では、ファイルのダウンロード先を$HOME/JAEA とする。

① pbvr_inSitu_1.00.tar.gz を解凍する。

```
pbvr_inSitu_1.00.tar.gz ⇒ C:¥pbvr
```

② 下記のホームページより glut-3.7.6-bin_x64.zip をダウンロードする。

[OpenGL/GLUT について]

http://ktm11.eng.shizuoka.ac.jp/lesson/modeling.html

③ glut-3.7.6-bin_x64.zip を解凍し、以下のディレクトリに配置する。

```
C:¥pbvr¥glut-3.7.6¥include¥GL¥glut.h
C:¥pbvr¥glut-3.7.6¥lib¥glut32.lib
```

④　以下のファイルを Microsoft Visual Studio 2017 で開く。

```
C:¥pbvr¥pbvr_inSitu_1.00¥pbvr.sln
```

⑤　下図 3-1 のように構成を「Release x64」にする。

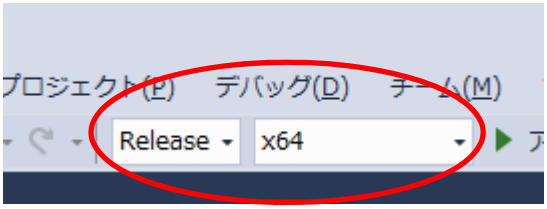


図 3-1PBVR クライアントのための VisualStudio2017 のビルド設定

⑥　メニューの[ビルド] ⇒ [ソリューションのビルド]を実行する。

⑦　上記の③で解凍された glut32.dll を以下のディレクトリに配置する。

```
C:¥pbvr¥pbvr_inSitu_1.00¥x64¥Release¥glut32.dll
```

4 In-Situ 可視化のセットアップ

シミュレーションに結合された粒子サンプラ、対話ノード上で動作するデーモン、そしてユーザ PC 上の PBVR クライアントが連携することで、バッチ処理されるシミュレーションが対話的に可視化される。これを実現するためには、幾つかの簡単な設定とポートフォワード接続が必要になる。

4.1. 環境変数の設定

デーモンと粒子サンプラは以下の表 4-1 に示す環境変数を利用しており、実行時には export コマンドで環境変数を設定する必要がある。

表 4-1 デーモンおよび粒子サンプラで参照する環境変数

環境変数名	説 明
VIS_PARAM_DIR	伝達関数ファイル（可視化パラメータ）の配置されるディレクトリ※1
PARTICLE_DIR	疑似シュミレーションコードが粒子データを出力するディレクトリ※1
TF_NAME	伝達関数のファイル名（拡張子は含まない）※2

※1 指定が無い場合、デーモンおよび粒子サンプラは各々が実行されているカレントディレクトリを検索する。

※2 指定が無い場合、デーモンおよび粒子サンプラは伝達関数名として default.tf を採用する。

4.2. 可視化パラメータの設定

粒子サンプラがシミュレーション結果のボリュームデータを可視化用粒子データに変換する際に、伝達関数や可視化する物理値のレンジ、画面解像度等の可視化パラメータが必要とされる。可視化パラメータはその各項目が伝達関数ファイルの中にタグベースで記述される。ユーザはソースパッケージの Example の中に格納してある default.tf を初起動時の伝達関数ファイルとして利用できる。

In-Situ PBVR を実行するために、ユーザは伝達関数ファイルを表 4-1 の VIS_PARAM_DIR で指定されるディレクトリに TF_NAME で指定されるファイル名で配置する必要がある。

ユーザはデーモンと PBVR クライアントを起動することで GUI により伝達関数ファイルの内容を編集できる。環境変数で指定された伝達関数ファイルはデーモンに読み取られ、PBVR クライアント上に内容表示され、編集後上書きされる。

4.3. ポートフォワード接続

デーモンと PBVR クライアントの間で粒子データや可視化パラメータはソケット通信により送受信される。遠隔地のスーパーコンピュータ上の対話ノードと手元の PC の間でソケット通信するために、ユーザは双方のポート間を ssh ポートフォワーディングで接続する必要がある。

4.3.1. 2点間リモート接続

手元の machineA と遠隔地に有る machineB の2点間でポートフォワーディングするには以下のコマンドを利用する。

```
machineA> ssh -L portnumA:hostnameB:portnumB username@machineB
```

上記のコマンドにおいて、portnumA は machineA のポート番号、hostnameB は machineB のホスト名、portnumB は machineB のポート番号である。hostnameB は machineB のターミナル上に表示されている場合が多く、また hostname コマンドでも確認できる。また、machineB のログインノードでポートフォワードが許可されている場合、hostnameB は特別なホスト名を入力する必要はなく、localhost が利用できる。

4.3.2. 多段リモート接続

遠隔地の 2 台のマシンでクライアントプログラム(machineA)とデーモンプログラム(machineB)を起動するが，セキュリティ上の理由等により machineC を経由して接続する例を示す。この場合も 2 点間の場合と同様に 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。

手順 1[ssh ポートフォワード A→C]

```
machineA> ssh -L 60000:localhost:60000 username@machineC
(machineA の 60000 番ポートを machineC の 60000 番ポートにフォワード)
```

手順 2[ssh ポートフォワード C→B]

```
machineC> ssh -L 60000:localhost:60000 username@machineB
(machineC の 60000 番ポートを machineB の 60000 番ポートにフォワード)
```

4.4. デーモンの起動とポートフォワード接続

デーモンは対話的処理が可能なノードあるいは、対話ジョブにおいて起動され、PBVR クライアントとソケット通信する。ユーザは手元の PC のターミナルから対話ノードへ ssh ポートフォワーディングし、デーモンのロードモジュールが配置されたディレクトリに移動し、以下のようにデーモンを起動する。

```
$ ./pbvr_daemon
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 PBVR クライアントを起動する。デーモン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。

```
$ ./pbvr_daemon -p 71000
```

デーモンは先述した環境変数（４.１節）を参照して粒子ファイルの集約や伝達関数ファイルの更新を行う。そして起動時に指定されたポートを通じて PBVR クライアントとデータの送受信を行う。

5 粒子サンプラ

シミュレーションコードに generate_particles 関数を挿入することで、粒子サンプラが In-Situ に可視化用粒子を生成できるようになる。この関数は kvs_wrapper.h で定義されており、粒子生成ライブラリを参照・リンクすることで使用可能となる。

シミュレーションに結合された粒子サンプラはシミュレーションのバッチ処理と共に実行され、先述した環境変数（4.1 節）と伝達関数ファイル（4.2節）を参照しながら計算結果のボリュームデータを可視化用粒子データに変換する。粒子サンプラは PARTICLE_DIR で指定されたディレクトリに粒子データファイルと、領域の最大最小値を記した t_pfi_coords_minmax.txt ファイルを出力する。粒子データファイルは、ヘッダファイル（kvsml）、座標ファイル（coord.dat）、色ファイル（color.dat）、法線ファイル（normal.dat）で構成されており、各タイムステップで 1 ノードにつき 1 組ずつ出力される。同時に、粒子サンプラは VIS_PARAM_DIR に伝達関数の変更履歴を記した$TF_NAME_タイムステップ.tf ファイルと、ヒストグラムや可視化対象の物理値のレンジを記した history_タイムステップ.txt ファイル、処理タイムステップ区間を記した state.txt を出力する。

5.1. 構造格子向けの粒子生成関数

```
#include “kvs_wrapper.h”
void generate_particles(
    int time_step, domain_patameters dom, Typs** volume_data, int num_volume_data );
```

この関数はシミュレーションのタイムステップや計算領域の情報、シミュレーション結果のボリュームデータを引数とする。

- int time_step：シミュレーションタイムステップ。
- domain_patameters dom：以下に示す計算領域を定義する構造体。

```
typedef struct
{
    float x_global_min;   //全計算領域の x 座標の最小値
    float y_global_min;   //全計算領域の y 座標の最小値
    float z_global_min;   //全計算領域の z 座標の最小値
    float x_global_max;   //全計算領域の x 座標の最大値
    float y_global_max;   //全計算領域の y 座標の最大値
    float z_global_max;   //全計算領域の z 座標の最大値
    float x_min;          //部分領域の x 座標の最小値
    float y_min;          //部分領域の y 座標の最小値
    float z_min;          //部分領域の z 座標の最小値
    int*   resolution;    //格子解像度 int resolution[3]へのポインタ
    float cell_length;    //格子の単位長さ
} domain_parameters;
```

- Typs** volume_data：シミュレーション結果のボリュームデータの配列へのポインタ。Type はユーザ指定の物理値の型であり、多変数のボリュームデータは 2 次元配列として定義される。格子解像度(X,Y,Z)のボリュームデータにおいて、n 番目の変数の位置(i,j,k)の値は volume_data[n][i+j*X+k*X*Y]で参照する。
- int num_volume_data：変数の数

5.2. 非構造格子向けの粒子生成関数

```
#include “kvs_wrapper.h”
  void generate_particles(
      int time_step, domain_parameters dom, Type** values, int nvariables, float* coordinates,
      int ncoords, unsigned int* connections, int ncells );
```

この関数はシミュレーションのタイムステップや計算領域の情報、シミュレーション結果のボリュームデータ、非構造格子の格子情報を引数とする。

- int time_step：シミュレーションタイムステップ。
- domain_patameters dom：以下に示す計算領域を定義する構造体。

```
typedef struct
{
    float x_global_min;   //全計算領域のｘ座標の最小値
    float y_global_min;   //全計算領域の y 座標の最小値
    float z_global_min;   //全計算領域の z 座標の最小値
    float x_global_max;   //全計算領域のｘ座標の最大値
    float y_global_max;   //全計算領域の y 座標の最大値
    float z_global_max;   //全計算領域の z 座標の最大値
} domain_parameters;
```

- Typs** volume_data：シミュレーション結果のボリュームデータの配列へのポインタ。Type はユーザ指定の物理値の型であり、多変数のボリュームデータは 2 次元配列として定義される。n 番目の変数の cell 番目の頂点上の値は volume_data[n][cell]で参照する。
- float* coordinates：頂点座標の配列へのポインタ。i 番目の頂点座標(x,y,z)は(coordinates[3*i], coordinates[3*i+1], coordinates[3*i+2])で参照する。
- int ncoords：頂点の数。
- unsigned int* connections：六面体要素を構成する頂点 ID の接続リストへのポインタ。六面体要素の構成を図 5-1 に示す。i 番目の六面体要素の n 番目の頂点は connections[6*i+n]で参照する。
- int ncells：要素の数。

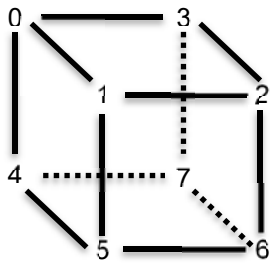


図 5-1　六面体要素の頂点接続

5.3. 階層格子向けの粒子生成関数

In-Situ PBVR ではメニーコア計算機のメモリレイアウトに最適化された階層格子構造（Block Structured AMR）をサポートしている。このタイプの格子は、N^3 の直交格子を最小の処理領域の単位（リーフ）と定義し、各階層でサイズの異なるリーフが接続されている。そのため Block Structured AMR は NxNxNxL(L はリーフ数)の四次元格子として定義される。図 5-2 に二次元の例を示す。

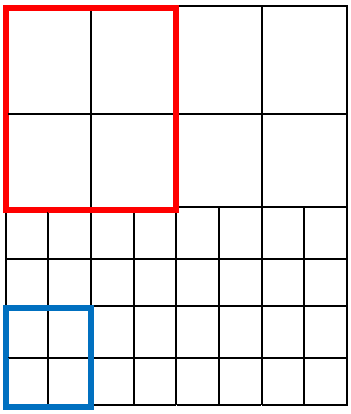


図 5-2 二次元の階層格子の例。上部が階層 Lv.1 で下部が階層 Lv.2.の格子。リーフが 2x2 の直交格子で定義され、赤が Lv.1 のリーフ、青が Lv.2 のリーフ。

```
#include “kvs_wrapper.h”
void generate_particles(
        int time_step, domain_patameters dom,
        std::vector<float>& leaf_length,
        std::vector<float>& leaf_min_coord,
        int nvariables, float** values);
```

この関数はシミュレーションのタイムステップや計算領域の情報、階層格子の構成情報、シミュレーション結果のボリュームデータを引数とする。

- int time_step：シミュレーションタイムステップ。
- domain_patameters dom：以下に示す計算領域を定義する構造体。

```
typedef struct
{
    float x_global_min;   //全計算領域のｘ座標の最小値
    float y_global_min;   //全計算領域の y 座標の最小値
    float z_global_min;   //全計算領域の z 座標の最小値
    float x_global_max;   //全計算領域のｘ座標の最大値
    float y_global_max;   //全計算領域の y 座標の最大値
    float z_global_max;   //全計算領域の z 座標の最大値
    int*   resolution;      //格子解像度 int resolution[4]へのポインタ
} domain_parameters;
```

・

・ std::vector<float>& leaf_length：リーフの長さの配列への参照。I 番目のリーフの長さは leaf_length[I]で参照する。

・ std::vector<float>& leaf_min_coord：リーフの最小位置座標の配列への参照。I 番目のリーフの座標は(leaf_min_coord[3*I], leaf_min_coord[3*I+1], leaf_min_coord[3*I+2])で参照する。

・ float** values：シミュレーション結果のボリュームデータの配列へのポインタ。多変数のボリュームデータは 2 次元配列として定義される。格子解像度(X,Y,Z,L)のボリュームデータにおいて、n 番目の変数の位置(i,j,k,l)の値は values[n][i+j*X+k*X*Y+l*X*Y*Z]で参照する。

・ int nvariables：変数の数

5.4. 粒子サンプラの組込み

ユーザのシミュレーションコードに generate_particles 関数を組み込みコンパイルすることで、In-Situ 可視化が可能になる。この章ではテストシミュレーションコードを例とした粒子サンプラの組み込み手順を示す。テストシミュレーションコードは、水素の電荷密度の式により各タイムステップで格子頂点上に電荷密度の値を計算する。そのため計算するクラス名は Hydrogen となっている。粒子サンプラの組み込みがない状態のコードは以下のようになっている。

```
#include "Hydrogen.h"
#include <iostream>
#include <mpi.h>
int main( int argc, char** argv )
{
    MPI_Init(&argc, &argv);
    Hydrogen hydro;
    int time_step = 0;
    for(;;)
    {
        hydro.values;
        time_step++;
    }
    MPI_Finalize();
    return 0 ;
}
```

上記ソースコードにおいて、Hydrogen クラスは for 文中の hydro.values で各タイムステップのボリュームデータを出力している。

5.4.1. 領域情報の定義と粒子生成関数の追加

generate_particles 関数は構造体 domain_parameters によって領域情報を取得する。ユーザはシミュレーションコードから得られる領域情報を構造体にコピーする必要がある。

```
int mpi_rank;
MPI_Comm_rank( MPI_COMM_WORLD, &(mpi_rank) );
int resol[3] = { hydro.resolution.x(), hydro.resolution.y(), hydro.resolution.z() };
domain_parameters dom = {
    hydro.global_min_coord.x(),
    hydro.global_min_coord.y(),
    hydro.global_min_coord.z(),
    hydro.global_max_coord.x(),
    hydro.global_max_coord.y(),
    hydro.global_max_coord.z(),
    hydro.global_region[mpi_rank].x(),
    hydro.global_region[mpi_rank].y(),
    0.0,
    resol,
    hydro.cell_length
};
```

上記コードでは、Hydrogen の部分領域の情報を得るために MPI のランク数を取得している。

領域情報とシミュレーション結果のボリュームデータを generate_particles 関数に入力する。generate_particles 関数はタイムステップループの内部でシミュレーション後にコールされる。Hydrogen の例の場合、以下のように挿入される。

```
int time_step = 0;
for(;;) {
    generate_particles( time_step, dom, hydro.values, hydro.nvariables )
    time_step++;
}
```

6 PBVR クライアント

PBVR クライアントはデフォルトでデーモンとから受信した最新タイムステップの粒子データを描画する。粒子データはビューワ上に OpenGL により描画される。PBVR クライアントは伝達関数を含む可視化パラメータを編集する GUI を提供し、可視化パラメータをデーモンに送信する。デーモンと PBVR クライアント間のデータの送受信はソケット通信により任意のポート番号を通して行う。

6.1. 起動方法

PBVR クライアントは以下のコマンドで実行する。

$ pbvr_client [コマンドラインオプション]

表 6-1　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
-p	ポート番号	60000	ソケット通信ポート番号
-viewer	100 ～ 9999 × 100～9999	620×620	PBVR クライアント解像度
-shading	{L/P/B},ka,kd,ks,n	-	シェーディング方法 ※1

※1. 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

6.2. 終了方法

PBVR クライアントの終了は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。Ctrl+c キーを押すと，時刻更新のタイミングで PBVR クライアントはデーモンと同期し、両方が終了する。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。また、デーモンを Ctrl+c キーで終了させてしまうと，PBVR 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ため注意すること。

6.2.1. 強制終了

何らかの理由により Ctrl+c キーの押下で PBVR クライアントとデーモンが終了しない場合は、以下に示すように，ps コマンドでクライアントとデーモン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 PBVRViewer
  PID TTY          TIME CMD
19582 pts/6    00:00:00 PBVRViewer
$ kill -9 19582
```

【デーモンプロセスの強制終了】

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

6.3. GUI の構成と操作方法

6.3.1. ビューワ

図 6-1 に示すビューワには，可視化用粒子データを用いてボリュームレンダリングしたが表示される。

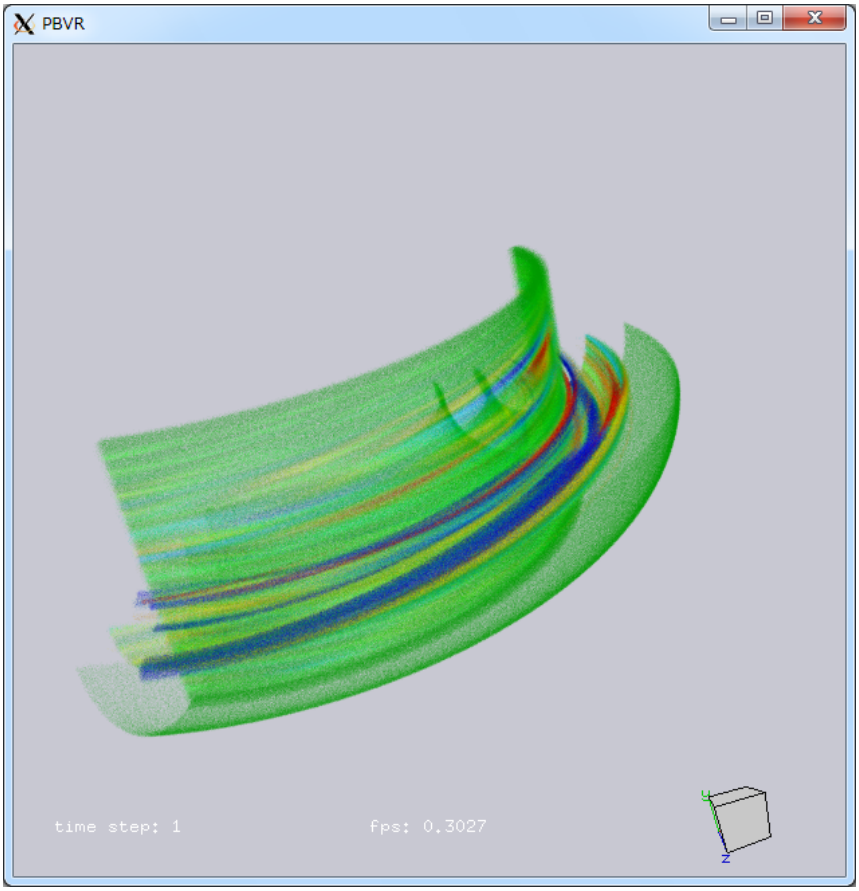


図 6-1　ビューワ

【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）

6.3.2. メインパネル

クラアイントのメインパネルを図 6-2 に示す。パネル内の各項目について以下に説明する。

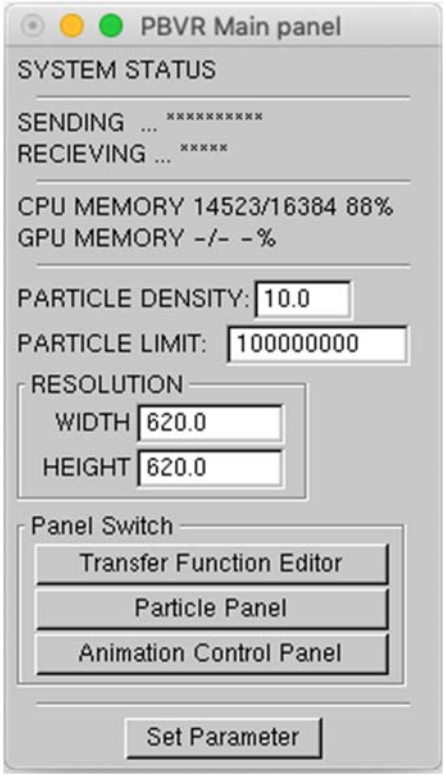


図 6-2　メインパネル

・PARTICLE DENSITY

　　粒子密度の倍率を指定する。高くすると粒子数が増加し、全体の不透明度が増す。

・PARTICLE LIMIT

　　伝達関数誤指定等による粒子数の爆発を避けるため，サーバ側で生成する粒子数の上限値を指定する。この値を高くすると画質が向上する。

・RESOLUTION

　　ビューワの解像度を指定する。

・Transfer Function Editor ボタン

　　伝達関数エディタを表示する。伝達関数エディタの詳細については後述。

・Particle Panel ボタン

　　粒子統合エディタを表示する。粒子統合エディタの詳細については後述。

・Animation Control Panel ボタン

　　動画作成用パネルを表示する。動画作成用パネルの詳細については後述。

・SENDING

　　データ送信中であることを示す。

・RECIEVING

　　データ受信中であることを示す。

• CPU MEMORY

システムメモリの使用状況（単位：MB）を表示する。

• GPU MEMORY

GPU レンダリング GPU メモリの使用状況（単位：MB）を表示する。

• Set parameter ボタン

クライアントのパネルで指定したパラメータをデーモンへ送信する。

6.3.3. 伝達関数エディタ

ユーザは伝達関数エディタを用いて，物理値にマッピングする色・不透明度の関数（伝達関数）を設計できる。伝達関数エディタはメインパネルの Transfer Function Editor ボタンを押すと現れる。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，伝達関数エディタでは

（1） 色と不透明度に対する独立な変量の指定

（2） 代数式による新たな変量の定義

座標 X,Y,Z, 変量 q1,q2,q3…を用いたユーザ指定の代数式がボリュームデータを合成する。

（3） 代数式による伝達関数の合成

ユーザ指定の代数式が 1 次元伝達関数 t1～t5 を合成し多次元伝達関数を生成する。

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 6-3 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。

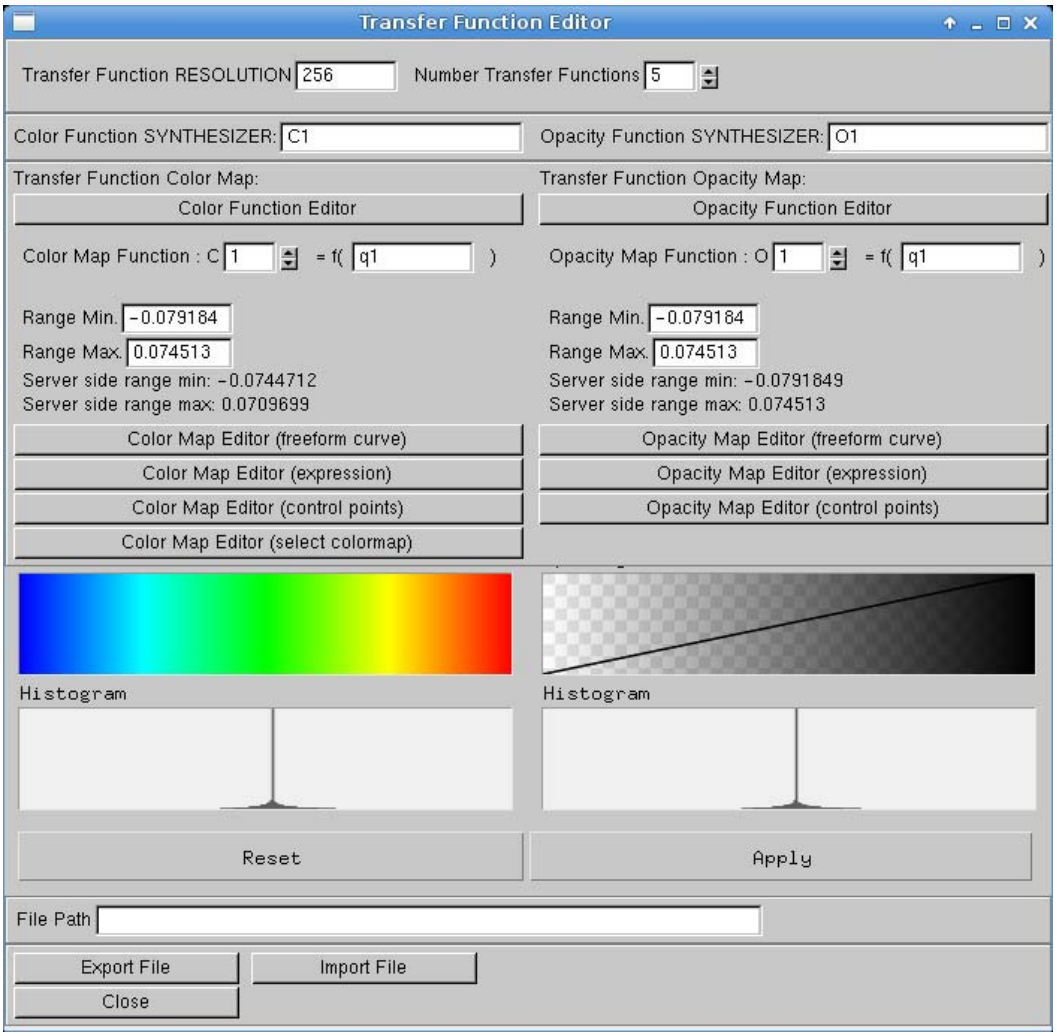


図 6-3 伝達関数エディタパネル

【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

- Transfer Fuction RESOLUTION

  伝達関数の解像度を指定する。

- Number Transfer Fuctions

  作成可能な伝達関数の制限数を指定する

- Color Fuction SYNTHESIZER

  C1～C[N]の名称で作成した伝達色関数同士の合成式を指定する。※1

- Opacity Fuction SYNTHESIZER

  O1～O[N]の名称で作成した伝達不透明度関数同士の合成式を指定する。※1

- 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。

- File Path

  伝達関数ファイルを保存，読み込みする際のファイルパスを指定する。

- Export File 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。

- Inport File 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。

- Close ボタン

  伝達関数エディタパネルをクローズする。

※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 カラーマップ指定機能

【Transfer Fuction Color Map カテゴリ】

Color Fuction SYNTHESIZER の伝達関数合成式に記述する伝達色関数名に対応する変量と色の伝達関数を作成、表示を行う。

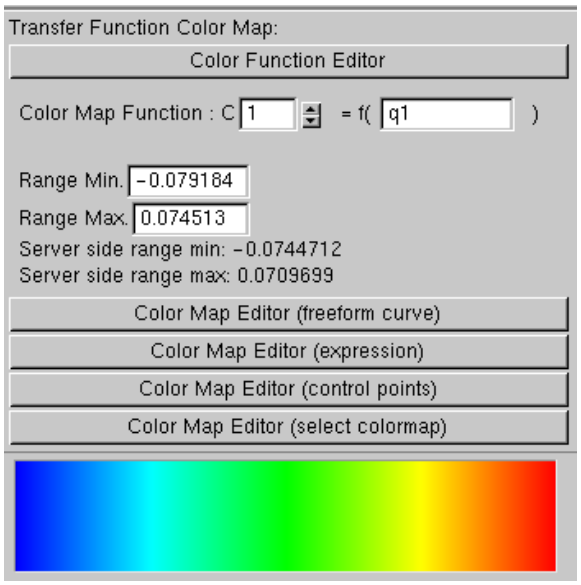


図 6-4　Transfer Fuction Color Map カテゴリ

- Color Function Editor ボタン

  Color Function Editor 画面を表示して、C1～C[N]の伝達色関数を作成、選択する。※1

- Color Map Function

  C1～C[N]の名称で伝達色関数の表示、作成する。※1

  選択した伝達関数名に対応する（合成）変量の定義式を指定する。定義式で使用できる変量の名称は以下。

  物理量：q1, q2, q3,・・・qn

  座標値：X, Y, Z

- Range Min

  選択した伝達関数名に対する変量の最小値を指定する。

- Range Max

  選択した伝達関数名に対する変量の最大値を指定する。

- Server side range min

  Variable 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を表示する。

- Server side range max

  Variable で指定した（合成）変量について，サーバ側で取得した最大値を表示する。

- Color Map Editor (freedom curve)ボタン

  選択した伝達関数名に対応する変量と色の伝達関数をマウスによる自由曲線入力で作成するパネルを出す。

- Color Map Editor (expression)ボタン

選択した伝達関数名に対応する変量と色の伝達関数を数式記述で作成するパネルを出す。

- Color Map Editor (control points)ボタン

  選択した伝達関数名に対応する変量と色の伝達関数を制御点指定で作成するパネルを出す。

- Color Map Editor (select colormap)ボタン

  選択した伝達関数名に対応する変量と色の伝達関数をあらかじめ用意されているカラーバーから選択して作成するパネルを出す。

- Color

  本エディタで作成した，変量と色の伝達関数のカラーバーが表示される。

※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 ボタンにより、Color Function Editor 画面を表示して、C1～C[N]の伝達色関数を作成、選択する。[N]は Number Transfer Fuctions にて指定した伝達関数の制限数の値である。

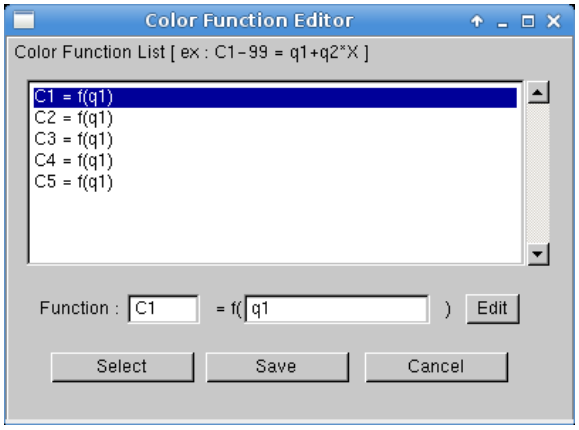


図 6-5 Color Function Editor 画面

- 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

- Edit ボタン

  Color Function List に反映する

- 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

Color Map Editor (freedom curve)ボタンにより、選択した伝達関数名に対応する変量と色の伝達関数をマウスによる自由曲線入力で作成するパネルを出す。

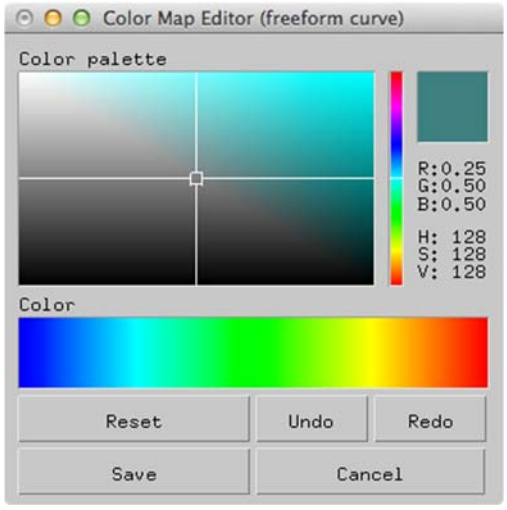


図 6-6 Color Map Editor (freedom curve)パネル

- 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 と併せて作成した色が右上に表示される。
- Color
  カラーバー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。
- 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。

6.3.3.1.3. Color Map Editor (expression)

Color Map Editor (expression)ボタンにより、選択した伝達関数名に対応する変量と色の伝達関数を数式記述で作成するパネルを出す。

図 6-7 Color Map Editor (expression)パネル

- 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。
- R
  色のR成分の伝達関数式を記述する。
- G
  色のG成分の伝達関数式を記述する。
- B
  色のB成分の伝達関数式を記述する。
- 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。

6.3.3.1.4. Color Map Editor (control points)

Color Map Editor (control points)ボタンにより、選択した伝達関数名に対応する変量と色の伝達関数を制御点指定で作成するパネルを出す。

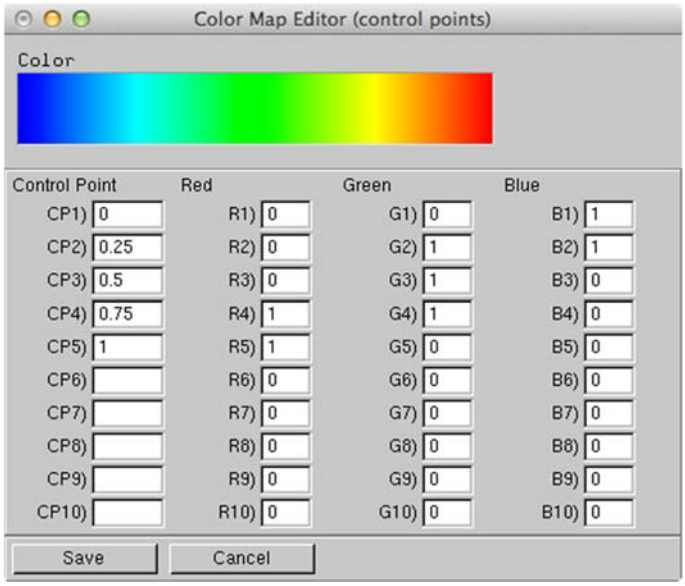


図 6-8　Color Map Editor (control points)パネル

- 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。
- Control Point
  制御点（最大 10 個）の値を CP1)～CP10)に指定する。
- Red
  制御点の値に対応する色の R 成分値を指定する。
- Green
  制御点の値に対応する色の G 成分値を指定する
- Blue
  制御点の値に対応する色の B 成分値を指定する
- 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。

6.3.3.1.5. Color Map Editor (select colormap)

Color Map Editor (select colormap)ボタンにより、選択した伝達関数名に対応する変量と色の伝達関数をあらかじめ用意されているカラーバーから選択して作成するパネルを出す。

図 6-9　Color Map Editor (select colormap)パネル

・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

・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。

6.3.3.2 不透明度指定機能

【Transfer Fuction Opacity Map カテゴリ】

Transfer Fuction name で選択した伝達関数名に対応する変量と不透明度の伝達関数を作成する。

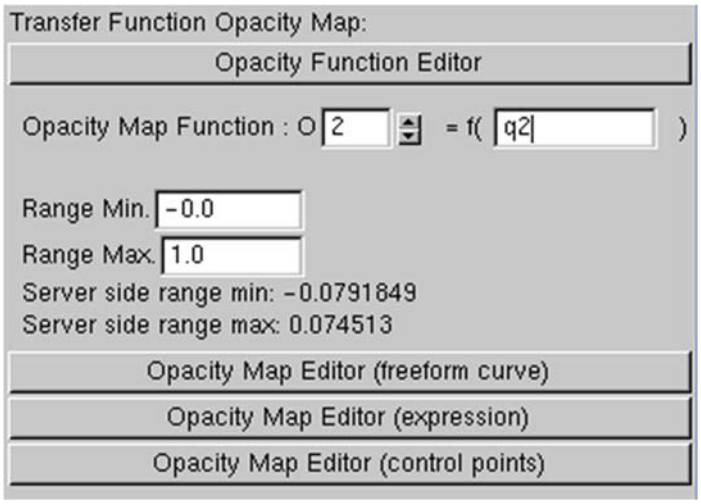


図 6-10　Transfer Fuction Opacity Map カテゴリ

- Opacity Function Editor ボタン

  Opacity Function Editor 画面を表示して、O1～O[N]の伝達不透明度関数を作成、選択する。※1

- Opacity Map Function

  O1～O[N]の名称で伝達不透明度関数の表示、作成する。※1

  選択した伝達関数名に対応する（合成）変量の定義式を指定する。定義式で使用できる変量の名称は以下。

  物理量：q1, q2, q3,・・・qn

  座標値：X, Y, Z

- Range Min

  選択した伝達関数名に対する変量の最小値を指定する。

- Range Max

  選択した伝達関数名に対する変量の最大値を指定する。

- Server side range min

  Variable 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を表示する。

- Server side range max

  Variable で指定した（合成）変量について，サーバ側で取得した最大値を表示する。

- Opacity Map Editor (freedom curve)ボタン

  選択した伝達関数名に対応する変量と不透明度の伝達関数をマウス操作による自由曲線入力で作成するパネルを出す。

- Opacity Map Editor (expression)ボタン

  選択した伝達関数名に対応する変量と不透明度の伝達関数を数式記述で作成するパネルを出す。

・Opacity Map Editor (control point)ボタン

選択した伝達関数名に対応する変量と不透明度の伝達関数を制御点指定で作成するパネルを出す。

・Opacity

本エディタで作成した，変量と不透明度の伝達関数曲線を表示する。

※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。

6.3.3.2.1. Opacity Function Editor

Opacity Function Editor ボタンにより、Opacity Function Editor 画面を表示して、O1～O[N]の伝達不透明度関数を作成、選択する。

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

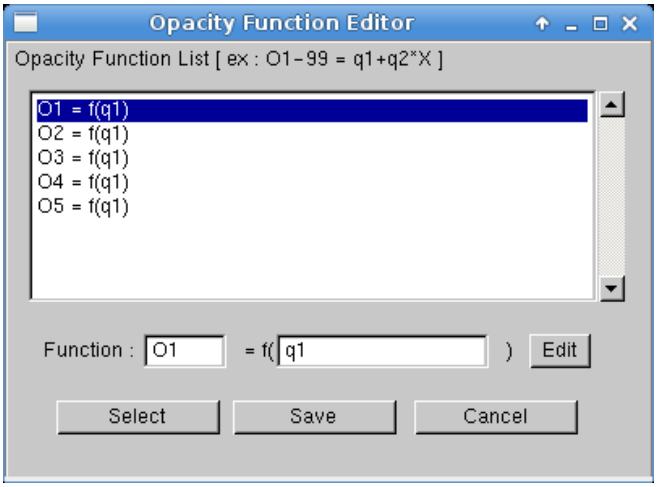


図 6-11　Opacity Function Editor 画面

・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

・Edit ボタン

Opacity Function List に反映する

・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 (freedom curve)

OpacityMap Editor (freedom curve)ボタンにより、選択した伝達関数名に対応する変量と不透明度の伝達関数をマウス操作による自由曲線入力で作成するパネルを出す。

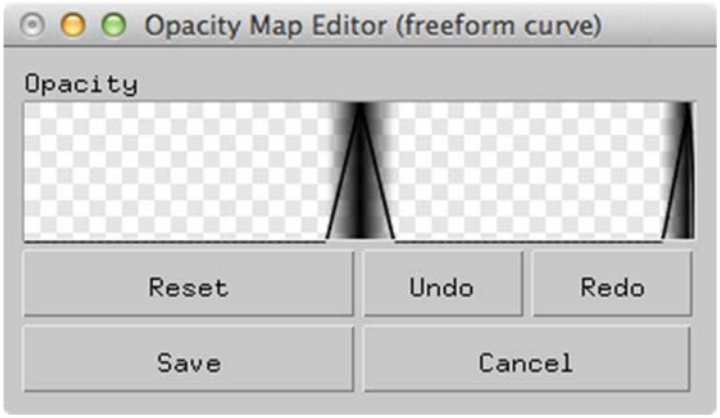


図 6-12　Opacity Map Editor (freeform curve)パネル

- Opacity
  不透明度の伝達関数をマウス操作で作成する。左ドラッグで自由曲線を描く。右クリックで２点間の線形補間直線を描く。
- 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。

6.3.3.2.3. Opacity Map Editor (expression)

Opacity Map Editor (expression)ボタンにより、選択した伝達関数名に対応する変量と不透明度の伝達関数を数式記述で作成するパネルを出す。

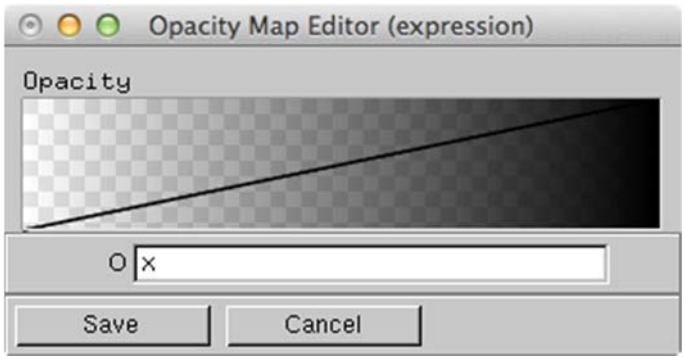


図 6-13　Opacity Map Editor (expression)パネル

・Opacity

　　O で入力した数式に基づき，不透明度の伝達関数を指定する曲線が描かれる。

・O

　　不透明度の伝達関数を指定する曲線の数式を記述する。

・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。

6.3.3.2.4. Opacity Map Editor (control point)

Opacity Map Editor (control point)ボタンにより、選択した伝達関数名に対応する変量と不透明度の伝達関数を制御点指定で作成するパネルを出す。

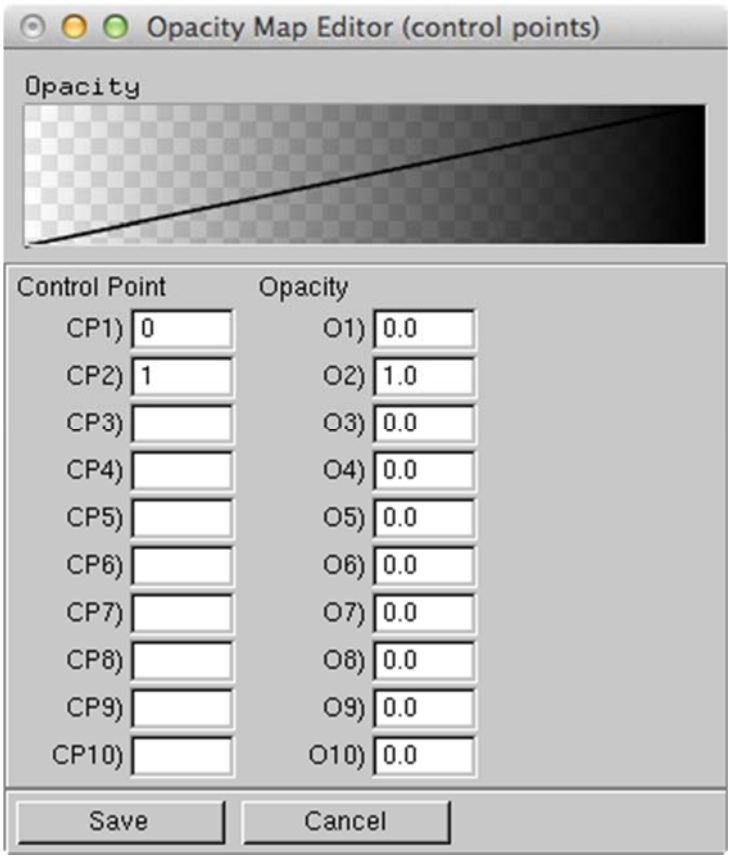


図 6-14　Opacity Map Editor (control points)パネル

- Opacity（上段）
  本パネルで作成した不透明度の伝達関数を指定する折れ線が描かれる。
- Control Point
  制御点（最大 10 個）の値を CP1)～CP10)に指定する。
- Opacity（中段右側）
  制御点の値に対応する不透明度を指定する。
- 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。

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 6-2　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 6-15 に示す。各ウィジットの動作は以下のとおり。

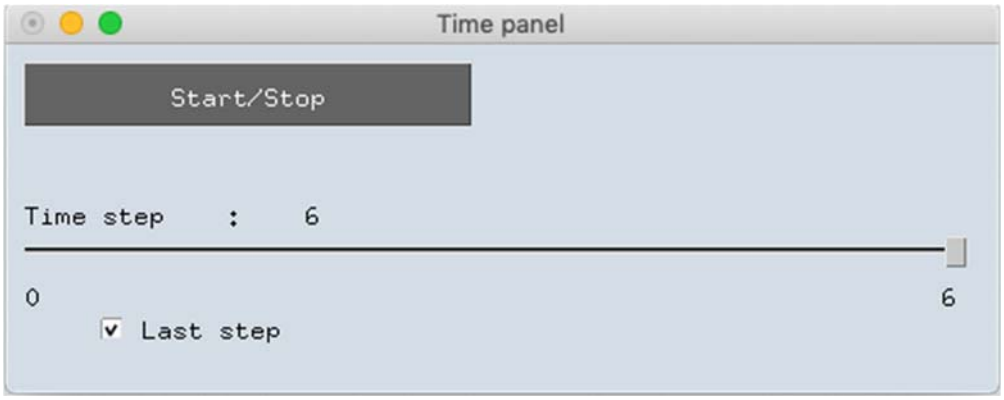


図 6-15 タイムステップ制御パネル

・Time step

レンダリングを行うタイムステップを指定する。

・Last step（チェックボックス）

常に更新済み、または更新後の最新ステップの粒子ファイルをレンダリングする。

・Start/Stop

デーモン/クライアント間の通信を開始または停止する。

ボタンが「赤」で停止状態。「濃いグレー」で通信状態。

6.3.5. 粒子統合エディタ

複数の粒子データを統合して表示する粒子統合エディタを図 6-16 に示す。粒子統合エディタはメインパネルの Particle Panel ボタンを押すと現れる。粒子統合エディタの操作方法は以下の通り。

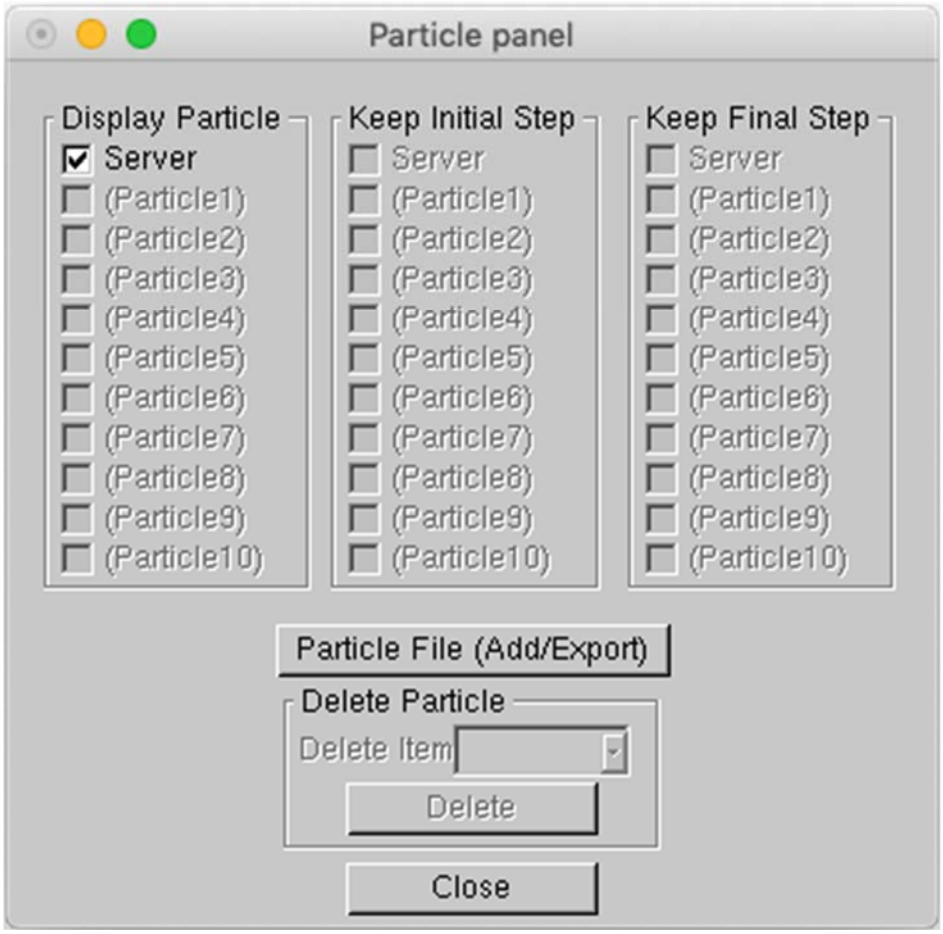


図 6-16　粒子統合エディタ

・Display Particle

統合表示の対象となる粒子データの一覧を示す。粒子データはデーモンから送られてきたデータ，もしくは，ローカルのディレクトリから読み込んだデータ（最大 10 個まで）から構成される。

①Server チェックボックス

デーモン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。

②(Particle1)～(Particle10)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と，読み込んだ粒子データファイル名が表示され，統合表示の対象としてチェックできるようになる。

• Keep Initial Step

開始タイムステップが異なる粒子データ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の一覧を示す。

• Keep Final Step

終了タイムステップが異なる粒子データ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の一覧を示す。

• Particle File（Add/Export）ボタン

Particle File パネルを表示する。

• Delete Particle プルダウンメニュー

Display Particle の一覧から削除する粒子データを選択する。

• Delete ボタン

Delete Particle プルダウンメニューで選択した粒子データを粒子統合エディタの一覧から削除する。

• Close ボタン

粒子統合エディタをクローズする。

粒子統合エディタの動作を、図 6-17（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 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 6-4 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 6-5 に表示されるタイムステップを示す。

図 6-17　タイムステップの異なる粒子統合の動作

表 6-3　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	×	１	２	３	４
クライアント	０	１	２	３	×

表 6-4　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	2	3	4
クライアント	0	0	0	0	0

表 6-5　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	×	1	2	3	4
クライアント	3	3	3	3	3

6.3.5.1 PARTICLE FILE パネル

粒子データファイルの読み込みと保存に関する操作を行うパネルで，Particle Panel の Particle File 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。

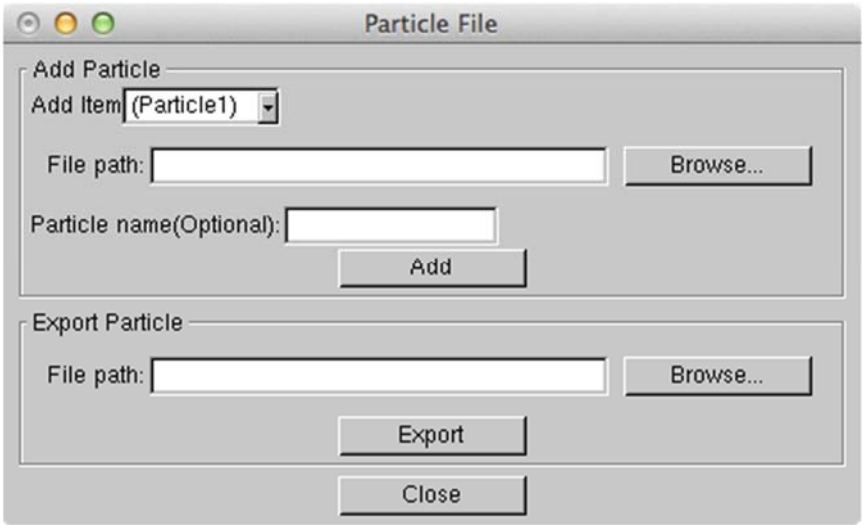


図 6-18　PARTICLE FILE パネル

・Close ボタン

Particle File パネルをクローズする。

【Add Particle カテゴリ】

・Add Item プルダウンメニュー

クライアント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を追加するときに，どの名前に対して追加するかを選択する。既に一覧へ追加済みの名前を選択した場合は，後から読み込んだ粒子データで上書きする。

・File path 欄

読み込む粒子データファイル名を指定する。

・Browse ボタン

ファイルダイアログを表示して，File path 欄へ入力する粒子データファイル名を指定する。

・Particle name (Optional)欄

粒子データを粒子統合エディタの一覧へ追加するときの名前を記述する。省略すると，File path 欄で指定したファイル名が表示される。

・Add ボタン

File path 欄で指定した粒子データを，粒子統合エディタの一覧へ追加する。

【Export Particle カテゴリ】

・File path 欄

出力する粒子データファイル名を指定する。

・Browse ボタン

ファイルダイアログを表示して，File path 欄へ入力する粒子データファイル名を指定する。

・Export ボタン

メモリ上にある粒子データを統合して File path 欄で指定したファイルへ出力する。

6.3.5.2 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。**エラー! 参照元が見つかりません。**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**エラー! 参照元が見つかりません。**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。

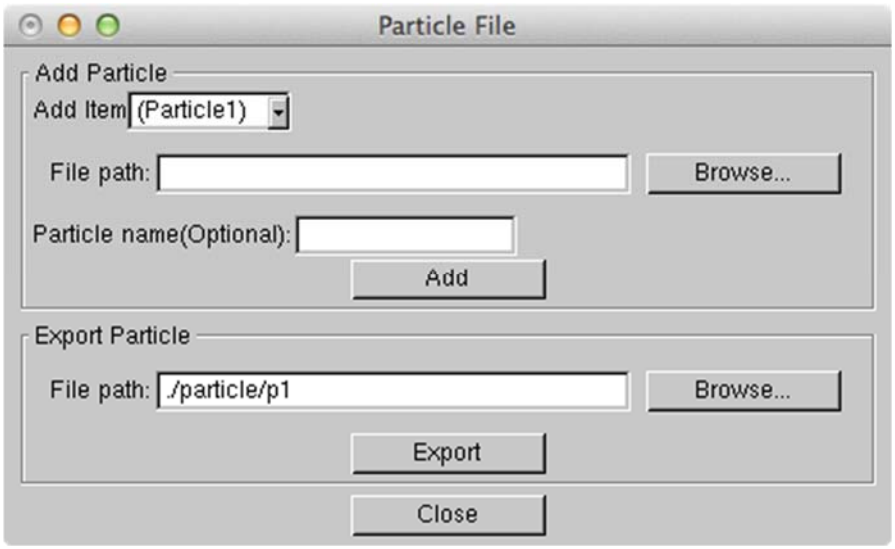


図 6-19　Particle File パネル（粒子データ保存前後）

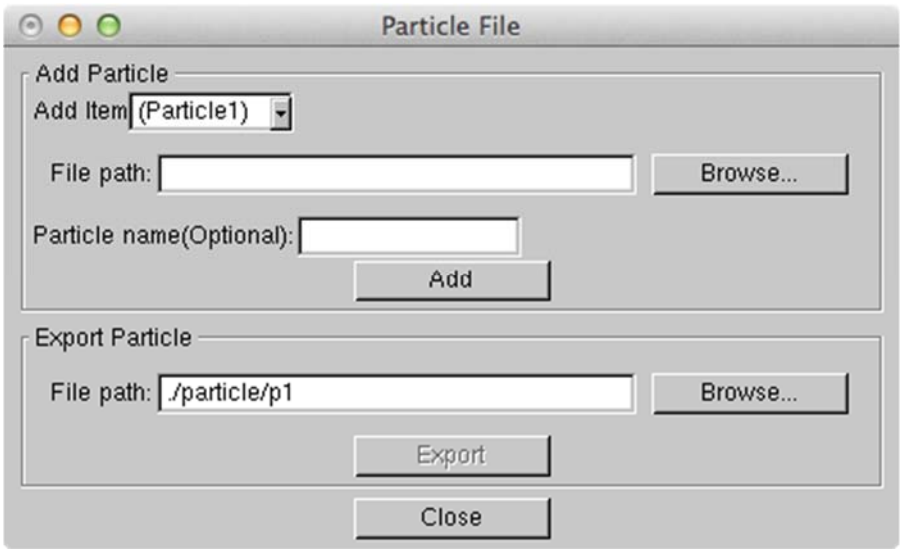


図 6-20　Particle File パネル（粒子データ保存中）

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 6-21 に示す。パネル内の各項目について以下に説明する。

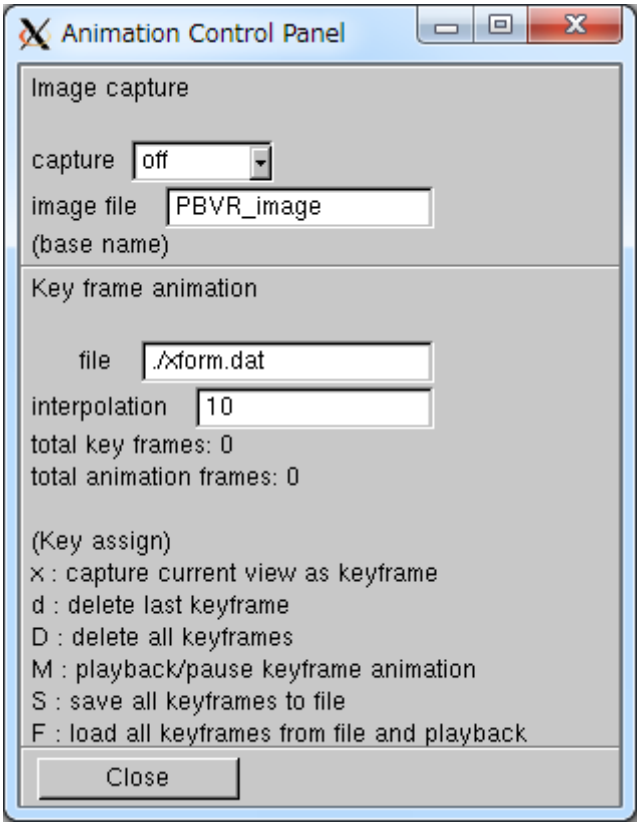


図 6-21　動画作成用パネル

・capture プルダウンメニュー

イメージキャプチャの開始／終了を措定する。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。

- Close ボタン

  動画作成用パネルをクローズする。

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 F（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 6-6 に示す。

表 6-6　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
F	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。

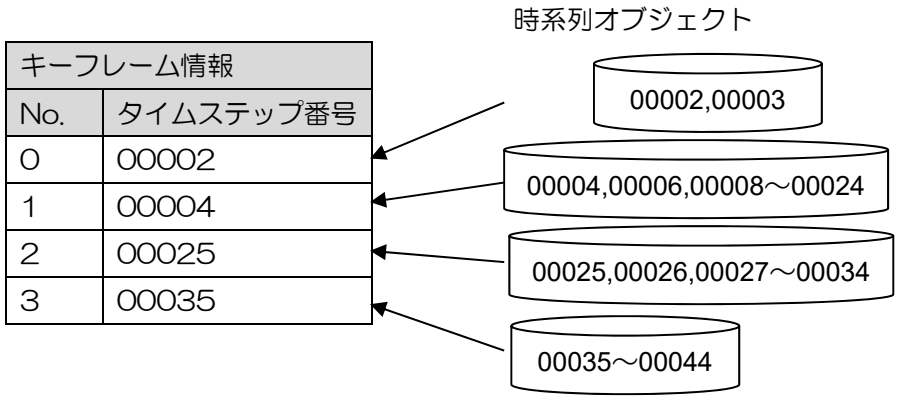
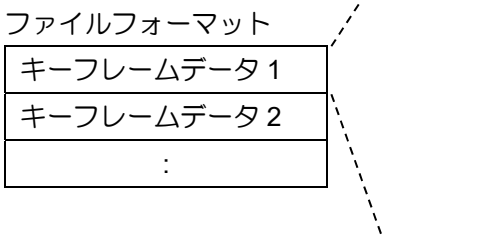


図 6-22　時系列データのキーフレームアニメーション構成

図 6-22 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 6-23 に示す。



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 6-23　キーフレーム情報ファイルのフォーマット

7 サンプルの実行

ソースコードパッケージに含まれる Example を用いて、Linux/Mac から機構所有のメニーコアクラスタ JKNL 及びスーパーコンピュータ ICEX にポートフォワード接続し、In-Situ PBVR を実行する例を示す。また Windows から遠隔のサーバにアクセスし In-Situ PBVR を実行する例を示す。

7.1. ICEX

ICEX 上で非構造格子用の In-Situ PBVR を対話型ジョブにて実行する手順を示す。この例では、RSA 公開鍵及び ssh のホスト名・ユーザ名の設定が済んでいるものとする。またソースパッケージが /home/g1/ユーザ ID/ に配置されているものとする。はじめに、対話型ジョブを実行するシェルプログラムを作成し、以下に配置する。

ICEX:/home/g1/ユーザ ID/pbvr_inSitu_1.00/Example/Hydrogen_unstruct/run.sh

```
#!/bin/sh
#PBS -q t432
#PBS -l select=1:ncpus=24:mpiprocs=4:ompthreads=6
#PBS -P pbvr_daemon@PG18019
#PBS -l walltime=1:00:00
#PBS -I
```

１つ目のターミナルを立ち上げ、以下のコマンドを実行する。
[ターミナル１]

```
$ ssh ICEX
% cd /home/g1/ユーザ ID/pbvr_inSitu_1.00/Example/Hydrogen_unstruct
% qsub run.sh
> hostname
r16i0n0
> export VIS_PARAM_DIR=/home/g1/ユーザ
ID/pbvr_inSitu_1.00/Example/Hydrogen_unstruct
> export PARTICLE_DIR=/home/g1/ユーザ
ID/pbvr_inSitu_1.00/Example/Hydrogen_unstruct/jupiter_particle_out
> export TF_NAME=jupiter
> cd $PBS_O_WORKDIR
> ./pbvr_daemon -p 61000
```

２つ目のターミナルで、以下のコマンドを実行する。

［ターミナル２］

```
$ ssh -L 61000:r16i0n0:61000 ユーザ ID@ICEX
  ⇒ ホスト名は、pbvr_daemon を起動しているターミナル１の hostname を指定する。
```

３つ目のターミナルで、以下のコマンドを実行する。

［ターミナル３］

```
$ cd /Users/admin/pbvr_inSitu_1.00/Client
$ ./pbvr_client -p 61000
```

４つ目のターミナルで、以下のコマンドを実行する。

［ターミナル４］

```
$ ssh ICEX
% cd /home/g1/ユーザ ID/pbvr_inSitu_1.00/Example/Hydrogen_unstruct
% qsub run.sh
> export VIS_PARAM_DIR=/home/g1/ユーザ
ID/pbvr_inSitu_1.00/Example/Hydrogen_unstruct
> export PARTICLE_DIR=/home/g1/ユーザ
ID/pbvr_inSitu_1.00/Example/Hydrogen_unstruct/jupiter_particle_out
> export TF_NAME=jupiter
> cd $PBS_O_WORKDIR
> cp jupiter_old.tf jupiter.tf
> mpijob ./run
```

7.2. JKNL

JKNL 上で階層構造格子用の In-Situ PBVR を対話型ジョブにて実行した際のコマンドを示す。以下の例では、RSA 公開鍵及び ssh のホスト名・ユーザ名の設定が済んでいるものとする。またソースパッケージが /home/g1/ユーザ ID/ に配置されているものとする。はじめに、対話型ジョブを実行するシェルプログラムを作成し、以下に配置する。

JKNL:/home/g1/ユーザ ID/pbvr_inSitu_1.00/Example/Hydrogen_AMR/run.sh

```
#!/bin/sh
#PBS -q mc64
#PBS -l select=1:ncpus=24:mpiprocs=4:ompthreads=6
#PBS -P pbvr_run@PG18019
#PBS -l walltime=01:00:00
#PBS -I
```

１つ目のターミナルを立ち上げ、以下のコマンドを実行する。

[ターミナル１]

```
$ ssh ユーザ ID@jknlfs.tokai-sc.jaea.go.jp
$ cd /home/g1/ユーザ ID/pbvr_inSitu_1.00/Example/Hydrogen_AMR
$ qsub run.sh
$ hostname
jknl3
$ export VIS_PARAM_DIR=/home/g1/ユーザ ID/pbvr_inSitu_1.00/Example/Hydrogen_AMR
$ export PARTICLE_DIR=/home/g1/ユーザ
ID/pbvr_inSitu_1.00/Example/Hydrogen_AMR/jupiter_particle_out
$ export TF_NAME=jupiter
$ cd $PBS_O_WORKDIR
$ ./pbvr_daemon -p 61000
```

２つ目のターミナルを立ち上げ、以下のコマンドを実行する。

[ターミナル２]

```
$ ssh -L 61000: jknl3:61000 ユーザ ID@jknlfs.tokai-sc.jaea.go.jp
  ⇒ ホスト名は、pbvr_daemon を起動しているターミナル１の hostname を指定する。
```

３つ目のターミナルを立ち上げ、以下のコマンドを実行する。

[ターミナル３]

```
$ cd /Users/admin/pbvr_inSitu_1.00/Client
$ ./pbvr_client -p 61000
```

４つ目のターミナルを立ち上げ、以下のコマンドを実行する。

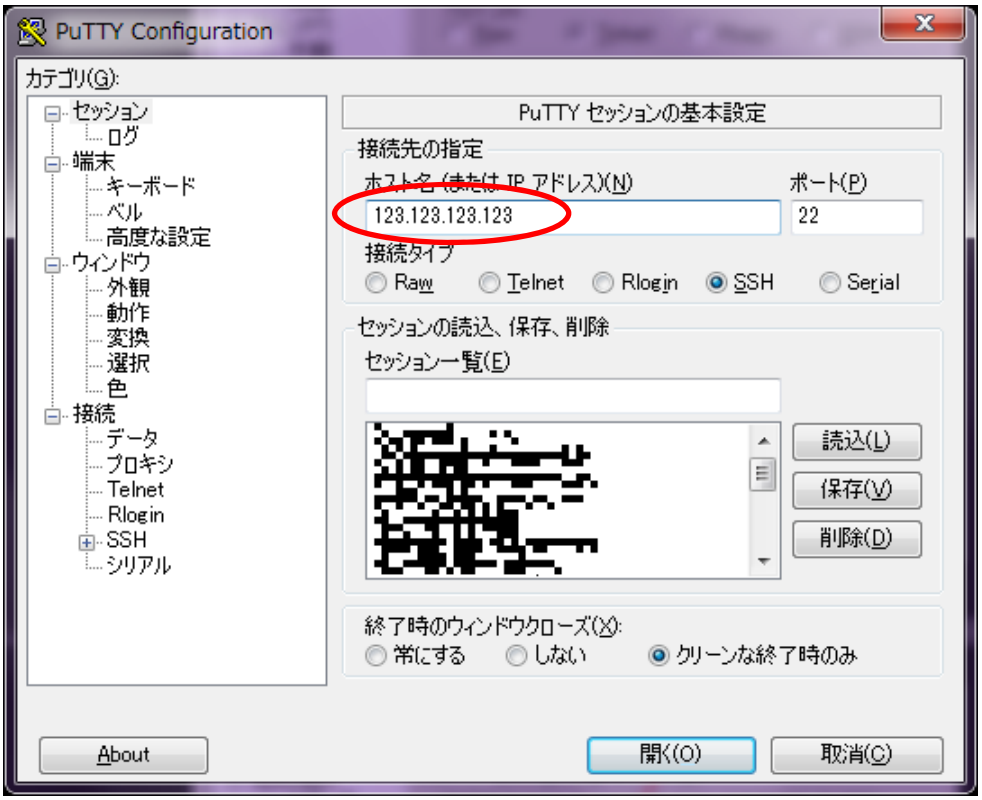
[ターミナル４]

```
$ ssh ユーザ ID@jknlfs.tokai-sc.jaea.go.jp
$ cd /home/g1/ユーザ ID/pbvr_inSitu_1.00/Example/Hydrogen_AMR
$ qsub run.sh
$ export VIS_PARAM_DIR=/home/g1/ユーザ ID/pbvr_inSitu_1.00/Example/Hydrogen_AMR
$ export PARTICLE_DIR=/home/g1/ユーザ
ID/pbvr_inSitu_1.00/Example/Hydrogen_AMR/jupiter_particle_out
$ export TF_NAME=jupiter
$ cd $PBS_O_WORKDIR
$ cp jupiter_old.tf jupiter.tf
$ mpiexec ./run
```

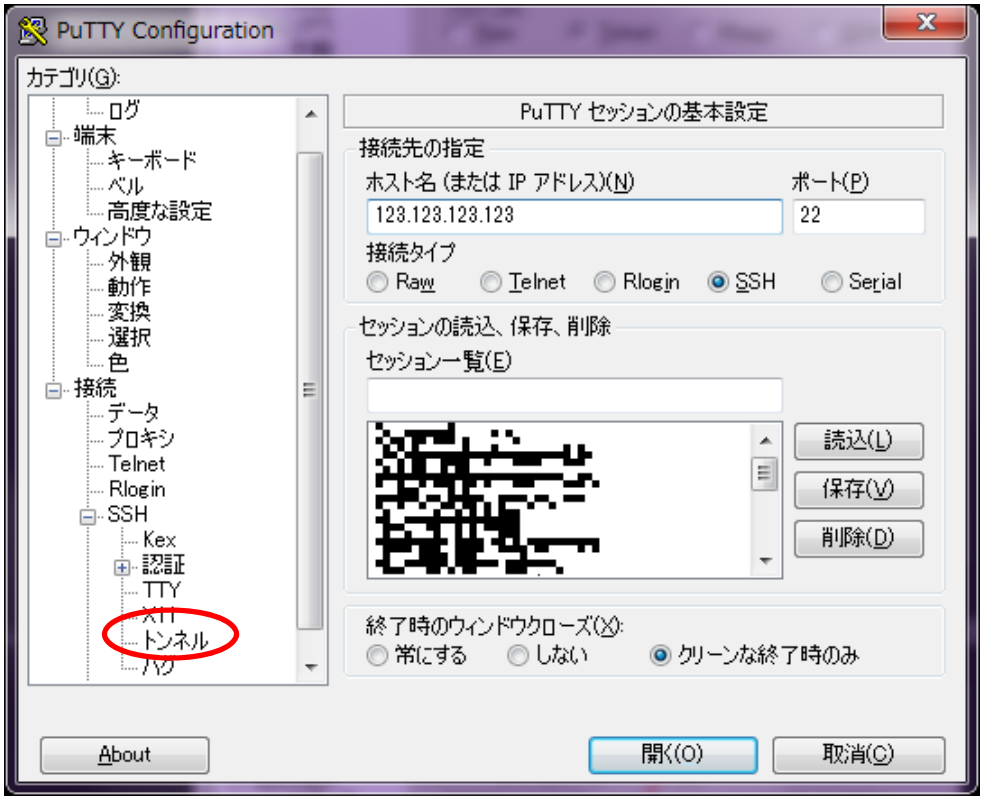
7.3. **Windows**

Windows クライアントから遠隔サーバにリモート接続する際の手順を以下に示す。尚、以下の手順では、クライアントのポート 60000 からサーバのポート 60000 に転送するものとしている。

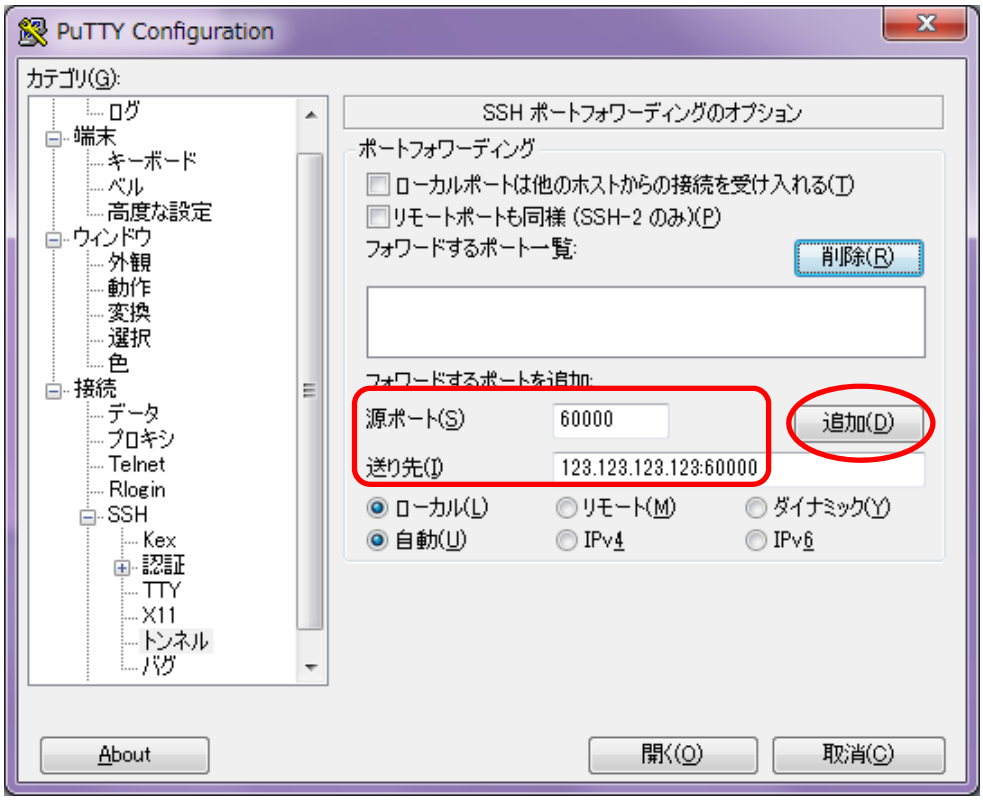
① puttyjp を起動し、接続するサーバのアドレスを入力する。



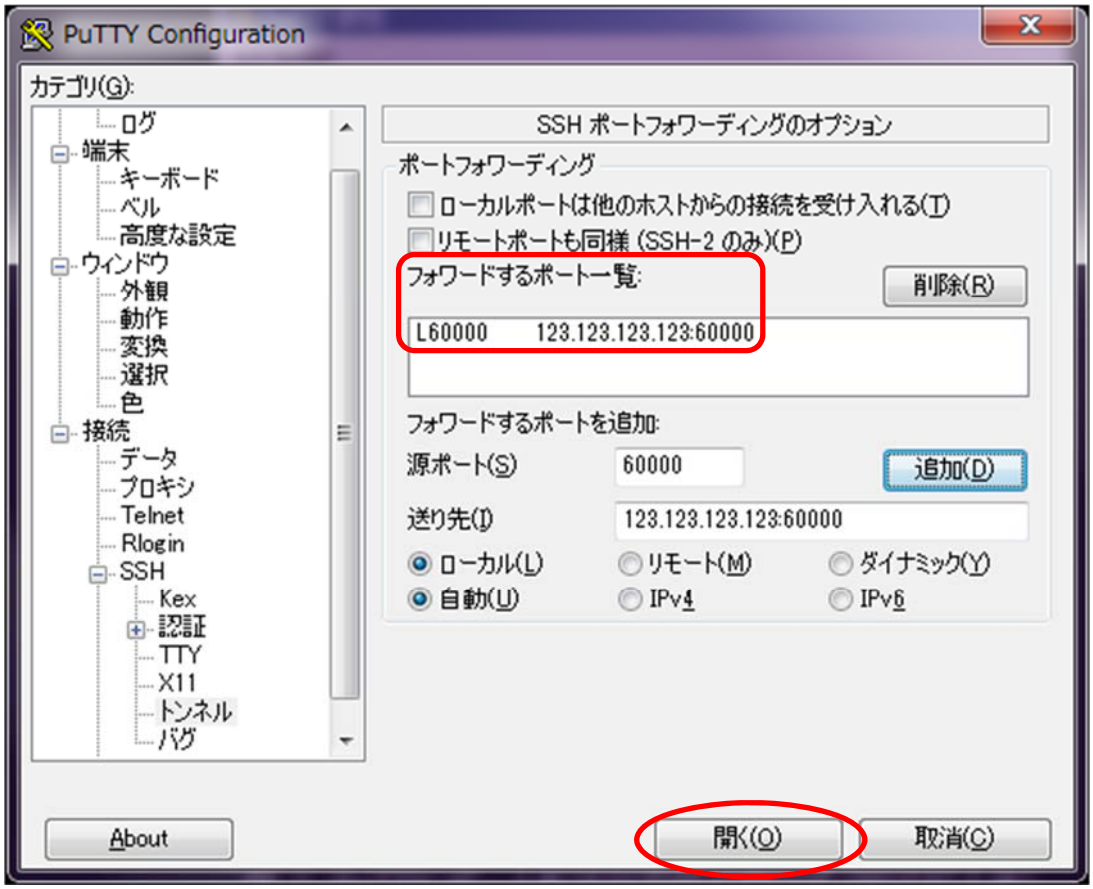
② 左側のカテゴリから「接続 ⇒ SSH ⇒ トンネル」を選択する。



③ 源ポートに転送元のポート番号、送り先にサーバのポートを設定し、「追加」ボタンを押下する。



④ フォワードするポート一覧にポート転送の設定が追加されたら、「開く」ボタンを押下してサーバと接続する。



⑤ サーバに接続しているターミナルからデーモンを起動する。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00/Example_C/Hydrogen
$ export VIS_PARAM_DIR=$HOME/JAEA/pbvr_inSitu_1.00/Example_C/Hydrogen
$ export PARTICLE_DIR=$HOME/JAEA/pbvr_inSitu_1.00/Example_C/Hydrogen
/jupiter_particle_out
$ export TF_NAME=jupiter
$ ./pbvr_daemon -p 60000
```

⑥ Visual Studio 2017 付属の「VS 2017 用 x64_x86 Cross Tools コマンド プロンプト」を起動し、以下のコマンドを実行する。

```
>set TIMER_EVENT_INTERVAL=1000
>cd C:¥pbvr¥pbvr_inSitu_work¥x64¥Release
>pbvr_client.exe -p 60000
```

⑦　サーバに接続しているターミナルからテストコードを起動する。

```
$ cd $HOME/JAEA/pbvr_inSitu_1.00/Example_C/Hydrogen
$ export VIS_PARAM_DIR=$HOME/JAEA/pbvr_inSitu_1.00/Example_C/Hydrogen
$ export PARTICLE_DIR=$HOME/JAEA/pbvr_inSitu_1.00/Example_C/Hydrogen
/jupiter_particle_out
$ export TF_NAME=jupiter
$ mpiexec -n 4 ./run
```