Remote Visualization Tool PBVR (v.2.0)

User Guide

Jan. 2023

Japan Atomic Energy Agency

Center for Computational Science & e-Systems

Version number	Date revised	Revised chapter	Revised content
1.04	2015.3.31	-	Release
1.05	2015.5.18	5.2	Parameter file function was added on Server
		5.4.2	File button was added on MainPanel
		5.4.6	Particle data output function was added
1.06a	2015.10.30	5.4.2	CROP panel was enabled for Mac
		5.4.3	Histogram function was added on Transfer Function Editor
		5.4.5	Image file production function was updated (file output directory, key frame animation)
1.07	2016.2.1	1.2	ICEX and VTK library were added to the platform list
		2	The package was updated including serial versions and PBVR Filter for VTK was added
		3.4	Data formats and parameter files were extended including STL, PLOT3D, and VTK
		4.2	A new command line option “-pd” was added, “-plimit” was modified, and “-sl” was removed
		5.2	A new command line option “-pd” was added, “-plimit” was modified, “-sl” was removed, and “-pin” was extended up to “-pin10”
		5.4.5	Particle panel was added
		6.4	An example of particle integration was added
1.071	2016.2.24	2	The package was updated. Compilation and installation were changed.
		5.4.2	Transfer Function Editor, Particle Panel and Animation Control Panel buttons were added to Main Panel.
1.09	2017.3.2	4.2	New command line options “-Bs”, “-Be” and “-Bd” were added.
		4.2.1	A usage of command line option “-vin” was extended for multiple pfi files.
		4.2.2	Added about processing of distributed files.
		5.2	- Number of panels that opened when client started was changed. - Command line option “-vin” was extended for multiple pfi files.

		5.4.2	- **The following three buttons were added on MainPanel. "Transfer Function Editor", "Particle Panel", and "Animation Control Panel".** - **A checkbox "no-repeat sampling until Transfer Functions be edited" was added on MainPanel, and following three buttons were added on MainPanel. "Legend Panel", "Coordinate Panel", and "Viewer Control Panel".** - **Display Particle Number was added.**
		5.4.3	**"Transfer Function Editor" was activated from MainPanel, and close button was added.**
		5.4.3.3	**An action when NaN appears by the arithmetic processing of function editor was added.**
		5.4.5	**"Particle panel" was activated from MainPanel, and close button was added.**
		5.4.6	**"Animation Control Panel" was activated from MainPanel, and close button was added.**
		5.4.7	**"Legend panel" was added.**
		5.4.8	**"Coordinate panel" was added.**
		5.4.9	**"Viewer Control panel" was added.**
1.10	**2017.3.15**	**3**	**"filter" was deleted. By Integration of "Filter program" and "Server program"**
		3	**"IN/OUT files" were added.**
		5.4.2	**"Main Panel" was changed.**
		5.4.3	**Editor of "Transfer function" was changed.** **"Function changes" and "GUI changes" by TFS function expansion works**
		4.2	**Launching Method: Program Arguments:** **"pin deletion", "fin addition"** **Changes by Integration of Filter program and server program**
		5.2	**Launching Method: Program Arguments:** **"pin deletion", "fin addition"** **Changes by Integration of filter program and server program**

		6.1	Launching Program: Program arguments: “pin deletion”, “fin addition” Changes by Integration of filter program and server program
		6.5	Batch mode processing: Program arguments: “pin deletion”, “fin addition” Changes by Integration of filter program and server program
1.11		3	Added description about Filtered version program.
1.12	2020.6.01	-	GUI of Client program was changed. Related images and descriptions are updated.
1.13	2020.9.17	2.3	Instead of built-in KVS, new Client program uses user installed KVS.
1.13.1	2020.10.16	5.3.1.1	Min/Max range in transfer function editor is modified.
1.14	2020.11.05	1.3 2.1.3 2.3	Windows build procedure is updated using 2017 auxiliary environment on MSVC2019.
1.15	2021.5.1	-	Added tool bar on rendering screen
1.17	2022.1.30		Added polygon composition function
1.171	2022.2.1		Added time series polygon composition function
1.172	2022.5.23		Modified bug on rendering. Added CPU rendering mode.
2.0	2021.01.01		Source codes of CS/IS-PBVR are integrated. Pre-defined colormap updated

Table of Contents

1 Introduction

1.1 Overview

This document is a user guide for Particle Based Volume Rendering (PBVR), a remote visualization system developed at the Center for Computational Science & e-Systems in Japan Atomic Energy Agency. PBVR provides high-speed remote visualization of large-scale volume data by making use of a visualization library KVS (https://github.com/CCSEPBVR/KVS), and by employing the particle-based rendering algorithm from the Koyamada Visualization Laboratory in Kyoto University. PBVR consists of the following three components.

1) PBVR Filter
   PBVR Filter reads volume data and divides it into sub-volumes, each of which becomes the unit to be processed in parallel visualization.
2) PBVR Server
   PBVR Server receives the sub-volumes and applies parallel visualization with PBVR's particle generation method.
3) PBVR Client
   PBVR Client renders the particle data with Open GL using CPU/GPU.

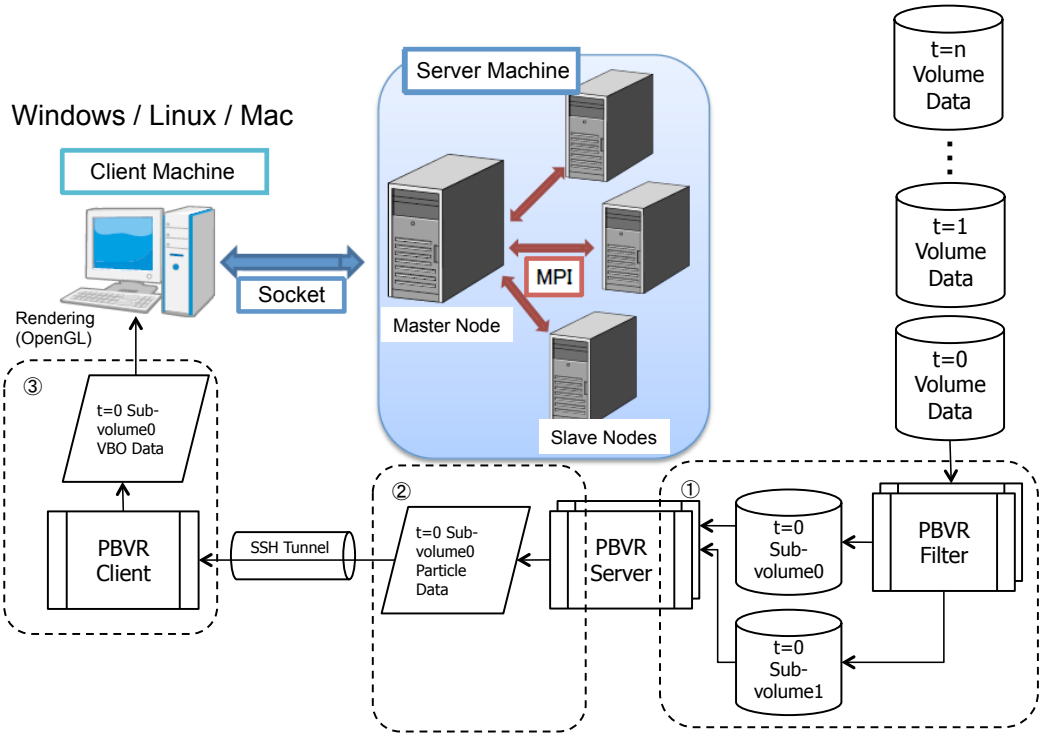


Figure 1.1-1 The system configuration of PBVR

PBVR Viewer implements a multivariate visualization function that can be used for various simulations. In 3D data visualization, color and opacity are assigned to volume data, and these are called the transfer function. In conventional visualization, a one-dimensional transfer

function that assigns color and opacity to a single physical value is used. PBVR Viewer provides a transfer function editor that allows users to design multivariate transfer functions using algebraic expressions. The transfer function editor allows users to edit 1D transfer functions for each variable and to write multidimensional transfer functions using arbitrary algebraic expressions with these variables. In this algebraic expression, basic operators, elementary functions, and differential operators can be used.

In the visualization of 3D simulations, displaying polygon data of boundary conditions and compositing it with calculation results is useful for understanding complex shapes. PBVR Viewer can composit the particle-based volume rendering and the polygon data. In conventional rendering methods, the visibility ordering based on alpha blending is the bottleneck, making composite polygons and volumes difficult. In this application, however, the Stochastic Rendering Compositor, which processes polygon rendering in the same framework as particle-based rendering, enables efficient composite rendering of polygons and volumes.

1.2 System Requirements

The remote visualization system PBVR is cross-platform program, and it works on Linux, Mac, and Windows. And the filter and server programs are works on various supercomputers such as FUGAKU and SGI8600. The system is verified for the following platforms and compilers.

●Supercomputer

Platform	CPU(architecture)	Compiler
OakforestPACS	Inetel Xeon Phi（KNL）	Intel
SGI8600	Intel Xeon Gold	Intel
FUGAKU	A64FX（ARM）	Fujitsu

●PBVR Filter/Server/Client

Platform	OS	Compiler	Library
Linux 64bit	Kubuntu18.04	g++ 5.3.1	OpenGL, Qt5.12
Mac 64bit *2	OSX12	clang 10.0.0	OpenGL, Qt5.12
Windows 64bit*3	Windows10	MSVC2017 *2	OpenGL, Qt5.12 Visual C++ runtime component

*1. KMATH is a high-performance pseudorandom number generator library, which was developed at RIKEN Advanced Institute for Computational Science. Installing KMATH_RANDOM further requires the NTL library.

*2. It means Microsoft Visual Studio C++ included in Microsoft Visual Studio 2017.

1.3 Environment Setting

Under the Windows environment, Qt and Microsoft Visual Studio (MSVC), which are integrated development environments (IDEs), are used to build this program. The version information required for that purpose is described below.

1.3.1 Usage for Qt

The Qt Creator IDE is provided by the installer which can be downloaded from the official Qt page (https://www.qt.io/download-qt-installer). It automatically installs the Qt package for each platform. There are online installer and offline installer in Qt installer, and Qt5.12 provided by offline installer to build this program.

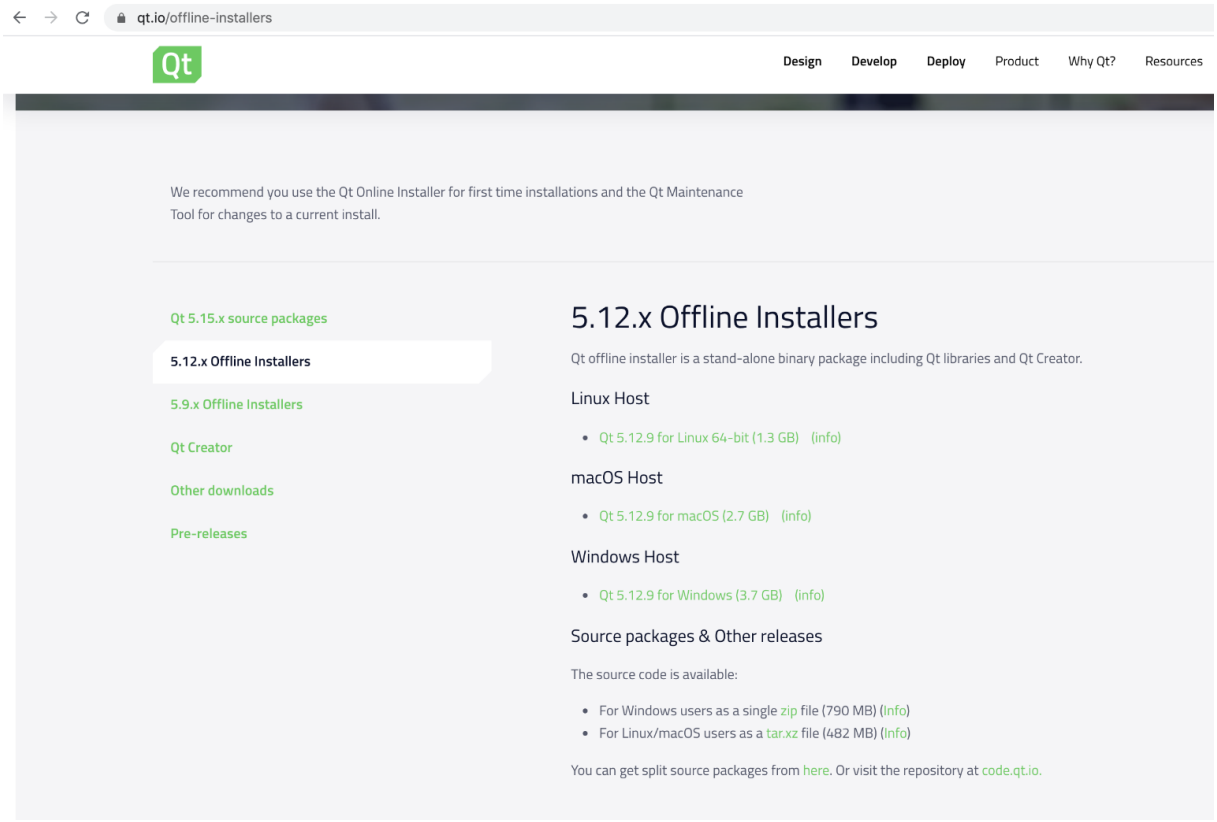


Figure 1.3-1Offline installer for Qt

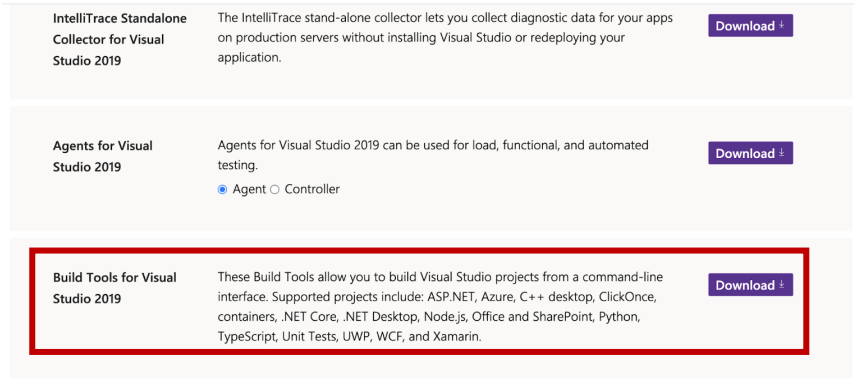
It is necessary to select the following items used to build the client program from the installer setting dialog.

- Tools/QtCreator
- Qt *.*.*/Sources
- Qt *.*.*/msvc*** 64bit（Required only Windows）

..* means version in the above list.

1.3.2 Usage of MicrosoftVisualStudio

Qt uses MSVC 2017 to build the program, but this version is not available from the official Microsoft web page. However, the MSVC installer includes an option to install auxiliary tools for building with MSVC 2017.



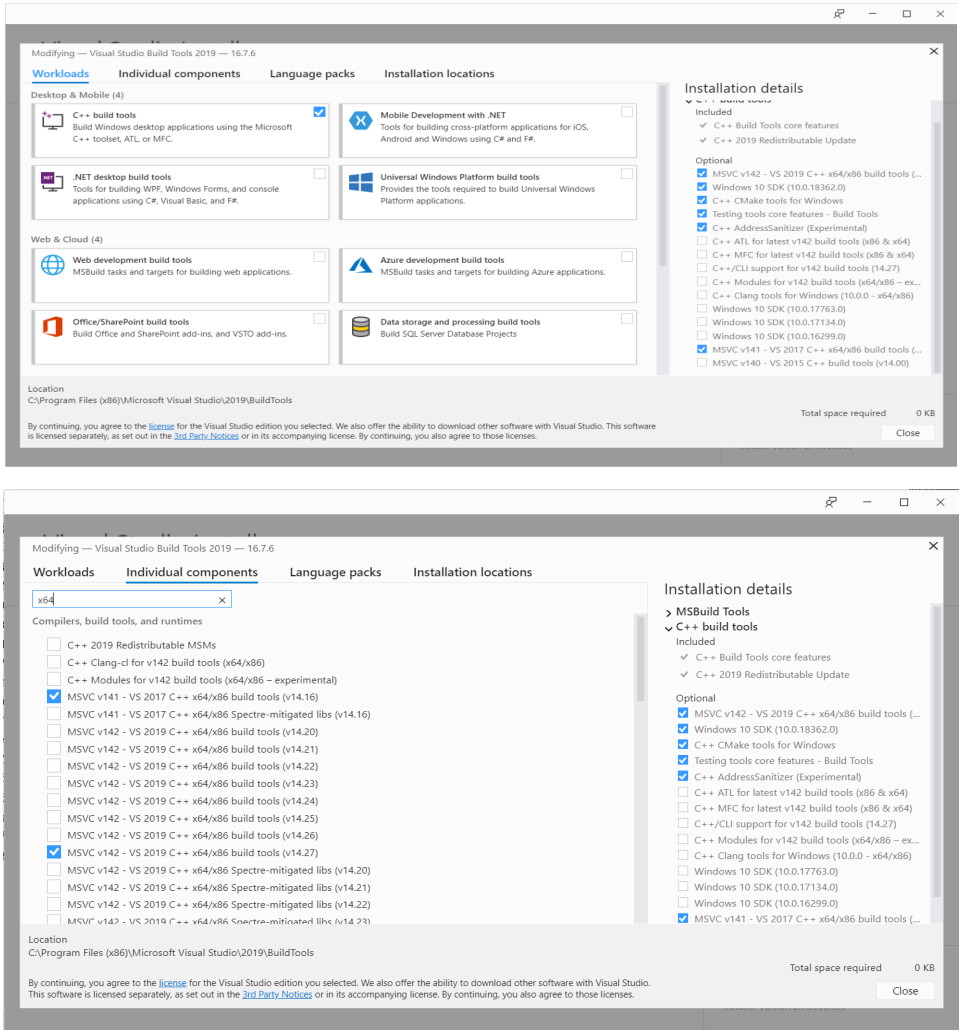


Figure 1.3-2 Tools for MSVC2017

1.3.3 Qt Setting

In order to use MSVC2017 with Qt Creator, copy (clone) the compiler settings of MSVC2019 to MSVC2017 of ABI parameters.

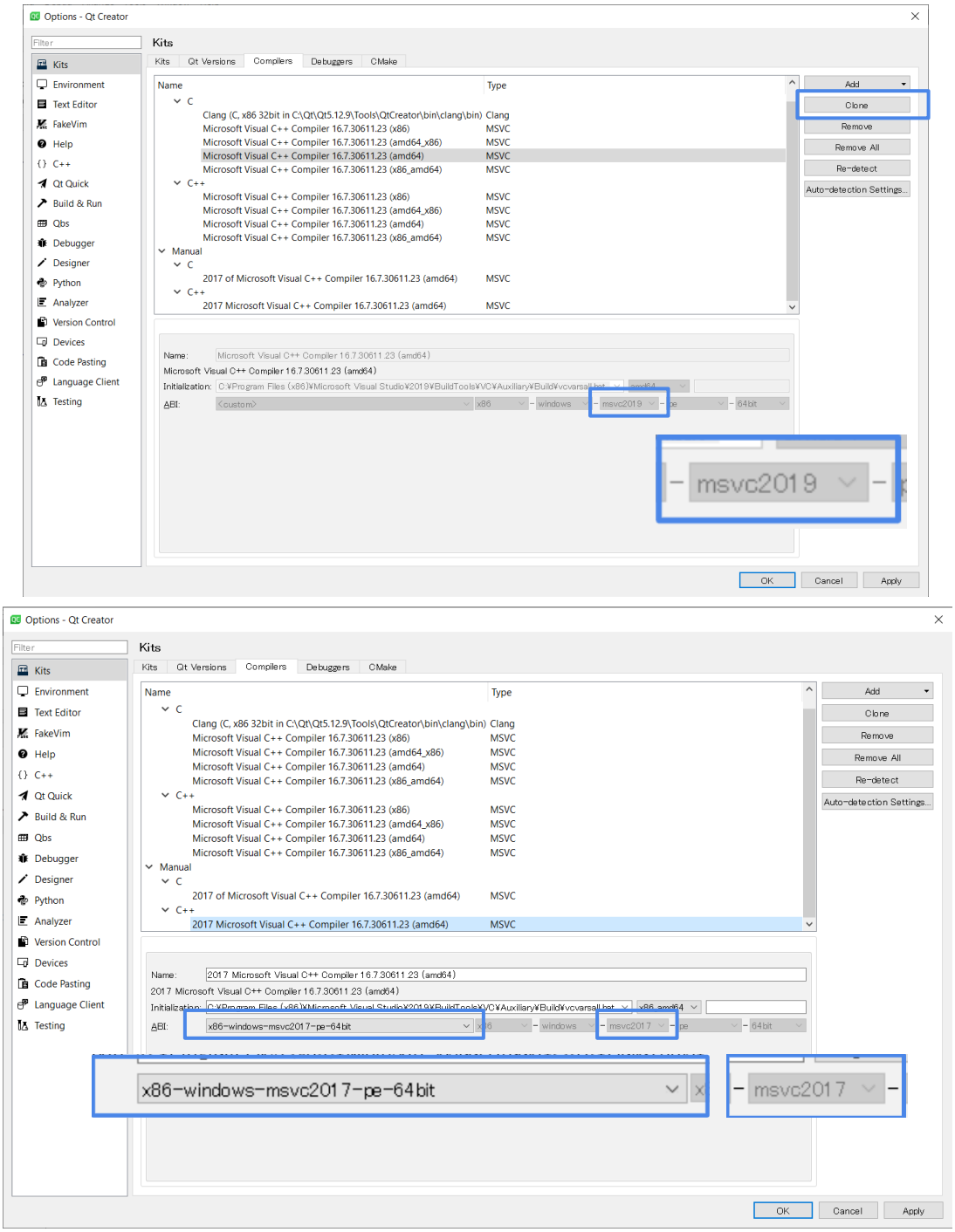
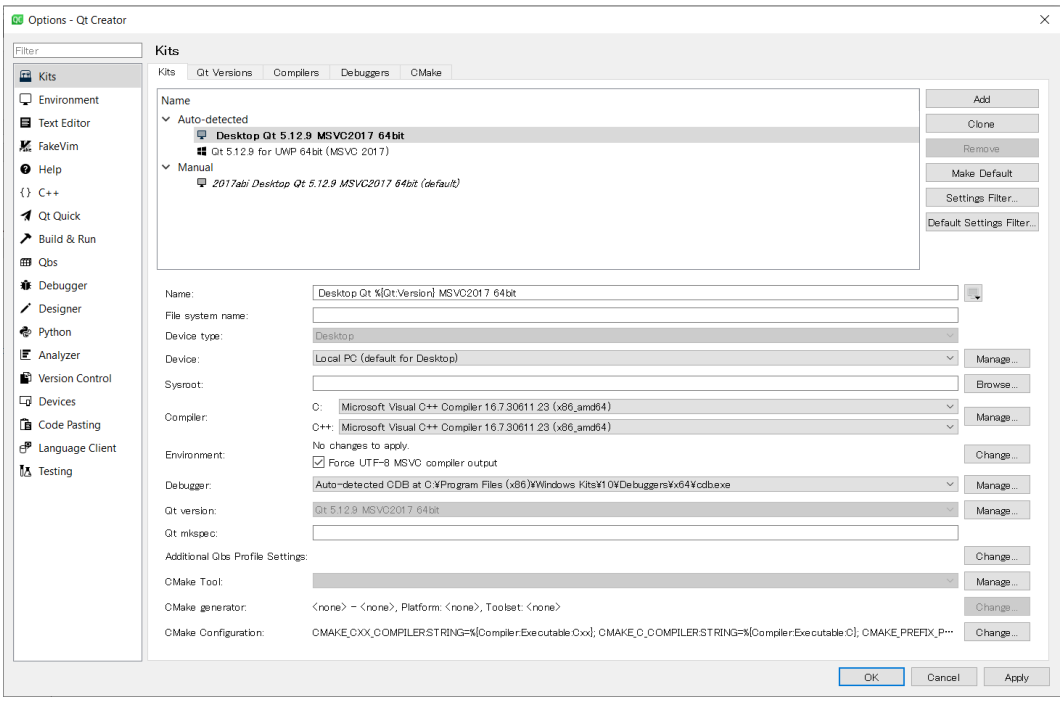


Figure 1.3-3 Clone of compilation settings to ABI parameters

Above cloning is repeated to C/C++ compilers. Next, Kits setting is cloned to Compiler terms.



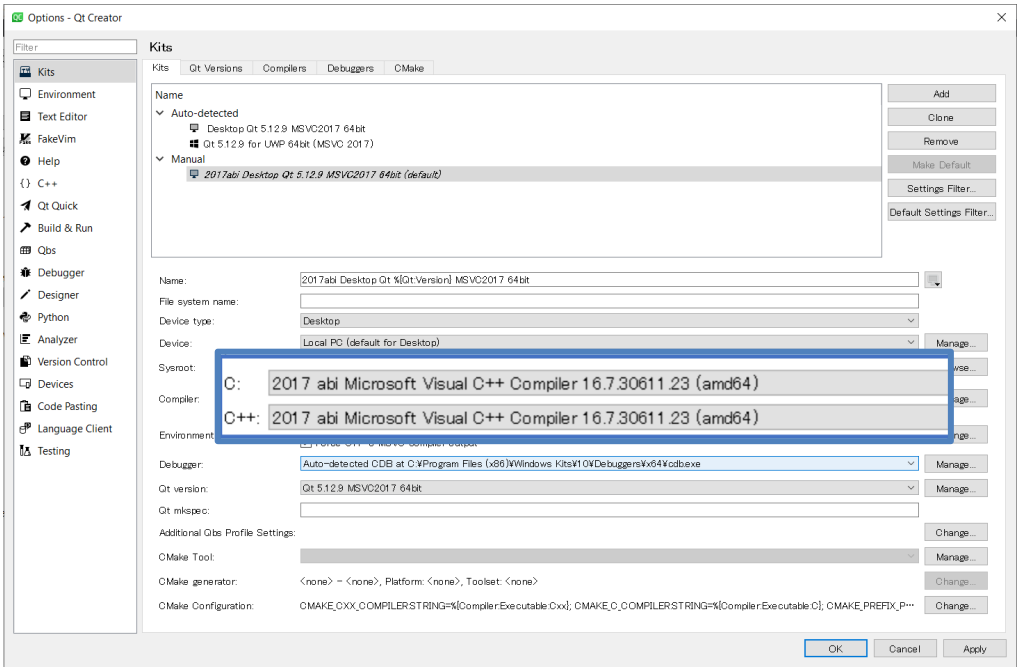


Figure 1.3-4 Copy of Kits setting

2 Installation

PBVR's filter, server, and client programs run on Linux, Mac, and Windows and are available in sequential and OpenMP parallelized binaries. The binaries for supercomputer SGI8600 and Fugaku are also available, which are massively parallelized with MPI+OpenMP.

2.1 PBVR Filter

PBVR Filter is implemented in C and is shipped in two versions: (1) is a MPI+OpenMP version for massively parallel computing and (2) an OpenMP version for thread parallel computing. The following table lists the load module packages. Load modules supporting the VTK format have suffix “_vtk”. The VTK library is needed to compile them. Choose the relevant load modules and copy them to a directory that is specified in the PATH environment parameter. When the copying operation finishes, the installation is complete.

Table 2.1-1 List of load modules for PBVR Filter.

Platform	**Parallelization**	**Name of load module**
Linux 64 bit	Serial	pbvr_filter_linux
	OpenMP	pbvr_filter_linux_omp
Mac 64 bit	Serial	pbvr_filter_mac
	OpenMP	pbvr_filter_mac_omp
Windows 64 bit	Serial	pbvr_filter_win
	OpenMP	pbvr_filter_omp_win
SGI8600	MPI+OpenMP	pbvr_filter_s86_mpi_omp
FUGAKU	MPI+OpenMP	pbvr_filter_fugaku_mpi_omp

*1. The load modules for supercomputers are used only in computing nodes. Therefore, for login nodes and post-processing nodes with Linux, use the load modules built for Linux.

2.2 PBVR Server

PBVR Server is implemented in C++ and is shipped in three versions: (1) a serial processing version, (2) an OpenMP version for thread parallel computing, and (3) an MPI+OpenMP version for massively parallel computing. The following table lists the load module packages. Choose the suitable load modules, and copy them to a directory that is specified in the PATH environment parameter. When the copying finishes, the installation is complete.

Table 2.2-1 List of load modules for PBVR Server

Platform	**Parallelization**	**Name of load module**
Linux 64 bit	Serial	pbvr_server_linux
	OpenMP	pbvr_server_linux_omp
Mac 64 bit	Serial	pbvr_server_mac
	OpenMP	pbvr_server_mac_omp
Windows 64 bit	Serial	pbvr_server.exe
	OpenMP	pbvr_server_omp.exe
SGI8600*1	MPI+OpenMP	pbvr_server_s86_mpi_omp
FUGAKU	MPI+OpenMP	pbvr_server_fugaku_mpi_omp

*1. The load modules for supercomputers are used only in computing nodes. Therefore, if the login nodes and the post-processing nodes are on Linux servers, use the load modules that are compiled for Linux.

2.3 PBVR Client

PBVR Client is implemented in C++ and uses Qt and OpenGL. The following table lists the load modules stored in the *client* directory of the load module package. Choose the relevant load modules, and copy them to a directory that is specified in the PATH environment parameter.

Table2.3-1 List of load modules for PBVR Client

Platform	**Parallelization**	**Name of load module**
Linux 64 bit	pthread	pbvr_client_linux
Mac 64 bit *1	pthread	pbvr_client _mac
Windows 64 bit *2	pthread	pbvr_client.exe

*1. This load module is built for Intel-based Macs, and requires Rosetta to be installed on Macs with M1 chips.

*2. In order to operate the Windows version load module, it is necessary to separately place the GLUT dynamic library in a directory which has a path or in the same directory as the load module. The GLUT dynamic library, glut32.dll, is available from the OpenGL site (3.1.2).

3 Build

The filter and server programs are implemented in C++ and can be built by switching between a sequential processing version, an OpenMP version for threaded parallel processing, and an MPI+OpenMP version for massively parallel processing. The client program is implemented in C++, Qt5 and OpenGL, and can be built by switching between CPU and GPU renderers using a config file. The composition of polygon and particle data (6.3.5) is only possible with the GPU renderer.

3.1 Filter and Server Program

The filter and the server program are compiled by pbvr.conf and Makefile under the PBVR/ directory in the source code package. pbvr.conf specifies the settings for making and compiling PBVR Filter and PBVR Server. The source code package is as follows.

Table3.1-1 Components of source code package

Directory・File	Detail
PBVR/	
KMATH/	Pseudorandom number generator library KMATH
KVS/	Visualization library KVS (for Server program) *2
glui/	Widget library for GUI
Client/	OpenGL ver. PBVR Client programs
FunctionParser/	Function editor library
Common/	Common library for protocols, communications
Filter/	PBVR Filter programs
Server/	PBVR Server programs
arch/	Compilation setting files
pbvr.conf *1	Configuration setting file
Makefile *1	Make file for directory PBVR/

*1. In Windows, the MSVC solution file pbvr.sln is used instead of pbvr.conf and Makefile.

*2. This KVS is not used for Client program. This is customized KVS for the particle generation to compiled at parallel environment.

The parameters in pbvr.conf (Table3.1-2) are used to specify which functions are installed.

Table3.1-2 List of parameters in pbvr.conf

parameter	Value	Detail
PBVR_MACHINE	String	Compilation setting files under arch/
PBVR_MAKE_CLIENT	0 or 1	Support of Open GL ver. PBVR Client
PBVR_MAKE_FILTER	0 or 1	Support of PBVR Filter
PBVR_MAKE_SERVER	0 or 1	Support of PBVR Server
PBVR_SUPPORT_KMATH	0 or 1	Support of KMATH (Server only) *1
PBVR_SUPPORT_VTK	0 or 1	Support of PBVR VTK (Filter only)

*1. In Windows and Mac, KMATH is unavailable, and TynyMT is used.

For PBVR_MACHINE, specify the compilation setting file in directory arch/ from those listed in the following table.

Table3.1-3 List of compilation setting files

Filename	Use
Makefile_machine_gcc	Serial compilation using gcc
Makefile_machine_gcc_omp	OpenMP compilation using gcc
Makefile_machine_gcc_mpi_omp	MPI+OpenMP compilation using gcc
Makefile_machine_intel	Serial compilation using intel
Makefile_machine_intel_omp	OpenMP compilation using intel
Makefile_machine_intel_mpi_omp	MPI+OpenMP compilation using intel
Makefile_machine_s86_mpi_omp	MPI+OpenMP compilation using JAEA's supercomputer For Intel compiler and mpt lib.
Makefile_machine_fugaku_clang	FUGAKU clang mode compilation
Makefile_machine_fugaku_trad	FUGAKU trad mode compilation

3.1.1 Linux and Mac

On a Linux or Mac system, build the source code and install it as follows.

1) Edit pbvr.conf in directory PBVR/ depending on your environment.
   The following example shows the parameters for building OpenMP versions of PBVR Client, PBVR Filter, and PBVR Server using gcc compiler.
   #Example of pbvr.conf

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

   To utilize Qt ver. PBVR Client, disable OpenGL ver. PBVR Client with describing PBVR_MAKE_CLIENT = 0.
2) Compile in directory PBVR/ as follows.
   $make
   Following load modules are generated under PBVR/ directory.
   PBVR Filter: Filter/pbvr_filter
   PBVR Server: Server/pbvr_server
   PBVR Client: Client/pbvr_client
3) Copy the generated load modules to an arbitrary directory that is specified in the PATH environment parameter.

3.1.2 Windows

In Windows, uncompress the source code package and build the source code as follows.

1) Install GLUT
   i) Download *glut-3.7.6-bin_x64.zip(64bit)* from the link below.
   https://user.xmission.com/~nate/glut.html
   ii) Extract the following files:
   glut.h
   glut32.lib
   glut32.dll
   In the setting of pbvr.sln (described later), the above files are set in the following directories.
   ✓ C:¥pbvr¥glut-3.7.6¥include¥GL¥glut.h
   ✓ C: ¥pbvr¥glut-3.7.6¥lib¥ glut32.lib
2) Extract *server* on a Windows machine that has MSVC.
3) Open pbvr.sln with MSVC.
4) To select the function to be used in each environment, select “Solution Properties” from MSVC's Solution Explorer. Then, select whether to support each project in "Configuration

Properties". When using the Qt version client, clear the Client check box to disable the OpenGL version client support.

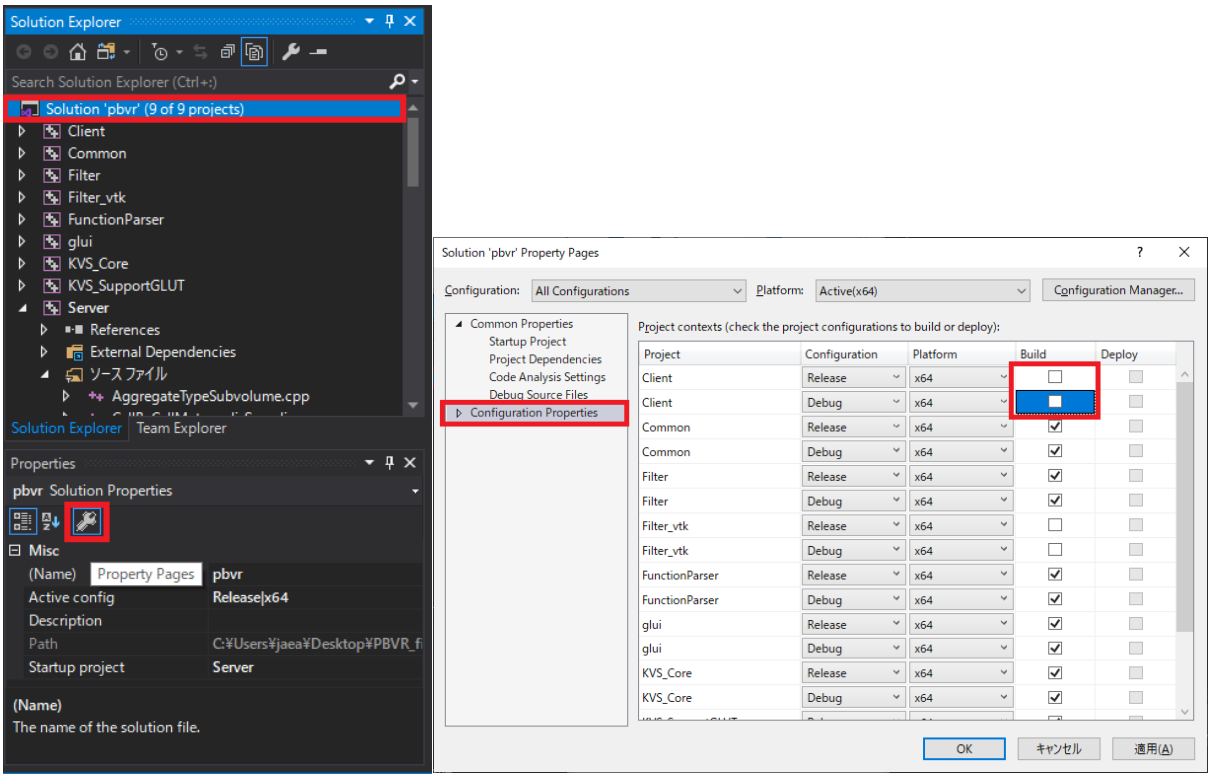


Figure 3.1-1 Inactivation of client support of OpenGL

5) To generate a program parallelized by OpenMP, enable "OpenMP support" in the following project file.
   - ✓ Client
   - ✓ Common
   - ✓ Filter
   - ✓ FunctionParser
   - ✓ glui
   - ✓ KVS_Core
   - ✓ KVS_SupportGLUT
   - ✓ Server

   Select each project from MSVC's Solution Explorer. Then, select whether to support OpenMP in each project from "Configuration Properties" -> “C/C++” -> “Language”
6) Choose **Release** and **x64** from the pull-down list as shown in Figure3.1-2.

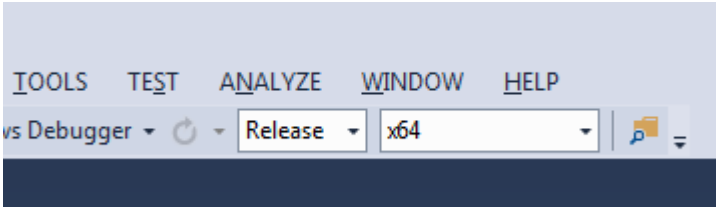


Figure3.1-2 Build configuration for MSVC

7) Go to the menu **Build** > **Build Solution**.

The load modules pbvr_filter.exe, pbvr_server.exe, and pbvr_client.exe are created under *¥¥x64¥Release*.

3.1.3 Filter for VTK data

VTK6.0 or later is required to compile and install PBVR Filter for VTK data. Refer to the VTK website (http://www.vtk.org/) for how to install the VTK library. In the installation, do the following in CMake-gui.

1) Turn on the BUILD_SHARED_LIBS option.
2) Choose “Release” for the CMAKE_BUILD_TYPE option.
3) Set the VTK installation directory to CMAKE_INSTALL_PREFIX.

On each environment, PBVR Filter is compiled as follows.

Installation for Linux and Mac

Execute the following compilation commands.

```
$ export VTK_VERSION=n.n
$ export VTK_LIB_PATH=/usr/local/lib
$ export VTK_INCLUDE_PATH=/usr/local/include/vtk-n.n
$ make -f makefile.linux vtk
```

Here, n.n denotes the version of the VTK library. Each path should be modified depending on the VTK installation directory.

Installation for Windows

Set the following environment parameters using
Control Panel > System > Property > Environment.

Parameter	*Value*
VTK_LIB	d:¥environments¥VTK¥lib
VTK_VERSION	n.n
VTK_INCLUDE_PATH	d:¥environments¥VTK¥include¥vtk-n.n

Here, n.n denotes the version of the VTK library. Each path should be modified depending on the VTK installation directory.

3.2 Client Program

The client program is built using the visualization library KVS (https://github.com/naohisas/KVS) and the integrated development environment Qt Creator.

3.2.1 Installation of KVS library

For details on how to install KVS, please refer to the Wiki on the download page. However, this client program is not built using the original KVS, but using a proprietary version of KVS (https://github.com/CCSEPBVR/KVS) with improved framebuffer handling and other features.

As environment variables for installation, specify the KVS installation path in KVS_DIR and add the commands for KVS in PATH.

Windows
Create the following environment variables from System Preferences and set their values.

Variable	Value
KVS_DIR	C:¥Program Files¥kvs (Anywhere you want)
PATH	%PATH%;%KVS_DIR%¥bin

Linux/Mac
Set the following environment variables from a terminal.

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

In addition, for Macs with the M1 chip, it is necessary to switch the architecture. arm64 architecture is used on M1 Macs, but Qt5.12, on which the client program was developed, does not support arm64. On the M1 Mac, please install Rosetta beforehand and switch the architecture to x86-64 by executing the following command in the terminal before the build of KVS.

```
arch -x86_64 bash
```

Set configuration by specifying kvs.conf as following.

```
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
```

The Mac environment does not use GLUT and GLEW for KVS installation. However, the Linux/Windows environment uses GLUT and GLEW, and you need to install and set the path accordingly.

Linux

Installation of GLUT

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

Installation of GLEW

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

Setting the install path to the environment variables KVS_GLUT_DIR and KVS_GLEW_DIR.

```
export KVS_GLUT_DIR=”Install path for GLUT”
export KVS_GLEW_DIR=”Install path for GLEW”
```

Windows

1. Download glut built in 64bit mode from following URL.
   http://coskx.webcrow.jp/mrr/for_students/LectGLCG/distribution/index.html
   The names of glut files included in the download file are 32bit (glut32.dll and glut32.lib), but they are built for 64bit.
2. Download the 64-bit version of GLEW from the GLEW download page (http://glew.sourceforge.net).

3. Unzip the downloaded file and copy the following files contained in it to the specified folder.

Name	Destination
glut32.dll glew64.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

Create the environment variables KVS_GLUT_DIR and KVS_GLEW_DIR from System Preferences and set the GLUT and GLEW installation location (in the example above, to C:¥PBVR_Dev¥OpenGL).

Compile and install KVS on the terminal.

Windows
Launch the developer command prompt of Visual Studio and type nmake command to build KVS by MSVC. In the following example, the directory of the KVS source code is C:SRC¥KVS.

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

Linux/Mac
Build KVS from a terminal using make. In the example below, the directory of the KVS source code is set to /SRC/KVS.

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

See the wiki on the download page for more options on compiling.

3.2.2 Setting for Qt Creator

The client program is configured in qtpbvr.conf. Edit qtpbvr.conf as follows to enable PBVR_MODE=CS.

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

Open the project file QtClient.pro included in the source code with Qt Creator to configure and build the client program.

1. Select “File” -> “Open File or Project”.
2. Select the QtClient.pro file from the QTPBVR/QtClient directory.
   PBVRClient/QtClient/QtClient.pro
3. Select “QtClient” from Active Project.
4. Select “Build Settings” and check “Shadow Build”.
5. Set “../build” in “Build Directory”.

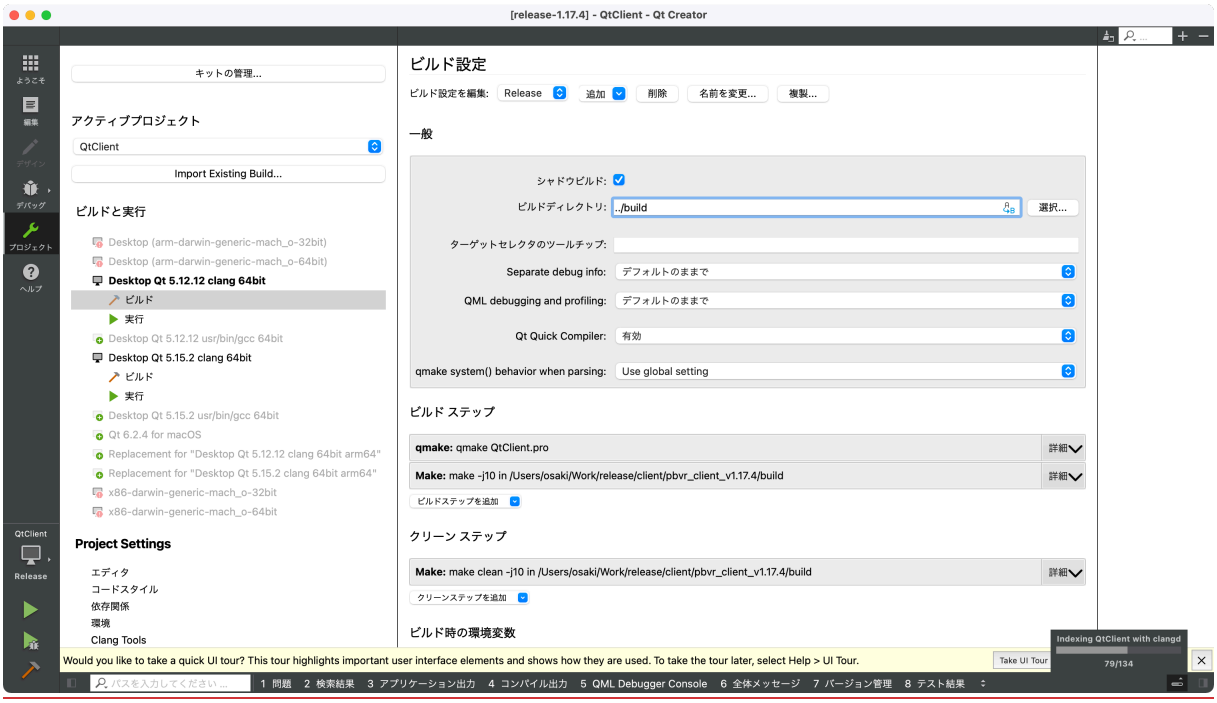


Figure 3.2-1 Client Configuration

Next, set KVS environment variables in your project. In "Build Settings", press "Detail" in "Build Environment" to expand the section, and the environment variables KVS_DIR (where KVS is installed), KVS_SOURCE (where KVS source code is installed), and GLEW_DIR (where GLEW is installed) is added. On Windows, also add %GLEW_DIR%¥bin to the PATH.

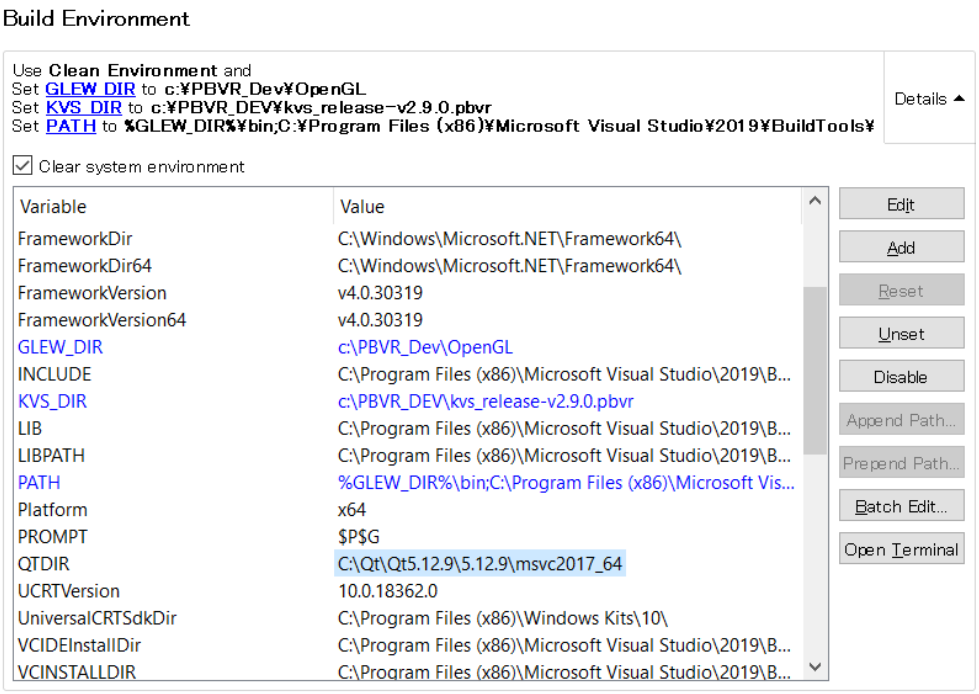


Figure 3.2-2 Setting of KVS environmental variables

Next, Client program can be built.

1. Select Edit on the left toolbar.
2. Right click on the “QtClient” project and select "Run qmake".
3. Right click the “QtClient” project and select "Build".

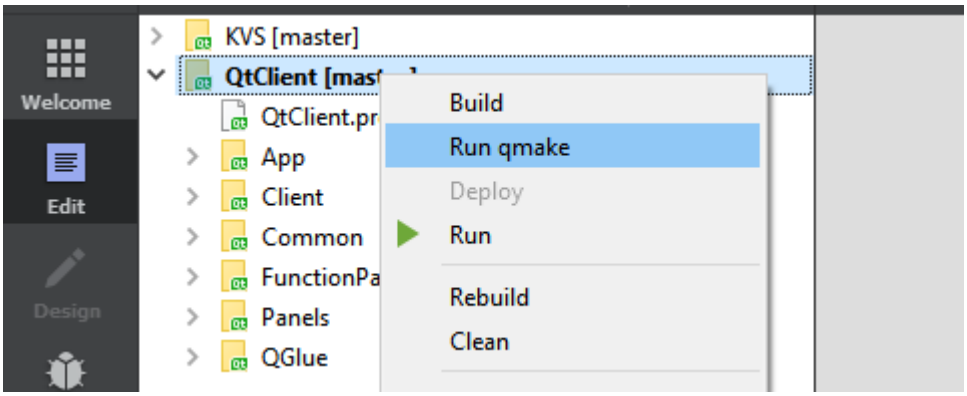


Figure 3.2-3 Build of Client Program

3.2.3 Deployment on Windows

This step is only necessary on Windows, and only if you want to deploy an application that can be relocated from the build location.

C:> cd C:¥PBVR_Dev¥QTPBVR¥build¥App
C:> windeployqt –release pbvr_client.exe

C:> cp C:¥PBVR_Dev¥OpenGL¥bin¥*.dll .

Next, copy shader programs from installed KVS to App folder.

Source: /include/Core/Visualization/Shader/
Destination: /App/include/Core/Visualization/Shader/

The App folder can now be relocated and executed from another location on the same machine, or on another Windows machine.

3.2.4 CPU or GPU Renderer

The project will build default with GPU renderer. The GPU renderer is built by default setting. But CPU renderer can be built for non GPU environment such as supercomputer.
To build the CPU renderer, open pbvr_client -> QtClient -> qtpbvr.conf and change the variable specifying the build mode as follows.

From
REND_MODE = GPU
To
REND_MODE = CPU

4 PBVR Filter

PBVR Filter is an independent part of the PBVR system. PBVR Filter divides time-series volume data, which then become the input for parallel processing in PBVR Server. In addition, PBVR Filter generates Sub-volume data for the purpose of visualization. The data decomposition is based on the octree model. PBVR Filter divides structured grid data and unstructured grid data into user-specified octree regions in order to generate the input files for parallel processing by PBVR Server.

4.1 Data Decomposition Model

As shown in Figure4.1-1, the octree data structure divides each edge of a cuboid in half, recursively. Therefore, each cuboid has eight child cuboids, each of which has a single parent.

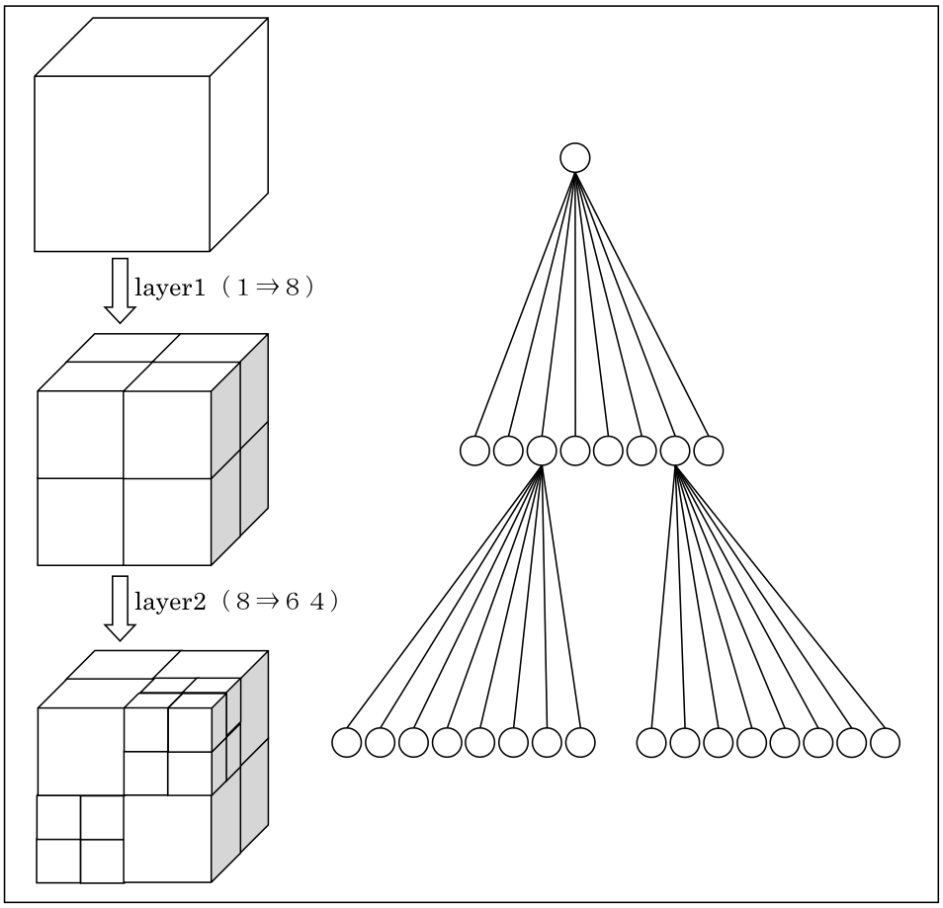


Figure4.1-1 Space partitioning with the octree data structure

As shown inFigure4.1-2, the boundaries of the child-cuboids are computed by dividing the sum of the minimum and maximum coordinates values by two. Given a point in the domain,

which cuboid containing the point can be determined by comparing the coordinates of the vertex and the boundaries.

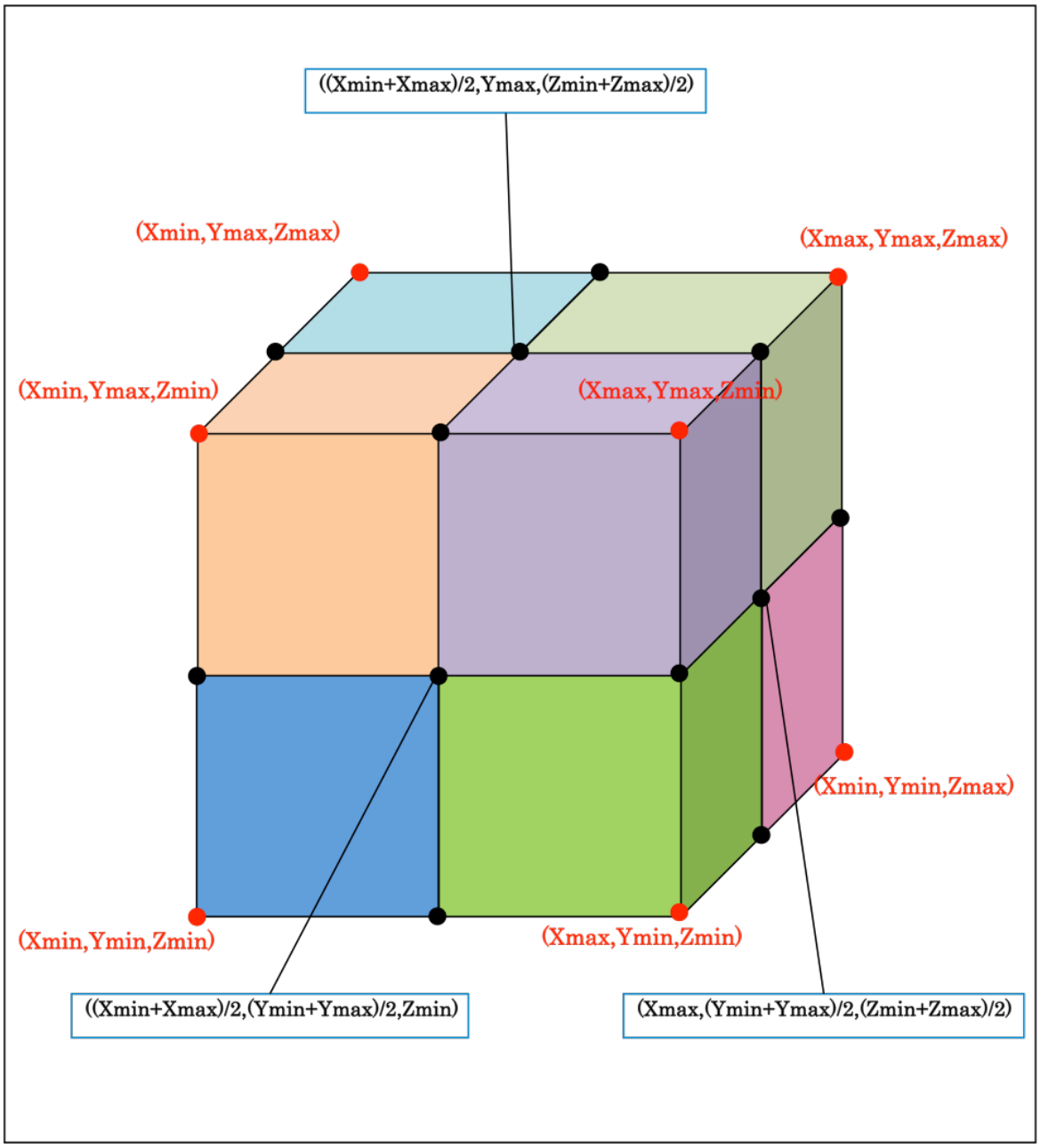


Figure4.1-2 Coordinates of the boundaries in the octree data structure.

4.2 Launching PBVR Filter

The following examples show how to launch PBVR Filter. Note that PBVR Filter requires parameters that are specified in a parameter file. The name of the parameter file is specified in the command line when launching PBVR Filter. If no parameter file name is given or a non-existent file name is provided, the execution of PBVR Filter will fail.

Examples:

Launch the MPI+OpenMP version of PBVR Filter with *N* processes:

```
$ mpiexec –n N filter param.txt
```

Launch the OpenMP version:

```
$ filter param.txt
```

*1. In both cases, the number of OpenMP threads is set in the environment parameter OMP_NUM_THREADS.

4.2.1 Launching PBVR Filter with VTK Support

Depending on your environment, an environment parameter needs to be set to launch PBVR Filter with VTK support.

Installation for Linux

Set the following parameter.

```
$ export LD_LIBRARY_PATH=${VTK_LIB_PATH}:$LD_LIBRARY_PATH
```

Installation for Mac

Set the following parameter.

```
$ export DYLD_LIBRARY_PATH=${VTK_LIB_PATH}:$DYLD_LIBRARY_PATH
```

Here, n.n denotes the version of the VTK library. Each path should be modified depending on the VTK installation directory.

Installation for Windows

Set the following environment parameters using
Control panel > System > Property > Environment.

Parameter	*Value*
Path	d:¥Program¥VTK6.3.0¥bin

The path should be specified to the bin directory under the VTK installation directory.

4.3 File Formats

This section describes the file formats that are read or written by PBVR Filter. All binary data in input and output files are single precision, without a header or footer, and in little endian format. The following three file formats are available: (1) the split format (which actually make use of kvsml format), (2) the sub-volume aggregate format, and (3) the step aggregate format (Figure4.3-1). The split format generates independent files for each time step for each sub-volume. However, in this format, the number of files grows exponentially as the number of layers in the octree increases. This problem can be avoided by using either of the other two file formats. The sub-volume aggregate format aggregates files over different time steps (but for the same sub-volume) to a single file. Conversely, the step aggregate format aggregates files over different sub-volumes (for the same time step) to a single file. The following sections explain these three file formats in detail.

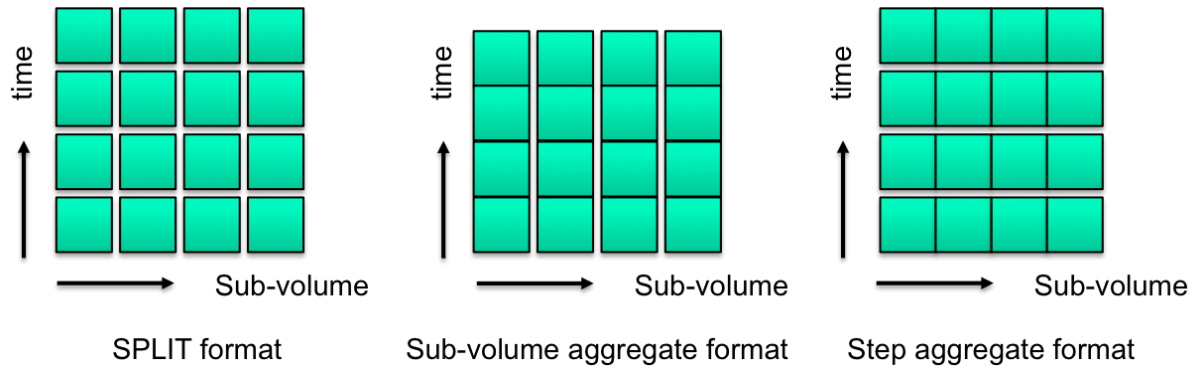


Figure4.3-1 Output file formats available for PBVR Filter

4.3.1 Input Data Format

PBVR Filter can process the following data formats as input.

1) AVSFLD binary data*1
2) AVSUCD ascii and binary data*1
3) STL binary data*2
4) PLOT3D binary data*3
5) VTK Legacy binary data *4

*1. Details of AVS data formats can be found in the AVS manual or at http://www.cybernet.co.jp/avs. AVSUCD binary data in “data” format can be used. However, the geom and the data_geom formats are not supported. 2D and 3D elements in Table4.7-1 and their mixed elements are supported.

*2. Details of STL data formats can be found at https://en.wikipedia.org/wiki/STL_(file_format).

*3. Refer the details of PLOT3D data formats can be found at

http://ntrs.nasa.gov/archive/nasa.

*4. Details of VTK data formats can be found at http://www.vtk.org/. PBVR Filter can process VTK structured points, VTK structured grid, VTK rectilinear grid, VTK unstructured grid, and VTK polygonal data.

4.3.2 Endian

The binary files used in PBVR Filter are in little endian format. Conversion is necessary on a big endian machine if the input data files are not already in little endian format.

4.3.3 Filter Output Information File (PFI)

A PFI file is a binary data file that summarizes the information about an input volume. It contains the following data:

Total number of nodes (int)
Total number of elements (int)
Element type (int) *1
File type (int) *2
Number of files (int) *3
Number of components (int)
Beginning time step (int)
Ending time step (int)
Number of sub-volumes (int) *4
Minimum *X*-coordinate value of the entire 3D space (float)
Minimum *Y*-coordinate value of the entire 3D space (float)
Minimum *Z*-coordinate value of the entire 3D space (float)
Maximum *X*-coordinate value of the entire 3D space (float)
Maximum *Y*-coordinate value of the entire 3D space (float)
Maximum *Z*-coordinate value of the entire 3D space (float)
Number of nodes for sub-volume 1 (int)
Number of nodes for sub-volume 2 (int)
Number of nodes for sub-volume 3 (int)
:
Number of nodes for sub-volume *n* (int)
Number of elements for sub-volume 1 (int)
Number of elements for sub-volume 2 (int)
Number of elements for sub-volume 3 (int)
:
Number of elements for sub-volume *n* (int)
Minimum *X*-coordinate value of sub-volume 1 (float)
Minimum *Y*-coordinate value of sub-volume 1 (float)
Minimum *Z*-coordinate value of sub-volume 1 (float)
Maximum *X*-coordinate value of sub-volume 1 (float)
Maximum *Y*-coordinate value of sub-volume 1 (float)
Maximum *Z*-coordinate value of sub-volume 1 (float)
Minimum *X*-coordinate value of sub-volume 2 (float)
Minimum *Y*-coordinate value of sub-volume 2 (float)
Minimum *Z*-coordinate value of sub-volume 2 (float)
Maximum X-coordinate value of sub-volume 2 (float)
Maximum Y-coordinate value of sub-volume 2 (float)
Maximum Z-coordinate value of sub-volume 2 (float)
:
Minimum *X*-coordinate value of sub-volume *n* (float)
Minimum *Y*-coordinate value of sub-volume *n* (float)
Minimum *Z*-coordinate value of sub-volume *n* (float)
Maximum *X*-coordinate value of sub-volume *n* (float)
Maximum *Y*-coordinate value of sub-volume *n* (float)
Maximum *Z*-coordinate value of sub-volume *n* (float)
Minimum value of variable 1 for time step 1
Maximum value of variable 1 for time step 1
Minimum value of variable 2 for time step 1

Maximum value of variable 2 for time step 1
:
Minimum value of variable *N* for time step 1
Maximum value of variable *N* for time step 1
:
Minimum value of variable 1 for time step *m*
Maximum value of variable 1 for time step *m*
Minimum value of variable 2 for time step *m*
Maximum value of variable 2 for time step *m*
:
Minimum value of variable *N* for time step *m*
Maximum value of variable *N* for time step *m*

*1. Element types are defined in Table4.7-1.

*2. Set the int value to 0, 1, or 2 in order to specify one of the following file formats:

0: SPLIT format

1: sub-volume aggregate format

2: step aggregate format

*3. The number of files used when the input file format is sub-volume aggregate format.

*4. The number of sub-volumes is 8^{n_layer}, for example:

n_layer = 0 : 1

n_layer = 1 : 8

n_layer = 2 : 64

n_layer = 3 : 512

n_layer = 4 : 4,096

n_layer = 5 : 32,768

n_layer = 6 : 262,144

n_layer = 7 : 2,097,152

4.3.4 SPLIT File Format

In addition to the data files, when the split file format is used, two files are produced for each sub-volume. The first is the element configuration file and it describes which nodes are in each cell. The second is the node coordinate file and it specifies the coordinates of the nodes. Moreover, there is a parameter file for each sub-volume for each time step. This file assigns the parameters (physical quantities) to each node. All three types of files are in kvsml format. Thus, the total number of files is

Number of sub-volumes × 2 + *Number of sub-volumes* × *Number of time steps* × 2.

Example:

If *n_layer* is 7 and the number of time steps is 100, then the total number of files is 423,624,704.

4.3.4.1 File Naming Convention

In PBVR, files in the SPLIT format have the following naming convention.

prefix_XXXXX_YYYYYYY_ZZZZZZZ.kvsml	: kvsml file（ASCII format）
prefix_YYYYYYY_ZZZZZZZ_connect.dat	: element configuration file (binary format)
prefix_YYYYYYY_ZZZZZZZ_coord.dat	: node coordinate file（binary format）
prefix_XXXXX_YYYYYYY_ZZZZZZZ_value.dat	:parameter file（binary format）

Here, prefix, ‘XXXXX’, ‘YYYYYYY’, and ‘ZZZZZZZ’ should be replaced with the following strings.

‘prefix’	: Arbitrary string of characters that are allowed in a file name
‘XXXXX’	: Number of steps（5 digits）
‘YYYYYYY’	: Sub-volume index (7 digits)
‘ZZZZZZZ’	: Total number of sub-volumes（7 digits）

4.3.4.2 kvsml File Format

```
<?xml version="1.0" ?>
<KVSML>
    <Object type="UnstructuredVolumeObject">
        <UnstructuredVolumeObject cell_type=" type of elements">
            <Node nnodes="number of nodes in the sub-volume">
              <Value veclen="number of variables">
                <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="number of elements in the sub-volume">
              <Connection>
                <DataArray type="uint"    file=" prefix_YYYYYYY_ZZZZZZZ_connect.dat" format="binary" />
              </Connection>
            </Cell>
        </UnstructuredVolumeObject>
    </Object>
</KVSML>
```

4.3.4.3 Format of Element Configuration File

Node 1 of element 1
Node 2 of element 1
:
Node *n* of element 1
Node 1 of element 2
Node 2 of element 2
:
Node *n* of element 2
Node 1 of element 3
Node 2 of element 3
:
Node *n* of element 3
:
Node 1 of element *N*
Node 2 of element *N*
:
Node *n* of element *N*

4.3.4.4 Format of Node Coordinate File

X-coordinate value of node 1
Y-coordinate value of node 1
Z-coordinate value of node 1
X-coordinate value of node 2
Y-coordinate value of node 2
Z-coordinate value of node 2
X-coordinate value of node 3
Y-coordinate value of node 3
Z-coordinate value of node 3
:
:
X-coordinate value of node *m*
Y-coordinate value of node *m*
Z-coordinate value of node *m*

4.3.4.5 Variable File

Variable 1 of Node 1
Variable 1 of Node 2
Variable 1 of Node 3
:
Variable 1 of Node *n*
Variable 2 of Node 1
Variable 2 of Node 2
Variable 2 of Node 3
:
Variable 1 of Node *n*
Variable *m* of Node 1
Variable *m* of Node 2
Variable *m* of Node 3
:
Variable *m* of Node *n*

4.3.5 Sub-volume Aggregate Format

In sub-volume aggregate format, information on element configurations, node coordinates, and parameters for all time steps are gathered into a single file for each sub-volume. By specifying the number of files (as explained in Section 5.1.1.1), one can aggregate information for several sub-volumes into an arbitrary number of files from 1 to the number of sub-volumes. (If *n_layer* is 7, then the number of files is 2,097,152.)

4.3.5.1 Naming Convention

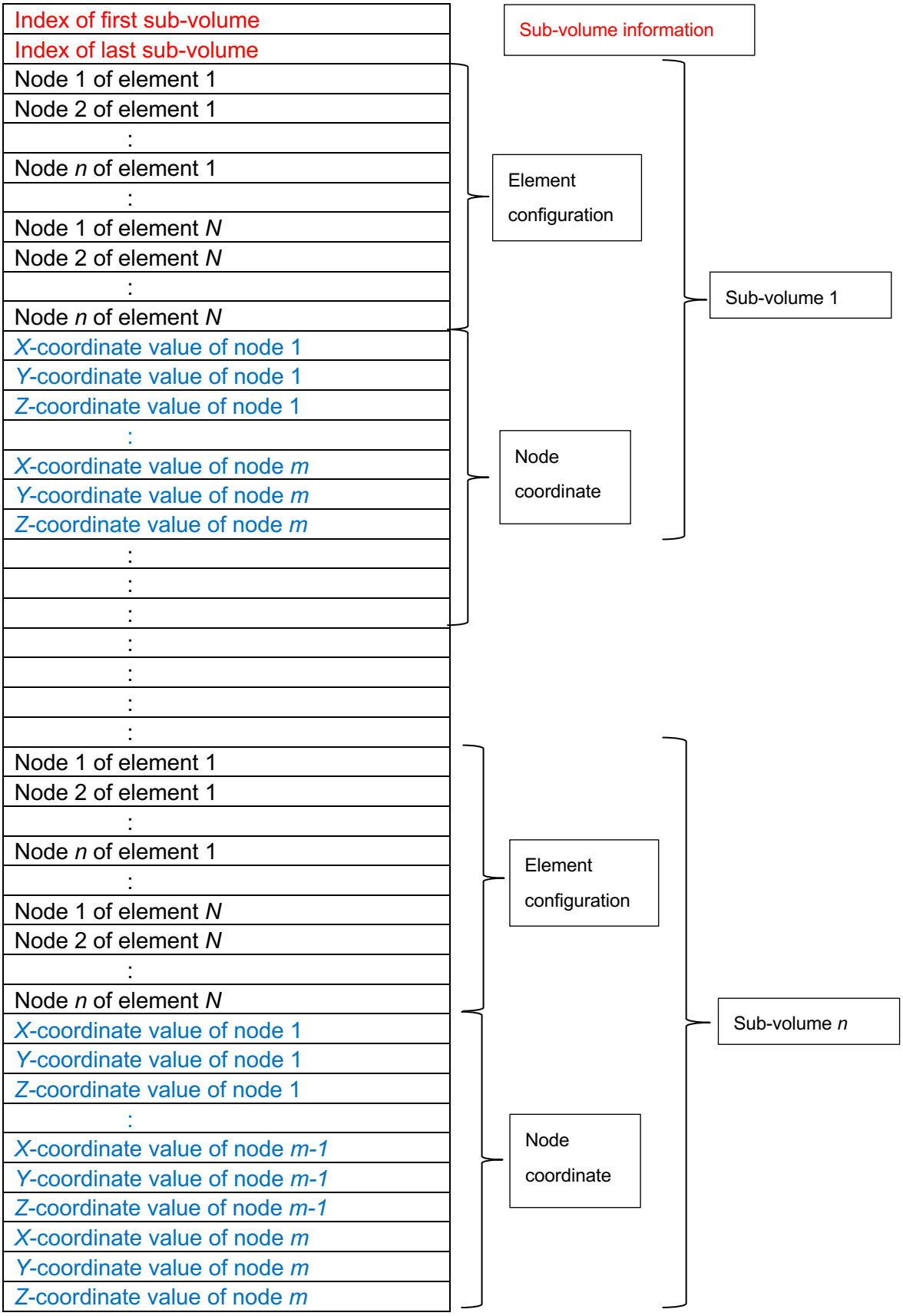
In PBVR, files in the sub-volume aggregate format have the following naming convention.

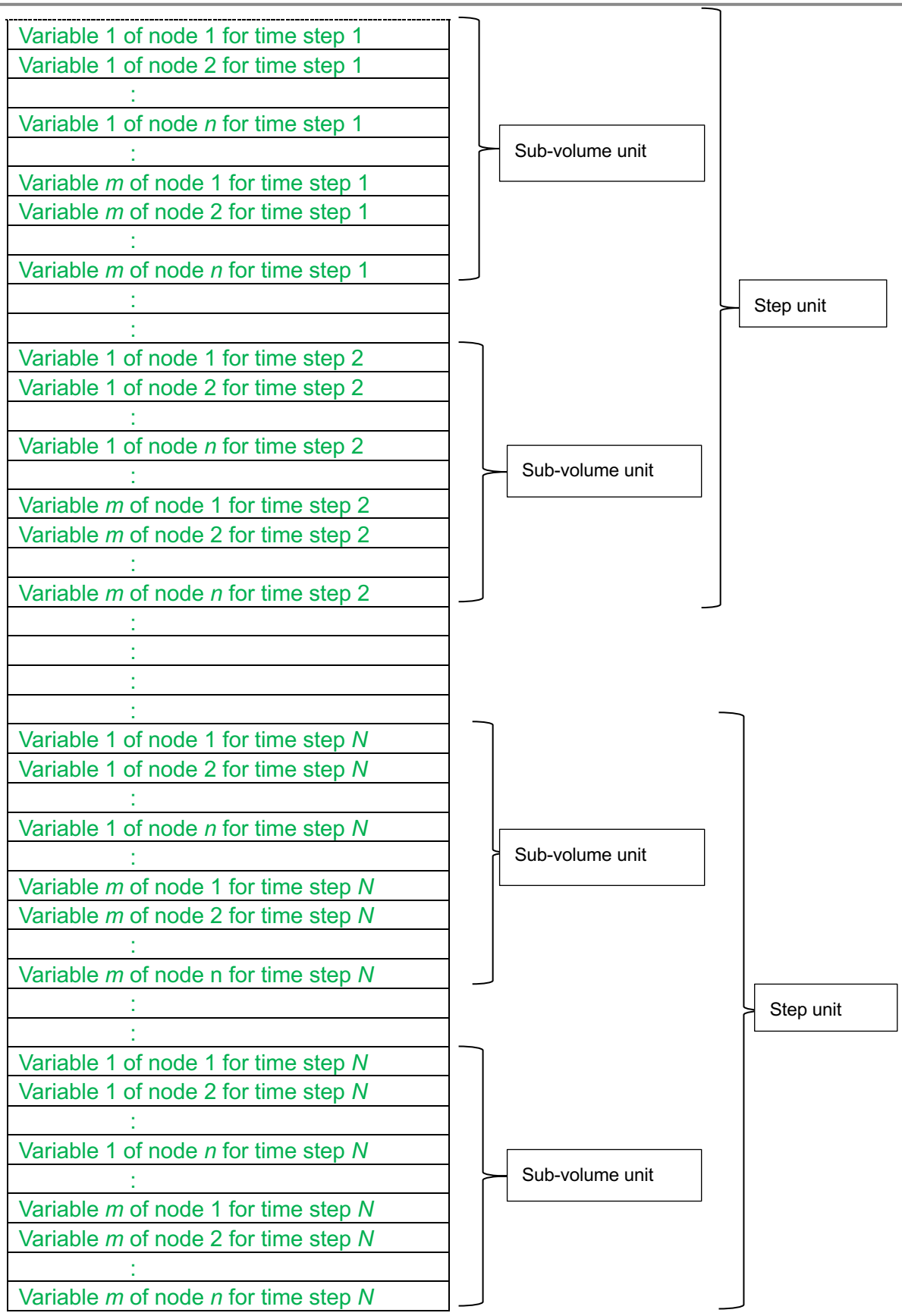
prefix_YYYYYYY_ZZZZZZZ.dat (binary format）

Here, ‘prefix’, ‘XXXXX’, ‘YYYYYYY’, and ‘ZZZZZZZ’ should be replaced with the following strings.

prefix	: Arbitrary string of characters that are allowed in a file name
YYYYYYY	: File number（7 digits）
ZZZZZZZ	: Total number of files（7 digits）

4.3.5.2 File Format




Variable 1 of node 1 for time step 1
Variable 1 of node 2 for time step 1
:
Variable 1 of node n for time step 1
:
Variable m of node 1 for time step 1
Variable m of node 2 for time step 1
:
Variable m of node n for time step 1
:
:
Variable 1 of node 1 for time step 2
Variable 1 of node 2 for time step 2
:
Variable 1 of node n for time step 2
:
Variable m of node 1 for time step 2
Variable m of node 2 for time step 2
:
Variable m of node n for time step 2
:
:
:
:
Variable 1 of node 1 for time step N
Variable 1 of node 2 for time step N
:
Variable 1 of node n for time step N
:
Variable m of node 1 for time step N
Variable m of node 2 for time step N
:
Variable m of node n for time step N
:
:
Variable 1 of node 1 for time step N
Variable 1 of node 2 for time step N
:
Variable 1 of node n for time step N
:
Variable m of node 1 for time step N
Variable m of node 2 for time step N
:
Variable m of node n for time step N
Sub-volume unit
Sub-volume unit
Sub-volume unit
Step unit
Step unit

4.3.6 Step Aggregate Format

In step aggregate format, there is an element configuration file and a node coordinate file. These two files contain information on all sub-volumes. A parameter file is produced for each step. Therefore, the total number of files is the number of steps + 2.

4.3.6.1 File Naming Convention

In PBVR, files in the step aggregate format have the following naming convention.

prefix_connect.dat	: element configuration file （binary format）
prefix_coord.dat	: node coordinate file （binary format）
prefix_XXXXX_value.dat	:parameter file （binary format）

Here, ‘prefix’ and ‘XXXXX’ should be replaced with the following strings.

prefix	: Arbitrary string of characters that are allowed in a file name
XXXXX	: Number of steps（5 digits）

4.3.6.2 Element Configuration File Format

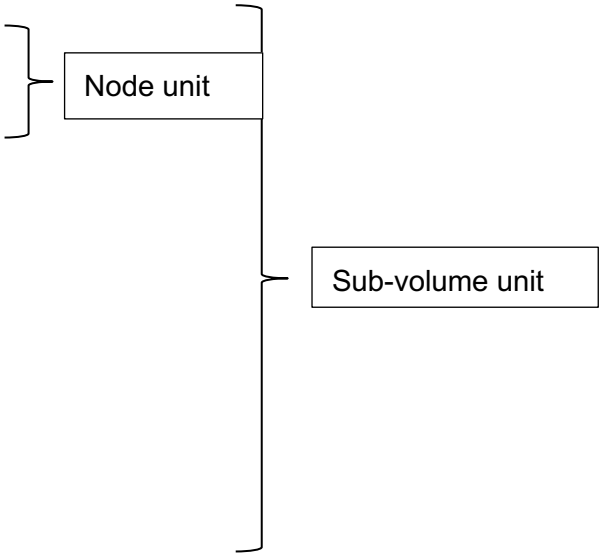
Node 1 of element 1 for sub-volume 1
Node 2 of element 1 for sub-volume 1
:
Node *n* of element 1 for sub-volume 1
Node 1 of element 2 for sub-volume 1
Node 2 of element 2 for sub-volume 1
:
Node *n* of element 2 for sub-volume 1
Node 1 of element 3 for sub-volume 1
Node *2* of element 3 for sub-volume 1
:
Node *n* of element 3 for sub-volume 1
:
Node 1 of element *N* for sub-volume 1
Node 2 of element *N* for sub-volume 1
:
Node *n* of element *N* for sub-volume *1*
:
Node 1 of element 1 for sub-volume *M*
Node 2 of element 1 for sub-volume *M*
:
Node *n* of element 1 for sub-volume *M*
Node 1 of element 2 for sub-volume *M*
Node 2 of element 2 for sub-volume *M*
:
Node *n* of element 2 for sub-volume *M*
Node 1 of element 3 for sub-volume *M*
Node *2* of element 3 for sub-volume *M*
:
Node *n* of element 3 for sub-volume *M*
:
Node 1 of element *N* for sub-volume *M*
Node 2 of element *N* for sub-volume *M*
:
Node *n* of element *N* for sub-volume *M*

Element unit

Sub-volume unit

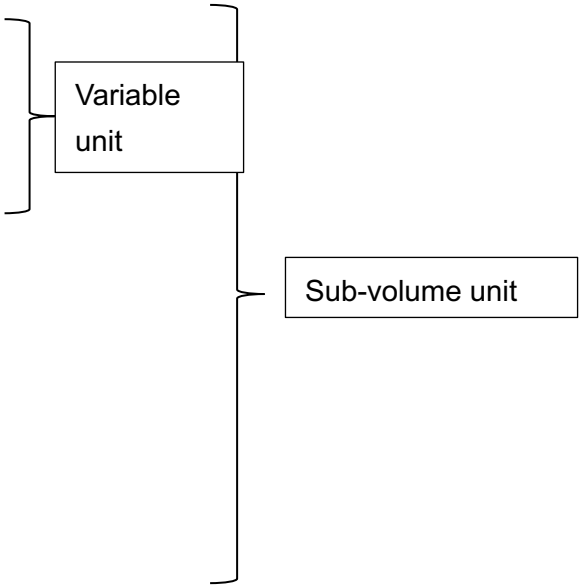
4.3.6.1 Node Coordinate File Format

X-coordinate of node 1 for sub-volume 1
Y-coordinate of node 1 for sub-volume 1
Z-coordinate of node 1 for sub-volume 1
X-coordinate of node 2 for sub-volume 1
Y-coordinate of node 2 for sub-volume 1
Z-coordinate of node 2 for sub-volume 1
X-coordinate of node 3 for sub-volume 1
Y-coordinate of node 3 for sub-volume 1
Z-coordinate of node 3 for sub-volume 1
:
:
X-coordinate of node *m* for sub-volume 1
Y-coordinate of node *m* for sub-volume 1
Z-coordinate of node *m* for sub-volume 1
:
:
:
X-coordinate of node 1 for sub-volume *M*
Y-coordinate of node 1 for sub-volume *M*
Z-coordinate of node 1 for sub-volume *M*
X-coordinate of node 2 for sub-volume *M*
Y-coordinate of node 2 for sub-volume *M*
Z-coordinate of node 2 for sub-volume *M*
X-coordinate of node 3 for sub-volume *M*
Y-coordinate of node 3 for sub-volume *M*
Z-coordinate of node 3 for sub-volume *M*
:
:
X-coordinate of node *m* for sub-volume *M*
Y-coordinate of node *m* for sub-volume *M*
Z-coordinate of node *m* for sub-volume *M*



4.3.6.2 Variable File Format

Variable 1 of node 1 for sub-volume 1
Variable 1 of node 2 for sub-volume 1
Variable 1 of node 3 for sub-volume 1
:
Variable 1 of node *n* for sub-volume 1
Variable 2 of node 1 for sub-volume 1
Variable 2 of node 2 for sub-volume 1
Variable 2 of node 3 for sub-volume 1
:
Variable 2 of node *n* for sub-volume 1
Variable *m* of node 1 for sub-volume 1
Variable *m* of node 2 for sub-volume 1
Variable *m* of node 3 for sub-volume 1
:
Variable *m* of node *n* for sub-volume 1
:
:
:
Variable 1 of node 1 for sub-volume *M*
Variable 1 of node 2 for sub-volume *M*
Variable 1 of node 3 for sub-volume *M*
:
Variable 1 of node *n* for sub-volume *M*
Variable 2 of node 1 for sub-volume *M*
Variable 2 of node 2 for sub-volume *M*
Variable 2 of node 3 for sub-volume *M*
:
Variable 2 of node *n* for sub-volume *M*
Variable *m* of node 1 for sub-volume *M*
Variable *m* of node 2 for sub-volume *M*
Variable *m* of node 3 for sub-volume *M*
:
Variable *m* of node *n* for sub-volume *M*



4.4 Parameter File

The parameter file is in ASCII format. It is used in PBVR Filter for AVSFLD/UCD, PLOT3D, STL, and VTK data. The name of the parameter file is specified on the command line when running PBVR Filter. Table4.4-1 lists the parameters.

Table4.4-1 List of PBVR Filter input parameters

Parameter name	Parameter detail	Default value	Notes
in_dir	Input file directory	'.'	Directory path for input files *1
field_file	AVSFLD file name	-	*2, *3, *4
stl_binary_file	STL file name	-	*2
Plot3d_config_file	PLOT3D configuration file name	-	*2, *3
vtk_file	VTK file name	-	*2, *3, *5
vtk_in_prefix	Prefix of time series VTK data files	-	*2, *3, *5
vtk_in_suffix	Suffix of time series VTK data files	-	*2, *3, *5
ucd_inp	AVSUCD file name	-	ASCII format*2
in_prefix	Prefix of time series AVSUCD data files	-	Binary format*2
in_sufix	Suffix of time series AVSUCD data files	-	Binary format*2
format	Step number format for time series data	"%05d"	
out_dir	Output file directory	'.'	Directory path of output files *1
out_prefix	Output file prefix	'output.'	
start_step	Starting step number	'1'	*6
end_step	Ending step number	'1'	*6
n_layer	Number of octree layer	'0'	An integer from '0' to '7'
output_type	File format	'0'	'0': SPLIT format '1': sub-volume aggregate '2': step aggregate
file_number	Number of output files	'0'	An integer greater than 0. When set to '0', the number of sub-volume is used. Valid only in Sub-volume aggregate file format.
mpi_volume_div	Number of MPI parallelism in sub-volume	'1'	The total number of MPI processes is given by mpi_volume_div × mpi_step_div. *7

mpi_step_div	Number of MPI parallelism in time step	‘1’	The total number of MPI processes is given by mpi_volume_div × mpi_step_div *7
mpi_div	Configuration of 2D MPI parallel processing	‘2’	‘0’: defined by mpi_volume_div’ and mpi_step_div. ‘1’: automatic with priority on sub-volume decomposition. ‘2’: automatic with priority on step decomposition. Options 1 and 2 do not work when mpi_volume_div and mpi_step_div are set.
multi_elem_type	Flag on mixed element type unstructured grid	‘0’	‘0’: data with a single element type ‘1’: data with multiple element types
temp_delete	Flag on temporary files produced by processing mixed element data	‘1’	‘0’: keep temporary files ‘1’: delete temporary file

*1. Directories can be specified either with an absolute path or a relative path, although tilde (~) cannot be used as an abbreviation for the home directory.

*2. One of the following options, field_file, stl_binary_file, plot3d_config_file, vtk_file, vtk_in_prefix(suffix), ucd_inp, or in_prefix(suffix) should be given.

*3. When the input data are 2D or 3D structured grid data, the output data are converted to unstructured grid data with linear quadrilateral or hexahedral elements, respectively.

*4. The file can refer only to the parameters ‘nstep’, ‘ndim’, ‘dim1’, ‘dim2’, ‘dim3’, ‘veclen’, ‘coord [123]’, and ‘variable.

*5. Five VTK Legacy data formats (VTK Structured Points, VTK Structured Grid, VTK Rectilinear Grid, VTK UnstructuredGrid, and VTKPolygonalData) are automatically recognized by PBVR Filter.

*6. Valid only for time series data.

*7. When ‘mpi_volume_div’ and ‘mpi_step_div’ are specified, an error occurs if the value of ‘mpi_volume_div’ × ‘mpi_step_div’ is not identical to the number of processes.

4.4.1 PLOT3D configuration file

PLOT3D data formats are described by a PLOT3D configuration file. Here, usebytecount should be set to be 1 for Fortran and 0 for C binary data, respectively.

Parameter name	Use	Default value
coordinate_file_prefix	Prefix of coordinate file	-
coordinate_file_suffix	Suffix of coordinate file	-
coordinate_mode_precision	Precision (float | double)	double
coordinate_mode_usebytecount	1 for true, 0 for false	true
coordinate_mode_endian	Endian (little | big)	little
coordinate_mode_iblanks	1 for true, 0 for false	false
solution_file_prefix	Prefix of solution file	-
solution_file_suffix	Suffix of solution file	-
solution_mode_precision	Precision (float | double)	double
solution_mode_usebytecount	1 for true, 0 for false	true
solution_mode_endian	Endian (little | big)	little
function_file_prefix	Prefix of function file	-
function_file_suffix	Suffix of function file	-
function_mode_precision	Precision (float | double)	double
function_mode_usebytecount	1 for true, 0 for false	true
function_mode_endian	Endian (little | big)	little

4.5 MPI Parallel Processing

This section describes how the computation is divided in MPI parallel processing. As an example, consider processing data with 50 time steps and 8 sub-volumes.

1) Partitioning the set of time steps first
   - If the number of processes is equal to or less than the number of the time steps, divide the number of time steps by the number of processes.

     Example:

     If there are 8 processes exist, each process treats 6 time steps × 8 sub-volumes, or 7 time steps × 8 sub-volumes.
   - If the number of processes is larger than the number of time steps, each process handles the data for only a single time step. The number of sub-volumes for each process is specified in the following manner. First, allocate the same number of processes to each time step, and then divide the number of sub-volumes by this number of processes.

     Example:

     If128 processes are used, PBVR Filter works with 50 × 2 = 100 processes (with the residue of 28 processes), and each process treats 1 time step × 4 sub-volumes.
2) Partitioning the set of sub-volumes first
   - If the number of processes is equal to or less than the number sub-volumes, divide

the number of sub-volumes by the number of processes.

Example:

If 8 processes are used, each process treats 50 steps × 1 sub-volume.

- When the number of processes is larger than the number of sub-volumes, each process handles the data for only a single sub-volume. The number of time steps for each process is specified in the following manner. First, allocate the same number of processes to each sub-volume, and then divide the number of time steps by this number of processes.

Example:

- If 128 processes are used, then 16 processes are used for each sub-volume, giving 8 × 16 = 128 processes (with a residue of 0 processes), and each process treats 3 times steps × 1 sub-volume or 2 time steps × 1 sub-volume.

3) Employing a parallelization that is more complex

- If the parallel parameters mpi_volume_div and mpi_step_div are both specified, an error occurs if mpi_volume_div × mpi_step_div is not equal to the number of processes.

4.6 Execution in the Staging Environment of the K computer

This section describes how to execute PBVR Filter in the staging environment of the K computer. When launching PBVR Filter, the parameter file and staging parameters must be consistent with each other. Depending on the output data format of PBVR Filter, multiple processes can write to a single file. In this case, the output location should be a shared domain on the local file system that is accessible to all processes.

4.6.1 Execution Shell Script and Parameter File

```
#!/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"                                  #Use rank directory
#PJM --stgin  "rank=* ./filter              %r:./"        #Stage  in  for  load
module·····①
#PJM --stgin  "rank=* ./param.txt           %r:./"        #Stage in for file·····
·····②
#PJM --stgin  "rank=0 /data/ucd/ucd*.dat    0:../"        #Stage  in  for  shared
file·····③
#PJM --stgout "rank=* %r:../output*.dat        ./"        #Stage out for resulting
file·····④
#PJM --stgout "rank=* %r:./pbvr_filter.*  ./LOG/"         #Stage  out  for  file·····
·····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. Transfer the load module to the rank directory of each process.
2. Transfer a parameter file to the rank directory of each process.
3. Transfer input data to the shared domain in the local file system.
4. Transfer output data from the shared domain to a directory in the global file system.
5. Transfer log and error files from the rank directory to a directory in the global file system.
6. When launching the load module in the rank directory of each process, specify the parameter file (which lies in the rank directory of each process) in the command line argument.

```
#
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. Specify the path for input data files. (The path should be provided as a relative path. The above sample reads input data from a shared domain.)
8. Specify the path for output data file. (The path should be given as a relative path. The above sample writes output data to a rank directory for each process by using of SPLIT file format.)
9. Specify an output file format. (The above sample uses the SPLIT format.)

4.6.2 Input/Output Files and Directories

This section describes the relation between input and output files for PBVR Filter and the directories in the staging environment. Output data in SPLIT format can be written in a rank directory, while output data in the other formats require a shared directory for data aggregation.

Table4.6-1 Table of input and output files and directories on the K computer

I/O	File type		Rank directory	Shared domain
Input	Parameter file		Yes *1	Yes
	Input data		Yes *2	Yes
Output	Output data	SPLIT format	Yes	Yes
		Step aggregate format	No	Yes
		Sub-volume aggregate format	No	Yes
	Log & error file		Yes *3	No

*1. The parameter file is read only from rank 0.
*2. The size and number of the input files should not exceed the resource limits of the staging environment (800 files/node and 14GB/node).
*3. The output directory is always a rank directory.

4.7 Unstructured Grid Data with Mixed Elements

When unstructured grid data contains several element types, PBVR Filter first generates UCD binary data blocks for each element type. These blocks are then divided into sub-volumes,

which are read by PBVR Server. By setting the parameter multi_element_type to 1 in the parameter file, PBVR Filter produces a sub-volume for each element type.

```
#
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
```

Output files are generated for each element type. The file names have a two-digit prefix that represents the element type. The following list shows the prefix for each element type.

Table4.7-1 List of element types

Element name	Element type code
Triangle Linear	2
Quadrilateral Linear	3
Tetrahedron Linear	4
Pyramid	5
Prism	6
Hexahedron Linear	7
Triangle Quadratic	9
Quadrilateral Quadratic	10
Tetrahedral Quadratic	11
Hexahedral Quadratic	14

For example, for the above parameter file, if the input data consist of linear tetrahedral elements and quadratic tetrahedral elements, the following output files are generated.

Table4.7-2 File names for mixed elements

Original mixed elements data		Linear tetrahedral data	Quadratic tetrahedral data
MULTI001.dat		04-div001_-	11-div001_-
MULTI002.dat		04-div002_-	11-div002_-
MULTI003.dat	⇒	04-div003_-	11-div003_-
MULTI004.dat	Decompose	04-div004_-	11-div004_-
MULTI005.dat		04-div005_-	11-div005_-
:		:	:
MULTI020.dat		04-div020_-	11-div020_-

5 PBVR Server

In PBVR, the volume data is converted to the particle data, and the particle data is projected onto the screen to create the visualization result. PBVR Server reads sub-volume files, which are produced by PBVR Filter, and performs parallel particle generation of the particle data. The particle data is transferred to PBVR Client via socket communication.

5.1 Launching PBVR Server

PBVR can run on supercomputers to generate particle data either in batch mode or in client–server mode. In client–server mode, the interactive processing is realized by connecting PBVR Client and PBVR Server via a socket communication. Stand-alone processing on PCs or workstations is also possible by launching PBVR Client and PBVR Server in client–server mode on the same machine. The following example shows how to launch PBVR Server:

Examples:

Launch the MPI+OpenMP version, and use *N* processes

```
$ mpiexec -n N pbvr_server
```

Launch the OpenMP version

```
$ pbvr_server
```

*1. Since the MPI+OpenMP version of PBVR Server operates with master-slave MPI processing, the number of processes *N* should be specified by the number of slave processes + 1.

*2. In both processing modes, the number of OpenMP threads is set with environment parameter OMP_NUM_THREADS.

*3. In Windows, these commands should be launched from Visual Studio 2013 x64 Native Tools command prompt.

The method of socket communication between the remote and the user PC is described in (5.2), the launch of the client program is described in (6.1).

Table5.1-1 List of command line options for the PBVR Server program

Option	Launch mode *1	Possible values	Default values	Functionality
-h	CS,B	-	-	This shows the list of available options and parameters
-B	B	-	-	To launch in the batch mode
-pa	B	File name	-	Visualization parameter file *2
-pd	B	Real number	1.0	Particle density
-S	B	u, m, r	u	Method for sampling particles u: uniform sampling m: metropolis sampling r: rejection sampling
-plimit	B	1-99999999	1000000	Maximum number of particles
-vin	B	File name	-	Input volume data (a PFI or PFL file) *3
-pout	B	File name	./	Name of the output particle data file *3
-p	CS	Port number	60000	Port number for socket communication
-viewer	B	100-9999 ×100-9999	620×620	Viewer resolution
-Bd	B	-	-	Create particle files separately without aggregating the sub-volumes
-Bs	B	Integer of 0 or more	First step of specified PFI file group	First time step for visualizing
-Be	B	Integer of 0 or more	Last step of specified PFI file group	Last time step for visualizing

*1. In launch mode, CS and B denote client-server mode and batch mode, respectively.

*2. The visualization parameter file is explained at 5.1.1.

*3. This option specify a relative of an absolute path of PFI file generated for filtered volume data or PFL file (5.1.1.1) in distributed processing. Do not omit the file extension. If this option conflicts with the option in the parameter file specified with '−pa', the latter is ignored.

*4. This generates a set of particle data files with names "[file name]_[time step]_[number of sub-volumes]_[sub-volume index].kvsml,"

where [file name] is the prefix specified with this option. If the prefix is omitted, the prefix 'server' will be inserted automatically.

5.1.1 Launching PBVR Server in Batch Mode

For command line option '-B' is given, PBVR Server is launched in batch mode. The following example shows how to launch PBVR Server in the batch mode (for the MPI+OpenMP version).

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

In this example, the input data file *./data/case.pfi* is processed with the visualization parameter file *./param.in* to output the following particle data.

```
./output/case_XXXXX_YYYYYYY_ZZZZZZZ.kvsml
```

Here

XXXXX	: Number of steps (5 digit)
YYYYYYY	: Index for the sub-volume (7 digit)
ZZZZZZZ	: Total number of Sub-volumes (7 digit)

Usually, all the sub-volumes for each time step are integrated, so both YYYYYYY and ZZZZZZZ are 1. If you want to output particle data for each of the sub-volumes without the integration, command line option -Bd must be specified when starting the server in batch mode. The visualization parameter file is specified with the command line option -pa. This file is generated in client–server mode interactively. Large-scale data processing in batch mode is executed using this file as is or with relevant values for the parameters.

5.1.1.1 Processing of Distributed Files

For visualization, this system integrates multiple volume data files from distributed environments. The volume data files are filtered one by one. One or more PFI files are generated for each volume data file. If two or more PFI files are needed for an input volume data file, then their files names are defined in a PFL file. The PFL file is specified with the command line option "-vin".

The first line of a PFL line is "#PBVR PFI FILES". The PFI file names are written from the second line, using the absolute path or relative path of the PFL file. The following example shows the contents of a PFL file:

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

5.1.2 Launching PBVR Server in Client-Server Mode

If the command line option “-B” is not specified, PBVR Server is launched in client–server mode, as in this example:

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

When “Waiting for connection” appears, as in the above example, and PBVR Server is waiting for socket communications with PBVR Client, launch PBVR Client in another terminal. In client–server mode, the input volume data name should be given to PBVR Client rather than to PBVR Server.
The default port number for the socket communication is 60000. To change the port number, use the command line option ‘-p’:

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

5.2 Connecting Client and Server via Socket Communication

The client program and the server program perform socket communication using the port connected by port forwarding, and send and receive particle data and visualization parameters. This section describes about the port forwarding using ssh.

5.2.1 Local Connection

The following example shows how to launch both PBVR Client and PBVR Server on a single machine ‘machineA’. In this example, they cooperate using the default port number 60000 of ‘machineA’.

Step 1 [Launch PBVR Server]
```
machineA> mpiexec -n 5 pbvr_server
```
Step 2 [Launch PBVR Client]
```
machineA> pbvr_client -vin filename
```

5.2.2 Remote Connection

This section shows an example in which the local machine (machineA) and remote machine (machineB) are connected by ssh port forwarding, and the client program is started on

machineA and the server program is started on machineB. Basically, the startup method after connecting ssh port forwarding is the same as standalone.
ssh port forwarding is a mechanism to transfer the port on the network using the route communicated by ssh called ssh tunnel. There are three types of port forward: local forward, remote forward, and dynamic forward. In this system, local forward is used to connect the port of the client PC to the target via the ssh server. The outline and command of ssh local forward are shown below.

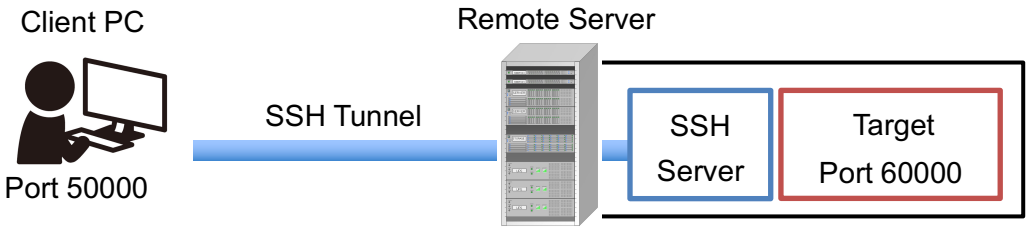


Figure 5.2-1 Abstract of local forward

[Command of SSH local forward]
ClientPC> ssh -L 50000:{Target Address}:60000 {Remote Server Address}

In the ssh local forward command, if the remote server and the target match, the target address will be “localhost”. Also, any port number can be used except the port numbers assigned to TCP and UDP, and in this example, the port numbers 50000 and 60000 are used.
The following shows an example in which port 50000 of machineA is connected to port 60000 of machineB using ssh port forwarding and the client program and server program are started on each machine.

Step1 [SSH port forwarding]
machineA> ssh -L 50000:localhost:60000 username@machineB
（The 50000 port of machineA is forwarded to the 60000 port of machineB. Since machintB itself is the target, the target address is localhost.）

Step2 [Launch PBVR Server]
machineB> mpiexec –n 5 pbvr_server

Step3 [Launch PBVR Client]
machineA> pbvr_client –vin filename

The following shows an example in which a user enters an interactive node interactB from a login node of machineB and connects port 50000 of machineA to port 60000 of interactB.

[SSH port forwarding]
machineA> ssh -L 50000:interactB:60000 username@machineB

5.2.3 Testing SSH Port Forwarding Connection

To check if SSH port forwarding is available, use the following test program, which simply transfers characters input from PBVR Server to PBVR Client. This program is available from the link below.

"C for Linux 2" Mitsuyuki Komata, SYUWA System, Inc., September 2005 (Japanese).
http://www.ncad.co.jp/~komata/c4linux2/

Launch PBVR Server
server port_number

Launch PBVR Client
client server_hostname port_number

5.2.4 Remote Connection from SSH Client

This section shows a port forwarding (local forward) using SSH client software Tera Term in Windows environment.

1) Launch TeraTerm and hit cancel in the "New connection" dialog.

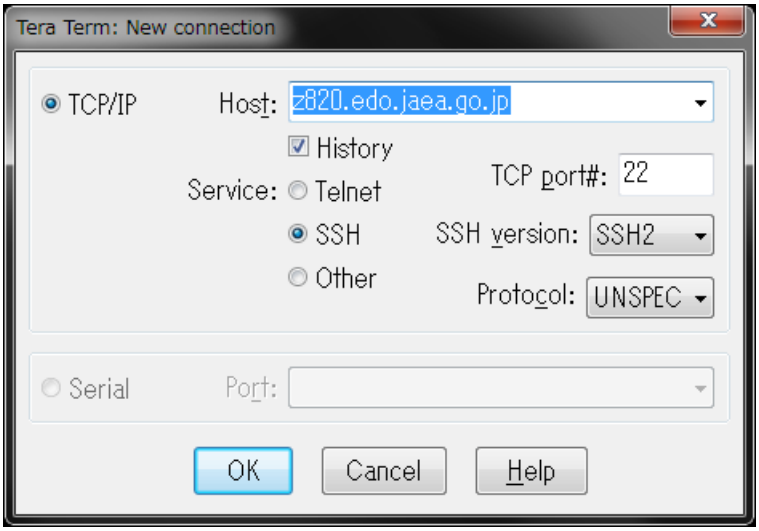


Figure5.2-2 Tera Term dialog 1)

2) Select **Setup** > **SSH Transfer** from the menu bar. Click **Add…** in the **Forwarding Setup** dialog. Check the dialogue box "Display remote X application on local X server".

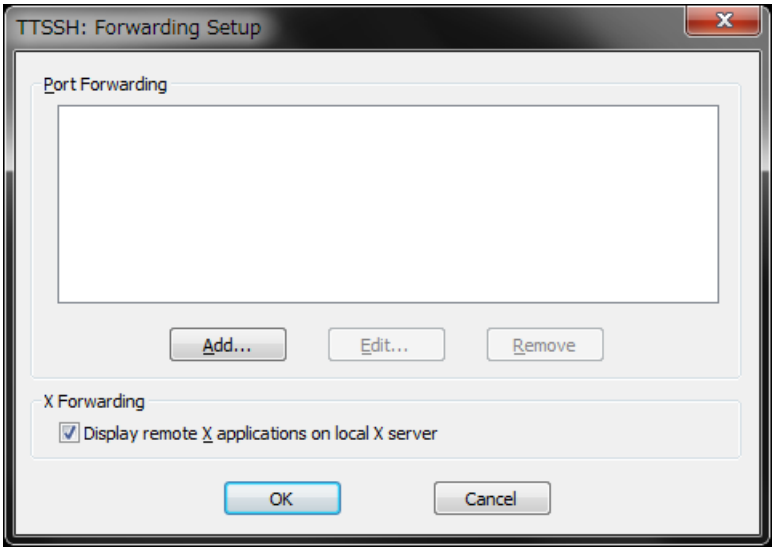


Figure5.2-3 Tera Term dialog 2)

3) In the **Select Direction for Forwarded Port** dialog, select **Forward Local Port** (it means the local forwarding) and enter the port number to be used for PBVR Client, which is corresponds to the port 50000 in Figure 5.2-1. In the **to remote machine** text field, enter the domain name or the IP address of the server, which is corresponds to the target address in Figure 5.2-1. In the **port** field, enter the port number to be used on PBVR Server, which is corresponds to the port 60000 in Figure 5.2-1. Click on **OK** to complete the setup of port forwarding.

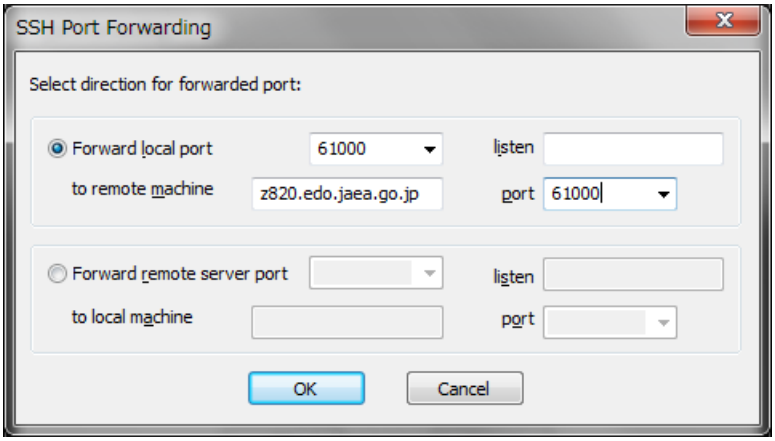


Figure5.2-4 Tera Term dialog 3)

4) Connect to the server. Select **File** > **New Connection** from the menu bar. In the **New Connection** panel, enter the host name of the serve which is corresponds to the remote server address in Figure 5.2-1, and click on **OK**. In the **SSH Authentication** panel, enter the user name and passphrase, or specify the location of the private key file, and click on **OK**.

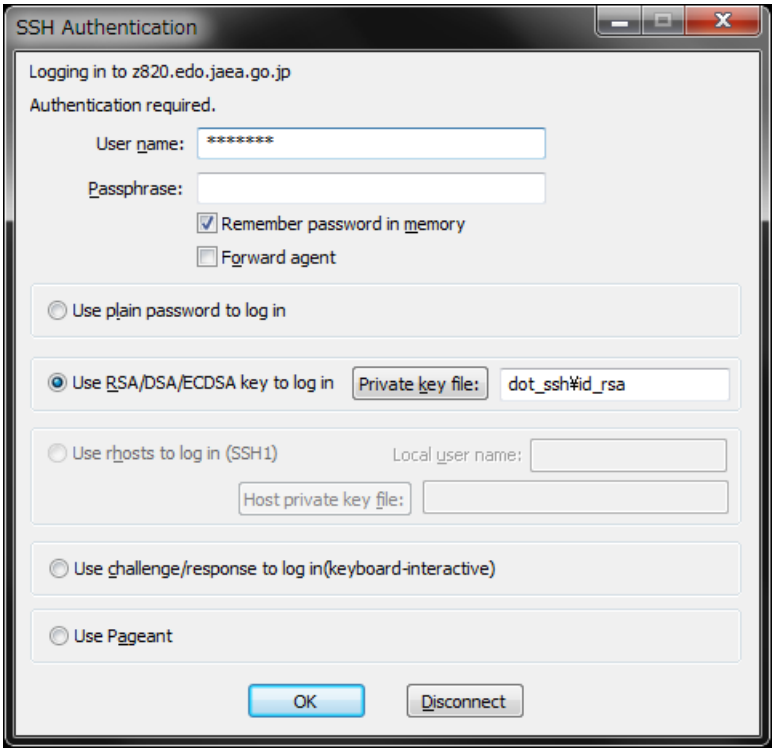


Figure5.2-5 Tera Term dialog 4)

5.3 Visualization on Front-End Server

PBVR allows not only client/server type visualization via socket communication, but also visualization on a front-end server via VNC or X Window System. Both client and server programs are started on the front-end server for the local connection (5.2.1), GUI using X Window System, or GUI using VNC desktop screen containing the GUI is transferred to the user PC.

VNC

If OSMesa and VNC server are installed on the front-end server, PBVR can run on the front-end server and be visualized on the VNC viewer. The way to use the VNC server depends on the operation of the front-end server, so please contact the administrator.

X Window System (Linux)

Since Linux supports the X Window System by default, PBVR can be used remotely by simply launching with the local connection. Login to the front-end server (machineA) with two terminals and start the client and server programs respectively.

Step1: Launch the server program on terminal 1.
UserPC:$ ssh usename@machineA
machineA:$ mpiexec –n 5 pbvr_server

Step2: Launch the client program on terminal 2. “XY” option is required for X Window System.
UserPC:$ ssh -XY usename@machineA
machineA:$ pbvr_client –vin filename

X Window System (Windows)

Installation of Xming and TereTerm is required to use X Window System on Windows.

Step1: Download and install Xming and Xming-fonts.
Step2: Set X11 forwarding for TeraTerm (5.2.4).
Step3: Launch Xming using Xlaunch
Step4: Launch TeraTerm and connect the front-end server.
The subsequent steps are the same as in the Linux case.

X Window System (Mac)

There is a X Window System application Xquartz for Mac OSX, but PBVR doesn’t support Xquartz.

6 PBVR Client

PBVR Client can operate either in client–server mode or in stand-alone mode. In client–server mode, PBVR Client receives particle data that is rendered using OpenGL based on primitives generated in PBVR Server. PBVR Client also gets visualization parameters (a transfer function etc.) via user interaction and sends the parameters to PBVR Server. In this way, PBVR Client controls the volume rendering process in PBVR Server. The data transfer between PBVR Client and PBVR Server uses a socket communication with a user-specified port number. In contrast, when PBVR is in stand-alone mode, it reads and displays particle data generated by PBVR Server operating in batch mode. Launching the server program is described in (5.1), the socket communication between the remote and the user PC is described in (5.2).

6.1 Launching PBVR Client

The following examples show how to launch the client program in client-server mode and to do so in stand-alone mode.

[Launch PBVR Client in client-server mode] *1
$ pbvr_client -vin [sub-volume file name *2] [command line options]

[Launch PBVR Client in stand-alone mode]
$ pbvr_client [particle data file name] [command line options]

*1. Client–server mode requires that PBVR Server has already been started.
*2. The file name for a sub-volume can be specified with the absolute or the relative path to the PFI file.

As with PBVR Server, if two or more PFI files are needed for an input volume data file, create a PFL file and specify it with the command line option “-vin” (Section 5.1.1.1).

Table6.1-1 List of command line options for client

Option	Launch mode *1	Parameter value	Default parameters	Function
-h	CS,SA	-	-	List of options and parameters
-pd	CS	Real number	1.0	Particle density
-S	CS	u, m, r	u	Particle sampling method u: uniform sampling m: metropolis sampling r: rejection sampling
-plimit	CS	1~99999999	1000000	Particle limit *2
-tdata	CS	all, div	all	Particle data transfer method all: step batch transmission, div: sub-volume divide forwarding）
-pa	CS,SA	File name	-	Visualization parameter file
-vin	CS	File name	-	Name of the PFI or PFL file of input volume data *2
-tf	CS	File name	-	Name of the transfer function file *3
-p	CS	Port number	60000	Port number for socket communication
-viewer	CS,SA	100-9999 ×100-9999	620×620	Viewer resolution
-shading	CS,SA	{L/P/B}, ka, kd, ks, n	-	Shading method *4
-pout	CS,SA	File name	-	Output file name for particle data *5
-pin1	SA	File name	-	Input file name for particle data
-iout	CS,SA	Directory name	./	Output directory name for image files

*1. CS and SA denote client-server mode and stand-alone mode, respectively.

*2. This option specifies the relative or absolute path of a PFI file generated for filtered volume data or a PFL file for distributed processing. Do not omit the file extension. If this option conflicts with the option in the parameter file specified with “-pa”, the latter is ignored.

*3. Transfer function files are generated by hitting the **Export File** button in the **Transfer Function Editor**. To apply the transfer function specified in this option, hit the **Apply** button in the **Transfer Function Editor**. Alternatively, the transfer function file can be loaded also with the **Import File** button.

*4. This argument specifies the shading parameters.

L: Lambert Shading
This method ignores specular reflection during shading. Parameters “ka” and “kd” are the coefficients for ambient and diffusion, respectively. They can have a value between 0 and 1.

P：Phong Shading
This method adds specular reflection to Lambert shading. Phong shading imitates smooth metal and mirrors, which is sometimes called highlighting. Parameters “ka”, “kd”, “ks” (which are the coefficients for specular reflection and have values between 0 and 1) and “n” (which is the strength of the highlighting and has values between 0 and 100) are used.

B：Blinn-Phong Shading
This is a simplified form of Phong shading. Parameters “ka”, “kd”, “ks”, and “n” are used.

*5. This generates a series of particle data files that are named “[prefix]_[time index]_[number of sub-volumes]_[sub-volume index].kvsml”, where the prefix is specified by this option.

6.2 Terminating PBVR

6.2.1 Standard Termination

PBVR Client's rendering process for the time-series data starts at the initial time step, and continues to the final time step. When the final time step is rendered, PBVR Client returns to the initial time step to loop over the steps. To terminate PBVR Client, press **Ctrl+C** in the console running PBVR.

In client-server mode, pressing **Ctrl+C** in the client console terminates both PBVR Client and PBVR Server. Just before the termination, PBVR Client and Server will synchronize their time steps. However, PBVR Client ignores pressing **Ctrl+C** whenever the client-server communications are interrupted with the **Stop** button in the **Time Panel**.

6.2.2 Forced Termination

If PBVR Server is terminated other than by pressing Ctrl+C in PBVR Client's console, PBVR Client becomes stuck and cannot be terminated with Ctrl+C. Furthermore, even if Ctrl+C is pressed to terminate PBVR Client, both PBVR Client and Server can become stuck. This can happen if the time step is not updated due to heavy particle generation processes or some other reason. In this case, obtain the process IDs of PBVR Client and PBVR Server using the ps command in the console, and then force them to quit with the kill command as follows:

```
[Force the termination of a PBVR Client process]
$ ps -C pbvr_client
   PID TTY          TIME CMD
19582 pts/6    00:00:00 pbvr_client
$ kill -9 19582

[Force the termination of a PBVR Server process】
$ ps -C pbvr_server
   PID TTY          TIME CMD
19539 pts/5    00:00:00 pbvr_server
$ kill -9 19539
```

6.3 Using the PBVR Client GUI

The client program after startup not only displays the visualization result, but also provides a GUI that can interactively control visualization parameters.

6.3.1 Viewer

As shown in Figure6.3-1, **Viewer** renders the rendering result of particle data.

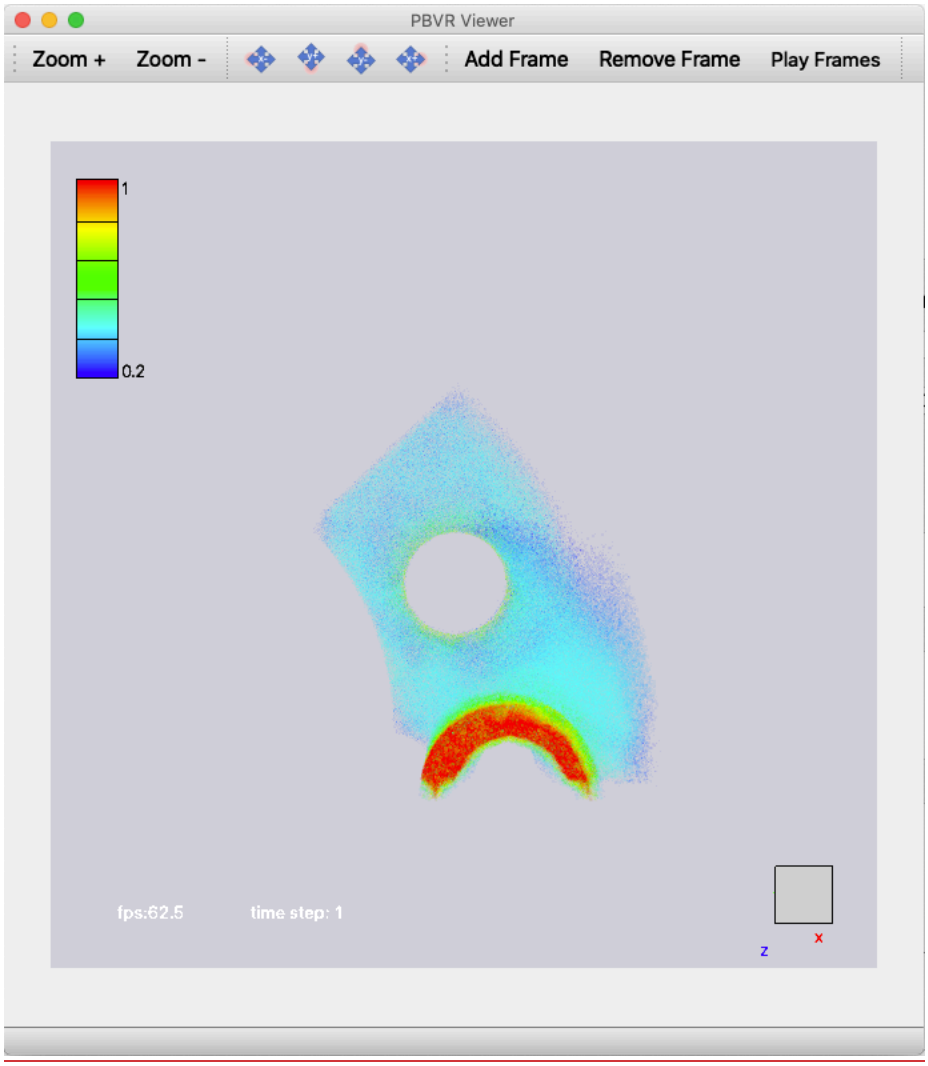


Figure6.3-1 **Viewer**

[Operations]

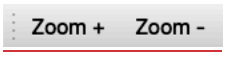
Rotation:	Dragging left with the mouse
Translation:	Dragging right with the mouse
Zoom:	Shift + left-dragging, or dragging while pressing the mouse wheel
Reset:	Home button (fn + left arrow on Macs)

[Display]

time step: Time step for the data displayed

fps: Frame rate (frame/sec)

[Tool bar]



Zoom in/out.


Translations.

Get keyframe/delete keyframe/play keyframe animation.

6.3.2 Tool Bar

Various functions of the client program are selected from the toolbar. The “File” tab controls the input and output of the visualization parameter file and transfer function.

The visualization parameter means a group of parameters that can be set by the client, such as viewer resolution, particle density, particle number limit, input volume data file name on the server (PFI file or PFL file), transfer function, etc. The visualization parameter file is a file in which they are described in the tag format.

The "File" tab and its functions are shown below.

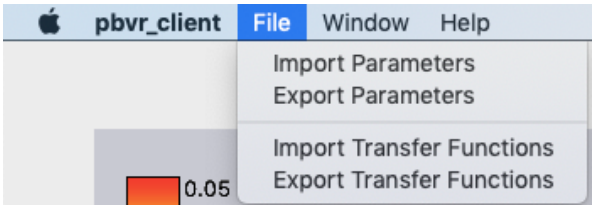


Figure 6.3-2 File tab

- **Import Parameters** specifies the visualization parameter file to be input.
- **Export Parameters** outputs the visualization parameter file.
- **Import Transfer Functions** specifies the input transfer function file.
- **Export Transfer Functions** outputs a transfer function file.

The "Window" tab and its functions are shown below.

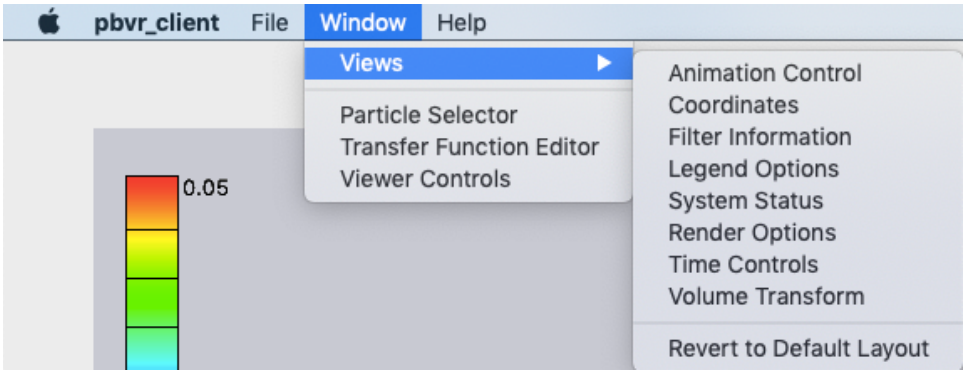


Figure 6.3-3 Window tab

- **Animation Control** displays a video creation panel.
- **Coordinates** displays a coordinates editor.
- **Filter Information** displays the input volume data property.
- **Legend Option** displays a legend panel for control the legend bar.
- **System Status** displays system property.
- **Render Options** controls rendering options.
- **Time Controls** displays a time step control panel.
- **Volume Transform** specifies geometry transformation for a rendering object.

- **Revert to Default Layout** returns the layout of GUI when this application starts.
- **Particle Selector** displays a particle panel.
- **Transfer Function Editor** displays a transfer function editor.
- **Viewer Controls** displays a viewer control panel.

6.3.2.1 FilterInfo

The Filter Info panel displays information about the input filtered volume data.

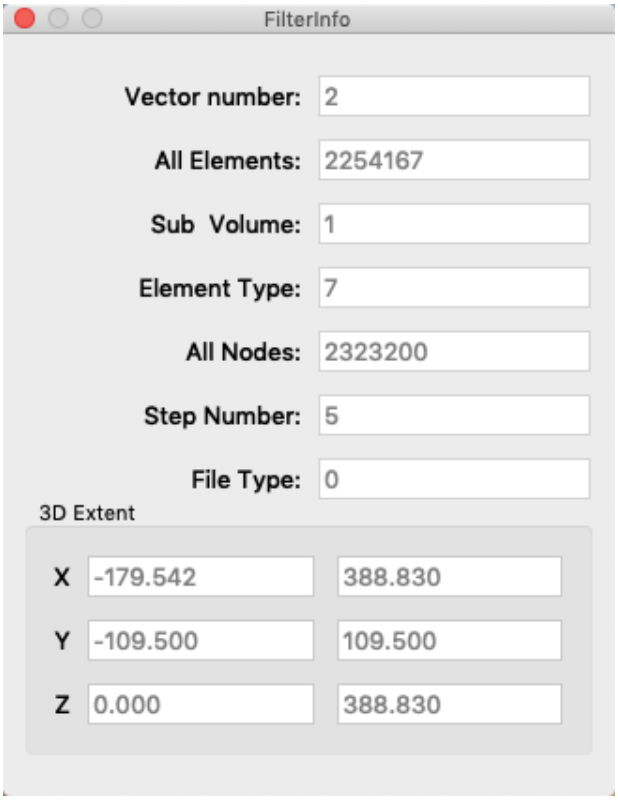


Figure 6.3-4 FilterInfo

- **Vector number** displays the number of variables consisting of the volume data.
- **All Elements** displays the number of elements consisting of the volume data.
- **Sub Volume** displays the number of sub-volume separated by the filter program.
- **Element Type** displays the type of element (Table4.7-1) consisting of the volume data.
- **All Nodes** displays the number of nodes of the volume data.
- **Step Number** displays the number of time steps.
- **File Type** displays the volume decomposition type (4.3).
- **3D Extent** displays the min-max X-Y-Z coordinates of the volume data.

System Status

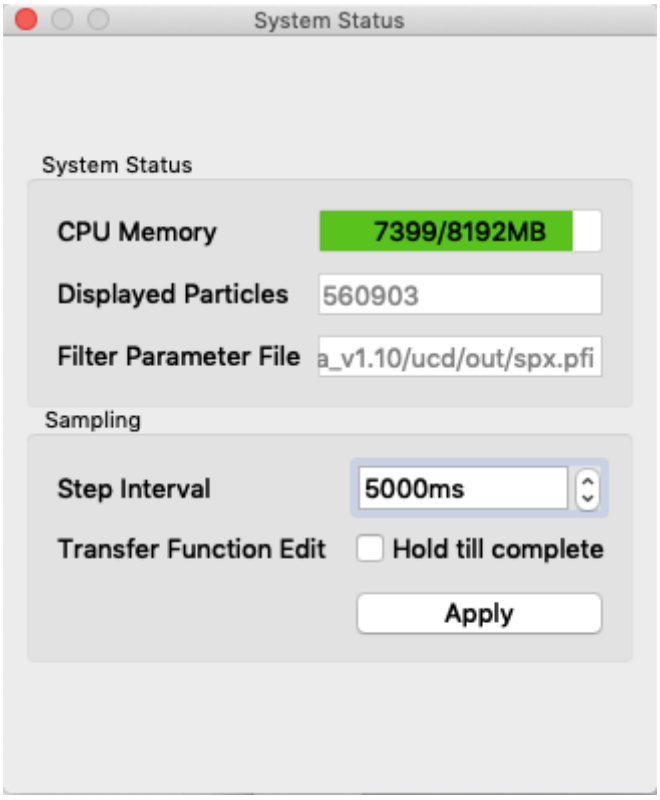


Figure 6.3-5 System status

- **CPU Memory** displays the system memory usage in megabytes.
- **Displayed Particles** displays the number of rendering particles.
- **Filter Parameter File** displays the name of input volume data.
- **Step Interval** specifies minimum of a time step update interval by [msec]. This is used to lengthen the update interval for time steps that are too short.
- **Transfer Function Edit _ Hold till complete** holds update of the transfer function till time step progress completed.

Render Options

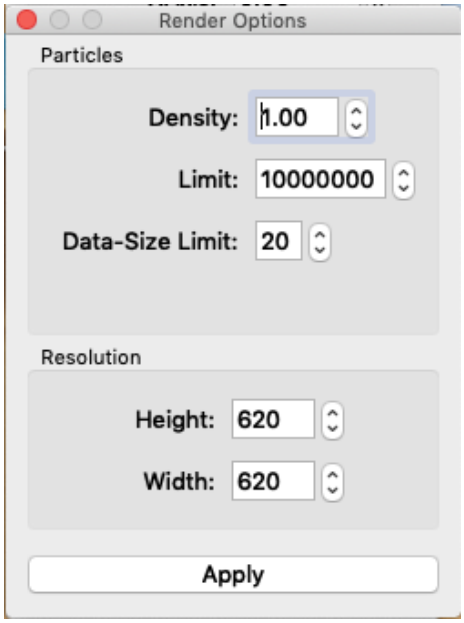


Figure 6.3-6 Render Options

Density specifies the particle density related to the depth of the image.

Limit specifies the upper limit of the number of particles generated by the server program in order to prevent the number of particles from exploding due to incorrect transfer function specification. If the number of particles exceeds this upper limit, the server program automatically reduces the image quality and adjusts so that the number of particles falls within this upper limit.

Data-Size Limit specifies the upper limit of the particle data size using [GB] generated by the server program in order to avoid explosion of the number of particles due to incorrect transfer function specification. If the particle data size exceeds this upper limit, particle generation is forced to stop.

Resolution specifies the viewer's resolution.

6.3.2.2 Volume Transform

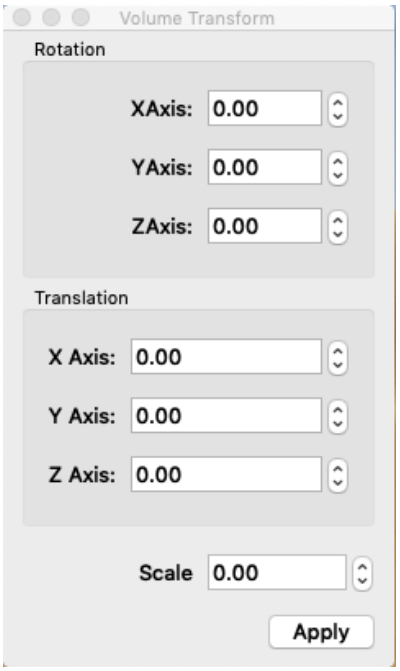


Figure 6.3-7 Volume Transform

- **Rotation** specifies the rotation angle (degree) about the x-, y-, and z-axis of the object.
- **Translation** specifies the translation of the object in the x, y and z directions.
- **Scale** specifies the scale rate of the object.

6.3.3 Transfer Function Editor

The transfer function editor enables to create transfer functions which assign the colors and opacity to physical values. In a standard rendering of a volume, a transfer function is defined with only one physical quantity. In contrast, PBVR allows for multi-dimensional transfer functions. It has the following three features:

1) It assigns a parameter to color and, independently, a parameter to opacity.
2) It defines both color and opacity with an arbitrary function of the *X*-, *Y*-, and *Z*-coordinates and parameters *q1*, *q2*, *q3,* ….
3) It algebraically synthesizes a multi-dimensional transfer function from one-dimensional transfer functions that are defined by color functions *C*1, *C*2, … and opacity functions *O*1, *O*2, ….

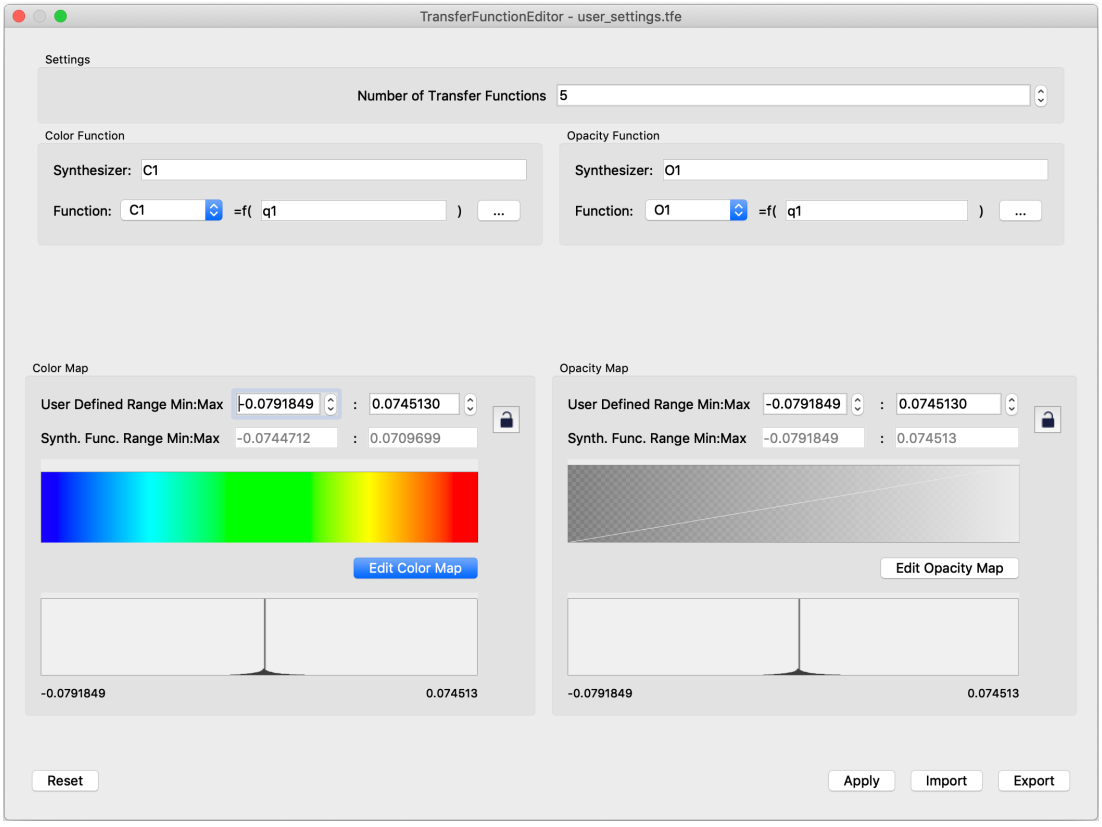


Figure 6.3-8 Transfer Function Editor

[Operations]

Scale change in histogram: Drag the mouse up/down in the histogram.

- **Number Transfer Functions** specifies the maximum number of transfer functions that can be created.
- **Color Function** category specifies the color functions and its argument which is synthesized

by the physical values.

- **Opacity Function** category specifies the opacity functions and its argument which is synthesized by the physical values.
- **Color Map** category edits the color functions.
- **Opacity Map** category edits the opacity functions.
- **Reset** button returns the transfer functions to initial state.
- **Apply** button sends the transfer function to the server program.
- **Export** button saves the transfer function file with same format of “-pa” option.
- **Import** button reads the transfer function file.

[Available operators for algebraic equations]
+, -, *, /, ^, sin(), cos(), tan(), sqrt(), log(), exp()

6.3.3.1 Color Map Specification

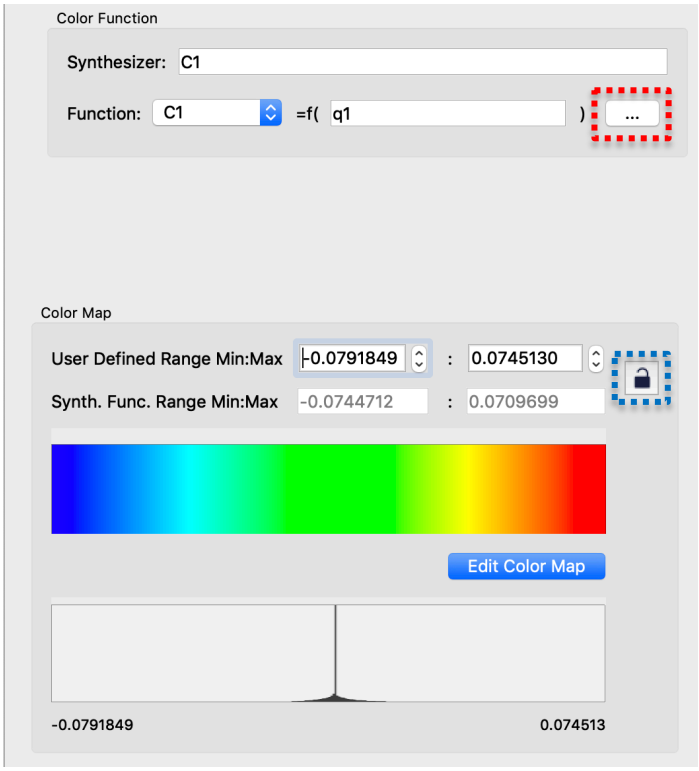


Figure 6.3-9 Top: Color Function category, bottom: Color Map category

[Color Function category]

- **Synthesizer** specifies the synthesis of the color function of C1 to C [N]. *1
- **Function** spin button selects the color function of C1 to C [N] for edit its argument.
- The **button surrounded by red dotted line** opens Color Function Editor which enables to synthesize the physical values as an argument of C1 to C [N].

[Color Map category]

- **User Defined Range Min:Max** specifies the minimum and maximum range of the color function which is assigned to the synthesized physical value.
- **Synth. Func. Range Min:Max** displays the minimum and maximum values of the synthesized physical value.
- **Edit Color Map** button opens Color Map Editor.
- **Histogram** displays a distribution of the synthesized physical value with the user defined range min:max.
- If the **Lock** button surrounded by blue dotted line is pressed, it is prohibited to input for the user defined range min:max and forced to apply the value of the synth. func. range min:max.

*1 [N] is the value of the limit number of the transfer function specified by Number of Transfer Functions.

6.3.3.1.1 . Color Function Editor

PBVR Client can define the arguments of the color functions C1 to C [N] using the algebraic formula. In the Color Function Editor, the algebraic formula is synthesized by the physical values. The names of the variables that can be used in the algebraic formula are shown below.

- Physical values：q1, q2, q3, .., qn.
- Coordinate values：X, Y, Z.

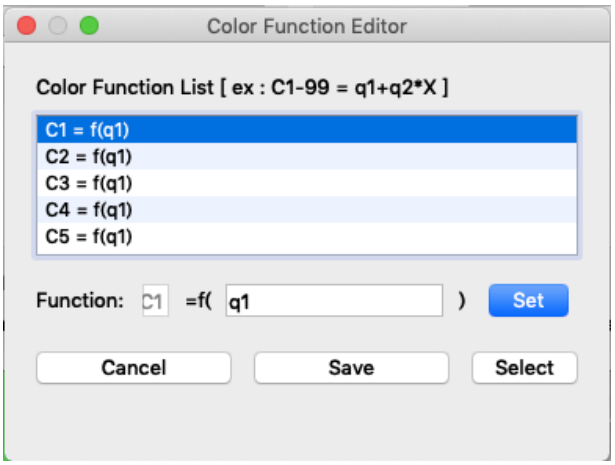


Figure 6.3-10 Color Function Editor

- **Color Function List** displays the created color functions.
- **Function: C[N] = f(algebraic formula)** displays a color function selected in Color Function List and enables to edit the algebraic formula.
- **Cancel** button close this panel.
- **Set** button applies the edited algebraic formula to Color Function List.
- **Save** button applies the Color Function List to the transfer function
- **Select** button applies a color function of Color Function List to Function.

6.3.3.1.2. Color Function Editor: Freeform Curve Edit Tab

In the Freeform Curve Edit tab of Color Map Editor, the color function can be created by the free form curve.

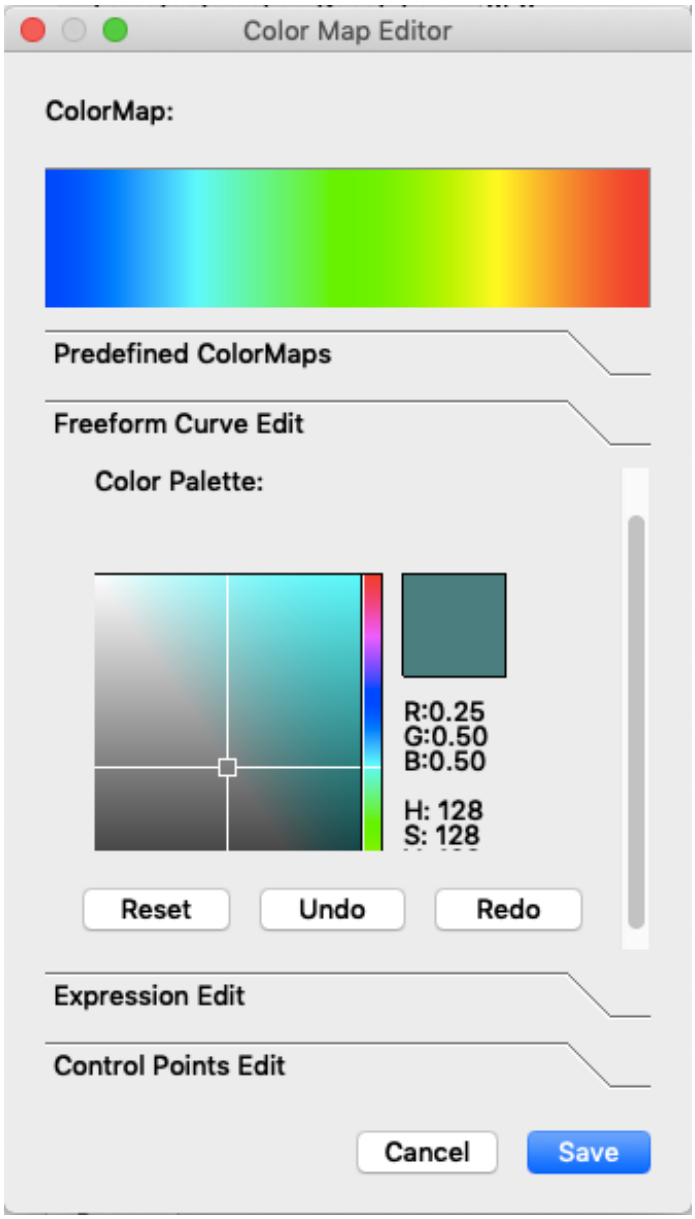


Figure 6.3-11 Color Map Editor: freeform curve tab

- **Color Palette** consists of saturation on the horizontal axis and brightness on the vertical axis, and these can be specified by the position of the cursor.
- The right side of the color palette is **RGB bar** and enables to specify the hue. Color Palette reflects the hue specified in RGB bar.
- **Reset** button returns this tab to default state.

- **Undo** button undo single action.
- **Redo** button redo single action.
- **Save** button applies created color function.
- **Cancel** button close this panel.

Blends the single color specified by the **Color palette** and the **RGB bar** with those from a standard spectrum. This is done by dragging the mouse along the color box while pressing the left mouse button. The blending ratio between the specified color and the color from the standard spectrum is determined by the vertical position of the mouse within the box. For example, if the mouse is traced across the upper edge of the color box from left to right, it is painted completely by the specified color. If the mouse is traced across the middle of the color box, the colors in the box are replaced with blended colors that are 50% of the original color and 50% of the specified color.

6.3.3.1.3. Color Map Editor: Expression Tab

In Expression Edit tab of Color Map Editor, the color functions can be created by algebraic expressions.

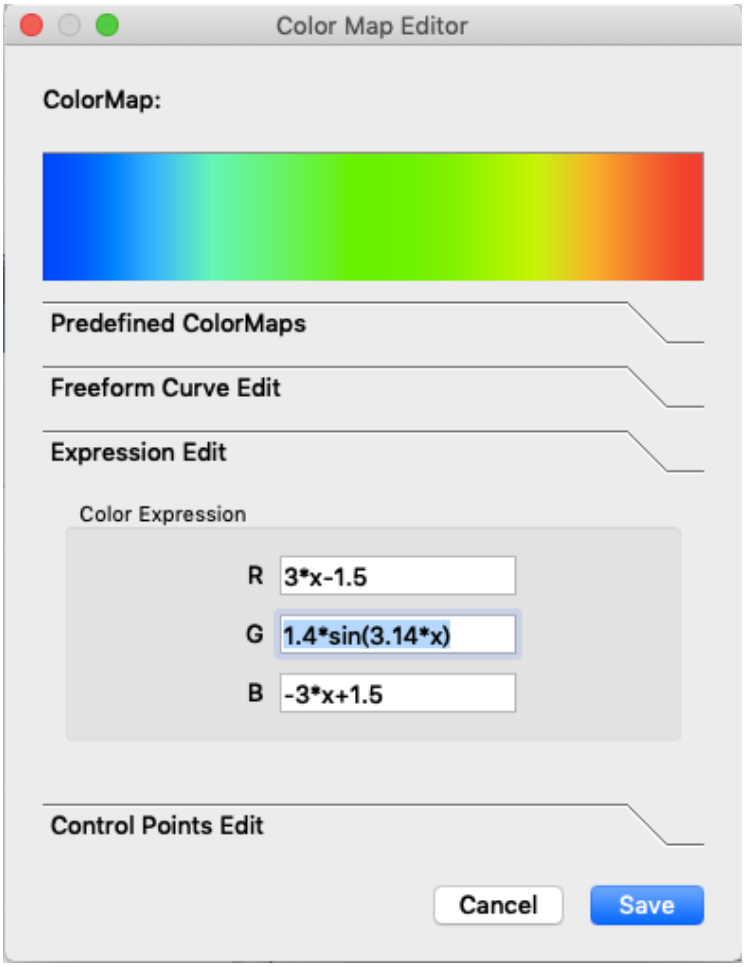


Figure 6.3-12 Color Map Editor: Expression tab

- **R** describes the color function of the red component of the color by algebraic expression.
- **G** describes the color function of the green component of the color by algebraic expression.
- **B** describes the color function of the blue component of the color by algebraic expression.

A variable of the color function is x, and the domain and range of the color function are 0 to 1.

6.3.3.1.4. Color Map Editor: Control Points Edit Tab

In Control Points Edit tab of Color Map Editor, the color functions can be created by control points.

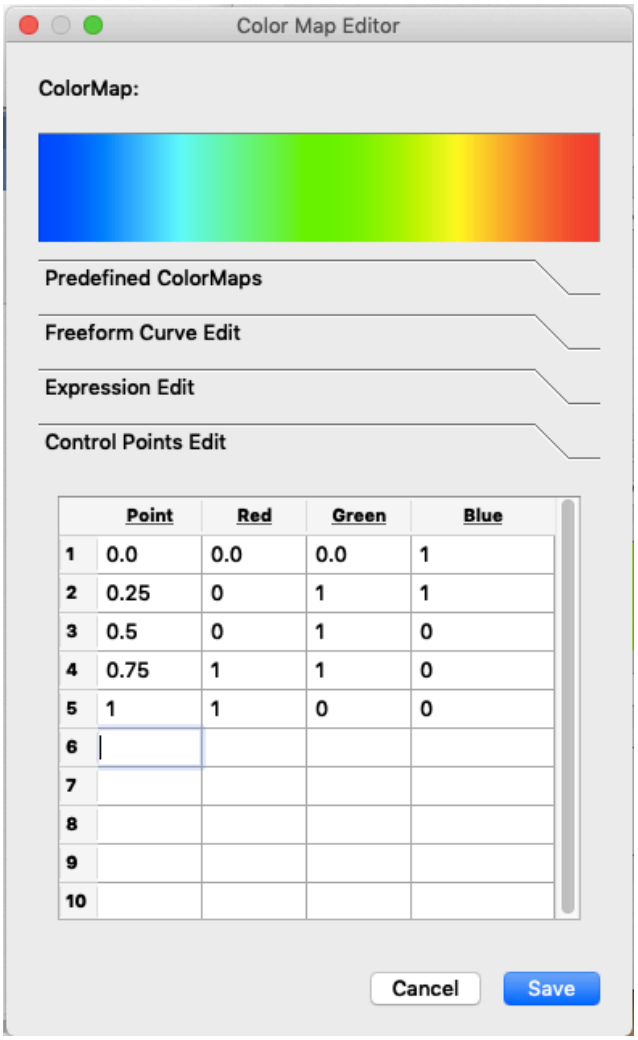


Figure 6.3-13 Color Map Editor: Control Points Edit Tab

- **Point** enables to specify the values of control points (up to 10). The domain is 0 to 1.
- **Red** enables to specify a red component value corresponding to the control point. The range is 0 to 1.
- **Green** enables to specify a green component value corresponding to the control point. The range is 0 to 1.
- **Blue** enables to specify a blue component value corresponding to the control point. The range is 0 to 1.

Each control point is interpolated with a piecewise linear function.

6.3.3.1.5. Color Map Editor: Predefined ColorMaps Tab

Predefined Color Maps tab of Color Map Editor provides the selection of predefined color functions.

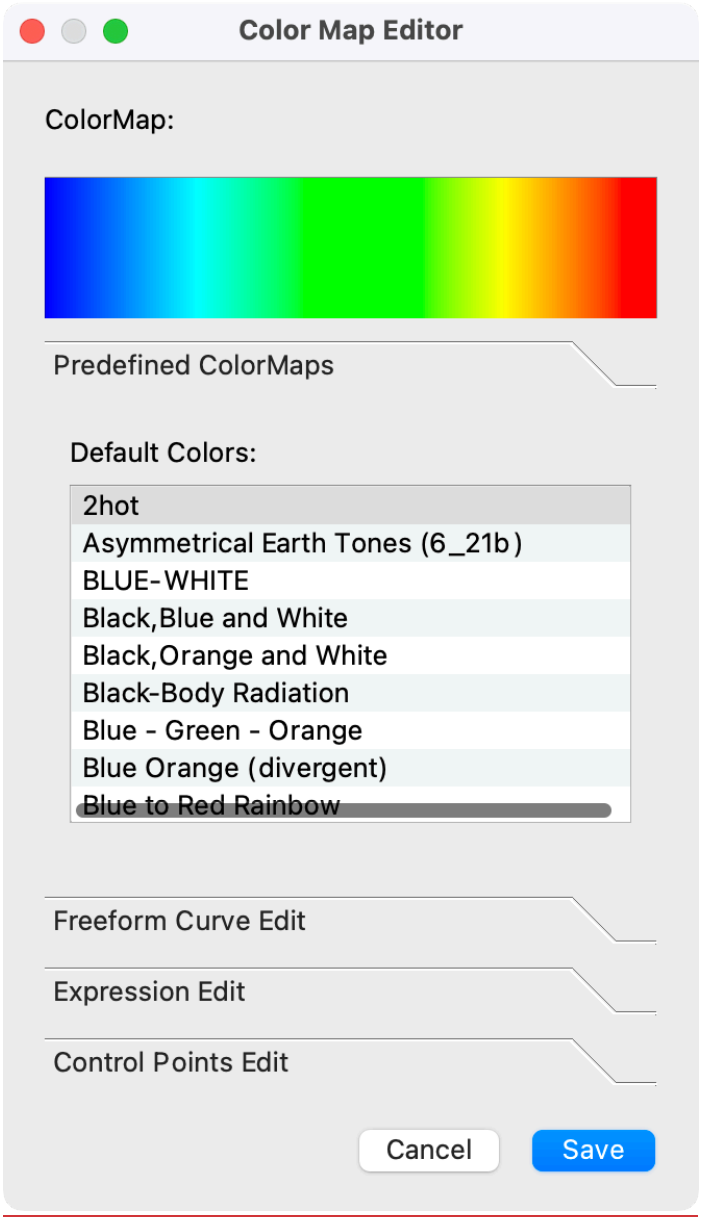


Figure 6.3-14 Color Map Editor: Predefined ColorMaps Tab

Default Colors: displays the list of the predefined color maps. The following templates are available.

- 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 Opacity Map Specification

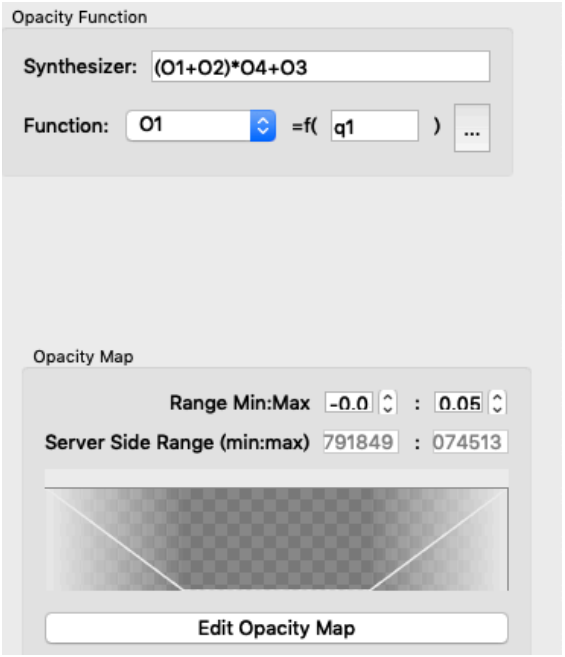


Figure 6.3-15 Top: Opacity Function category, bottom: Opacity Map category

[Opacity Function category]

- **Synthesizer** specifies the synthesis of the opacity function of O1 to O[N]. *1
- **Function** spin button selects the opacity function of O1 to O[N] for edit its argument.
- The **button surrounded by red dotted line** opens Opacity Function Editor which enables to synthesize the physical values as an argument of O1 to O[N].

[Opacity Map category]

- **Range Min:Max** specifies the minimum and maximum range of O1 to O[N].
- **Server Side Range (min:max)** displays the minimum and maximum values of the synthesized physical value
- **Edit Color Map** button opens Opacity Map Editor.

*1 [N] is the value of the limit number of the transfer function specified by Number of Transfer Functions.

6.3.3.2. 1 . Opacity Function Editor

PBVR Client can define the arguments of the opacity functions O1 to O [N] using the algebraic formula. In the Opacity Function Editor, the algebraic formula is synthesized by the physical values. The names of the variables that can be used in the algebraic formula are shown below.

- Physical values：q1, q2, q3, .., qn.
- Coordinate values：X, Y, Z.

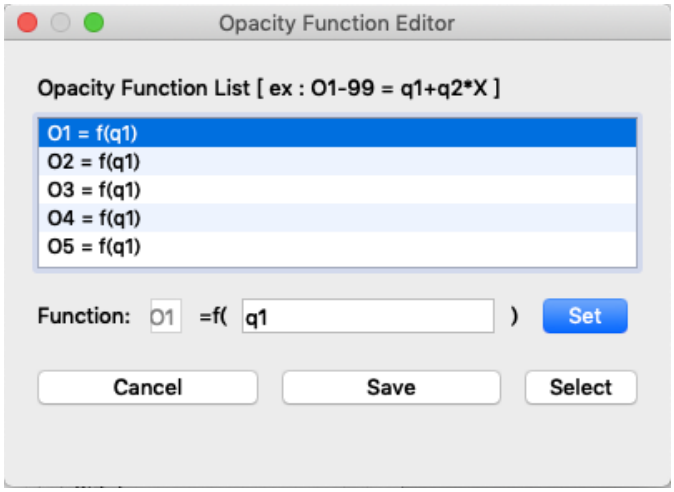


Figure 6.3-16　Opacity Function Editor

- **Opacity Function List** displays the created color functions.
- **Function: O[N] = f(algebraic formula)** displays a color function selected in Opacity Function List and enables to edit the algebraic formula.
- **Cancel** button close this panel.
- **Set** button applies the edited algebraic formula to Opacity Function List.
- **Save** button applies the Opacity Function List to the transfer function
- **Select** button applies a color function of Opacity Function List to Function.

6.3.3.2.２. Opacity Map Editor: Freeform Curve Tab

In the Freeform Curve Edit tab of Opacity Map Editor, the opacity function can be created by the free form curve.

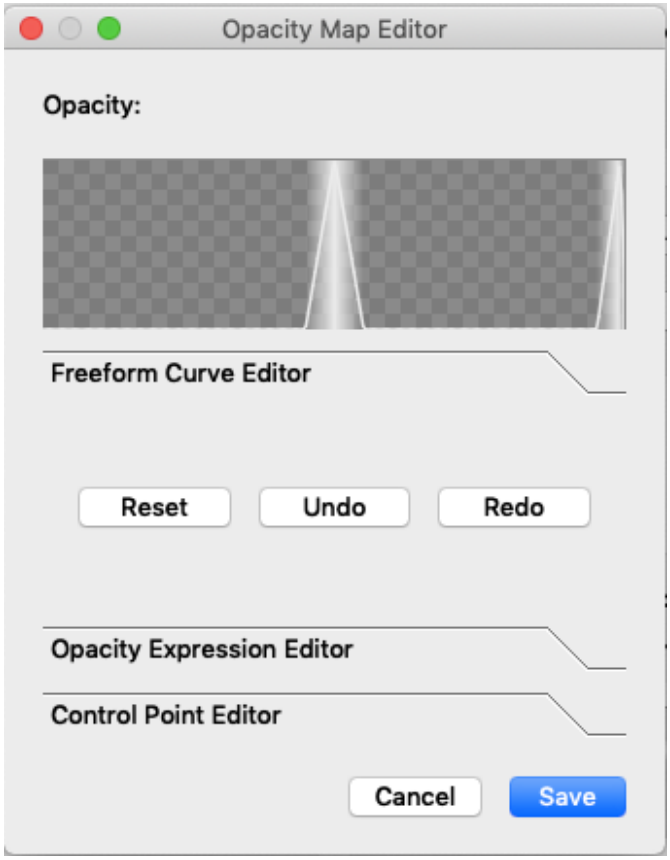


Figure 6.3-17 Opacity Map Editor: Freeform Curve Tab

- **Reset** button returns this tab to default state.
- **Undo** button undo single action.
- **Redo** button redo single action.
- **Save** button applies created color function.
- **Cancel** button close this panel.

In drawing the opacity function, a free-form curve can be drawn by left-dragging with the mouse and a linear interpolation line between two points can be drawn by right-click.

6.3.3.2.３. Opacity Map Editor: Opacity Expression Tab

In Expression Edit tab of Opacity Map Editor, the opacity functions can be created by algebraic expressions.

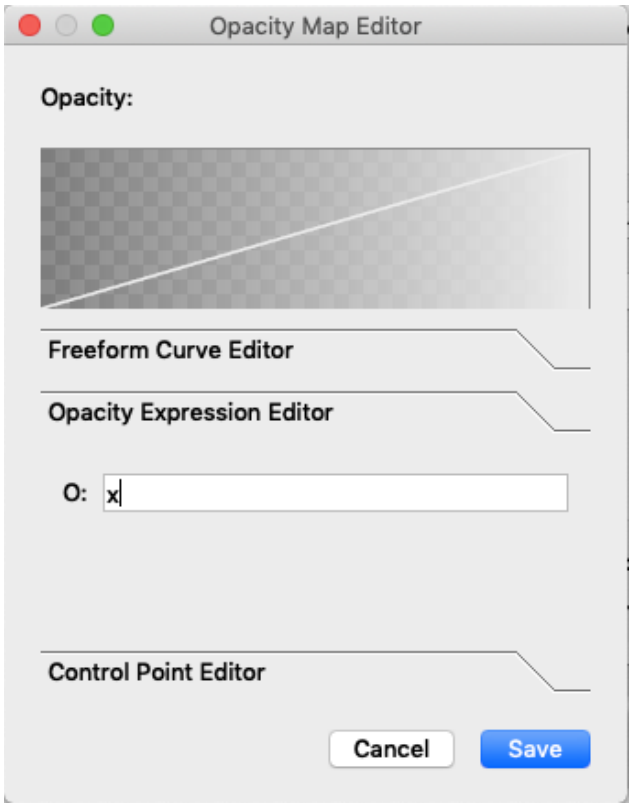


Figure 6.3-18 Opacity Map Editor: Opacity Expression Tab

- O: describes the color function of the blue component of the color by algebraic expression.

A variable of the opacity function is x, and the domain and range of the opacity function are 0 to 1.

6.3.3.2.4. Opacity Map Editor: Control Point Editor Tab

In Control Points Edit tab of Opacity Map Editor, the opacity functions can be created by control points.

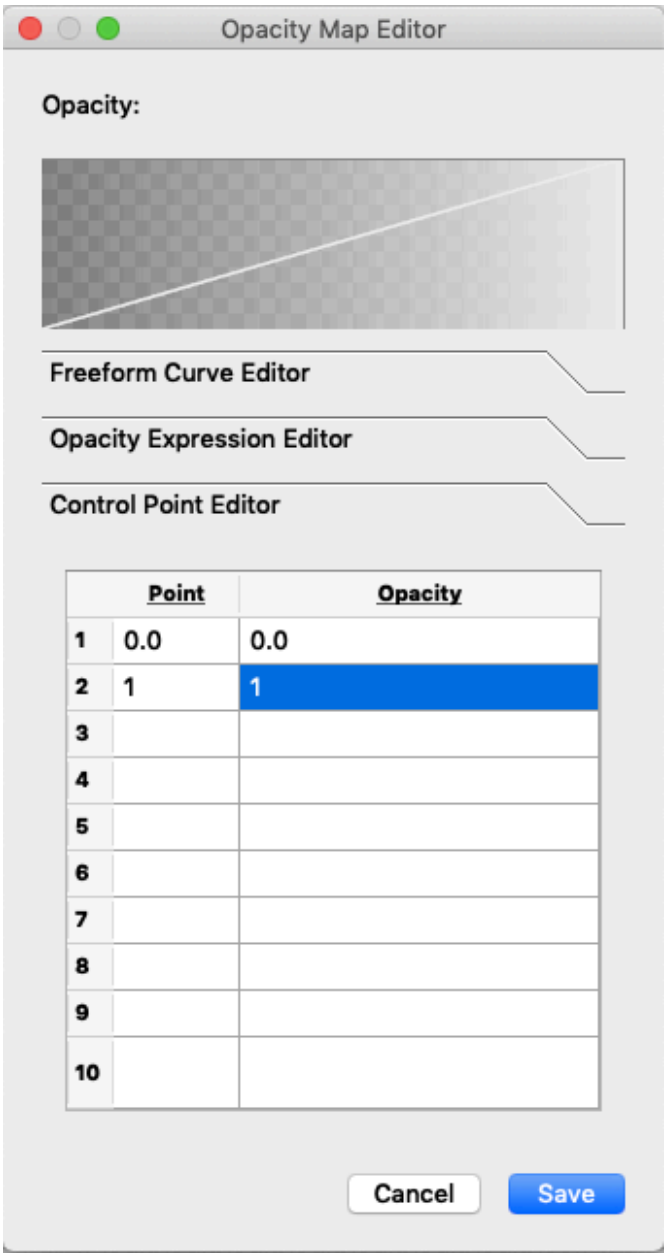


Figure 6.3-19 Opacity Map Editor: Control Point Editor Tab

- **Point** enables to specify the values of control points (up to 10). The domain is 0 to 1.
- **Opacity** enables to specify the opacity value corresponding to the control point. The range is 0 to 1.

Each control point is interpolated with a piecewise linear function.

6.3.3.3 Function Editor

Table 6.3-1 lists the built-in math operations available in the function editors. They can be used for the overall transfer functions, the parameters, and for the color and opacity curves.

Table 6.3-1 Math operations available in function editors

Math operation	**In function editors**
+	+
-	-
×	*
/	/
sin	sin(x)
cos	cos(x)
tan	tan(x)
log	log(x)
exp	exp(x)
square root	sqrt(x)
power	x^y

When arithmetic processing by the function editor produces NaN, PBVR outputs an error message and stops the drawing process.

6.3.4 Time panel

Figure6.3-20 shows **Time panel**, which specifies the time steps for the visualization. Each widget works as described below.

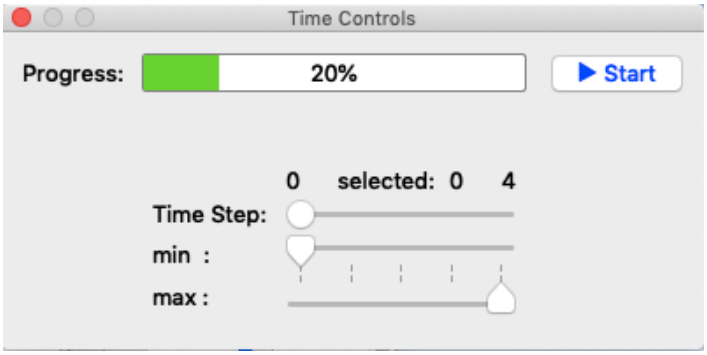


Figure6.3-20 Time panel

- **Progress** expresses the current time step as a percentage.
- **Time step** specifies the time step of the data to be rendered.
- **Min Time** specifies the minimum time step for the ROI.
- **Max Time** specifies the maximum time step for the ROI.
- **Start/Stop** starts/stops communications between PBVR Client and PBVR Server.

6.3.5 Particle and Polygon Composition

CS-PBVR supports composition of the volume rendering and polygon rendering. Figure6.3-21 shows **Particle panel**, which is used to integrate multiple particle datasets. Each widget works as described below.

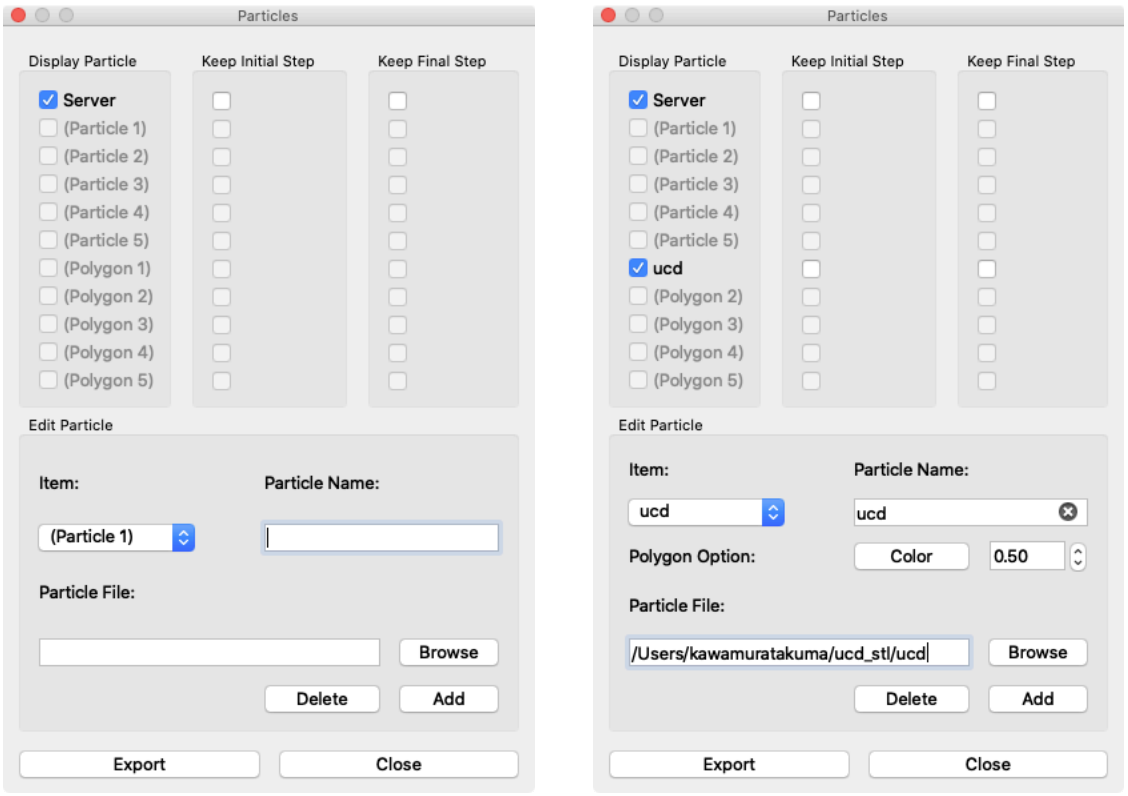


Figure6.3-21 Particle panel. Left: particle selection mode, right: polygon selection mode.

Display Particle category shows a list of particle and polygon datasets. The particle is sent from PBVR Server, or is loaded from local files (maximum 5 files). The polygon is loaded from local files (maximum 5 files).

Particle panel supports STL format polygon data. This function can process STL files that are divided into one file per time step according to the following naming convention.

prefix_*****.stl (***** means the number of time steps in a five-digit display.)

If the subpixel levels of the server particle data and local particle data is different, the server particle's subpixel level takes priority. In standalone mode, If multiple local particle data are to be displayed and the sub-pixel level of each particle data is different, the sub-pixel level of the later loaded particle data takes priority.

Keep Initial Step category specifies particle/polygon datasets, in which the initial step data is displayed before the time series starts, when integrated particle datasets start from different time steps. **Keep Final Step** category specifies particle/polygon datasets, in which the final step data is displayed after the time series ends, when integrated particle datasets end at different time steps.

[Display Particle/ Keep Initial Step/ Keep Final Step category]

- **Server check box** is activated when a particle dataset from PBVR Server is integrated with local particle data sets. This checkbox is not available in stand-alone mode.
- **(Particle1)- (Particle5) check box** are activated to integrate the particle datasets loaded from local files. The checkbox is not available before particle datasets are loaded via Particle file panel.
- **(Polygon1)- (Polygon5) check boxes** are activated to integrate the polygon datasets loaded from local files. The checkbox is not available before polygon datasets are loaded via Particle file panel.

Edit Particle category enables to add particle data or polygon from a local file on the client PC to the list of **(Particle1)- (Particle5)**, **(Polygon1)- (Polygon5)** and to save the integrated particle data as the particle file.

[Edit Particle category]

- **Item** spin button selects the item from the list of (Particle1)- (Particle5), (Polygon1)- (Polygon5) to add particle data.
- **Particle Name** text box enables to name the particle data and polygon read from the client PC. If this description is omitted, the file name displayed in the Particle File column is used.
- **Browse** button opens the file dialogue to load the particle data and polygon. The path of the loaded particle data and polygon file is displayed in the Particle File column. A path containing a double-byte character string cannot be specified.
- **Add** button adds the particle data and polygon displayed in the Particle File column to the item selected with the Item spin button. If you select an item that has already been added, overwrite it with the particle data that was load later.
- **Delete** button deletes the particle data and polygon added to the item currently selected with the Item spin button.
- **Export** button integrates the particle data in memory and outputs it to the file specified in the Particle File column.
- **Close** button closes Particle panel.
- **Polygon Option** appears in case of that polygon is selected at Item spin button. It can specify color and opacity of the polygon.

6.3.5.1 Example of volume and polygon composition

The following is an example of a composite the unstructured lattice data spx.inp and the polygon spx.stl of its boundary shape included in the sample data. Specify the path of spx.inp filtered by the -vin option when starting the server, and execute volume rendering on the client (Figure 6-22 left). Next, in the Edit Particle category of the Particle Integration panel, select Polygon from the Item spin box, and specify the path of spx.stl in Particle File. Next, specify

the data name "spx.stl" in Particle Name, and set the color and opacity. Then, check spx.stl in the Display Particle category (right side of Figure 6-22), and the boundary shape will be synthesized (center of Figure 6-22).

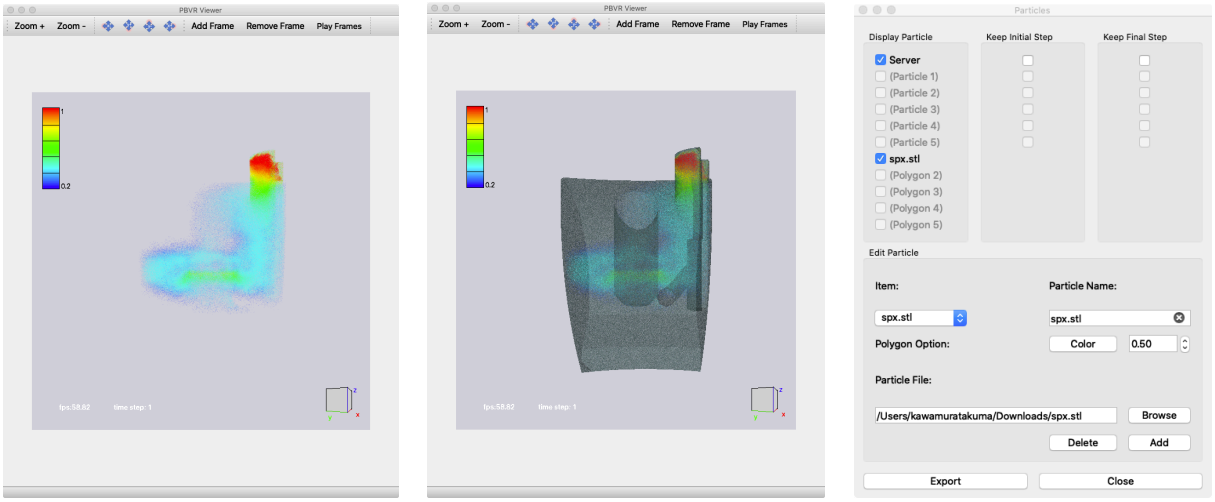


Figure 6-22 Left: volume rendering result of spx.inp; Center: composite view of spx.inp volume and boundary shapes; Right: settings of the Particle Integration panel in the composite view.

6.3.5.2 Example of multi-time steps composition

The behavior of the particle integration is explained using Figure 6.3-23 (sever side: 1~4 time steps, client side:0~3). If the server checkbox and the particle checkboxes are checked, all time steps are displayed as shown in Table 6.3-2. If the checkbox **Keep Initial Step** is checked for the client particles, only the first time step is displayed, as shown in Table 6.3-3. If **Keep Final Step** is checked for the client particles, only the final time step is displayed, as shown in Table 6.3-4.

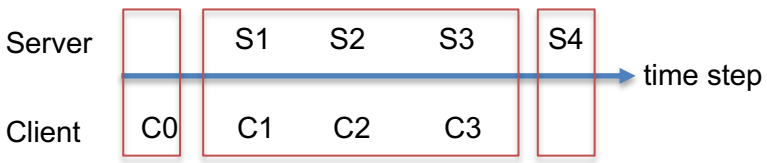


Figure 6.3-23 The behavior of the particle integration

Table 6.3-2 Particle datasets displayed by default.

	Step0	Step1	Step2	Step3	Step4
Server	-	S1	S2	S3	S4

Client	C0	C1	C2	C3	–

Table 6.3-3 Particle datasets displayed with Keep Initial Step for the server particles.

	Step0	Step1	Step2	Step3	Step4
Server	S1	S1	S2	S3	S4
Client	C0	C1	C2	C3	–

Table 6.3-4 Particle datasets displayed with Keep Final Step for the client particles.

	Step0	Step1	Step2	Step3	Step4
Server	–	S1	S2	S3	S4
Client	C0	C1	C2	C3	C3

6.3.6 Image file production

In Animation Controls, PBVR Client saves image data for the Viewer in two modes, both of which are played as a movie.

- Time series data mode
  The images are saved as time series data in BMP format. The image data files are converted and compressed into a movie file via free software, such as ImageMagic and ffmpeg.

- Key frame animation mode
  Geometrical information from an arbitrary time step is retained as a key frame. A series of key frames can be played as a key frame animation.

Figure6.3-24 shows the **Animation Control Panel**. Each widget works as described below.

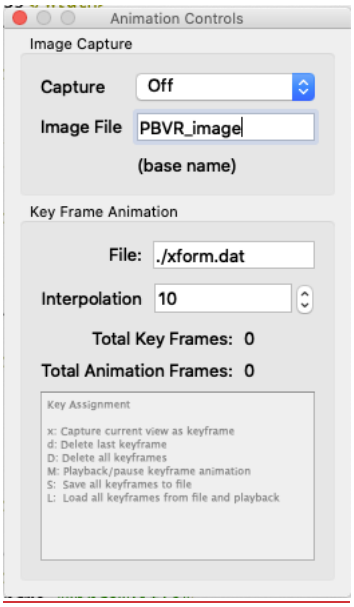


Figure6.3-24 Animation Controls Panel

- **Capture** spin box turns image production off or on.
- **Image File** specifies a prefix for the image data files. The default name is “PBVR_image”.
- **File** specifies a key frame file that contains geometrical data. The default name is ./xform.dat.
- **Interpolation** specifies the number of frames used for linear interpolation of the geometry data between two key frames in a key frame animation. The default value is 10.
- **Total Key Frames** shows the number of key frames stored in the current key frame animation. It is initialized to 0, and incremented (or decremented) by pressing “x” (or “d”). It is initialized to 0 again by pressing “D”.

- **Total Animation Frames** shows the number of total frames stored in the current key frame animation, which is calculated as
  (Total Key Frames – 1) x Interpolation

6.3.6.1 Image production

Image files are produced as follows.

1. Specify the prefix for the image files in **image file**.
2. Select “on” in the **capture** drop down menu.
3. A series of image files are saved at each time step.
4. Image production is stopped by selecting “off” in the **capture** drop down menu.

The image files are saved in the directory specified by the command line option “-iout”. If “-iout” option is not specified, they are saved in the current directory “./”. The following shows an example of image data files produced with the default prefix “PBVR_image”:

PBVR_image.00001.bmp
PBVR_image.00002.bmp
…

If the image files are produced from a key frame animation, which is explained later, the file names are modified by adding “_k” after the prefix.

PBVR_image_k.00001.bmp
PBVR_image_k.00002.bmp
…

6.3.6.2 Key frame animation of a still image

A key frame animation of a still image, which is obtained by pressing **Stop** in the **Time Panel**, is produced as follows.

[Capture key frames and save them in a file]

1) Specify a key frame file in **file**.
2) Activate the **Viewer** by clicking it.
3) Adjust the view and press “x” to store the geometry information for the view in memory.
4) Repeat (3).
5) Press “M” (i.e., shift+m) to play the key frame animation.
6) If the contents of the key frame animation are satisfactory, press “S” (i.e., shift+s) to

save the geometry information as a series in the key frame file.

[Play a key frame file]

1) Specify a key frame file in **file**.
2) Activate the **Viewer** by clicking it.
3) Press “L” (i.e., shift+f) to play the key frame animation stored in the key frame file.
4) Press “x” to add new key frames to the current key frame animation.

Table6.3-5 Keys used for controlling key frame animation

Key	Function
x	Add geometry information for the current view to the key frame data in memory.
d	Delete the last key frame.
D	Delete all key frames.
M	Play or pause the key frame data in memory.
S	Save the key frame data in memory to a key frame file.
L	Load a key frame file and play its key frame data

6.3.6.3 Key frame animation of time series data

A key frame animation of time series data can be produced as follows.

1) By pressing “x” while time series data are being rendered, both geometry information and a time step number are stored in memory.
2) Press “S” to save a series with geometry information and time step numbers in the key frame file.
3) Press “F” to load a series with geometry information and time step numbers in the key frame file and play a key frame animation. Here, if key frames are at unequal intervals, then interpolation frames, which are specified in **interpolation**, are non-uniform in time.

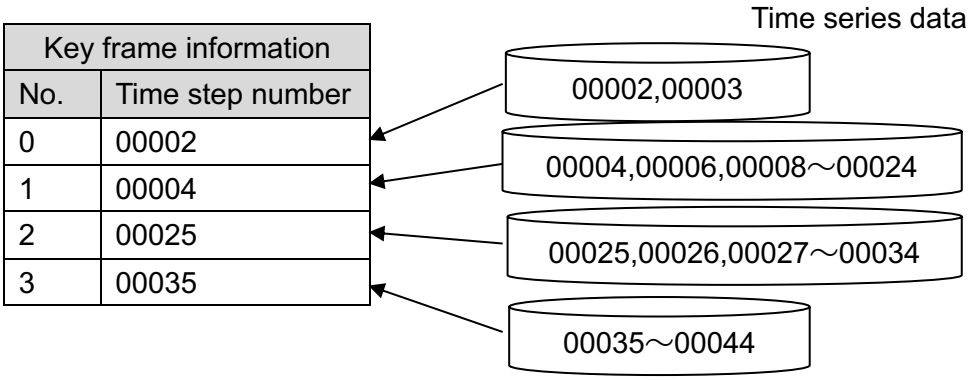


Figure6.3-25 Key frame animation for time series data

In the example shown in Figure6.3-25, if there are 10 interpolation frames between key frames, then 5 interpolation frames are assigned to the time steps 00002 and 00003 between key frames 0 and 1. On the other hand, between key frames 1 and 2, 10 interpolation frames are assigned to the time steps from 00004 to 00024. As a result, the time steps 00004, 00006, …, 00024 are shown in the key frame animation.

6.3.6.4 Key frame file format

A key frame file contains binary data with the following format.

File format

Key frame data 1
Key frame data 2
:

Type	Size (bytes)	data
int	4	Time step number
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

Figure6.3-26 Key frame file format

6.3.7 Legend panel

Figure6.3-27 shows the **Legend panel**. A legend is a bar showing how the values of a physical quantity are rendered as a color in the visualization. Each widget works as described below.

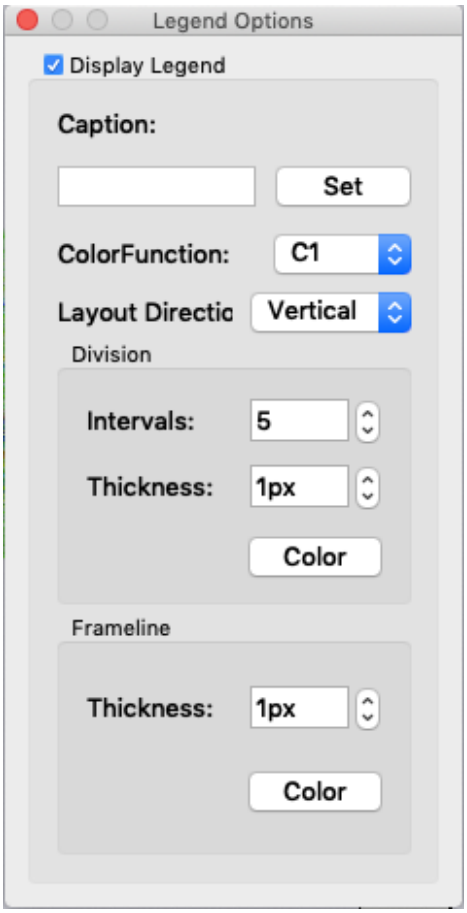


Figure6.3-27 Legend panel

- **Display Legend** checkbox turns on or off of showing the legend.
- **Caption** text box enables to enter a caption string for the legend and the contents are reflected by **Set** button.
- **Color Function** spin button selects the color map legend and the range.
- **Layout Direction** spin button selects the direction of the legend (vertical/horizontal).
- **Intervals** specifies the number of tick marks.
- **Thickness** in Division category specifies a thickness of the tick marks.
- **Color** in in Division category specifies a color of the tick marks.
- **Thickness** in Frameline category specifies the thickness of the frame border.
- **Color** in in Frameline category specifies a color of frame border.

Figure6.3-28 shows an example of legend.

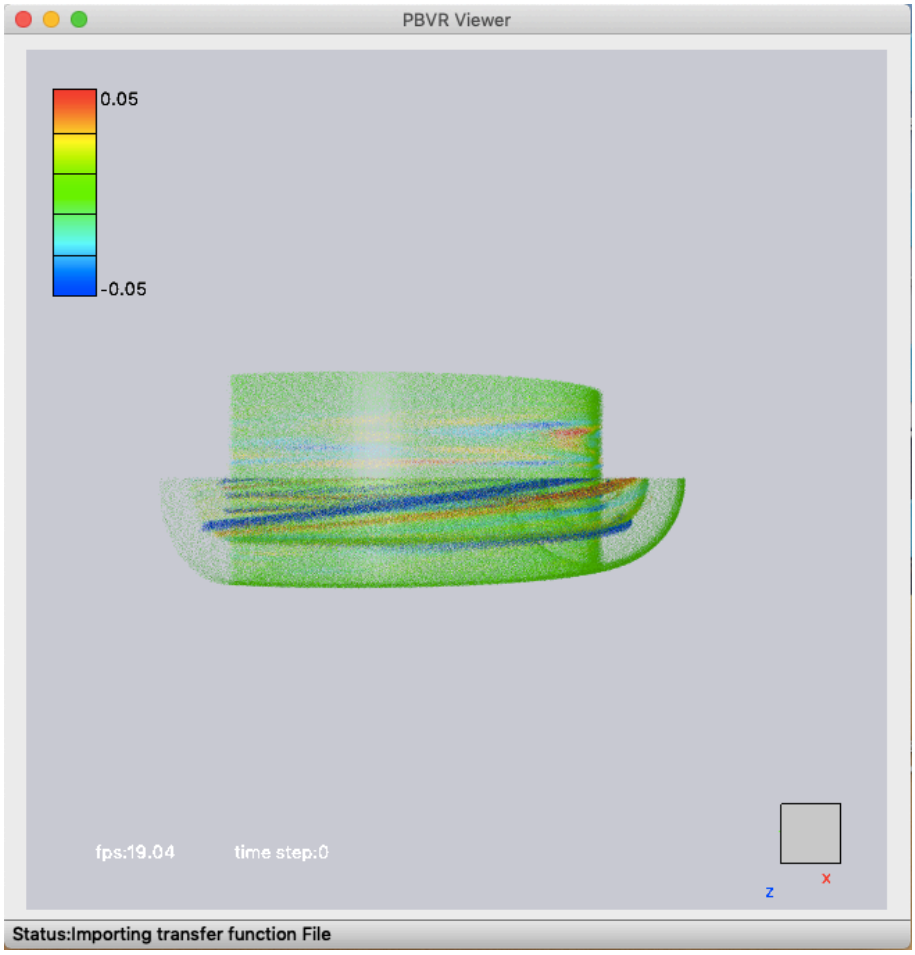


Figure6.3-28 Example of a legend

6.3.8 Coordinate Panel

Figure6.3-29 shows the **Coordinate panel**. This panel is used to specify a coordinate transformation for each axis. For example, you can change from Cartesian coordinates to cylindrical coordinates.

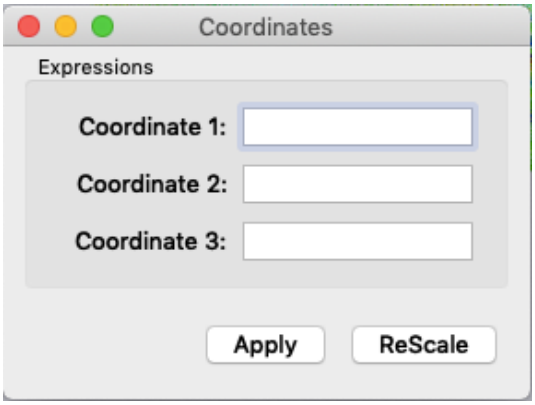


Figure6.3-29 Coordinate panel

- **Coordinate 1** specifies a formula for the new *X* coordinate. Empty or blank means just "X", i.e., the Cartesian coordinate.
- **Coordinate 2** specifies a formula for the new *Y* coordinate. Empty or blank means just "Y", i.e., the Cartesian coordinate.
- **Coordinate 3** specifies a formula for the new *Z* coordinate. Empty or blank means just "Z", i.e., the Cartesian coordinate.
- **Rescale** button rescales the viewer display using the maximum and minimum coordinate values of data displayed so far.
- **Apply** button sends a formula defined with this panel to the server.

The coordinate transformations are entered into the text boxes for coordinate axes 1–3. Variables used are the Cartesian coordinates (*X*, *Y*, and *Z*), physical quantities (*q*1, *q*2, …, *q*9), and time (*T*). The variables can be written in upper case (*X*, *Y*, *Z*, and *T*) or lower case (*x*, *y*, *z*, and *t*). In addition, the operations allowed are the same as those used in the transfer function editor (see Section 6.3.3.3). If a physical quantity specified does not exist, then 0 will be used.

6.3.9 Viewer Control Panel

Figure6.3-30 shows the **Viewer Control Panel**, which specifies the properties of the viewer. Each widget works as described below.

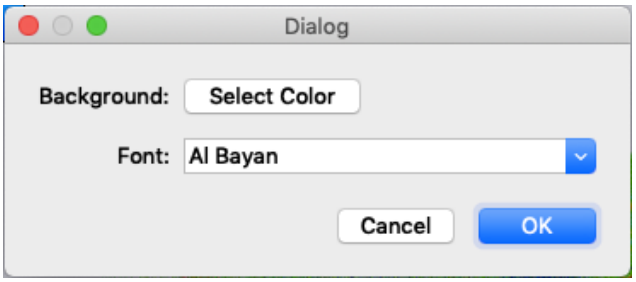


Figure6.3-30 Viewer control panel

- **Background** specifies the background color of the viewer.
- **Font** selects a font type and size of the character shown to a viewer

7 An Example with the Sample Dataset

The following sections demonstrate the usage of PBVR for a sample dataset *gt5d.tgz*.

7.1 Filtering Process

Uncompress *gt5d.tgz* to extract the following files under the directory *./gt5d*.

gt5d.fld:	An AVS field file
co3d.dat:	A coordinate data file
pd3d.dat:	The variable 1
psid.dat:	The variable 2
param.txt:	Input parameters for PBVR Filter
demo.tf:	A transfer function file for demonstration

Execute PBVR Filter with the following command (which invokes the OpenMP version).

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

The contents of param.txt are as follows:

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

The above example specifies the SPLIT file format (which is the default format), a single sub-volume (without sub-volume decomposition), and the same directory for both input and output. This filtering process generates the following files in the specified output directory.

case.pfi	: A PFI file
case_YYYYYYY_ZZZZZZZ_connect.dat	: An element configuration file
case_YYYYYYY_ZZZZZZZ_coord.dat	: A node coordinate file
case_XXXXX_YYYYYYY_ZZZZZZZ.kvsml	: A kvsml file
case_XXXXX_YYYYYYY_ZZZZZZZ_value.dat	: A variable file

7.2 Starting PBVR

The following is an example of connecting port 50000 of machineA to port 60000 of machineB using ssh port forwarding and starting client and server programs on each machine.

Step1 [ssh portforwarding]
machineA> ssh -L 50000:localhost:60000 username@machineB

Step2 [Launching server program]
machineB> mpixec -n pbvr_server
first reading time[ms]:0
Server initialize done
Server bind done
Server listen done
Waiting for connection ...

Step3 [Launching client program] In this example, the client program reads the transfer function file demo.tf at launching, used the Metropolis sampling for the particle generation, and set phong shading as the shading parameters.

machineA> pbvr_client -S m -vin ./gt5d/case.pfi -shading P,0.6,0.6,0.6,30

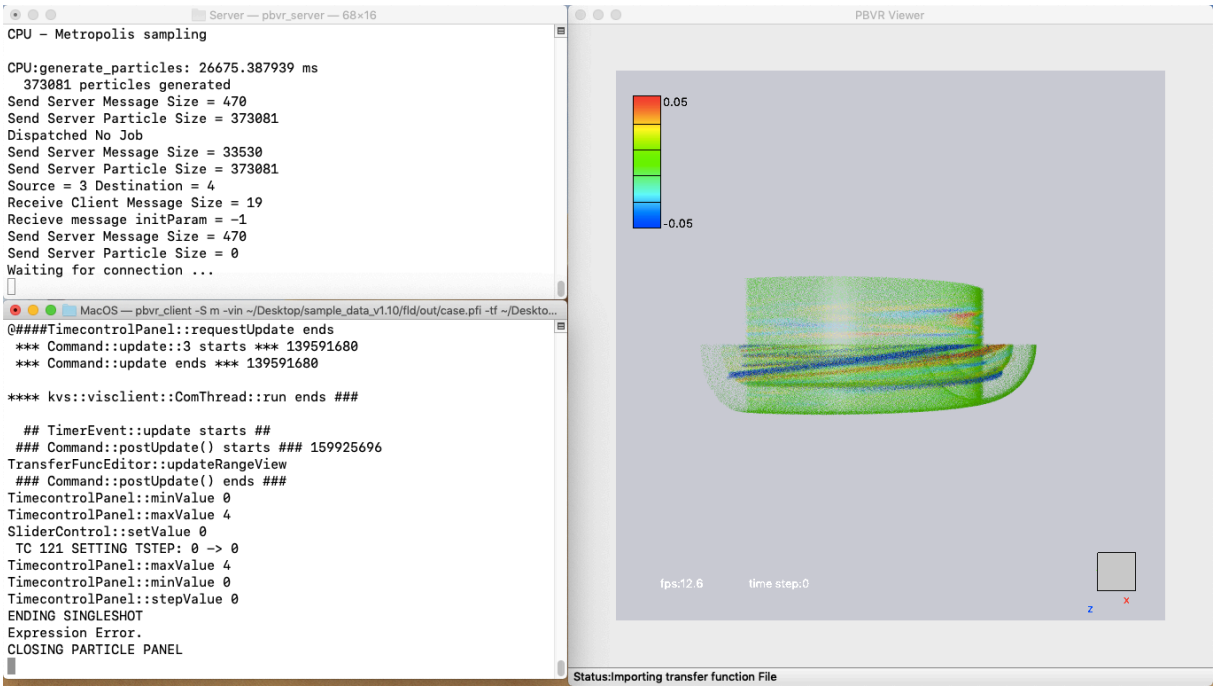


Figure7.2-1 PBVR GUI

7.3 Designing Transfer Functions

This section shows examples of visualizing *gt5d.fld*, using the multi-dimensional transfer function that is produced with the advanced transfer function design capability of PBVR. *gt5d.fld* contains structured grid volume data that consists of two variables.

7.3.1 Volume Rendering for a Single Variable

First, visualize the variable *q1* by setting the transfer function *t1* as shown in Figure7.3-1. In this example, the transfer function is designed with the **Transfer Function Editor**. The left part of the **Transfer Function Editor** shows the configuration for colors, while in the right part shows the opacity. Notice that this configuration is the conventional volume rendering for a single variable.

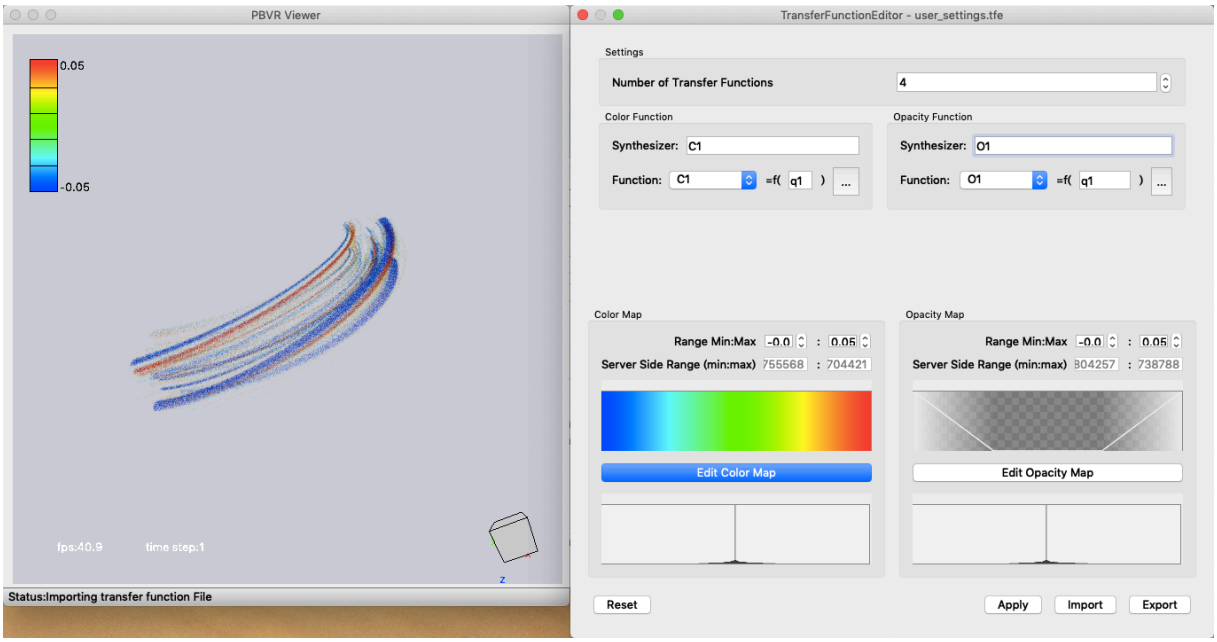


Figure7.3-1　Volume rendering result for the variable *q1*.

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

7.3.2 Multivariate Volume Rendering

The next example shows the result of a multivariate volume rendering, in which the variables *q1* and *q2* are synthesized as shown in Figure7.3-2. In this example, the colors are assigned to the variable *q1*, while the opacity is assigned to the variable *q2*. The opacity map extracts two torus surfaces, which are given by the iso-surfaces of the variable *q2*. The colors encode the distribution of the *q1* values in these iso-surfaces.

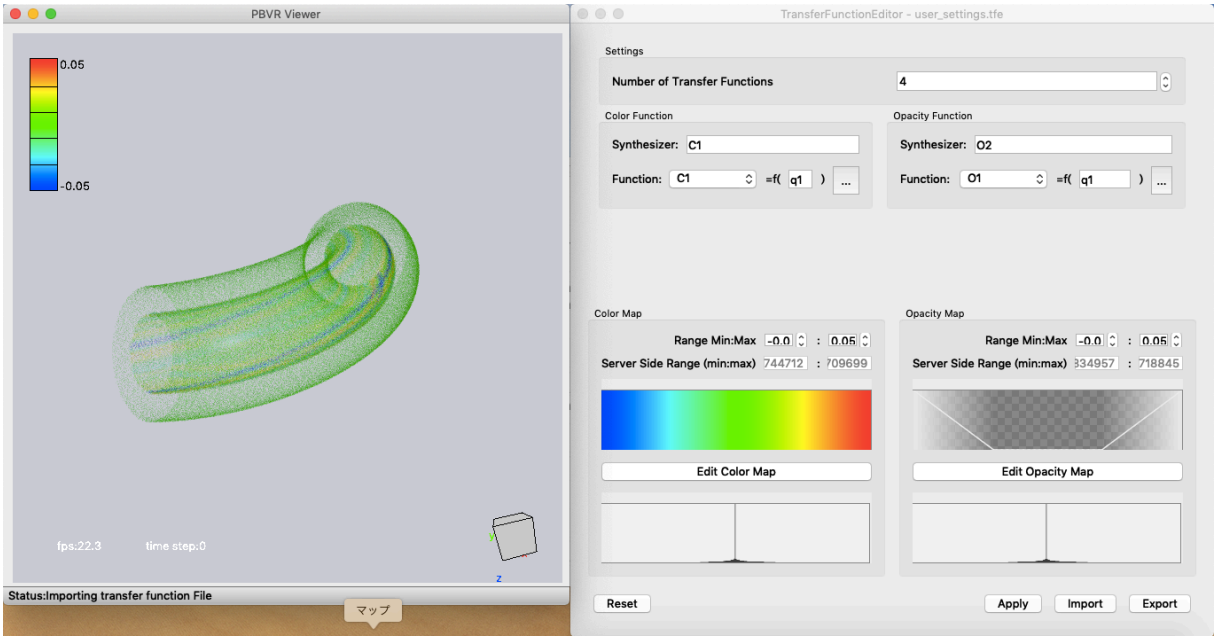


Figure7.3-2　Rendering a multivariate volume. The *q1* values are color-mapped onto the iso-surfaces of *q2*.

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

7.3.3 Slicing Volumes

Figure7.3-3 shows an application of PBVR's multivariate volume rendering after extracting a slice. With PBVR, an arbitrary function can be used to design a transfer function. In this example, the cylindrical surface ($X^2+Z^2=const.$) is extracted and the color of the variable *q1* is mapped onto it.

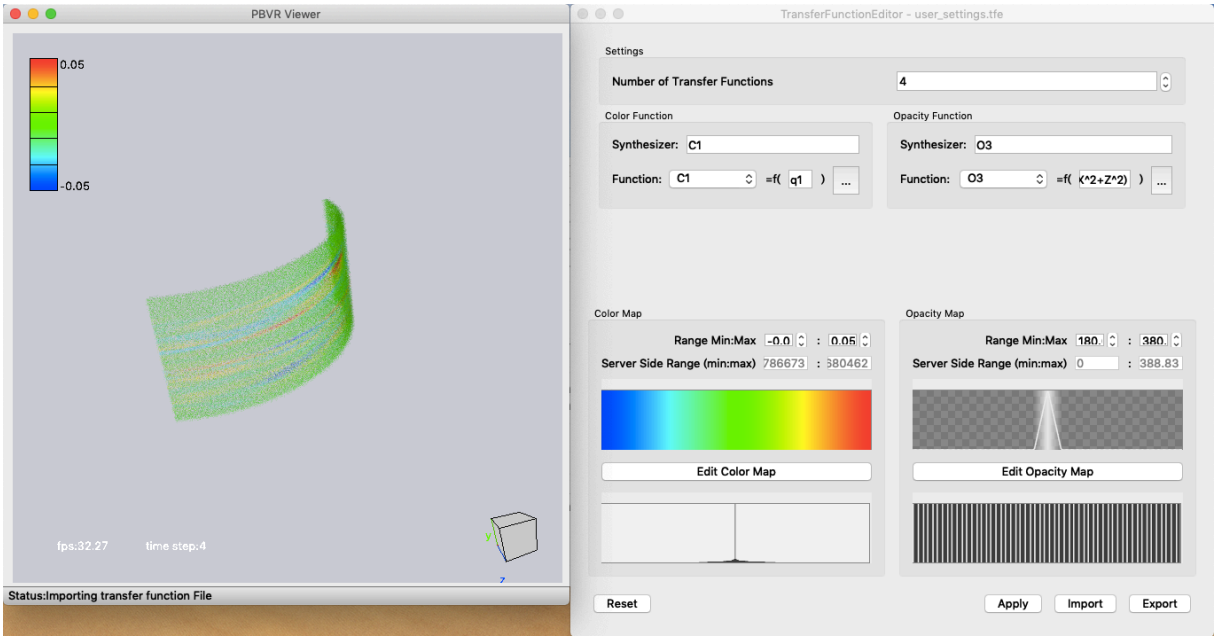


Figure7.3-3 Rendering after slicing the volume with PBVR's multivariate volume rendering capability.

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 Synthesis of Transfer Functions

This section explains how to synthesize transfer functions in PBVR. Figure7.3-4 shows a transfer function *O4,* whose opacity function makes the region $Y > 0$ transparent. By synthesizing the previously described transfer functions *O1*, *O2,* and *O3* together with a new transfer function *O4* as (*O1* + *O2*) * *O4* + *O3*, the individually extracted sub-regions can undergo a flexible composition using arithmetic operations. In this example, the colors of *t2* and *O3* are set to (*R*, *G*, *B*) = (0, 0, 0), while the color of *O4* is set to (*R*, *G*, *B*) = (1, 1, 1). In the above synthesis equation, the final colors are the rainbow color map defined for *O1*. On the other hand, the opacity of *O4* is multiplied by the sum of *O1* and *O2* in order to extract the lower half region ($Y < 0$) of *O1* and *O2*. Then, the resulting region is synthesized with the cylindrical surface given by *O3*.

As demonstrated by these examples, PBVR's ability to synthesize transfer functions is powerful. It can extract an arbitrary region for each parameter and then use a series of mathematical operations.

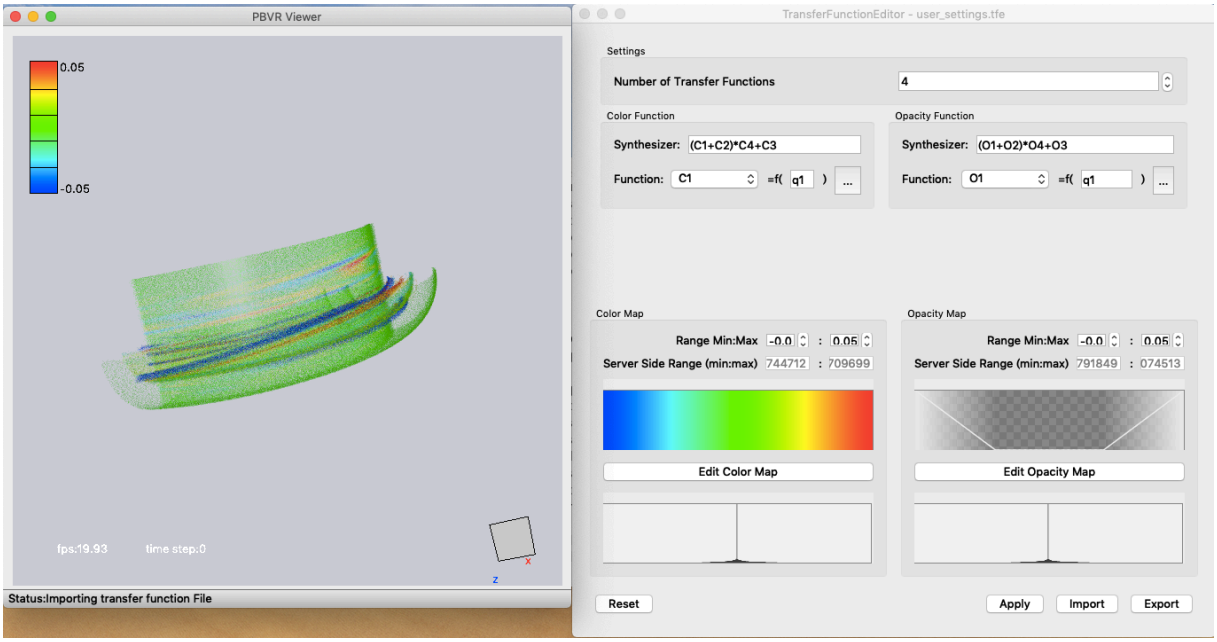


Figure7.3-4 Synthesizing transfer functions

Color Function SYNTHESIZER ：(C1+C2)*C4+C3

Opacity Function SYNTHESIZER ：(O1+O2)*O4+O3

7.4 Integration of Particle Dataset

While the previous section illustrates a visualization using volume rendering, iso-surfaces, and surface rendering via multi-dimensional transfer functions, a similar image composition can also be achieved by integrating multiple particle datasets. This section gives an example of a particle integration.

7.4.1 Saving Particle Datasets

Particle datasets are stored using the **Particle File sub-panel** in **Particle panel**. Figure7.4-1 shows an example using “Export Particle”. In this case, the following files are generated with the prefix “p1”:

./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

Here, XXXXX is the time step, YYYYYYY is the sub-volume number, and ZZZZZZZ is the number of sub-volumes. The files “colors”, “coords”, and “normal” contain the color, coordinates, and normal vector of each particle, respectively. On hitting the **Export** button, integrated particle data are stored in the above files. While saving, the **Export** button is de-activated as shown in Figure7.4-2. Once the whole time series data have been saved, the **Export** button becomes active again.

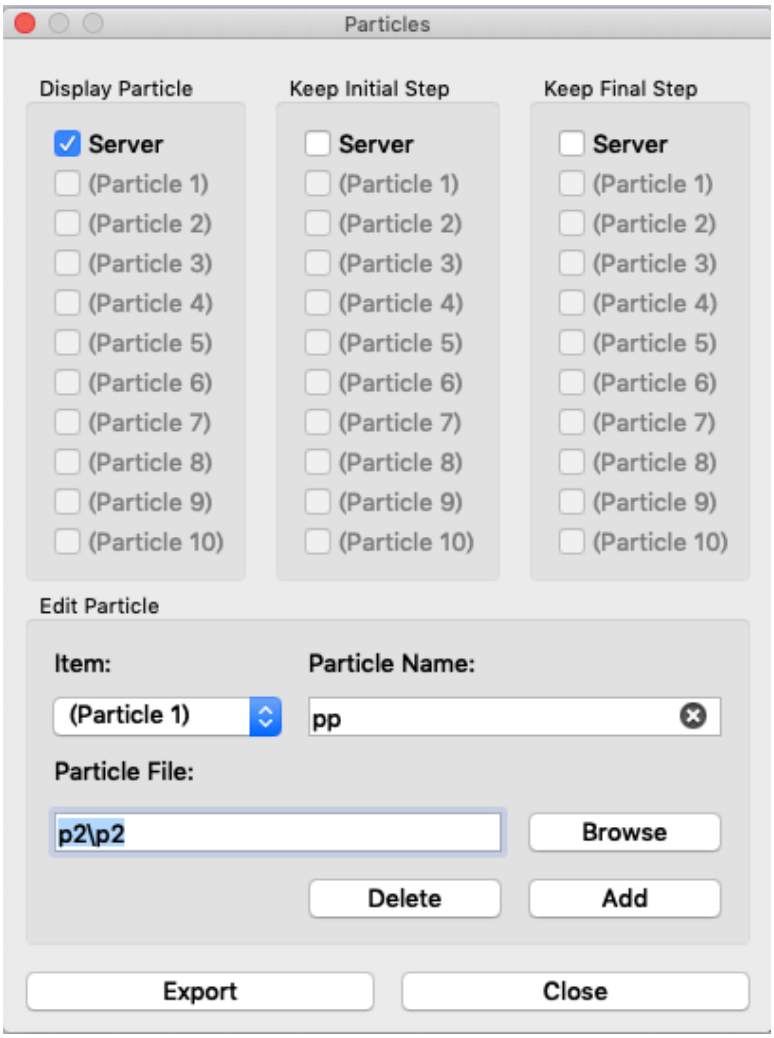


Figure7.4-1 Particle File panel (**Export** is active)

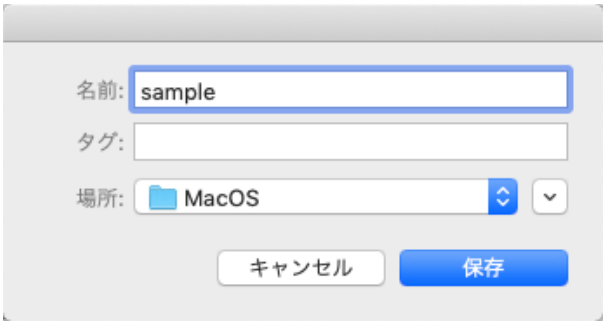


Figure7.4-2 Particle File panel (**Export** is inactive)

7.4.2 Loading Particle Dataset

In the following example, three particle datasets, p1, p2, and p3, which correspond to the images in Figure7.3-1 to Figure7.3-3, are loaded and integrated. Here, PBVR Client is launched in stand-alone mode with the following command:

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

p1–p3 are listed in the **Particle panel**. After turning on the **Display particle** checkbox for p1, the volume rendering is shown, as illustrated in Figure7.4-3. In addition, by turning on the **Display particle** checkboxes for p2 and p3, all three particle datasets are integrated, as in Figure7.4-4. The integrated particle data can be stored as a single particle dataset via **Particle File**. Note that to obtain images that are correctly integrated, all the particle datasets must be generated using the same particle densities and particle limits, which are specified by the command line options -pd and -plimit.

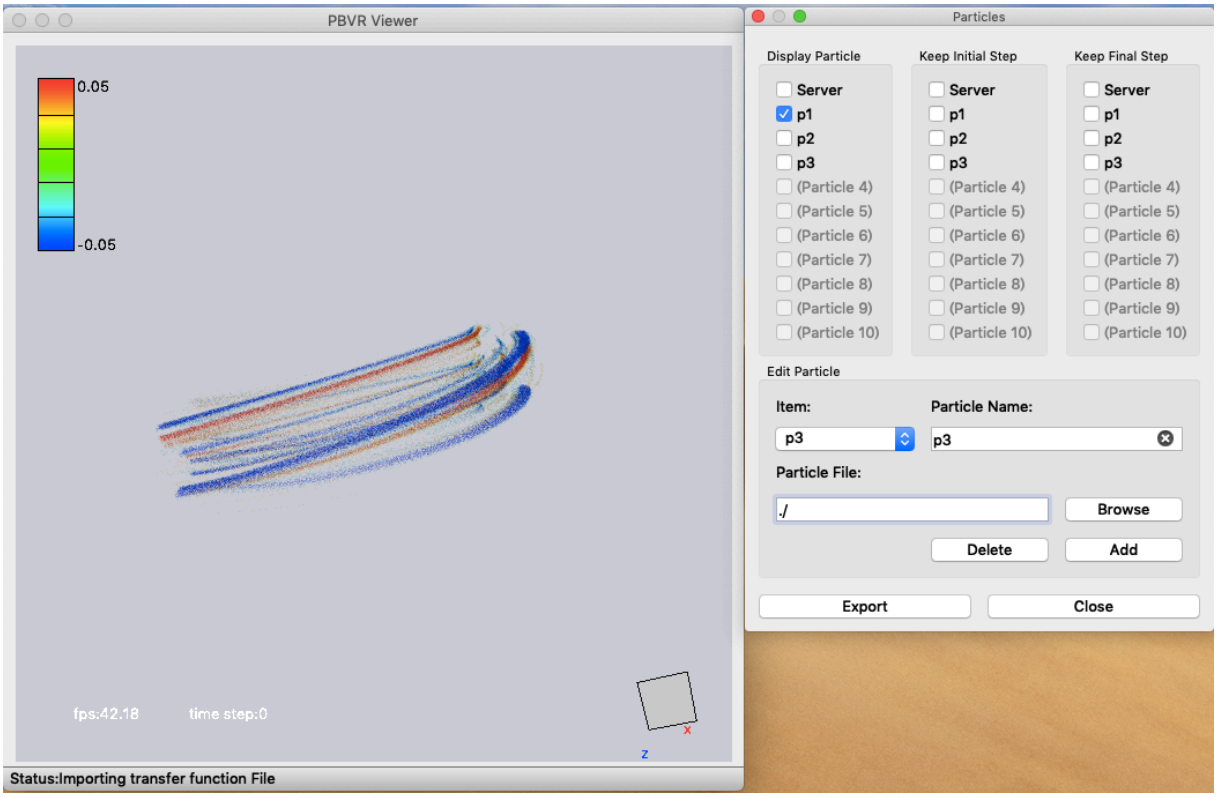


Figure7.4-3 Particle dataset p1

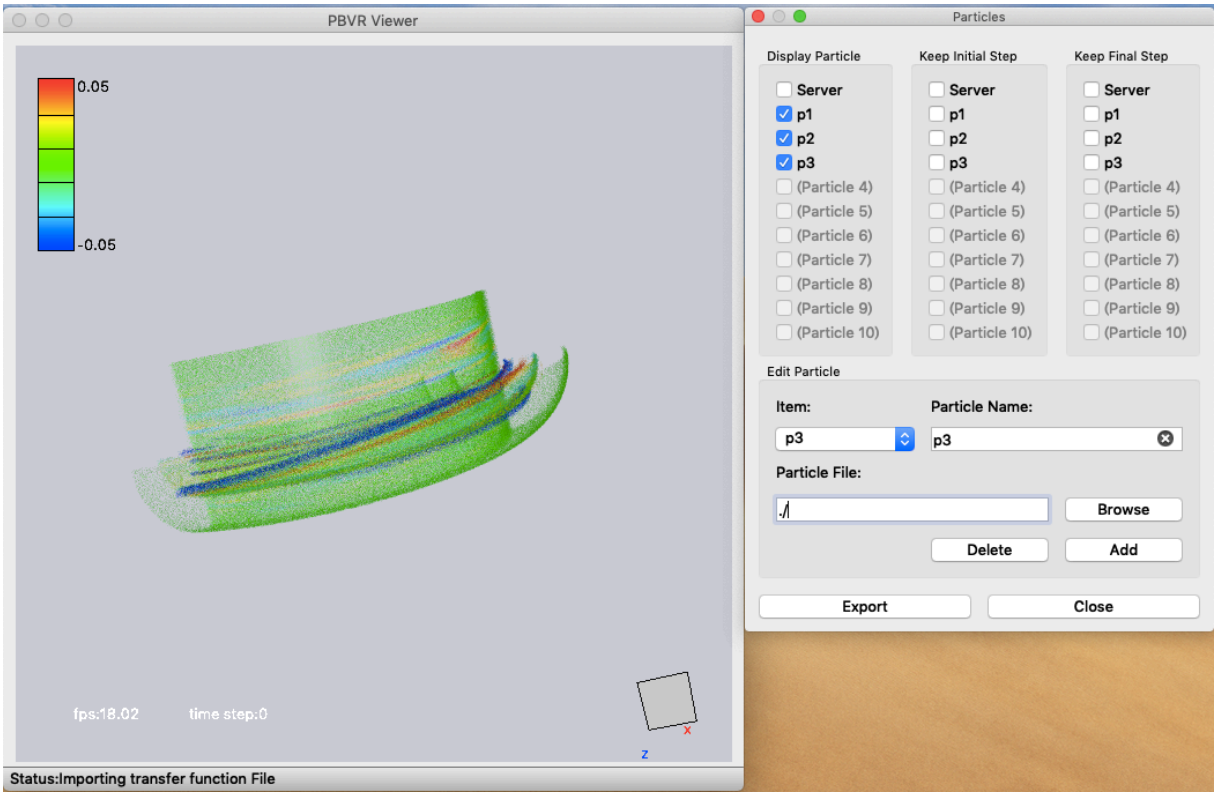


Figure7.4-4 Integration of p1, p2, and p3

7.5 Saving Results

After designing the transfer function, PBVR can save the resulting image and parameters in the following three ways:

1) Image output (Section 6.3.6)
   To save the results as images, select “on” in the **capture** drop-down list in the **Animation panel**. Bitmap image files (PBVR_image.xxxxx.bmp) are generated.
2) Transfer function file (Section 6.3.3)
   To generate a transfer function file, enter a file name for the transfer function in the **File Path** of the **Transfer Function Editor** and press **Export File**. This file can subsequently be loaded using **Import File**.
3) Visualization parameter file (Section 4.4)
   All the visualization parameters including the transfer function can be exported and used to run PBVR Server in batch mode. Open the **File panel** from the **Main panel**, specify the parameter filename and press **Export FILE**.

7.6 Batch Mode Example

This section shows how to run PBVR Server in batch mode using the visualization parameter file exported in the previous section. This mode is used for massively parallel processing on a supercomputer. In addition, it is also useful for high-speed rendering of time series data with PBVR Client in stand-alone mode, because it eliminates the latency due to particle generation and particle data transfer.

[Step 1] Launch PBVR Server (of OpenMP version) in batch mode

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

[Step 2] Launch PBVR in stand-alone mode

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