# Remote Visualization Tool PBVR (v1.05)

# User Guide

**March 2015**

**Japan Atomic Energy Agency**

**Center for Computational Science & e-Systems**

# Revision Record

| Version number | Date revised | Revised chapter | Revised content |
|---|---|---|---|
| 1.04-1 | 2015.3.31 | - | Release |
| 1.05-1 | 2015.6.9 | 4.2 | Table 10 (command line options) is revised due to updates in the Japanese manual as of 20150608<br>. |
| | | 5 | FILE Panel. Repeat level is eliminated. |
| | | 6 | Repeat level is eliminated. Added Figure 8. |

# 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 the KVS library, 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 as images using Open GL.

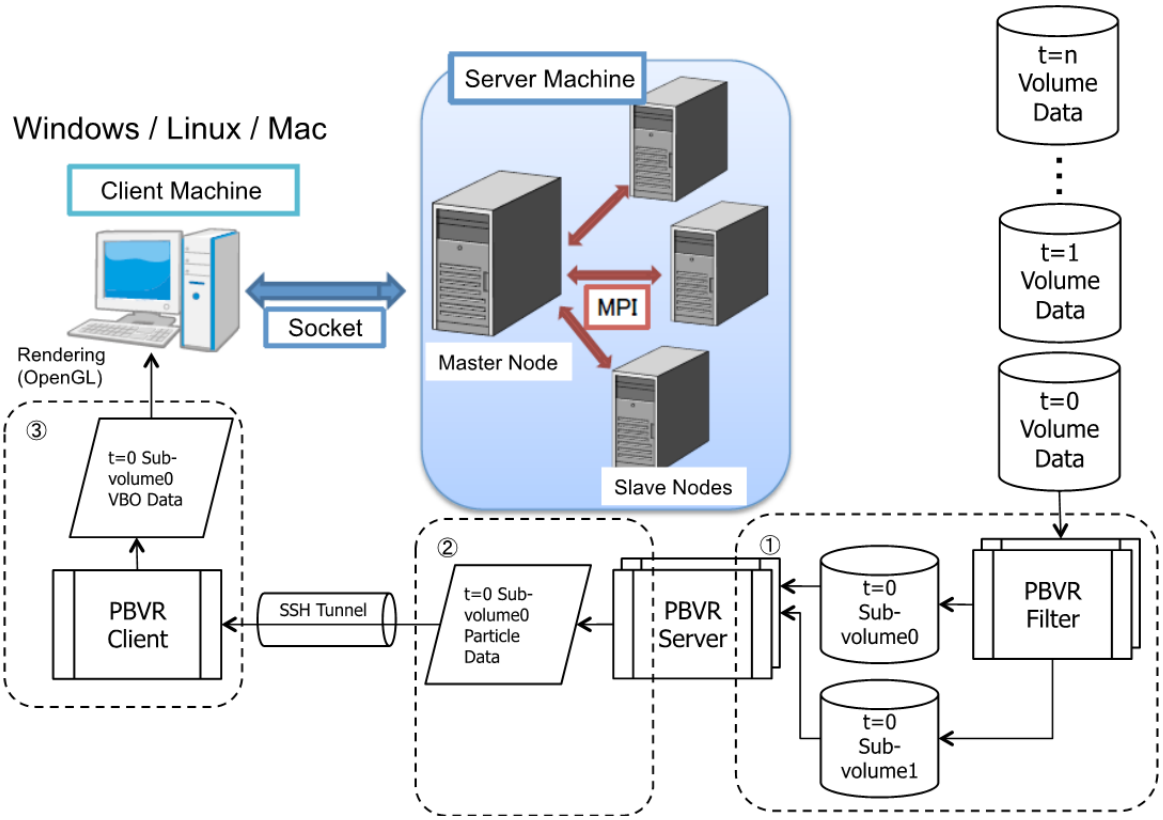



Figure 1 The system configuration of PBVR

## 1.2 System Requirements

The system is verified for the following platforms and compilers.

●PBVR Filter

| Platform | Compiler | Library |
|---|---|---|
| Linux 64bit | gcc version 4.4.6 | |
| Mac 64bit *2 | gcc version 4.8.2 | |
| Windows7 64bit | Visual Studio 2013 | |
| K Computer | Fujitsu Compiler | |
| FX10 | Fujitsu Compiler | |
| BX900 | Fujitsu Compiler | |

●PBVR Server

| Platform | Compiler | Library |
|---|---|---|
| Linux 64bit | g++ version 4.4.6 | KMATH_RANDOM, NTL *1 |
| Mac 64bit *2 | g++ version 4.8.2 | |
| Windows7 64bit | Visual Studio 2013 | |
| K Computer | Fujitsu Compiler | KMATH_RANDOM, NTL *1 |
| FX10 | Fujitsu Compiler | KMATH_RANDOM, NTL *1 |
| BX900 | Fujitsu Compiler | KMATH_RANDOM, NTL *1 |

●PBVR Client

| Platform | Compiler | Library |
|---|---|---|
| Linux 64bit | g++ version 4.4.6 | OpenGL, GLUT |
| Mac 64bit *2 | g++ version 4.8.2 | OpenGL, GLUT |
| Windows7 64bit | Visual Studio 2013 | OpenGL, GLUT |

*1. KMATH_RANDOM 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. On Macs, OpenMP is not available for the default gcc that is shipped with Xcode. To use the load modules or to compile the source code, install a newer version of gcc that works with OpenMP. The prebuilt binaries were compiled with gcc48 installed through MacPorts. To obtain this gcc version, run the following commands with root privilege in the terminal'.

```
# port install gcc48
# port select –set gcc mp-gcc48
```

# 2 Installation

PBVR consists of a load module package *load_module_v1.04.tgz* and a source code package *pbvr_v1.04.tgz*. The following sections show how to install PBVR Filter, PBVR Server, and PBVR Client.

## 2.1 PBVR Filter

PBVR Filter is implemented in C and is shipped with two versions. The first is an MPI+OpenMP version for massively parallel computing, and the second is an OpenMP version for thread parallel computing.

### 2.1.1 Installation of Prebuilt Binaries

The following table lists the load modules that are stored in the *filter* directory of the load module package. Choose the suitable load modules, and copy them to a directory that is specified in PATH environment variable. When the copying operation finishes, the installation is complete.

Table 1 List of load modules for PBVR Filter.

| **Platform** | **Parallelization** | **Name of load module** |
|---|---|---|
| Linux 64bit | OpenMP | filter_Linux_omp |
| | MPI+OpenMP | filter_Linux_mpi_omp |
| Mac 64bit | OpenMP | filter_mac |
| Windows7 64bit | OpenMP | filter_Win_omp.exe |
| K Computer *1 | OpenMP | filter_K_omp |
| | MPI+OpenMP | filter_K_mpi_omp |
| FX10 *1 | OpenMP | filter_FX10_omp |
| | MPI+OpenMP | filter_FX10_mpi_omp |
| BX900 *1 | OpenMP | filter_BX_omp |
| | MPI+OpenMP | filter_BX_mpi_omp |

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

### 2.1.2 Installation from Source Code

Alternatively, PBVR Filter can be installed from the source code. If this is desired, uncompress *filter.tgz*, which contains the source code, to an arbitrary directory. Compile the extracted source using suitable Makefiles specified in the list below, and copy the generated load

modules to an arbitrary directory that is specified in PATH environment variable. When the copying operation is finished, the installation is complete.

Table 2 List of Makefiles for the PBVR Filter

| Platform | Parallelization | Name of Makefile |
|---|---|---|
| Linux 64bit | OpenMP | Makefile.linux_omp |
| | MPI+OpenMP | Makefile.linux_mpi_omp |
| Mac 64bit | OpenMP | Makefile.mac |
| Windows7 64bit*1 | OpenMP | Makefile.win |
| K Computer*2 | OpenMP | Makefile.omp |
| | MPI+OpenMP | Makefile.mpi_omp |
| FX10 *2 | OpenMP | Makefile.fx10_omp |
| | MPI+OpenMP | Makefile.fx10_mpi_omp |
| BX900 *2 | OpenMP | Makefile.bx_omp |
| | MPI+OpenMP | Makefile.bx_mpi_omp |

*1. In a Windows environment, launch the Visual Studio 2013 x64 Native Tools command prompt and run nmake.

*2. Makefiles for supercomputers are designed for use with cross-compilers on login nodes.

## 2.2 PBVR Server

PBVR Server is implemented in C++ and is shipped in two versions. As with PBVR Filter, the first is an MPI+OpenMP version for massively parallel computing and the second is an OpenMP version for thread parallel computing.

### 2.2.1 Installation of Prebuilt Binaries

The following table lists the load modules that are stored in the *sever* directory of the load module package. Choose the suitable load modules, and copy them to a directory that is specified in PATH environment variable. When the copying operation is finished, the installation is complete.

Table 3 List of load modules for PBVR Server

| Platform | Parallelization | Name of load module |
|---|---|---|
| Linux 64bit | OpenMP | CPUServer_linux_omp |
| | MPI+OpenMP | CPUServer_linux_mpi_omp |
| Mac 64bit | OpenMP | CPUServer_mac |
| Windows7 64bit | OpenMP | PBVR_Server_win.exe |
| K Computer *1 | OpenMP | CPUServer_k_omp |
| | MPI+OpenMP | CPUServer_k_mpi_omp |

| | | |
|---|---|---|
| FX10 *1 | OpenMP | CPUServer_fx10_omp |
| | MPI+OpenMP | CPUServer_fx10_mpi_omp |
| BX900 *1 | OpenMP | CPUServer_bx_omp |
| | MPI+OpenMP | CPUServer_bx_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.

### 2.2.2 Installation from Source Code

As with PBVR Filter, PBVR Server can be installed from the source code. If this is desired, uncompress *server.tgz*, which contains the source, to an arbitrary directory. Compile the extracted source using suitable Makefiles in the list below, and copy the generated load modules to an arbitrary directory that is specified in PATH environment variable. When the copying operation is finished, the installation is complete.

Table 4 List of Makefiles for the PBVR Server program

| **Platform** | **Parallelization** | **Name of Makefile** |
|---|---|---|
| Linux 64bit *1 | OpenMP | makefile.linux_omp |
| | MPI+OpenMP | makefile.linux_mpi_omp |
| Mac 64bit *3 | OpenMP | makefile.mac |
| K Computer *2 | OpenMP | makefile.k_omp |
| | MPI+OpenMP | makefile.k_mpi_omp |
| FX10 *2 | OpenMP | makefile.fx10_omp |
| | MPI+OpenMP | makefile.fx10_mpi_omp |
| BX900 *2 | OpenMP | makefile.bx_omp |
| | MPI+OpenMP | makefile.bx_mpi_omp |

*1. The KMATH_RANDOM library for the Linux environment is compiled using OpenMPI. To use other MPI libraries, recompile KMATH_RANDOM as follows.

[Step 1]

Modify the following path in order to suit the computing environment.

'./server/kmath_linux/random/Makefile.machine'

[Step 2]

```
$ cd ./server/kmath_linux/random/c++
$ make
```

When the compilation of KMATH_RANDOM or NTL fails, modify the following lines in the Makefile in order to compile PBVR Server without these libraries.

Delete '–DKMATH' from lines starting with 'CXXFLAGS='

Delete '-lkm_random –lntl' from lines starting with 'LDFLAGS='

Note that in this case, the generation of pseudo random numbers in a node is dependent to other nodes when the MPI+OpenMP version is used. Therefore, the use of

KMATH_RANDOM is recommended for the MPI+OpenMP version if there is no specific reason to avoid it.

*2. The Makefiles for supercomputers are designed for use with cross-compilers on login nodes.

*3. The Mac version does not use KMATH_RANDOM nor NTL.

### 2.2.3 Installation in Windows

The followings show how to compile the Windows version of PBVR Server. The Windows version does not make use of KMATH_RANDOM nor NTL.

1) Extract *server* on a Windows machine that has Visual Studio 2013 installed.
2) On the *server* directory, run *¥¥project_vc¥PBVR_Server_win.sln* and launch Visual Studio 2013.
3) Choose **Release** and **x64** from the pull-down list as shown in Figure 2.

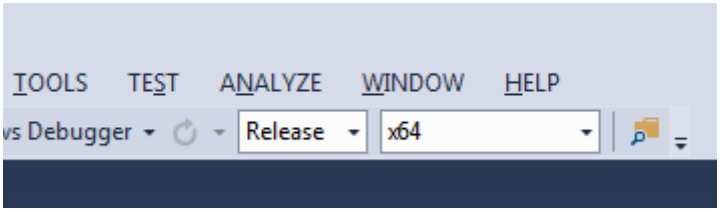



Figure 2 Build configuration of Visual Studio 2013

4) Go to the menu **Build** > **Build Solution**.
5) The load module *PBVR_Server_win.exe* is created under *¥¥project_vc¥x64¥Release*.

## 2.3 PBVR Client

PBVR Client is implemented in C++ and makes use of OpenGL.

### 2.3.1 Installation of Prebuilt Binaries

The following table lists the load modules stored in the *client* directory of the load module package. Choose the suitable load modules, and copy them to a directory that is specified in PATH environment variable.

Table 5 List of load modules for PBVR Client

| **Platform** | **Parallelization** | **Name of load module** |
|---|---|---|
| Linux 64bit | OpenMP | PBVRViewer_linux |
| Mac 64bit | OpenMP | PBVRViewer_mac |
| Windows7 64bit *1 | OpenMP | PBVRViewer_win.exe |

*1. For the Windows version, copy also ‘glut32.dll’ to the destination directory.

### 2.3.2 Installation from Source Code

Uncompress *client.tgz* to an arbitrary directory and compile it using the proper compilation script that is specified in this section. Once the load modules are generated, copy them to a suitable directory that is specified in PATH environment variable, and the installation is complete.

### 2.3.3 Installation in Linux and Mac

Execute the compilation script as follows.

```
$ cd ./client/PBVRViewer
$ sh build_cpu.sh
```

To recompile the source from scratch (rather than skipping object files already compiled in the past), run the script *rebuild_cpu.sh*.

### 2.3.4 Installation in Windows

1) Install GLUT
   i) Download *glut-3.7.6-bin_x64.zip(64bit)* from the link below.
   http://ktm11.eng.shizuoka.ac.jp/lesson/modeling.html
   ii) Extract the following files:
   *glut.h*
   *glut32.lib*
   *glut32.dll*
2) Install glui
   i) Open folder *glui-2.36¥src¥msvc*.
   ii) Open *glui.sln* with Visual Studio 2013.
   iii) Choose **Release x64** for build configuration as shown in Figure 2.
   iv) Right-click on **glui** in **Solution Explorer**, go to **Configuration Property** > **C/C++**, and add the file *glut.h* to the **Additional Include Directory** list.
   v) Open **Solution Explorer,** right-click on ‘glui’, and build the project.
3) Install KVS
   i) Open folder *KVS-src-1.1.1* and copy *build_win.bat* to a suitable folder.
   ii) Open *kvs.conf* and change the line ‘KVS_SUPPORT_GLEW = 1’ to ‘KVS_SUPPORT_GLEW = 0’.
   iii) Specify *folder_storing_PBVRVlewer.sln¥kvs* for *KVS_DIR* in the copied *build_win.bat* in i).
   iv) Specify the folder storing *glut.lib* for ‘KVS_GLUT_DIR’ in the copied *build_win.bat* in i).
   v) Go to **Start Menu** > **Visual Studio 2013** > **Visual Studio Tool** and open **VS2013 x64 Native Tools command prompt**.

vi) Go to ‘KVS-src-1.1.1’ and execute the .bat file created in i) iii) iv).

4) Install PBVR Client

i) Choose **Release x64** for the configuration as shown in Figure 2.

ii) Go to **Solution Explorer**, right-click on **PBVRViewer**, open the property **configuration property** > **C/C++**, and add the include path of ‘glut’ as an **additional include directory**.

iii) Go to **Solution Explore**r, right click on **PBVRViewer**, open the property **configuration property** > **linker**, and add the ‘lib’ path for ‘glut’ as an **additional library directory**.

iv) Go to M**enu** > **Build Solution** to execute the compilation.

v) A load module *PBVRViewer.exe* is created under *¥¥x64¥Release¥*.

vi) Before executing the module, place *glut32.dll* in the same directory.

# 3 PBVR Filter

## 3.1 Overview

PBVR Filter is independent from the PBVR system. PBVR Filter divides time-series volume data that will become the input of 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 grids data into user-specified octree regions in order to generate the input files of parallel processing by PBVR Server.

## 3.2 Data Decomposition Model

As shown in Figure 3, the octree data structure divides each edge of a cuboid in half, recursively. Therefore, each cuboid has eight child cuboids while each child cuboid has a single parent cuboid.

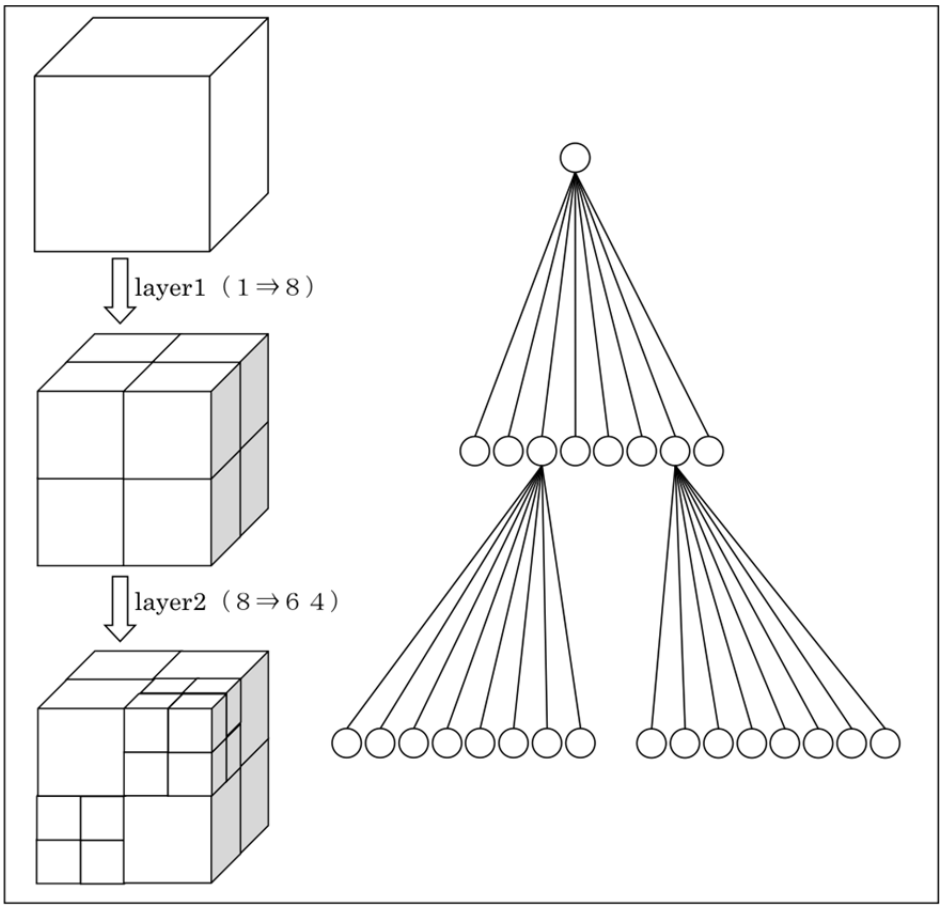



Figure 3 Space partitioning with the octree data structure

As shown in Figure 4, the boundaries of the child-cuboids are computed by dividing the sum of the minimum and maximum coordinate values by two. Given a point in the domain, the cuboid containing the point can be determined by comparing the coordinates of the vertex and the boundaries.

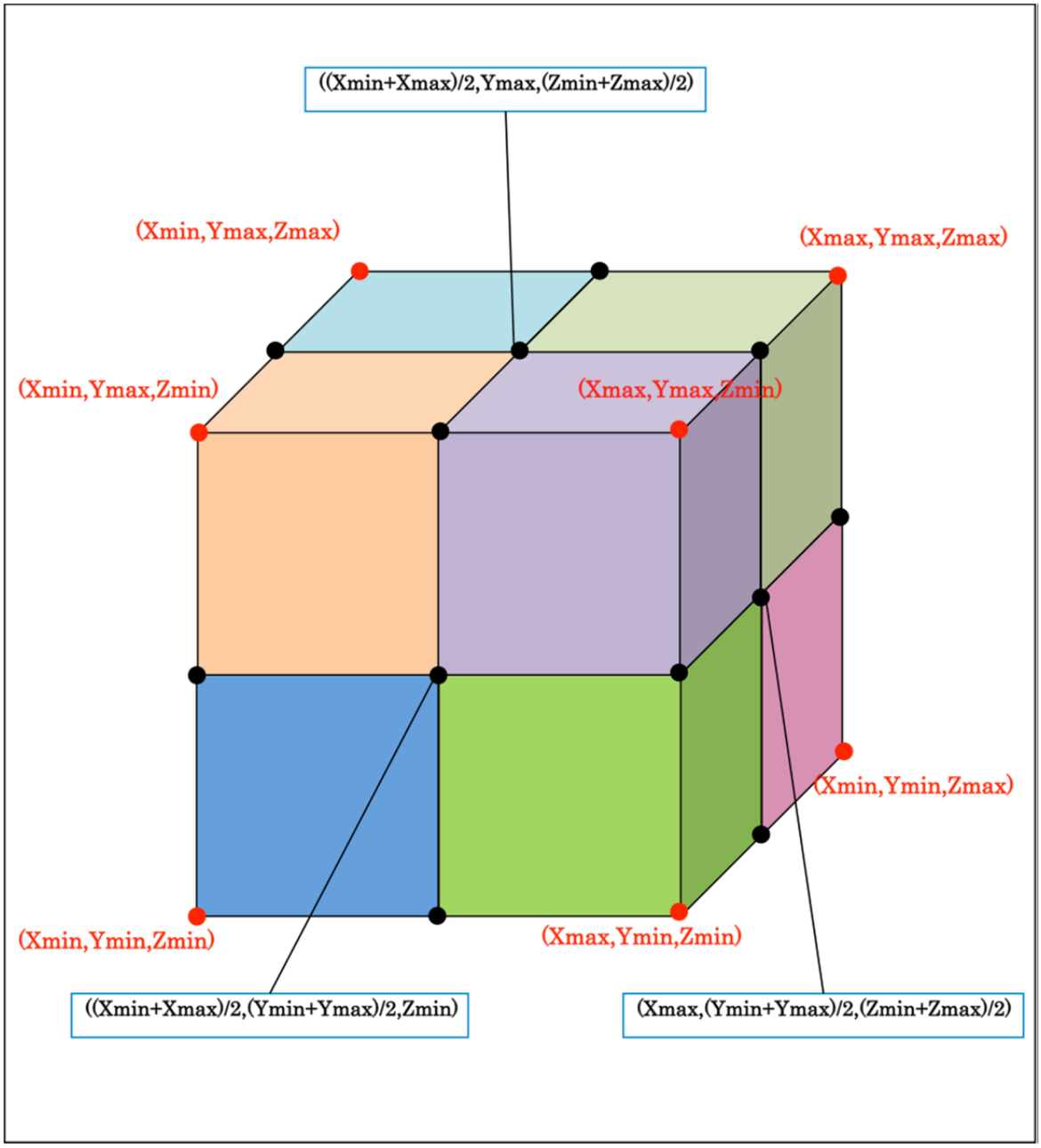



Figure 4 Coordinates of the boundaries in the octree data structure.

## 3.3 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 should be specified in the command line when launching PBVR Filter. When no parameter file name is given or a non-existent file name is provided, the execution of PBVR Filter fails.

**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 variable 'OMP_NUM_THREADS'.

*2. In Windows, the command can be provoked from Visual Studio 2013 x64 Native Tools command prompt.

## 3.4 File Formats

This section describes the file formats that are read/written by PBVR Filter. All binary format data in input/output files are given in single precision, without a header/footer, and in little endian. Three file formats are available: the SPLIT format (that actually make use of kvsml format), the sub-volume aggregate format, and the step aggregate format. (See Figure 5.) The SPLIT format generates independent files for each time step, for each sub-volume. However, in this format, the number of files grows explosively as the number of layers in octree increases. This problem can be avoided by using either of the other two file formats. The sub-volume aggregate format aggregates files at different time steps (but of the same sub-volume) to a single file. Conversely, the step aggregate format aggregates files of different sub-volumes (at the same time step) to a single file. The following sections explain these three file formats in detail.

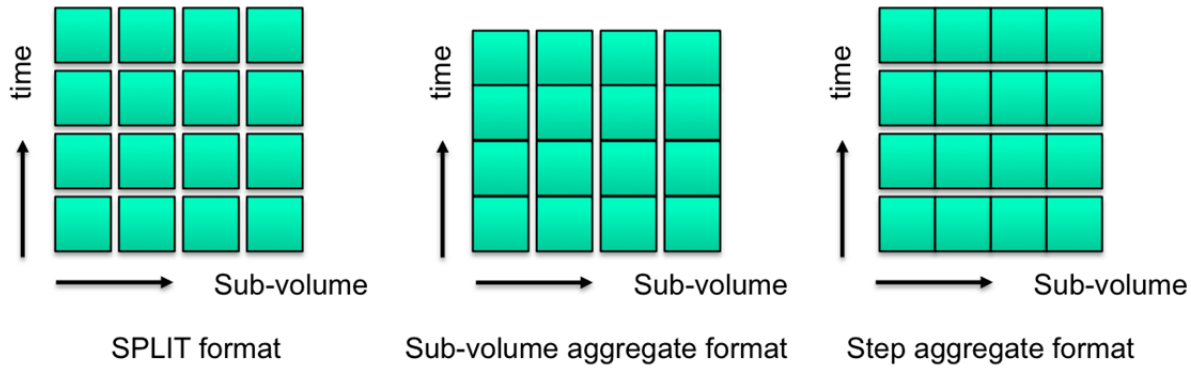



Figure 5 The output file formats available for PBVR Filter

### 3.4.1 Input Data Format

PBVR Filter can take the following AVS data formats as input.

1) AVSFLD binary data
2) AVSUCD binary data *1,2

*1. AVSUCD binary data with “data” format can be used. However, the geom and the data_geom formats are not supported.

*2. 2D/3D elements in Table 8 and their mixed elements are supported.

For more details of the AVS data formats, refer to the *AVS manual* or the following link (in Japanese).

http://www.cybernet.co.jp/avs/products/avsexpress/dataformat.html

### 3.4.2 Endian

The binary files used in PBVR Filter are in little endian. On a big endian machine, if input data files do not use the little endian format, conversion is necessary.

## 3.4.3 Filter Output Information File (.pfi)

A .pfi file is a binary data file that summarizes the information of the input volume.

| |
|---|
| 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 Table 8.

*2. Set the int value to 0-2 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, when the input file format is sub-volume aggregate format.

*4. The number of sub-volumes is $8^{n\_layer}$. Examples follow.

*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

### 3.4.4 SPLIT File Format

When the SPLIT file format is used, two files are produced for each sub-volume. The first is called an element configuration file. This file describes which of the nodes constitutes each cell. The second is called a node coordinate file, which specifies the coordinates of the nodes. In addition, each sub-volume gets another file for each time step. This file, which is called a variable file, assigns the values of variables (physical quantities) to each node. All these three types of files are formatted as a kvsml file. It is worth noting that the total number of files can be calculated as follows:

*The number of sub-volume* × 2 + *the number of sub-volume* × *the 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.

#### 3.4.4.1 File Name 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* | : variable file（binary format） |

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

| | |
|---|---|
| ‘prefix’ | : arbitrary string of characters that are allowed for a file name |
| ‘XXXXX’ | : number of steps（in 5 digits） |
| ‘YYYYYYY’ | : index for sub-volume (in 7 digits) |
| ‘ZZZZZZZ’ | : total number of sub-volumes（in 7 digits） |

### 3.4.4.2 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>
```

### 3.4.4.3 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* |

### 3.4.4.4 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* |

### 3.4.4.5 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* |

### 3.4.5 Sub-volume Aggregate Format

In sub-volume aggregate format, the information of element configuration, node coordinates, and variables of all time steps are gathered in a single file for each sub-volume. By specifying the 'Number of file' (which is explained in Section 3.5), one can aggregate the information of several sub-volumes into arbitrary number of files from one to the number of sub-volumes. (If *n_layer* is 7, then the number of files is 2,097,152.)

#### 3.4.5.1 Naming Convention

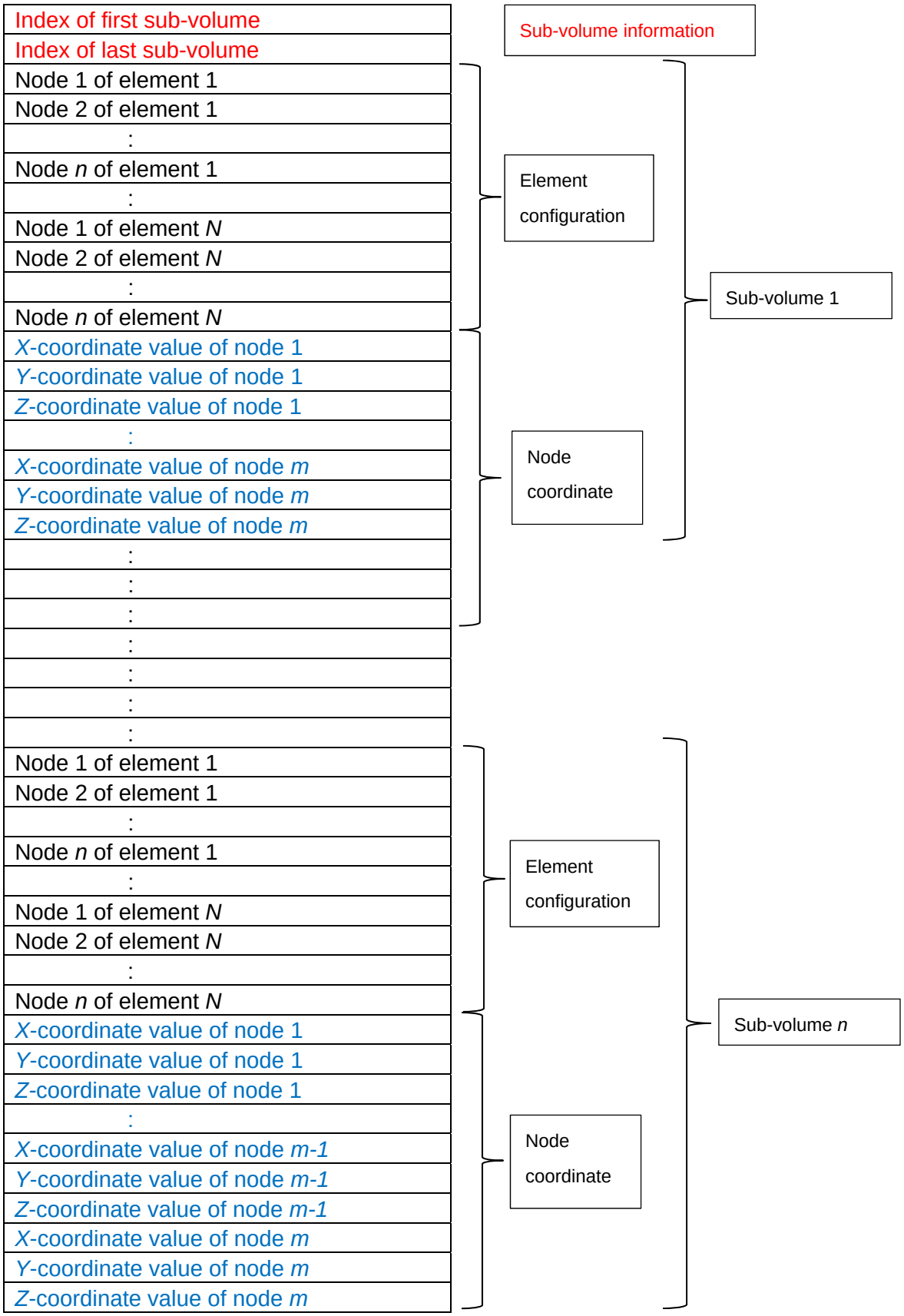
In PBVR, files in the sub-volume aggregate format have the following naming convention.

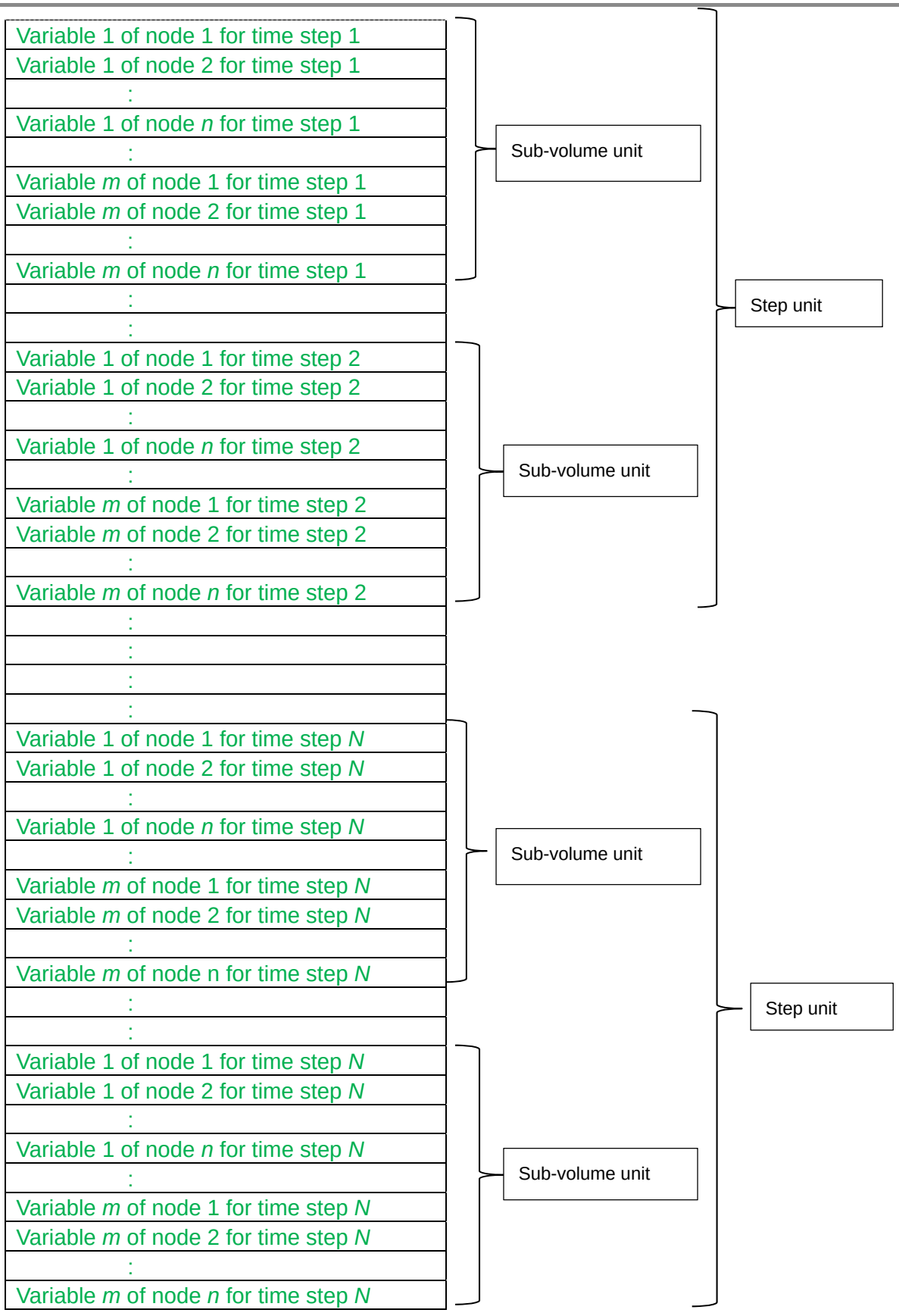
*prefix_YYYYYYY_ZZZZZZZ.dat* (A binary file）

'prefix', 'XXXXX', 'YYYYYYY', and 'ZZZZZZZ' should be replaced with the following strings.

| | |
|---|---|
| prefix | : arbitrary string of characters that are allowed for a file name |
| YYYYYYY | : file number（in 7 digits） |
| ZZZZZZZ | : total number of files（in 7 digits） |

### 3.4.5.2 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

## 3.4.6 Step Aggregate Format

The step aggregate format is made up of by an element configuration file and a node coordinate file. These two files contain the information of all the sub-volumes. A variable file is produced for each step. Therefore, the total number of files becomes the number of steps + 2.

### 3.4.6.1 File Name

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

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

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

prefix : arbitrary string of characters that are allowed for a file name
XXXXX: number of steps（in 5 digits）

### 3.4.6.2 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

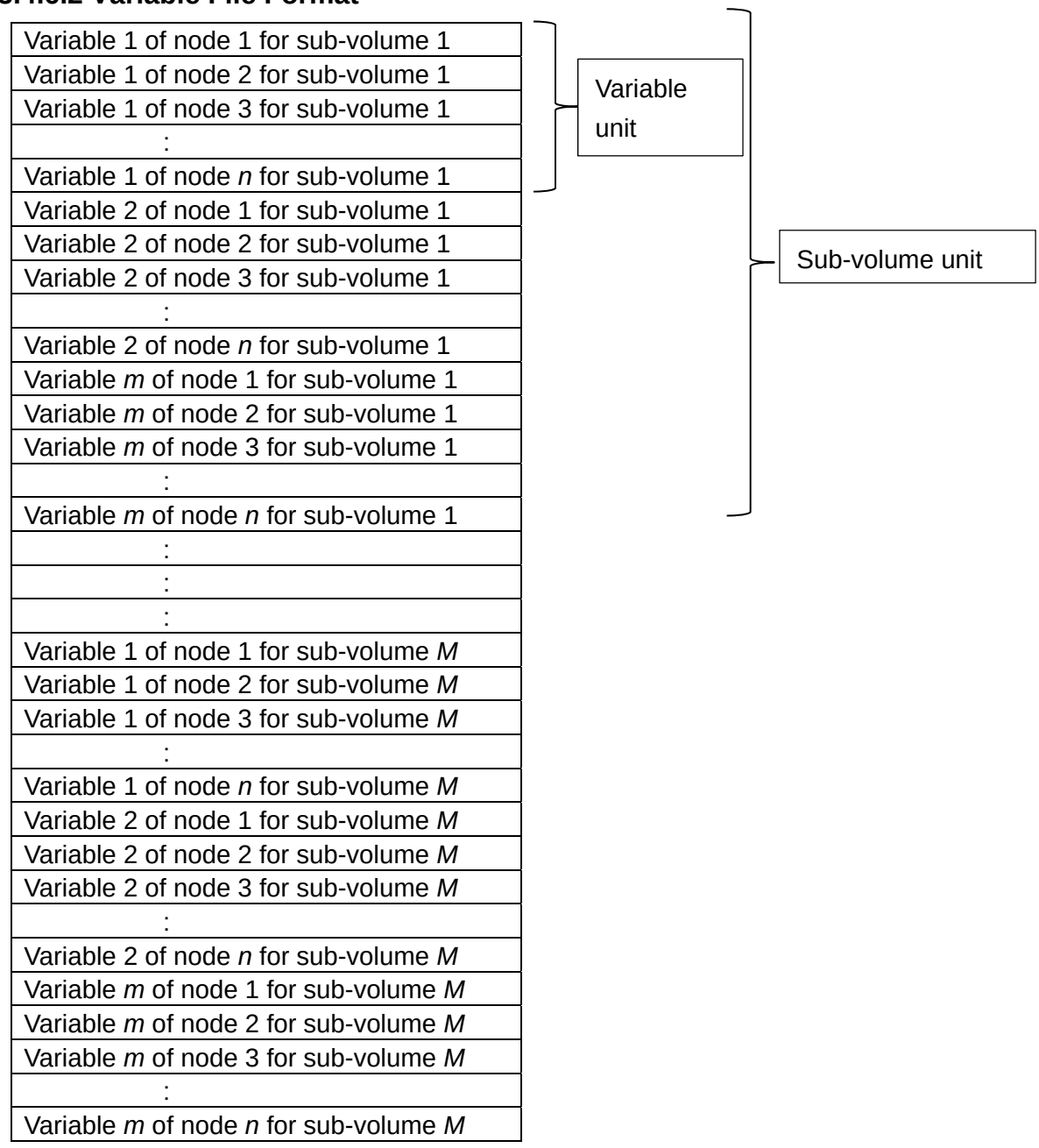
### 3.4.6.1 Node Coordinate File Format

| |
|---|
| *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* |

Node unit

Sub-volume unit

### 3.4.6.2 Variable File Format



## 3.5 Parameter File

The parameter file is in ASCII format. By specifying the file name in the command line when provoking PBVR Filter, the parameters inside are set as input to PBVR Filter. Table 6 lists the available parameters.

Table 6　List of PBVR Filter input parameters

| Parameter name | Parameter detail | Default value | Notes |
|---|---|---|---|
| in_dir | Input file directory | '.' | Directory path of input files *1 |
| field_file | .fld file name | - | Only for structured grid data *2,3 |
| in_prefix | Input file prefix | 'input.' | Only for unstructured grid data |
| in_sufix | Input file suffix | '.dat' | Only for unstructured grid data |
| out_dir | Output file directory | '.' | Directory path of output files *1 |
| out_prefix | Output file prefix | 'output.' | |
| Format | Step number format | '%05d' | Only for unstructured grid data |
| start_step | Starting step number | '1' | |
| end_step | Ending step number | '1' | |
| n_layer | Number of octree layer | '0' | An integer from '0' to '7' |
| output_type | File format | '0' | '0': SPLIT format<br>'1': sub-volume aggregate<br>'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. *4 |
| 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 *4 |

| | | | |
|---|---|---|---|
| mpi_div | Configuration of 2D MPI parallel processing | '2' | '0': defined by mpi_volume_div' and mpi_step_div.<br>'1': automatic with priority on sub-volume decomposition.<br>'2': automatic with priority on step decomposition.<br>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<br>'1': data with multiple element types |
| temp_delete | Flag on temporary files produced by processing mixed element data | '1' | '0': keep temporary files<br>'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. Only the parameters 'nstep', 'ndim', 'dim1', 'dim2', 'dim3', 'veclen', 'coord [123]', and 'variable' are referred.

*3. When 'field_file' is specified, input data is recognized as structured grid data. Otherwise, input data is treated as unstructured grid data. When input data is structured grid data, the output data is converted to unstructured grid data with linear hexahedral elements when the space is three dimensional. In contrast, it is converted to unstructured grid data with linear quadrilateral elements for a two-dimensional space.

*4. 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.

## 3.6 MPI Parallel Processing

This section describes the ways of dividing the computation in MPI parallel processing. As an example, consider processing data with 50 steps × 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 time steps by the number of processes.

  **Example:**

  Since 8 processes exist, each process treats 6 steps × 8 sub-volumes, or 7 steps × 8 sub-volumes.

- When the number of processes is larger than the number of time steps, each process

handles a single time step. The number of sub-volumes for each process is specified in the following manner. First, divide the number of processes by the number of time steps. Then, divide by the quotient the number of sub-volumes.

**Example:**

When 128 processes are used, PBVR Filter works with 50 × 2 = 100 processes (with the residue of 28 processes), and each process treats 1 step × 4 sub-volumes.

2) Partitioning the set of sub-volumes first

- When the number of processes is equal to or less than the number sub-volumes, divide all the sub-volumes by the number of processes.

  **Example**:

  When 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 a single sub-volume. The number of time steps for each process is specified in the following manner. First, divide the number of processes by the number of sub-volumes. Then, divide by the quotient the number of time-steps.

  **Example:**

  When 128 processes are used, the PBVR Filter program works with 8 × 16 = 128 processes (with the residue of 0 process), and each process treats 3 steps × 1 sub-volume or 2 steps × 1 sub-volume

3) Employing a parallelization that is more complex

- When the parallel processing number ‘mpi_volume_div’ and ‘mpi_step_div’ are specified, an error occurs if ‘mpi_volume_div × mpi_step_div’ does not agree with the number of processes..

## 3.7 Execution in Staging Environment of K computer

This section describes how to execute PBVR Filter in the staging environment on 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 may write to a single file. In such a case, the output location should be specified in a shared domain on the local file system that is accessible from all the processes.

### 3.7.1 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 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.)

## 3.7.2 Input/Output Files and Directories

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

Table 7　Table of input-output files and directories on 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 limit of the staging environment (800 files/node, 14GB/node).

*3. The output directory is always a rank directory.

## 3.8 Unstructured Grid Data with Mixed Elements

When unstructured grid data contains several element types, PBVR Filter firstly generates UCD binary data for each element type, and then divides the UCD binary data with a single element type into sub-volumes, which are read by the 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, and have file names with a 2 digit prefix that represents the element type. The following list shows the names of the elements and the corresponding prefix.

Table 8　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 |

When the input data with the above parameter file consists of linear tetrahedral elements and quadratic tetrahedral elements, the following output files are generated.

Table 9　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_- |

# 4 PBVR Server

## 4.1 Overview

PBVR Server reads sub-volume files, which are produced by PBVR Filter, and performs parallel visualization with the PBVR technique to generate particle data as visualization results.

## 4.2 Launching PBVR Server

PBVR can operate in supercomputers both in batch mode, which generates only particle data in batch processing, and in client-server mode, which generates particle data in interactive processing 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 the client-server mode on the same machine. The followings show how to launch PBVR Server.

**Examples:**

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

$ mpiexec -n N ./CPUServer

Launch the OpenMP version

$./CPUServer

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

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

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

Table 10 List of command line options for the PBVR Server program

| Option | Launch mode | Possible parameters | Default parameters | Functionality |
|---|---|---|---|---|
| -h | SB | - | - | This shows the list of available options and parameters |
| -B | B | - | - | To launch in the batch mode |
| -pa | B | File name | - | Visualization parameter file |
| -sl | B | 1-99 | 1 | Sub-pixel level *2 |

| | | | | |
|---|---|---|---|---|
| -S | B | u, m | u | Method for sampling particles<br>u: uniform sampling<br>m: metropolis sampling |
| -plimit | B | 1-99999 | 10 | Maximum number of particles ($10^6$) *2 |
| -vin | B | File name | - | Input volume data (a .pfi file) *2 |
| -pout | B | File name | ./ | Name of the output particle data file *3 |
| -p | S | Port number | 60000 | Port number for socket communication |
| -viewer | B | 100-9999<br>×100-9999 | 620×620 | Viewer resolution |

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

*2. If this option conflicts with the option in the parameter file specified with '−pa', the latter is ignored.

*3. 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.

### 4.2.1 Launching PBVR Server in Batch Mode

When the 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 ./CPUServer –B -vin ./data/case.pfi -pout ./output/case –sl 2 –rl 2 –plimit 10 –pa ./param.in

In this example, the input data *./data/case.pfi* is processed with the visualization parameter file *./param.in* and with command line options, '–sl 2 –rl 2 –plimit 10', to output the following particle data.

./output/case_XXXXX_YYYYYYY_ZZZZZZZ.kvsml

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

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 the batch mode is executed by using this file as it is, or with desirable modifications to the parameters.

### 4.2.2 Launching PBVR Server in Client-Server Mode

When the command line option ‘–B’ is not specified, PBVR Server is launched in the client-server mode. See the following example.

```
$ mpiexec -n 5 ./CPUServer
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 waits for a socket communications with PBVR Client, launch the PBVR Client in another terminal. In the client-server mode, 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 ./CPUServer -p 55555
```

### 4.2.3 Connecting Client and Server via Socket Communication

#### 4.2.3.1 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 ./CPUServer
```

Step 2 [Launch PBVR Client]

```
machineA> ./PBVRViewer -vin filename
```

#### 4.2.3.2 Remote Connection between Two Machines

The following example shows how to launch PBVR Client on a machine ‘machineA’ and PBVR Server program on another machine ‘machineB’ where the two machines are located at distant places. This example uses SSH port forwarding to connect the port 60000 of ‘machineA’ to the port 60000 of ‘machineB’. In this way, PBVR Server and Client on the two machines cooperate through the default port number 60000. Once the SSH port forwarding is established, the launching procedure is basically the same as that in stand-alone mode. In Windows, SSH port forwarding can be setup by using a third-party application such as TeraTerm or Putty.

Step1 [SSH port forwarding from machineA to machineB]

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

（Forwarding the 60000 port of machineA to the 60000 port of machineB）

Step2 [Launch PBVR Server]

machineB> mpiexec –n 5 ./CPUServer

Step3 [Launch PBVR Client]

machineA> ./PBVRViewer –vin filename

#### 4.2.3.3 Remote Connection with Several Machines

This section provides an example of connecting PBVR Server and PBVR Client on two remote machines ‘machineA’ and ‘machineB’ via ‘machineC’ for some reason, e.g. security. Once the SSH port forwading is established, the launching method is basically the same as the stand-alone mode, as wtih the two point remote concection mentioned before.

Step1 [SSH port forwarding from machineA to machineC]

machineA> ssh -L 60000:localhost:60000 username@machineC

（Forwarding the 60000 port of machineA to the 60000 port of machineC）

Step2 [SSH port forwarding from machineC to machineB]

machineC> ssh -L 60000:localhost:60000 username@machineB

（Forwarding the 60000 port of machineC to the 60000 port of machineB）

Step3 [Launch PBVR Server]

machineB> mpiexec –n 5 ./CPUServer

Step4 [Launch PBVR Client]

machineA> ./PBVRViewer –vin filename

#### 4.2.3.4 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

### 4.2.3.5 Connecting to Pre-post Server of K computer

This section shows an example of connecting a PC ('machineA') in s laboratory to the data processing server of the K computer (Pre-post server pps3) via the login node of the K computer (klogin).

Step1 [SSH port forward from machine to klogin]
machineA> ssh -L 60000:localhost:60000 username@k.aics.riken.jp
（Forwarding the 60000 port of machineA to the 60000 port of klogin）
Step2 [SSH port forward from K login node to pre-post server]
klogin> ssh -L 60000:localhost:60000 username@pps3
（Forwarding the 60000 port of klogin to the 60000 port of pps3）
Step3 [Launch PBVR Server]
pps3> mpiexec –n 5 ./CPUServer
Step4 [Launch PBVR Client]
machineA> ./PBVRViewer –vin filename
（Forwarding the 60000 port of klogin to the 60000 port of pps3）

### 4.2.3.6 Connecting with BX900

Members of Japan Atomic Energy Agency can connect a computer of the agency to BX900 via the intranet in order to use BX900 as PBVR Server with port forwarding. Contact the agency's information management room (ccse-tokaii-unyou@ml.jaea.go.jp) to apply for the visualization node (bxviz11). In addition, inform your machine's IP address. After the application, establish the connection in the following manner.

[Step 1] Login to BX900 with SSH
machine> ssh username@bxviz11.tokai-sc.jaea.go.jp

[Step 2] Write a shell script that will be used with qsub as BX900's interactive job.
See the following example of a shell script 'run.sh'.

```
#!/bin/sh
#@$-q t32
#@$-C JOB
#@$-lP 32
#@$-lp 1
#@$-cp 30:00
#@$-oi
#@$-eo
#@$-o output

cd ${QSUB_WORKDIR}
```

```
NPROC=32
NTHREAD=1

PARALLE=${NTHREAD}
OMP_NUM_THREADS=${NTHREAD}
export PARALLEL
export OMP_NUM_THREADS

cd ${QSUB_WORKDIR}
hostname
mpiexec -n ${NPROC} ./CPUServer.bx_mpi_omp -p 61000
```

[Step 3] In BX900, execute qsub with the above shell script being the interactive job.

BX900> qsub –l run.sh

Execute the hostname command in the job of the qsub, and note the result. In what follows, assume that the result is bx2049.

[Step 4] Start SSH port forwarding from the client terminal.

machine> ssh –p 61022 -L 61000:bx2049:61000 username@bxviz11.tokai-sc.jaea.go.jp

[Step 5] Launch PBVR Client

machineA> ./PBVRViewer_linux -p 61000 –vin pfi_filename

### 4.2.3.7 Local Connection in Windows

This section shows how to launch both PBVR Sever and PBVR Client on a single Windows machine. The Visual Studio 2013 x64 Cross Tools command prompt in Visual Studio 2013 is used as the terminal for launching the programs.

Step1 [Launch PBVR Server]

Windows> CPUServer.exe

Step2 [Set the client parameter for Windows]

Windows> set TIMER_EVENT_INTERVAL=1000

Step3 [Launch PBVR Client]

Windows> PBVRViewer.exe –vin filename

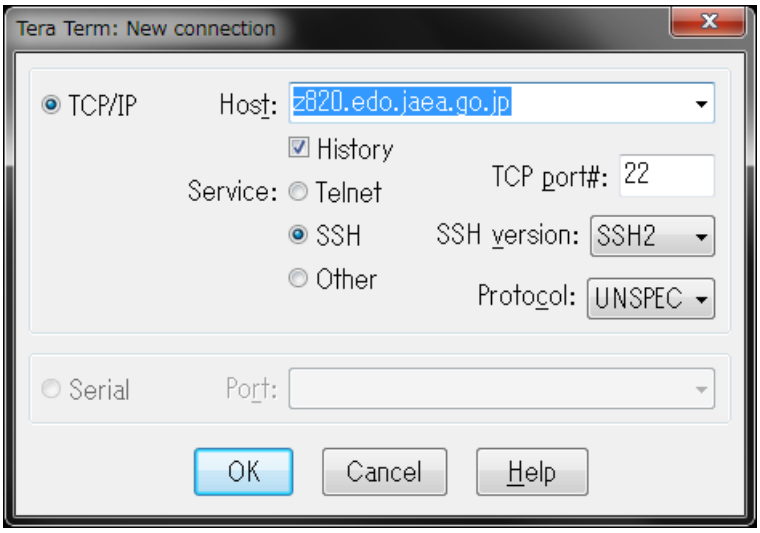
Another way of launching PBVR Server and Client is to execute a batch file with the following lines.

```
set TIMER_EVENT_INTERVAL=1000
start PBVR_Server_win.exe
PBVRViewer.exe –vin filename
```

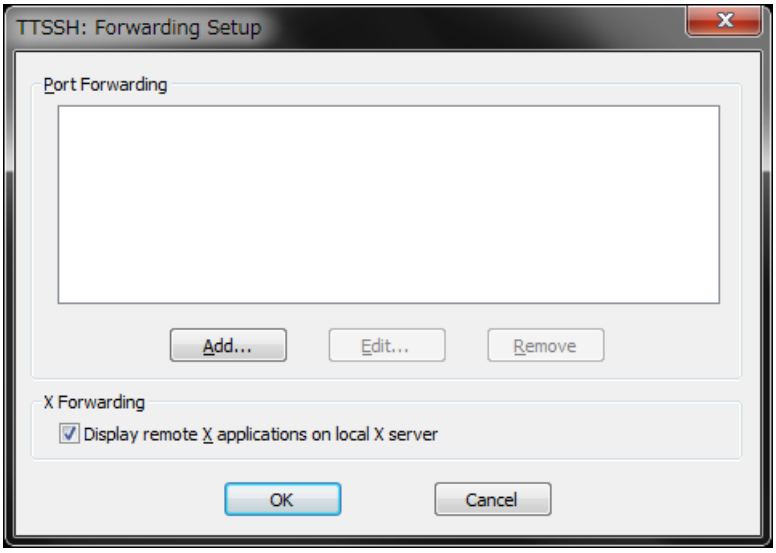
### 4.2.3.8 Remote Connection from Windows Client

To connect PBVR Client in a Windows machine to PBVR Server in a remote machine, setup port forwarding with the help of an SSH client software such as TeraTerm or Putty. The following shows an example for TeraTerm.

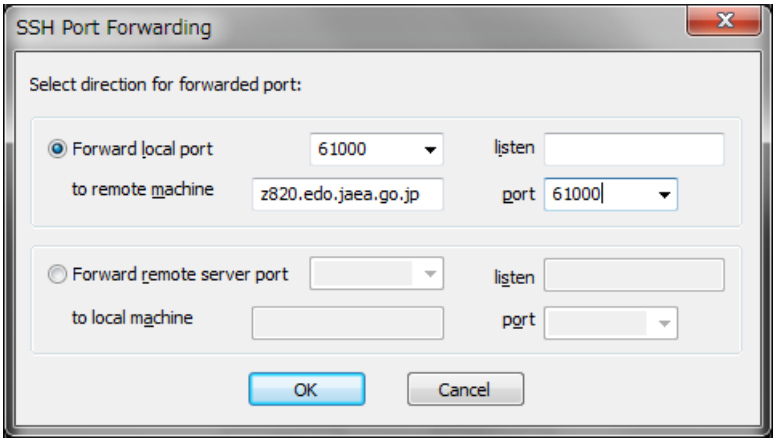
1) Launch TeraTerm and hit cancel in the “New connection” dialog.



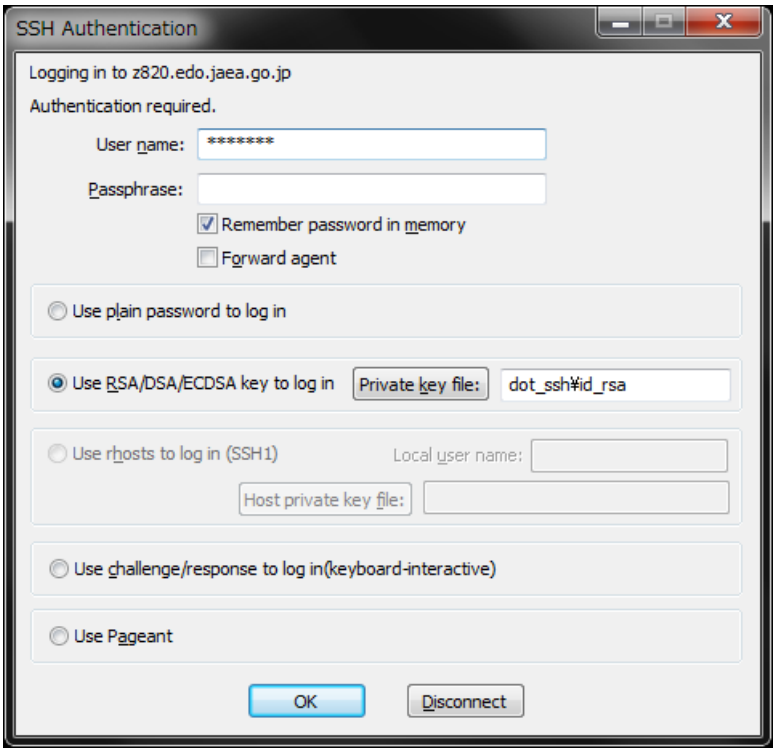

2) Select **Setup** > **SSH Transfer** from the menu bar. Click **Add…** in the **Forwarding Setup** dialog.




3) In the **Select Direction for Forwarded Port** dialog, select **Forward Local Port** and enter the port number to be used for PBVR Client. In the **to remote machine** text field, enter the domain name or the IP address of the server. In the **port** field, enter the port number to be used on PBVR Server. Click on **OK** to complete the setup of port forwarding.




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 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**.




The following procedures show how to launch PBVR Server and Client after establishing port forwarding. This example uses the Visual Studio 2013 x64 Cross Tools command prompt in Visual Studio 2013 as the terminal for launching PBVR Client.

Step1 [Launch PBVR Server]

Server> mpiexec –n 4 ./CPUServer –p port_number

Step2 [Set a client parameter for Windows]

Windows> set TIMER_EVENT_INTERVAL=1000

Step3 [Launch PBVR Client]

Windows> PBVRViewer.exe –vin filename –p port_number

Note that PBVR Client on a Windows machine can be launched also by executing a batch file with the following lines.

```
set TIMER_EVENT_INTERVAL=1000
PBVRViewer.exe –vin filename –p port_number
```

# 5 PBVR Client

## 5.1 Overview

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 geometric primitives generated in PBVR Server. Further, PBVR Client renders the data using OpenGL. PBVR Client also gets visualization parameters (the sub-pixel level, 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. 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.

## 5.2 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. When PBVR Client starts, it opens five panels: **Viewer**, **Main panel**, **Transfer Function Editor**, **Time Panel**, and **Animation Panel**.

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

Launch PBVR Client in stand-alone mode
$ ./PBVRViewer [particle data file name *3] [command line options *4]

*1. Client-server mode requires starting PBVR Server beforehand.
*2. The file name for sub-volume can be specified with the absolute or the relative path to the .pfi file.
*3. The particle data file must be stored in a directory without another kvsml file.
*4. In stand-alone mode, if the sub-pixel level (-sl), and the repeat level (-rl) are not specified in the command line options, the particle size becomes incorrect in the rendering process.

Table 11 List of command line option for client

| **Option** | **Launch mode *1** | **Parameter value** | **Default parameters** | **Function** |
|---|---|---|---|---|
| -h | C, S | - | - | Display the list of options and parameters |

| | | | | |
|---|---|---|---|---|
| -sl | C | 1-99 | 1 | Subpixel level |
| -rl | C | 1-99999 | 1 | Repeat level |
| -S | C | u, m | u | Particle sampling method<br>u: uniform sampling<br>m:metropolis sampling |
| -tdata | C | all, div | all | Particle data transfer method<br>all: step batch transmission,<br>div: sub-volume divide forwarding） |
| -pa | C, S | file name | - | Specify a parameter file |
| -vin | C | file name | - | Name of the .pfi file of input volume data |
| -tf *2 | C | file name | - | Name of the transfer function file |
| -p | C | port number | 60000 | Port number of socket communication |
| -viewer | C, S | 100-9999 ×100-9999 | 620×620 | Viewer resolution |
| -shading | C, S | {L/P/B}, ka, kd, ks, n | - | Shading method *4 |
| -pout *4 | C, S | directory name | ./pout | Output directory for particle data |

*1. C and S denote client-server mode and stand-alone mode, respectively.

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

*3. This argument specifies the shading parameters. PBVR's shading process gives shading effect by emphasizing the color tone of rendered objects. The reflection of light is modeled as the sum of diffusion reflection, specular reflection and ambient light. The parameters specified in the command line argument modify the intensity of these three components.

L: Lambert Shading

This method ignores specular reflection in the shading process.
Parameters ‘ka’ and ‘kd’ are the coefficient for ambient and diffusion, respectively.
They can have a value between 0-1.

P：Phong Shading
This method adds the specular reflection to Lambert shading. Phong shading imitates smooth metal and mirrors. (This is sometimes called highlight).
Parameter ‘ka’, ’kd’,’ks’ (coefficients for specular reflection lying between 0-1) and ’n’ (strength of highlight lying between 0-100) are used.

B：Blinn-Phong Shading
This is a shading model that simplifies Phong shading. Parameters ‘ka’, ‘kd’, ‘ks’, and ‘n’ exist.

*4. This generates a series of particle data files that are named “[file name]_[time index]_[number of sub-volumes]_[sub-volume index].kvsml”, where the [file name] is the prefix specified this option. If this option is omitted, the prefix “client” will be specified automatically,

## 5.3 Terminating PBVR

### 5.3.1 Standard Termination

PBVR Client's rendering process for the time-series data starts from 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 step. However, PBVR Client ignores pressing **Ctrl+C** whenever the client-server communications are interrupted with the **Stop** button in **Time Panel**.

### 5.3.2 Forced Termination

When PBVR Server is terminated not 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 might become stuck. This can happen if the time step is not updated due to heavy particle generation processes or some other reason. In such a 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 PBVRViewer
   PID TTY          TIME CMD
19582 pts/6    00:00:00 PBVRViewer
$ kill -9 19582
```

[Force the termination of a PBVR Server process】

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

## 5.4 Using PBVR Client GUI

### 5.4.1 Viewer

As shown in Figure 6, **Viewer** displays the rendering result of particle data.

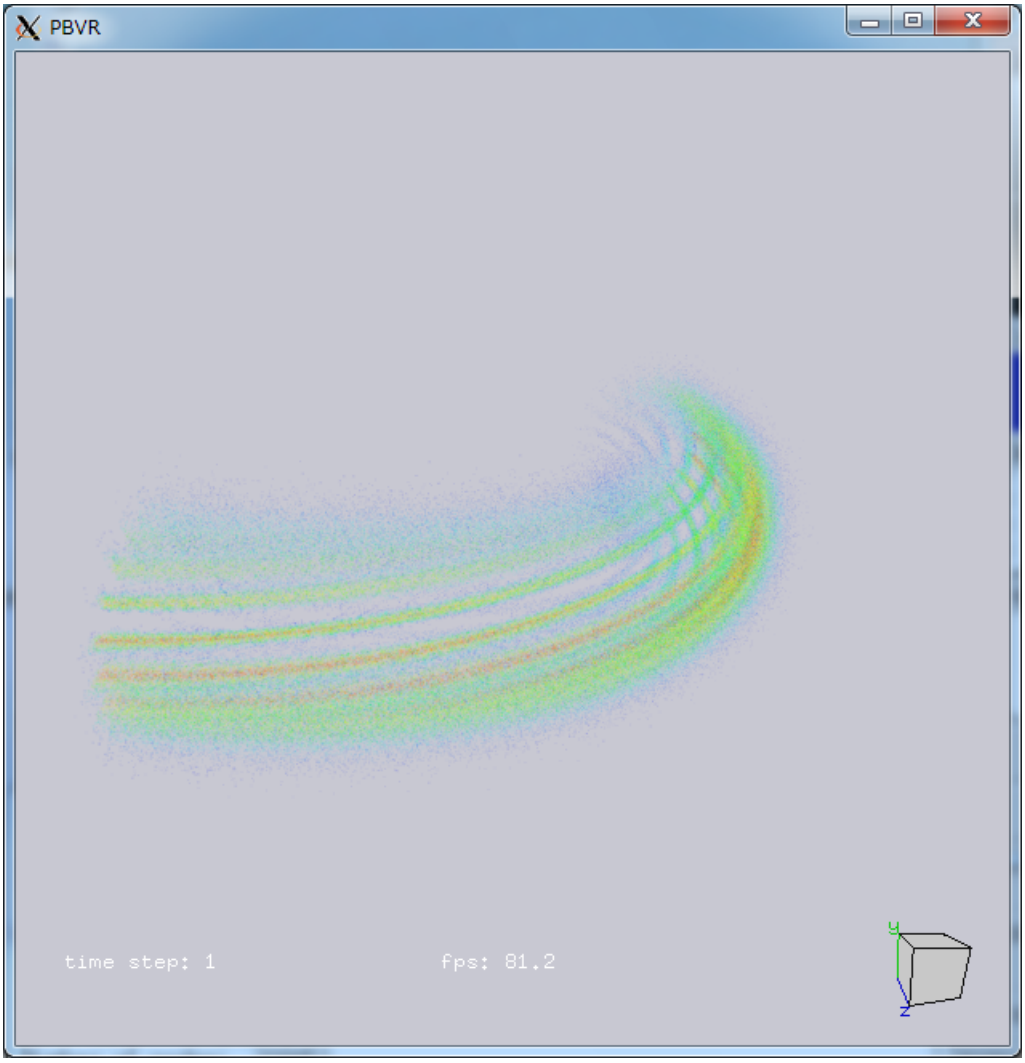



Figure 6 **Viewer**

[Operations]

Rotation: move the mouse while pressing the left-button

Translation: move the mouse while pressing the right-button

Zoom: scroll up/down the mouse wheel, or move the mouse up/down while pressing the **Ctrl** key

[Display]

**time step**: the time step of the displayed data

**fps**: the frame rate [frame/sec]

## 5.4.2 Main Panel

Figure 7 shows **Main panel** of PBVR Client. The items of the panel are described below.

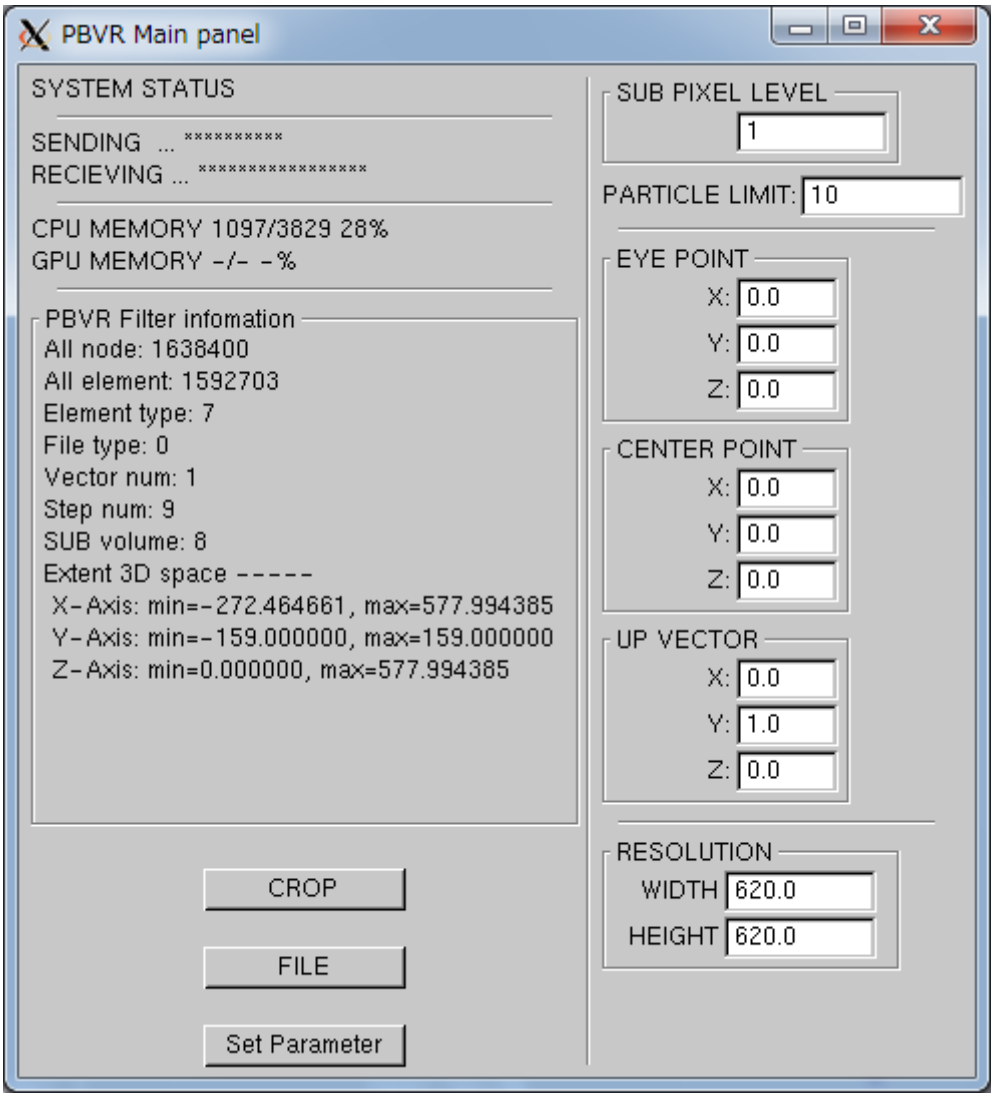



Figure 7 **Main panel**

- **SUB PIXEL LEVEL**

  Specifies the sub-pixel level value *L*'s, which control the image quality.

- **PARTICLE LIMIT**

  Specifies the maximum number of particles that are generated in PBVR Server. Use this to avoid the explosive increase of the number of particles (e.g. due to the false settings of a transfer function). The number is multiplied by $10^6$.

- **EYE POINT**

  Specifies the viewpoint.

- **CENTER POINT**

  Specifies the location where the camera looks at.

- **UP VECTOR**

  Specifies the up vector of the camera.
- **RESOLUTION**

  Specifies the Viewer’s resolution.
- **SENDING**

  Shows the progress of data transfer to the server program.
- **RECEIVING**

  Shows the progress of data transfer from the server program.
- **CPU MEMORY**

  Displays the system memory usage in megabytes.
- **GPU MEMORY**

  Displays the GPU memory usage in megabytes.
- **PBVR Filter information**

  Displays information about the volume data in PBVR Server, which is the contents of the .pfi file.
- **FILE**

  This button shows the **FILE Panel**, whose detail is described later.
- **CROP**（only supported for Linux and Windows by the current version）

  Displays **CROP Panel**, which is described in the next section.
- **Set parameter**

  Sends parameters specified in “Main panel” to the server program.

## 5.4.3 FILE Panel

**FILE Panel** is a panel for reading and writing visualization parameter files. This panel is shown when the **FILE** button is hit.

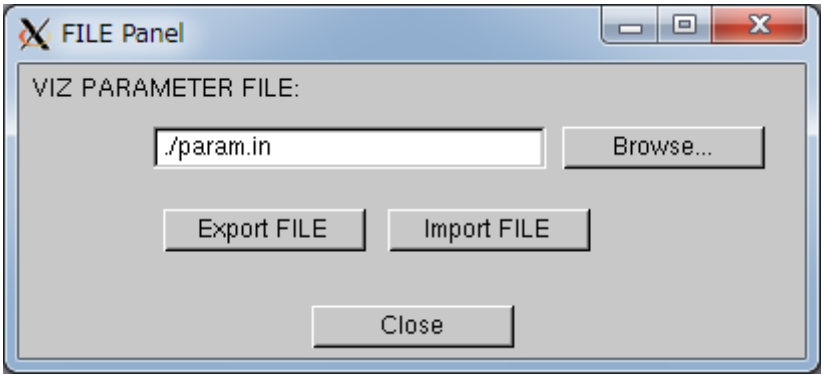



Figure 8 **FILE Panel**

- **VIZ PARAMETER FILE**

  specifies the path to a visualization parameter file.
- **Browse …**

  Opens a file dialog for specifying the path to a visualization parameter file.
- **Export FILE**

  Saves the current parameter settings to a visualization parameter file. In the Mac client, the parameters for the CROP functionalities are not saved.
- **Import FILE**

  Imports a visualization parameter file.
- **Close**

  Closes **FILE Panel**.

## 5.4.4 CROP Panel

**CROP Panel** is activated by hitting the **CROP** button in **Main panel**. Use **CROP panel** for operations related to extracting and rendering elements involved within the Region Of Interest (ROI). ROI can be specified with a cuboid, a sphere, or a cylinder. Note that **CROP panel** is not available on the Mac client.

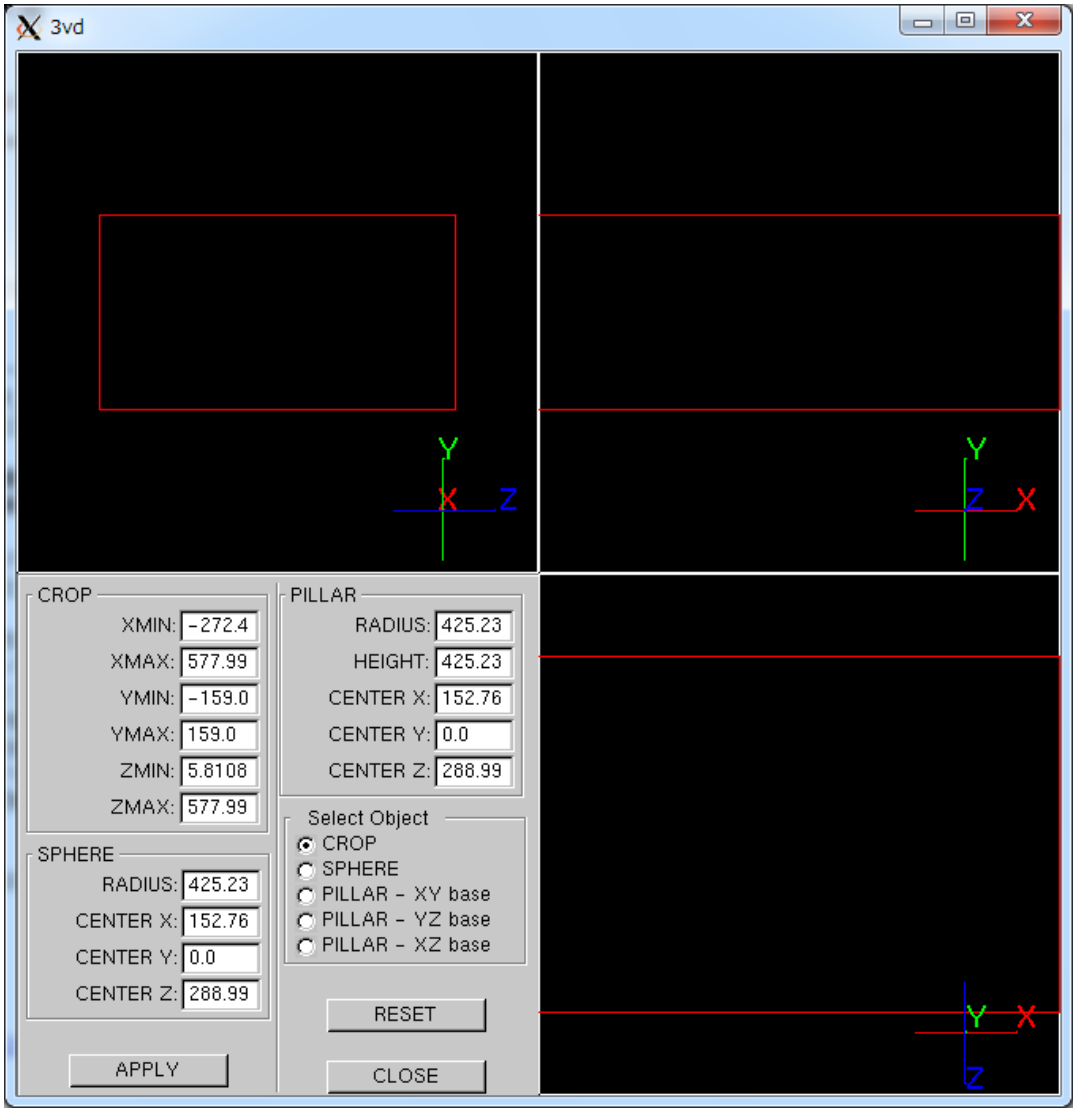



Figure 9 **CROP panel**

**Select Object**: Specifies the shape of the ROI

**CROP**: A cuboid

**SPHERE**: A sphere

**PILLAR-XY base**: A cylinder with a *X*-*Y* base

**PILLAR-YZ base**: A cylinder with a *Y*-*Z* base

**PILLAR-XZ base**: A cylinder with a *X*-*Z* base

**CROP**: Specifies the range of the cuboid

**SPHERE**: Specifies the center and radius of the sphere

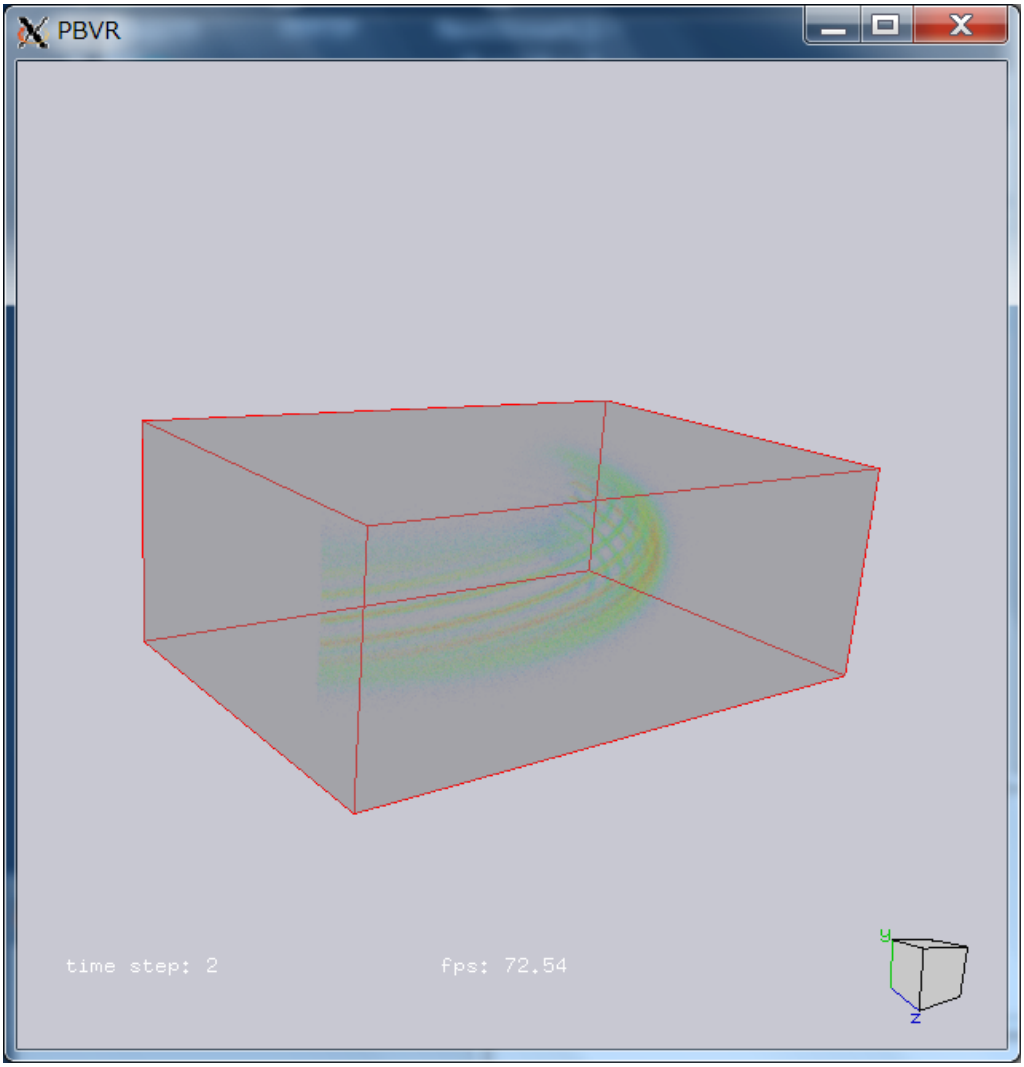
**PILLER**: Specifies the radius, the height, and the center coordinate values of the cylinder

**RESET**: Resets the **CROP** panel

**APPLY**: Extracts the ROI

**CLOSE**: Closes the panel

Displaying **CROP** panel overdraws the shape of the ROI in **Viewer** as in Figure 10.




Feature 10 **Viewer panel** interacting with CROP

## 5.4.5 Transfer Function Editor

**Transfer Function Editor** edits the transfer functions, which assigns a color/opacity to each scalar value for volume rendering. In a standard volume rendering, a transfer function is defined by only one physical quantity. In contrast, PBVR provides a new multi-dimensional transfer function design, which has the following three features:

1) Assign two independent variable quantities to color and opacity.
2) Define each variable quantity with an arbitrary function of the *X-Y-Z* coordinates and variables *q1*, *q2*, *q3*…
3) Synthesize a multidimensional transfer function from one-dimensional transfer functions *t1-t5* using equations.

This new transfer function design adds significant flexibility to visualization. **Transfer Function Editor** is shown in Figure 11. Each item in the panel is explained below.

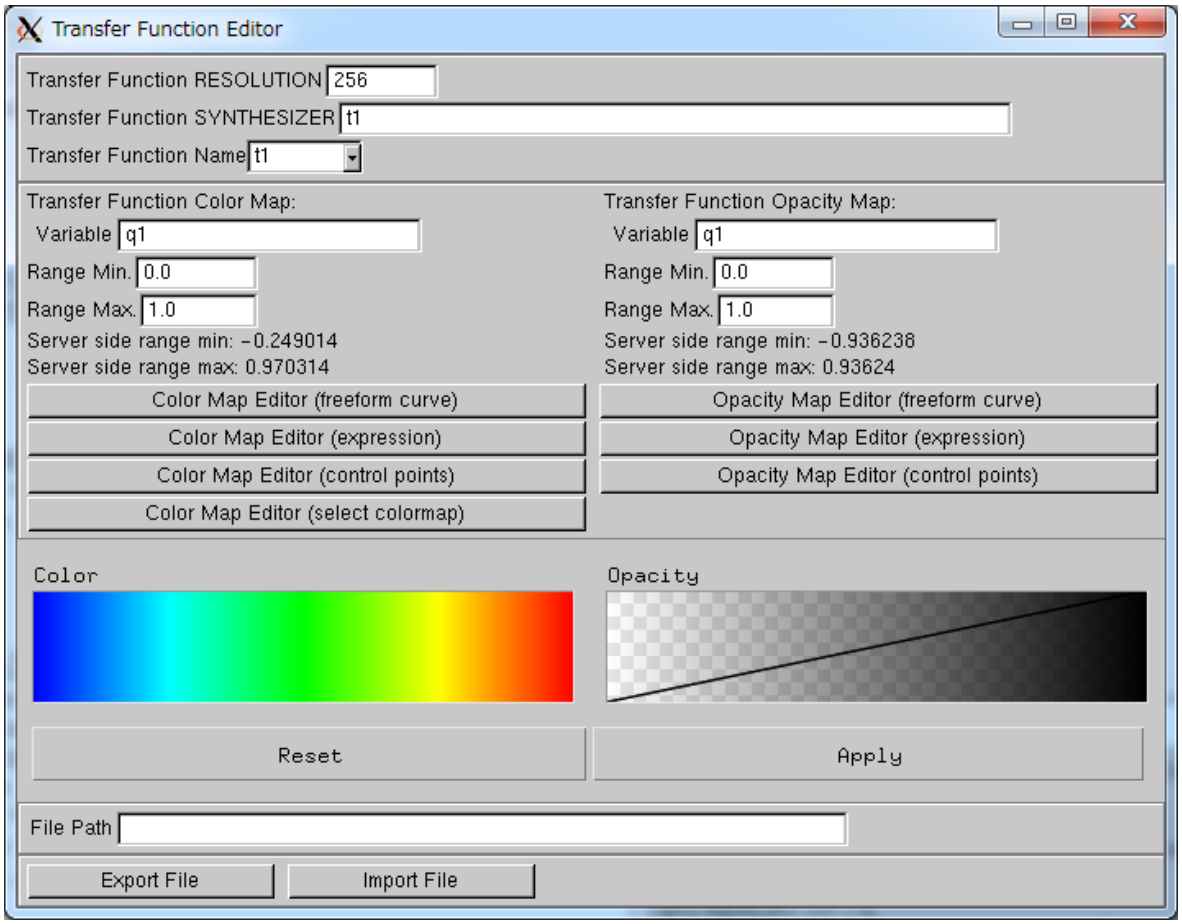



Figure 9 **Transfer Function Editor**

- **Transfer Function RESOLUTION**

  Species the resolution of the transfer function

- **Transfer Function SYNTHESIZER**

  Specifies a function to synthesize one dimensional transfer functions *t1-t5* *1

- **Transfer Function Name**

  Selects a transfer function (t1-t5) to edit with a pull-down menu.

- **Reset**

  Resets the panel.

- **Apply**

  Sends a transfer function defined with this panel to the server.

- **File Path**

  Specifies a file path for saving and loading a transfer function file.

- **Export File**

  Saves a transfer function defined with this panel to a file in the same format as the parameter file specified with the command line option '-pa'.

- **Import File**

  Loads a transfer function stored in a file to this panel

#### 5.4.5.1 Color Map Editor Panel

[**Transfer Function Color Map** category]

The GUI components in this category set a variable quantity and color for the transfer function specified with the **Transfer Function Name** field.

- **Color**

  Displays the colors that were assigned to the values of variable quantity by the transfer function.

- **Variable**

  Defines the (synthesized) variable quantity used for color of the selected transfer function. An equation can be entered, while the following variables are available.

  Physical quantities：q1, q2, q3, .., qn.

  Coordinate values：X, Y, Z.

- **Range Min**

  Specifies the minimum value of the specified variable quantity.

- **Range Max**

  Specifies the maximum value of the specified variable quantity.

- **Server side range min**

  Displays the minimum value of the (synthesized) variable quantity obtained in the server program.

- **Server side range max**

  Display the maximum value of the (synthesized) variable quantity obtained in the server program.

- **Color Map Editor (freeform curve)**

Displays a sub-panel, which specifies a transfer function with a freeform curve. Use the mouse to edit the freeform curve.

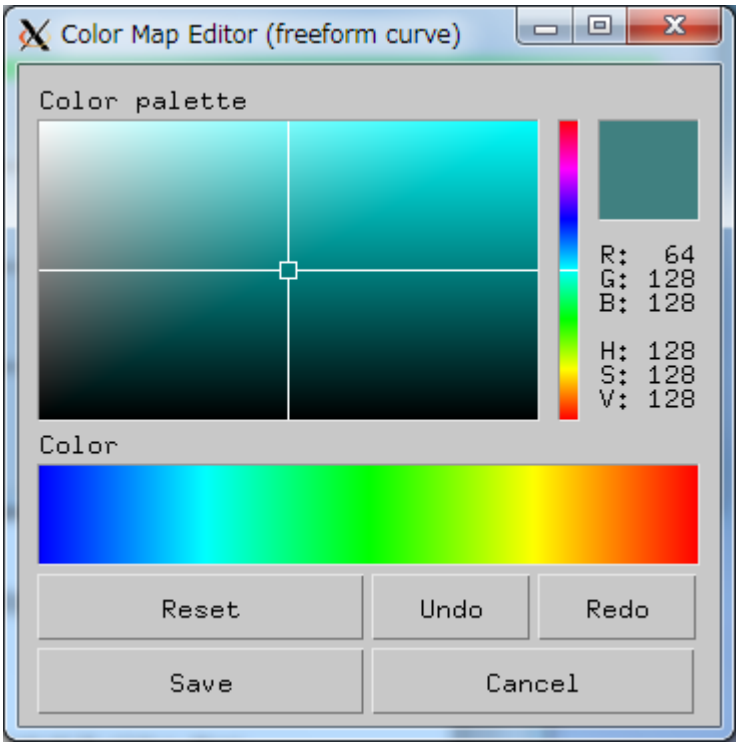



Figure 10 **Color Map Editor (freeform curve)** panel

- **Color palette**

  Specifies the saturation, the brightness, and the hue of a color with mouse cursor. On the left, the horizontal and vertical axes correspond to the saturation and brightness, respectively. The neighboring bar shows the hue.

- **RGB**

  Specifies the hue of the color by placing a mouse cursor. The upper-right box displays the color created by **Color palette** and **RGB bar**.

- **Color**

  Blends the colors in **Color** area with a color specified with **Color palette** and **RGB bar.** To specify the locations in **Color** area, trace the locations by dragging the mouse cursor while pressing the left mouse button. The blending ratio of the original color and the overpainting color is determined by the mouse cursor's vertical position. For example, when the upper edge of the color bar is traced from left to right, the **Color** bar is painted completely by the specified color rather than by blended colors; when the vertical center line of **Color** bar is traced, the colors are replaced with blended colors with 50% of the original color and 50% of the specified color.

- **Reset**

  Resets the panel.

- **Undo**

  Undoes the last mouse action.
- **Redo**

  Redoes the last mouse action undone.
- **Save**

  Saves the transfer function.
- **Cancel**

  Closes the panel.

- **Color Map Editor (expression)**

  Displays a panel to create a transfer function by taking equations as input.

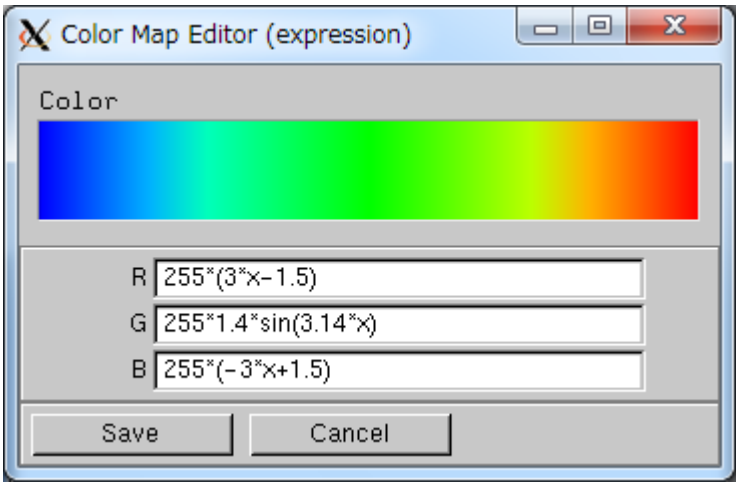



Figure 11 **Color Map Editor (expression) panel**

- **Color**

  Displays a color bar of a transfer function created in this panel.
- **R**

  Describes a transfer function of the R component of the color.
- **G**

  Describes a transfer function of the G component of the color.
- **B**

  Describes a transfer function of the B component of the color.
- **Save** button

  Saves a transfer function created in this panel.
- **Cancel**

  Closes the panel.

- **Color Map Editor (control points)**

  Displays a panel for creating a transfer function. This editor takes control points as input.

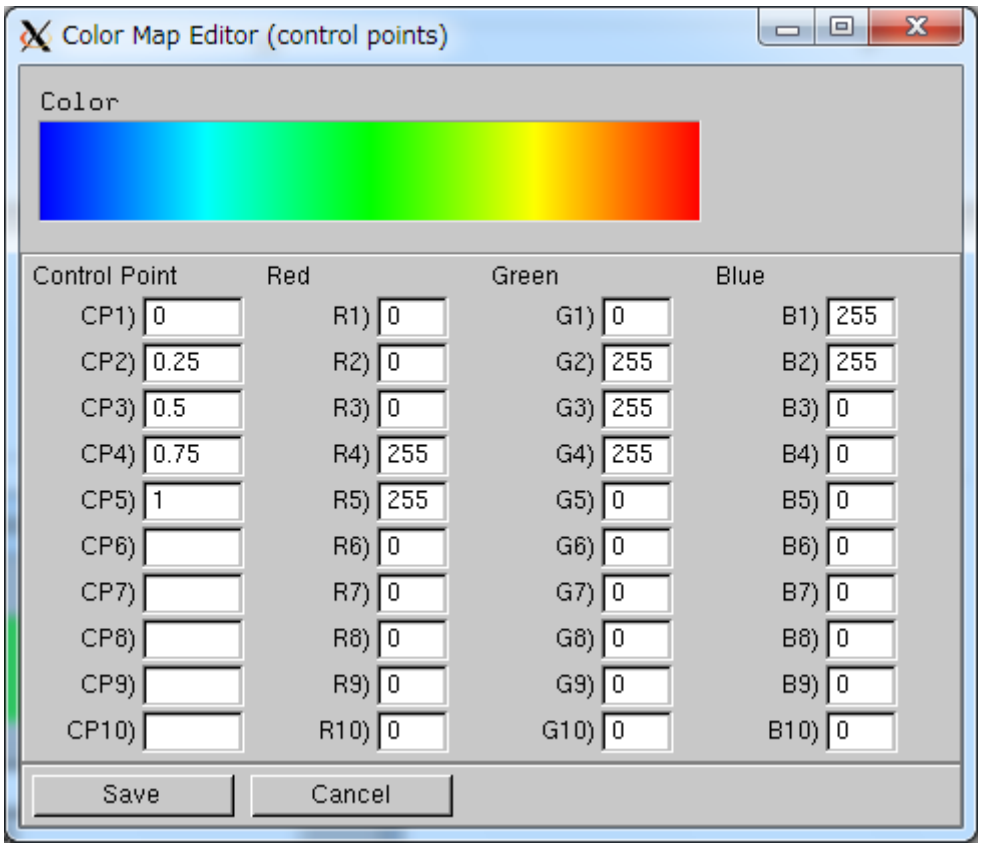



Figure 12 **Color Map Editor (control points) panel**

- **Color**

  Displays a color bar for the transfer function that is being defined with this panel.
- **Control Point**

  Specifies the values of (up to 10) control points with the fields **CP1)-10)** .
- **Red**

  Specifies the R component of the color at the control points.
- **Green**

  Specifies the G component of the color at the control points.
- **Blue**

  Specifies the B component of the color at the control points.
- **Save**

  Saves the transfer function.
- **Cancel**

  Closes the panel.

- **Color Map Editor (select colormap)**

  Displays a panel to create a transfer function from preset color bar templates.

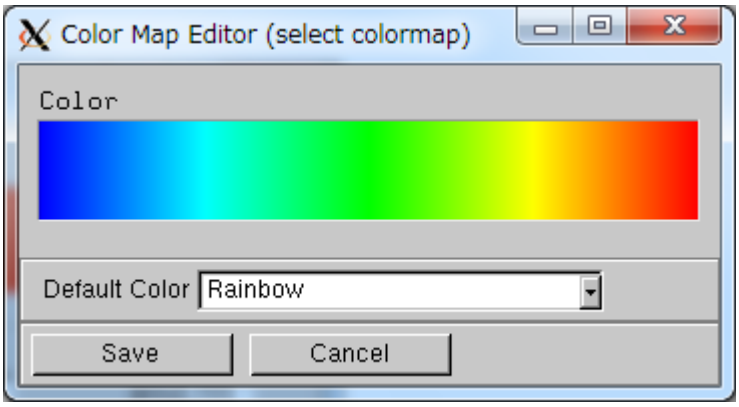



Figure 13 **Color Map Editor (select colormap) panel**

- **Color**

  Displays the color bar of the transfer function that is being created with this panel.

- **Default Color**

  Selects a color bar to be set as the transfer function. 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

- **Save**

  Saves the transfer function created with this panel.

- **Cancel**

  Closes the panel.

### 5.4.5.2 Opacity Editor

【**Transfer Function Opacity Map** Category】

The GUI components in this category set a variable quantity and color for the transfer function specified with the **Transfer Function Name** field.

- **Opacity**

  Displays the transfer function curve under edit.

- **Variable**

Defines the (synthesized) variable quantity used for the opacity of the selected transfer function. An equation can be entered, while the following variables are available.

Physical quantities: q1, q2, q3, .., qn.

Coordinate values: X, Y, Z.

- **Range Min**

  Specifies the minimum value of the variable quantity.

- **Range Max**

  Specifies the maximum value of the variable quantity.

- **Server side range min**

  Displays the minimum value of the (synthesized) variable quantity obtained in the server program.

- **Server side range max**

  Display the maximum value of the (synthesized) variable quantity obtained in the server program.

- **Color Map Editor (freeform curve)**

  Displays a panel for creating a transfer function with a freeform curve. Use the mouse to edit the freeform curve.

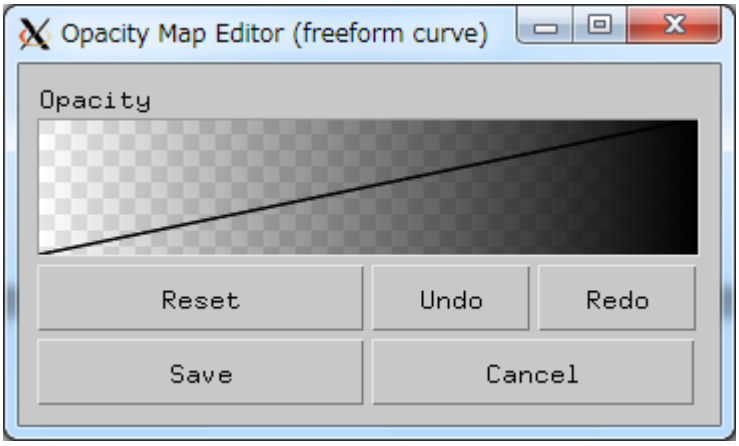



Figure 14 **Opacity Map Editor (freeform curve) panel**

- **Opacity**

  Specifies a transfer function for the opacity. A freeform curve is drawn by dragging the mouse while holding the left mouse button. A piecewise linear curve is drawn by specifying control points with right clicks.

- **Reset**

  Resets the panel.

- **Undo**

  Undoes the last mouse action.

- **Redo**

  Redoes the last mouse action undone.
- **Save**

  Saves the transfer function created with this panel.
- **Cancel**

  Closes the panel.

- **Color Map Editor (expression)**

  Display a panel to create a transfer function using equations.

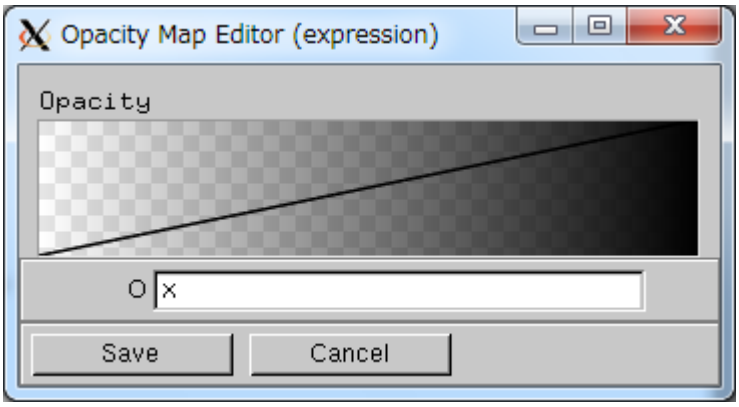



Figure 15 **Opacity Map Editor (expression) panel**

- **Opacity**

  Displays the transfer function for opacity specified by the equation in the field **O**.
- **O**

  Specifies the equation for the curve that specifies the transfer function of opacity.
- **Save**

  Saves the transfer function created with this panel.
- **Cancel**

  Closes the panel.

- **Color Map Editor (control point)**

  Displays a panel to create a transfer function by taking equations as input.

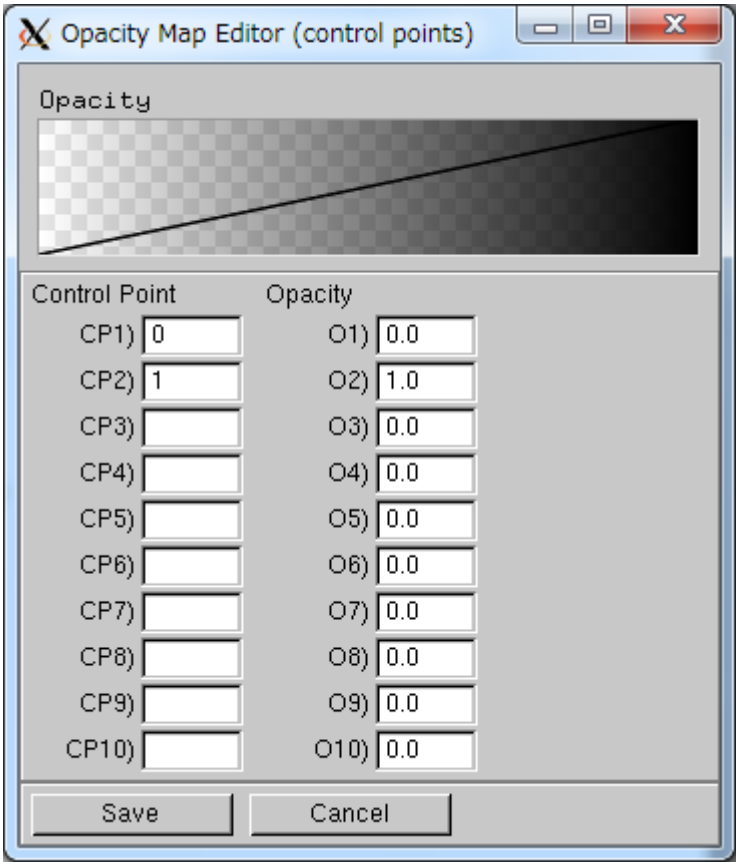



Figure 16 **Opacity Map Editor (control points) panel**

- **Opacity**（on the top）

  Displays the transfer function for the opacities specified in this panel.

- **Control Point**

  Specifies the values of (up to 10) control points in the fields **CP1)-10)** .

- **Opacity**（on the bottom right）

  Specifies the opacities at the control points.

- **Save**

  Saves the transfer function created with this panel.

- **Cancel**

  Closes the panel.

### 5.4.5.3 Function Editor

Table 12 lists the built-in math operations available in the function editor. They can be used to synthesize transfer functions and variable quantities, and to define colormap/opacity curves.

Table 12 Math operations 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 |

### 5.4.6 Time panel

Figure 19 shows **Time panel**, which specifies the time steps for visualization. Each widget works as described in the followings.

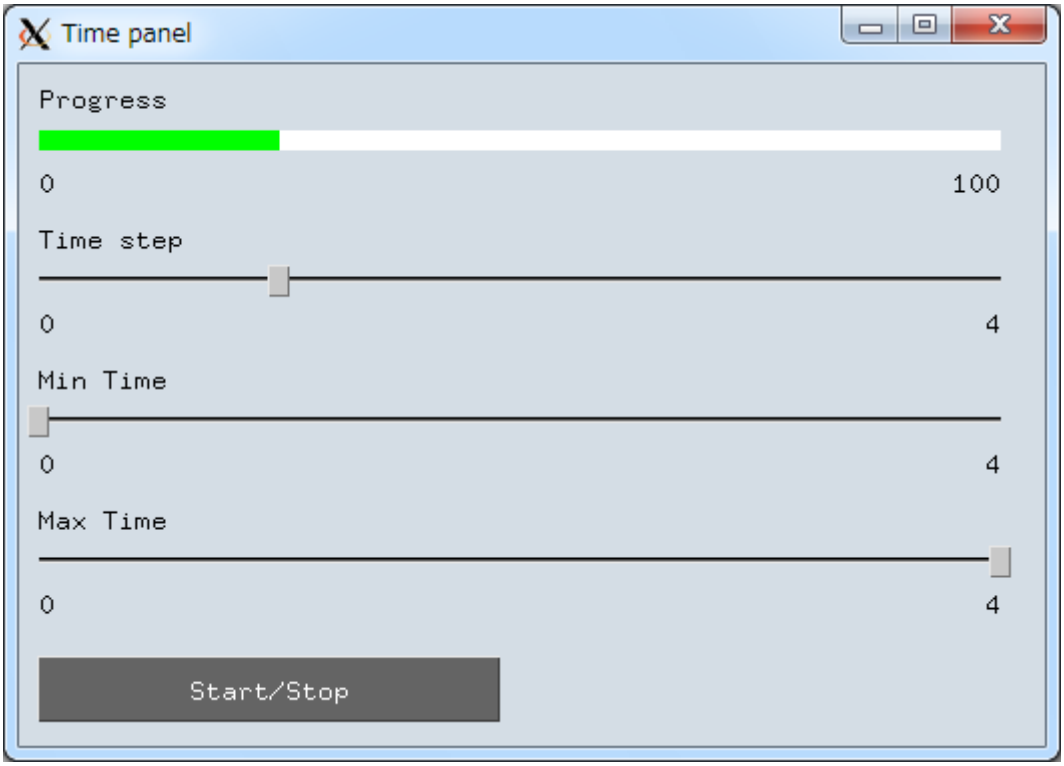



Figure 17 **Time panel**

- **Progress**

  Expresses the current time step as percentage.
- **Time step**

  Specifies the time step of the data to be rendered.
- **Min Time**

  Specifies the minimum time step for ROI.
- **Max Time**

  Specifies the maximum time step for ROI.
- **Start/Stop**

  Starts/stops the communication between PBVR Client and PBVR Server.

### 5.4.7 Animation panel

Image files displayed in **Viewer** are saved as sequentially numbered BMP image files so that one image file is generated for each time step. The sequence of image files can be converted and compressed into movie files like mpeg using free software such as ImageMagic and ffmpeg.

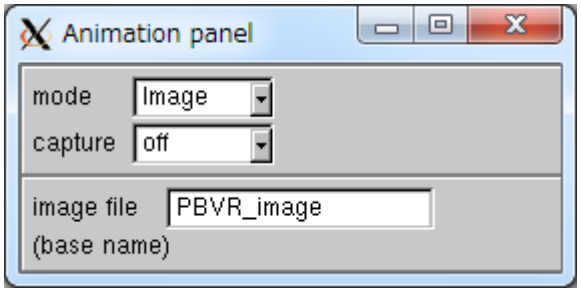



Figure 18 **Animation panel**

**mode**: Currently this field accepts only ‘image’.
**capture**: When turned on, sequentially numbered image files are produced.
**image file**: Specifies the prefix for the sequentially numbered image files.

Sequentially numbered image files are stored in the directory from which the PBVR Client is executed, with the following names.

*PBVR_image.00001.bmp*
*PBVR_image.00002.bmp*

# 6 An Example with the Sample Dataset *gt5d.tgz*

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

## 6.1 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 |
| *demo1.tf*: | A transfer function file for demonstration 1 |
| *demo2.tf*: | A transfer function file for demonstration 2 |

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

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

The contents of param.txt are the followings.

```
#
in_dir=./
field_file=gt5d.fld
out_dir=./
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 both for 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

## 6.2 Starting PBVR

[step 1] Launch PBVR Server (which is the OpenMP version)

```
$./CPUServer
first reading time[ms]:0
Server initialize done
Server bind done
Server listen done
Waiting for connection ...
```

[step 2] Launch PBVR Client. This example uses the metropolis sampling. The sub-pixel level is set to 2, and the particle limit is specified as 100 million particles.

```
$./PBVRViewer -S m -vin ./gt5d/case.pfi -plimit 100 -rl 2 -sl 4
```

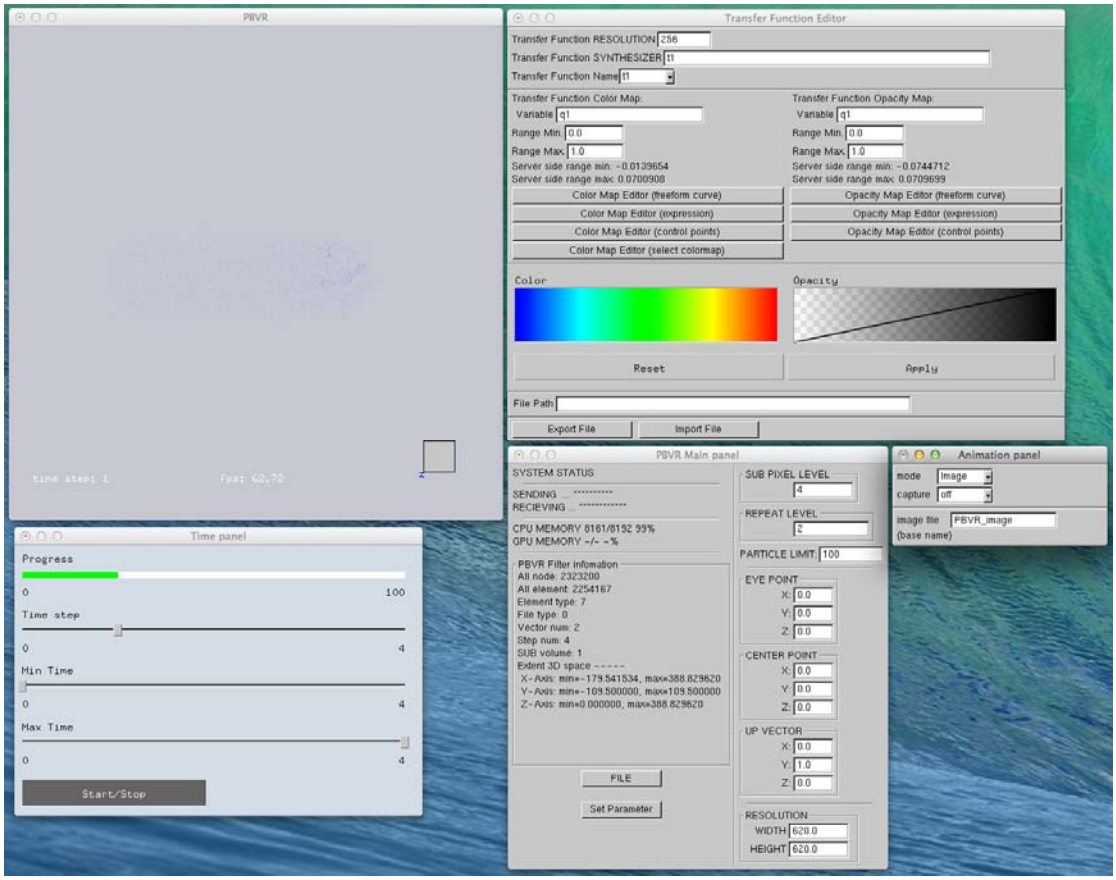



Figure 19 The GUIs of PBVR

## 6.3 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.

*1. The transfer functions used in section 6.3.1-6.3.3 are stored in *demo1.tf*.

*2. The synthesized transfer function used in section 6.3.4 is stored in *demo2.tf*.

### 6.3.1 Volume Rendering for a Single Variable

First of all, understand the variable *q1* by setting the transfer function *t1* as shown in Figure 22. In this example, the transfer function is designed with **Transfer Function Editor**. Shown in the left of **Transfer Function Editor** is the configuration of colors, while in the right is that of the opacities. Notice that this configuration is the conventional volume rendering for a single variable.

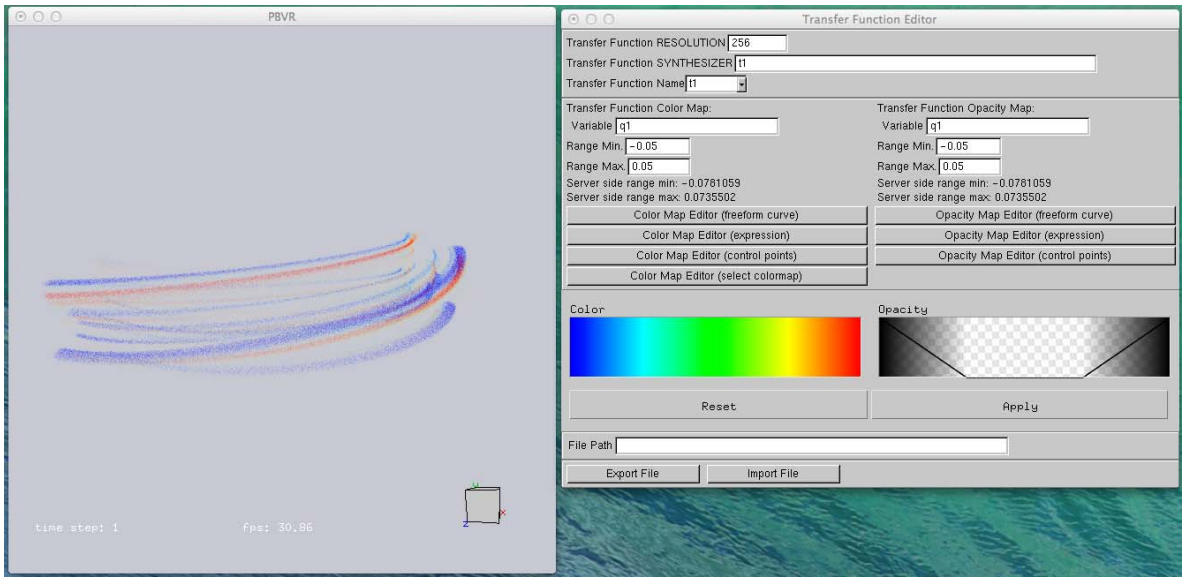



Figure 20 The volume rendering result for the variable *q1*.

## 6.3.2 Multivariate Volume Rendering

The next example shows the result of multivariate volume rendering, in which the variables *q1* and *q2* are synthesized as shown in Figure 23. In this example, the colors are assigned to the variable *q1*, while the opacities are 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.

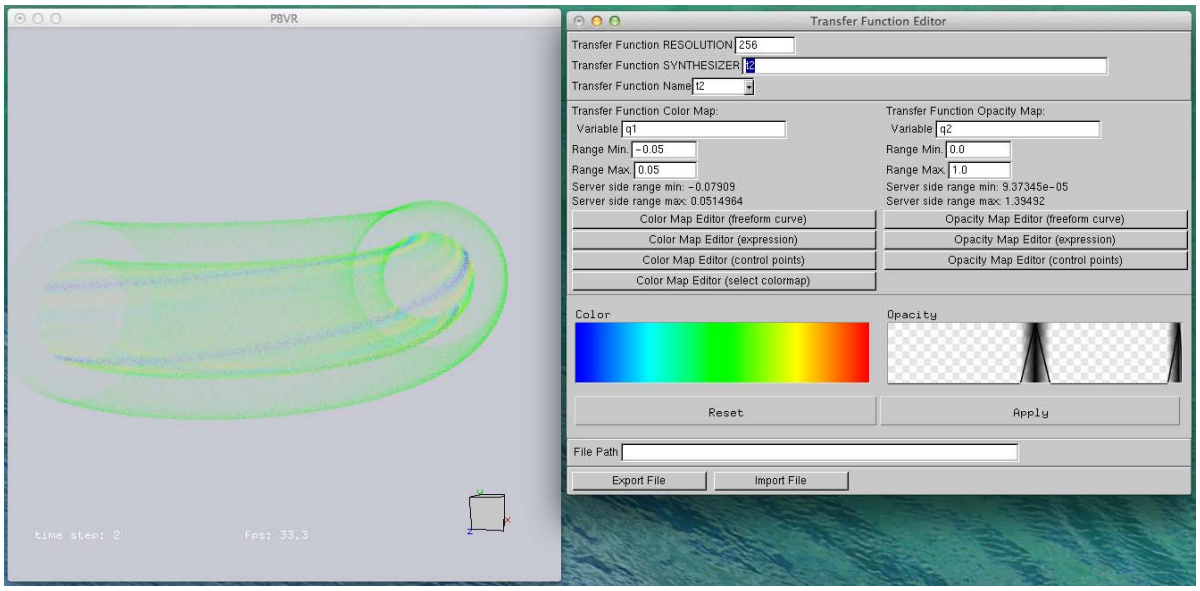



Figure 21 Rendering a multivariate volume. The *q1* values are color-mapped onto the iso-surfaces of *q2*.

## 6.3.3 Slicing Volumes

Figure 24 shows an application of PBVR's multivariate volume rendering for 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.

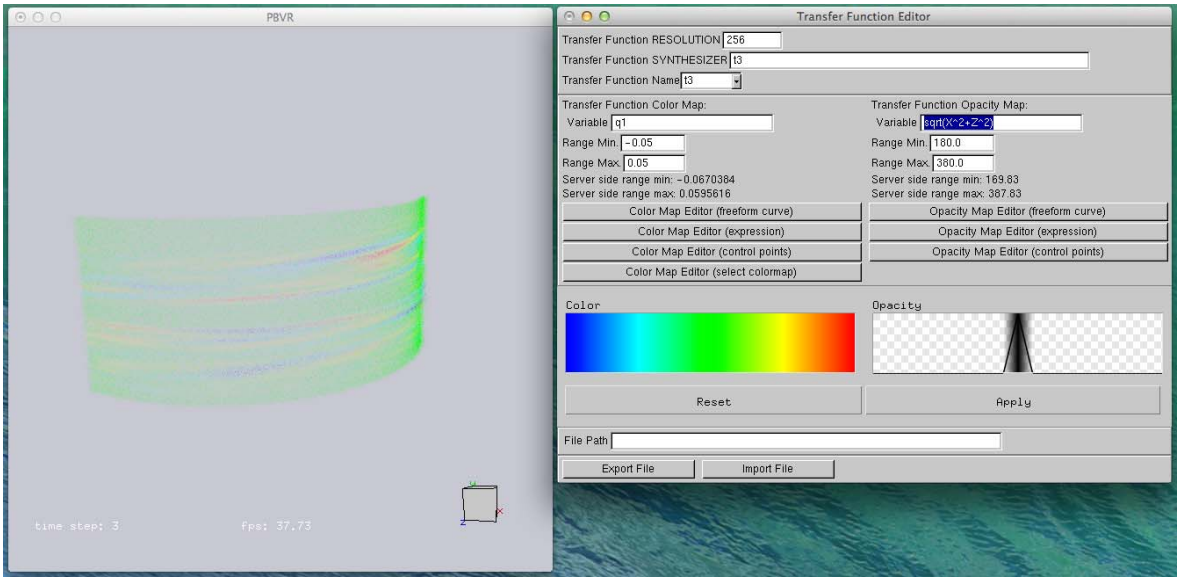



Figure 22　A rendering result for Slicing the volume with PBVR's multivariate volume rendering capability.

## 6.3.4 Synthesis of Transfer Functions

This section explains how to synthesize transfer functions in PBVR. Figure 25 shows a transfer function *t4,* whose opacity function makes the region $Y > 0$ transparent. By synthesizing the previously described transfer functions *t1*, *t2,* and *t3* together with a new transfer function *t4* as (*t1* + *t2*) * *t4* + *t3*, the individually extracted sub-regions can undergo flexible composition through arithmetic operations. In this example, the colors of *t2* and *t3* are set to (*R*, *G*, *B*) = (0, 0, 0), while the color of *t4* is set to (*R*, *G*, *B*) = (1, 1, 1). In the above synthesis equation, the final colors obey the rainbow colormap defined for *t1*. On the other hand, the opacity of *t4* is multiplied to the sum of *t1* and *t2* in order to extract the lower half region ($Y < 0$) of *t1* and *t2*. Then, the resulting region is synthesized with the cylindrical surface given by *t3*. As revealed in these examples, PBVR's ability to synthesize transfer functions is powerful considering the capability to extract arbitrary region for each variable and to carry out a preferred series of operations.

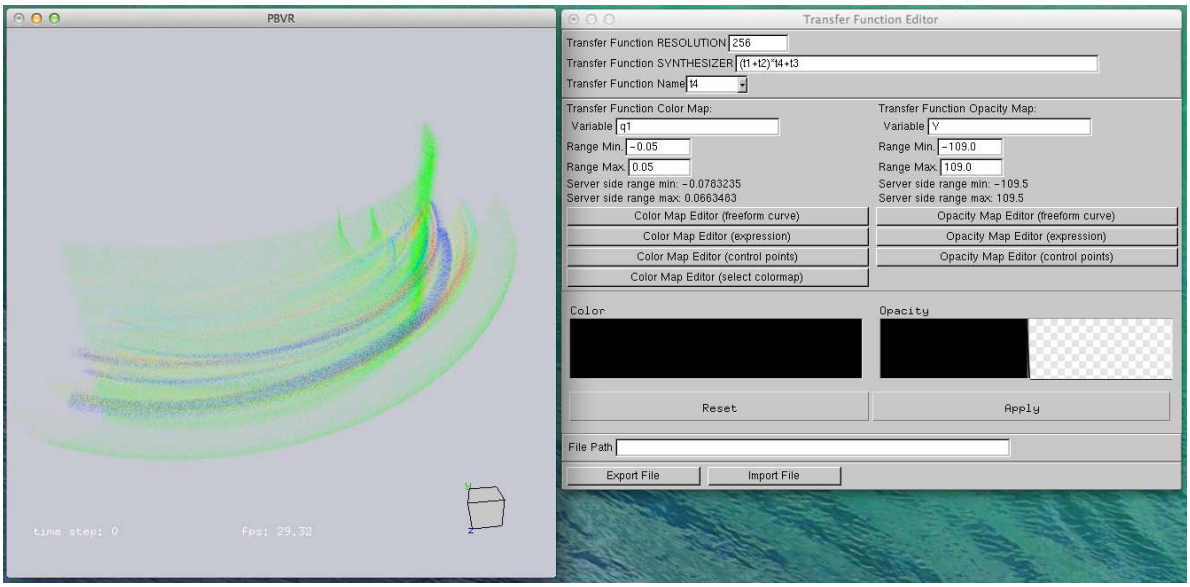



Figure 23 Synthesizing transfer functions

## 6.4 Saving Results

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

1) Image output

   In order to save the results as images, select **on** from the **capture** drop down menu in **Animation panel**. The bitmap image files (PBVR_image.xxxxx.bmp) are generated in the working directory.

2) Transfer function file

   In order to generate the transfer function file, write file name of the transfer function in **File Path** field of **Transfer Function Editor** and press **Export File**. Later, this file can be loaded by hitting **Import File**. Note that the input transfer function is not reflected until **Apply** is hit.

3) Visualization parameter file

   In order to run PBVR Server in batch mode, all the visualization parameters including the transfer function can be exported. Open **File panel** from **Main panel**, specify the parameter filename, and press **Export File**.

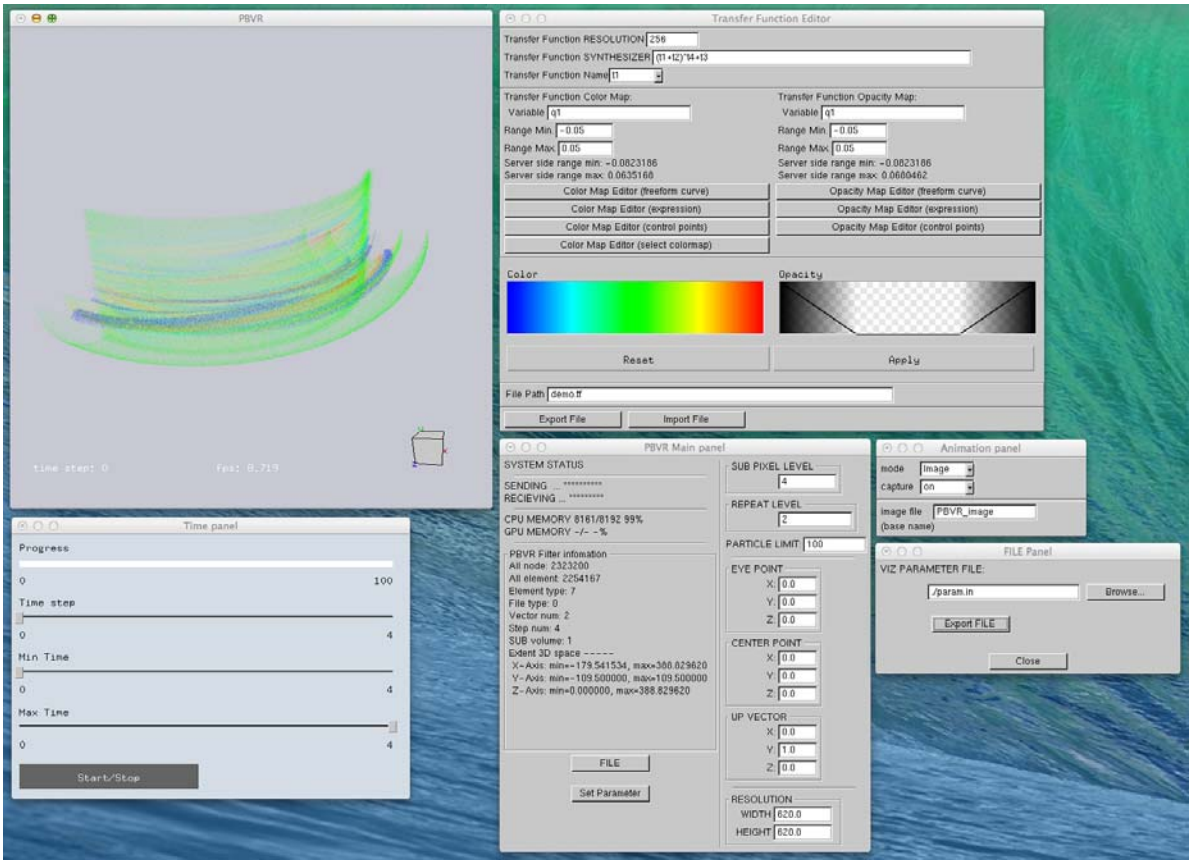



Figure 24 Exporting the visualization results

## 6.5 Example of Batch Mode

This section explains how to run PBVR Server in batch mode using the visualization parameter file exported in the previous section. This mode is developed for carrying out massively parallel processing with supercomputers. In addition, this mode is useful also for high speed rendering of time series data with PBVR Client in stand-alone mode, since the latency due to particle generation and particle data transfer can be eliminated.

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

```
$./CPUServer –B –vin ./gt5d/case.pfi –pout ./particle/ -S m -plimit 100 -sl 4 –pa ./param.in
```

[Step 2] Launch PBVR in stand-alone mode ∗2

```
$./PBVRViewer ./particle/ -sl 4
```

*1. In the current version, running PBVR Server in batch mode requires command line options for visualization parameters except of the transfer function.

*2. The sub-pixel should be specified also when launching PBVR Client in stand-alone mode, since they determine the size of particles in the rendering process.