

In-situ PBVR (v1.00) User Guide

March 2019

Japan Atomic Energy Agency

Center for Computational Science & e-Systems

Revision Record

Version number	Date revised	Revised chapter	Revised content
1.00	2019.03.28	-	Release

目次

1 Introduction	4
1.1. Overview of Particle Sampler.....	6
1.2. Overview of Daemon	6
1.3. Overview of PBVR Client.....	6
2 Package Configuration.....	8
2.1. Load Module Package.....	8
2.2. Source Code Package.....	10
3 Build.....	11
3.1. Daemon and Particle Sampler	11
3.2. PBVR Client	13
3.2.1. Linux • Mac	13
3.2.2. Windows.....	14
4 Setup of In-situ Visualization.....	16
4.1. Setting of Environment Variables	16
4.2. Setting of Visualization Parameters	16
4.3. Port Forwarding Connection	16
4.3.1. Remote Connection between Two Machines	17
4.3.2. Remote Connection with Several Machines	17
4.4. Launching Daemon and Port Forwarding	17
5 Particle Sampler	19
5.1. Particle Generation Function for Structured Grid.....	19
5.2. Particle Generation Function for Unstructured Grid	20
5.3. Particle Generation Function for AMR Grid	21
5.4. Connection into Simulation Code.....	22
5.4.1. Domain Information and Particle Generation Function.....	23
6 PBVR Client	24
6.1. Launching PBVR Client	24
6.2. Terminating PBVR Client	24
6.2.1. Forced Termination.....	25
6.3. Using PBVR Client GUI	26
6.3.1. Viewer.....	26
6.3.2. Main Panel	26
6.3.3. Transfer Function Editor.....	29

6.3.4. Time Panel	44
6.3.5. Particle Panel.....	45
6.3.6. Image File Production	49
7 Samples	54
7.1. ICEX	54
7.2. JKNL.....	56
7.3. Windows	58

1 Introduction

This document is a user guide for in-situ PBVR, a remote visualization system in a situation developed at the Center for Computational Science & e-Systems in Japan Atomic Energy Agency. In-situ visualization avoids large-scale data I/O and reliably visualizes a large-scale simulation by simultaneously visualizing the calculation results with the simulation. In-situ PBVR compresses large-scale data into visualization particle data and allows interactive viewpoint changes on a user PC. This system has been developed using C++ and realizes in-situ visualization using the particle-based volume rendering (PBVR) method and the visualization library KVS developed at Koyamada Laboratory, Kyoto University. The system consists of the three following components:

(1) Particle sampler

The particle sampler denotes visualization libraries that are coupled to the simulation code. It then converts the calculation results into visualization particles at each time step of the simulation. For the visualization particles, one file is output on the storage for each computing node. The particle sampler then generates visualization particles by referring to the visualization parameter file on the storage.

(2) Daemon

The daemon is an in-situ PBVR file-based control program that runs in an interactive node. It aggregates particle files on the storage and transfers them to the user PC via the network. Moreover, it receives visualization parameters sent by the PBVR client and outputs them as a file.

(3) PBVR client

The PBVR client operates on the user PC, projects particle data on the screen, and displays a visualized image on the viewer. The user can see the visualized image, adjust the visualization parameters (e.g., color and opacity), and send it to the daemon.

In-situ PBVR has a multivariate visualization function that can be applied to various simulations in a versatile manner. For conventional visualization, the spatial distribution of physical values is drawn by mapping a color and an opacity function (transfer functions) to one physical value. In-situ PBVR provides a transfer function synthesizer that enables designing multivariate transfer functions using algebraic expressions.

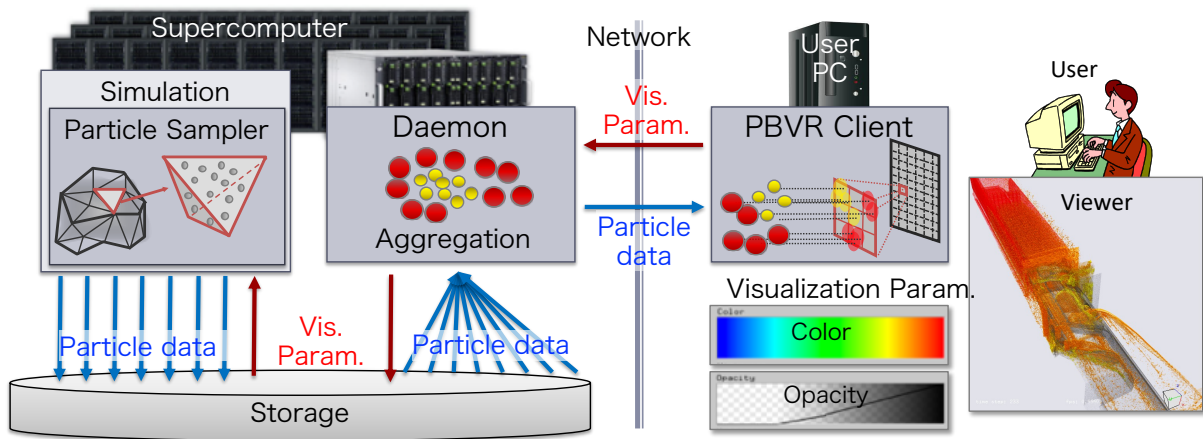


Figure 1 -1 Configuration of the In-situ PBVR framework

1.1. Overview of Particle Sampler

The particle sampler is parallelized by a hybrid of the MPI/OMP programming model, which further accelerates the generation of particle data using the SIMD operation. The particle sampler is MPI-parallelized without changing the domain decomposition of simulation and generates particles in element parallel using OpenMP for each decomposed domain. The synthesis of physical values, synthesis of transfer functions, and interpolation of physical values, which are required in multivariate visualization, are vectorized using SIMD operations.

Three types of particle samplers are available according to the simulation grid types: one for structured grids, one for unstructured grids, and one for hierarchical grids. The particle sampler supports the simulation code written in C/C++ and FORTRAN. For in-situ visualization, the particle sampler is inserted into the time step loop of the simulation. At this time, the multivariate data of simulation results, lattice data, and global coordinates of the region are input as arguments of the particle sampler. The memory layout of multivariate data also assumes an array-like structure.

1.2. Overview of Daemon

The daemon is executed in an interactive node or interactive job of the supercomputer, and is key to realize interactive visualization during batch processing. The daemon constantly monitors files on storage and collects particle files output using the particle sampler. This collection is asynchronous with the particle sampler and the simulation; hence, it does not impede the simulation performance. The particle file collection is parallelized using OpenMP, and the daemon aggregates the particle files into single-particle data and sends it to the PBVR client. Moreover, the daemon receives the visualization parameters sent from the PBVR client and updates the visualization parameter file on the storage used by the particle sampler.

For in-situ PBVR, the interactive node and the user PC need to be connected by port forwarding because the daemon and the PBVR client send and receive data by socket communication via the internet. Therefore, the daemon does not work and interactive visualization cannot be used if the port forwarding is not permitted on a supercomputer. Moreover, interactive visualization cannot be used even on a supercomputer that employs staging I/O, in which file output does not occur until the end of batch processing.

1.3. Overview of PBVR Client

The PBVR client is launched on the user PC and comprises a screen for displaying the visualization result and a “transfer function synthesizer” (TFS), which is a function for realizing multivariate visualization. TFS has a volume data synthesis function that combines variables included in the result data to generate new volume data as well as a transfer function synthesis function that combines multiple transfer functions. The user specifies the synthesis function using an algebraic expression on the TFS. The algebraic expression is transferred to the PBVR sampler as a visualization parameter, and the volume data and transfer function syntheses are performed in real-time using a computer algebra system. Users can utilize the initial mathematical functions and spatial derivatives of physical

variables as algebraic expressions on the TFS and flexibly design the transfer function for the multivariate data using mathematical expressions.

2 Package Configuration

In-situ PBVR is provided as a source code package and a load module package. The daemon and the PBVR client are programs that work by themselves. The particle sampler is provided as a library to be coupled to the simulation code and is composed of the following:

- (1) Particle generation library, providing the particle generation function
- (2) KVS library, providing the visualization function
- (3) Computer algebra library, providing the mathematical algebraic calculation function

2.1 . Load Module Package

The load modules of the particle samplers and the daemons are generated using JAEA's supercomputer ICEX and KNL cluster (JKNL). The PBVR clients are provided for Mac, Windows, and Linux. The users need to build from the source code for use in other environments. Table 2-1 to Table 2-7 show lists of the load modules.

Table 2-1 Particle generation libraries for the structured grid (a part of the particle sampler)

Platform	Parallelization	Load module
ICEX	MPI+OpenMP	InSituLib_struct/libInSituPBVR_icex.a
JKNL	MPI+OpenMP	InSituLib_struct/libInSituPBVR_jknl.a

Table 2-2 Particle generation libraries for the unstructured grid (a part of the particle sampler)

Platform	Parallelization	Load module
ICEX	MPI+OpenMP	InSituLib_unstruct/libInSituPBVR_icex.a
JKNL	MPI+OpenMP	InSituLib_unstruct/libInSituPBVR_jknl.a

Table 2-3 Particle generation libraries for the AMR grid (a part of the particle sampler)

Platform	Parallelization	Load module
ICEX	MPI+OpenMP	InSituLib_AMR/libInSituPBVR_icex_a
JKNL	MPI+OpenMP	InSituLib_AMR/libInSituPBVR_jknl.a

Table 2-4 KVS library (a part of the particle sampler)

Platform	Parallelization	Load module
ICEX		libkvsCore_icex.a
JKNL		libkvsCore_jknl.a

Table 2-5 Computer algebra library (a part of the particle sampler)

Platform	Parallelization	Load module
ICEX		libpbvrFunc_icexa
JKNL		libpbvrFunc_jknl

Table 2-6 Daemon

Platform	Parallelization	Load module
ICEX	OpenMP	pbvr_daemon_icex
JKNL	OpenMP	pbvr_daemon_jknl

Table 2-7 PBVR client

Platform	Parallelization	Load module
Linux	OpenMP	pbvr_client_linux
Mac	OpenMP	pbvr_client_mac
Windows ※1	OpenMP	pbvr_client_win

※1. Copy glut32.dll in the same directory for Windows version.

2.2. Source Code Package

The particle sampler, daemon, and PBVR client are generated by compiling the source code package in the user's environment. Table 2-8 shows a source tree of the in-situ PBVR package. The source code package contains a particle sampler-coupled test simulation code.

Table 2-8 Source tree of the In-situ PBVR

Directory/file name	Explanation
pbvr_inSitu_1.00/	1.00 is version number
 -pbvr.conf	Configuration file for the Makefile
 -Makefile	Makefile for particle sampler, daemon, and PBVR client
 -arch/	Setting files for various environments
 -Client/	PBVR client program
 -Common/	Protocol, communication, and common library
 -Daemon/	Daemon program
 -Example/	Samples of test simulation code
 C/	C version
 -Hydrogen_struct/	For the structured grid
 -Hydrogen_AMR/	For the AMR grid
 -Hydrogen_unstruct/	For the unstructured grid
 -Fortran/	Fortran version
 -Hydrogen_struct/	For the structured grid
 -Hydrogen_AMR/	For the AMR grid
 -Hydrogen_unstruct	For the unstructured grid
 -FunctionParser/	Computer algebra library
 -glui/	GUI widget library
 -InSituLib/	Particle generation library
 -struct/	For the structured grid
 -AMR/	For the AMR grid
 -unstruct/	For the unstructured grid
 -KVS/	KVS library
 -shell/	Sample shell script for supercomputer

3 Build

The source package build is controlled by editing the configuration file pbvr.conf according to the user's environment. Table 3-1 shows a list of the variables specified in pbvr.conf. Table 3-2 shows a list of build configuration files that can be used as the value of PBVR_MACHINE.

Table 3-1 Variables defined in pbvr.conf

Variable	Input	Explanation
PBVR_MACHINE	Literal	Build setting file name located in arch
PBVR_MAKE_CLIENT	0 or 1	0 ⇒ Building daemon and particle sampler 1 ⇒ Building PBVR client

Table 3-2 Build setting files

File name	Explanation
Makefile_machine_gcc	gcc compiler build setting
Makefile_machine_gcc_omp	gcc compiler +OpenMP build setting
Makefile_machine_gcc_mpi_omp	gcc compiler +MPI+OpenMP build setting
Makefile_machine_intel	intel compiler build setting
Makefile_machine_intel_omp	intel compiler +OpenMP build setting
Makefile_machine_intel_mpi_omp	intel compiler +MPI+OpenMP build setting
Makefile_machine_fujitsu	fujitsu compiler build setting
Makefile_machine_fujitsu_omp	fujitsu compiler+OpenMP build setting
Makefile_machine_fujitsu_mpi_omp	fujitsu compiler+OpenMP+MPI build setting
Makefile_machine_icex	ICEX compiler build setting
Makefile_machine_icex_omp	ICEX compiler+OpenMP build setting
Makefile_machine_icex_mpi_omp	ICEX compiler+OpenMP+MPI build setting
Makefile_machine_jknl_omp	KNL compiler+OpenMP build setting
Makefile_machine_jknl_mpi_omp	KNL compiler+OpenMP+MPI build setting

3.1 . Daemon and Particle Sampler

The steps to build a daemon and particle sampler library from a source package are shown below. The file download destination in the following procedure is \$ HOME / JAEA. Moreover, the particle sampler is linked to the test simulation code, attached to the source package after the library build.

- ① Unzip pbvr_inSitu_1.00.tar.gz.

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

- ② Edit pbvr_inSitu_1.00/pbvr.conf and specify the build setting.

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

PBVR_MACHINE ⇒ Specify setting the file name in pbvr_inSitu_1.00/arch

PBVR_MAKE_CLIENT ⇒ Specify 0 (for daemon and particle sampler)

- ③ The KVS library is built using OpenGL and GLUT functions, disabled when PBVR_MAKE_CLIENT = 0, and using the OpenGL and GLUT functions, enabled when PBVR_MAKE_CLIENT = 1. Therefore, if you build the client just before, you need to execute the following command to rebuild the KVS:

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

- ④ Build the whole package at the root of the source tree.

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

This build generates load modules for daemon and particle sampler in Table 3-3.

Table 3-3 Load modules of daemon and particle sampler

Directory	Load module	Explanation
KVS	libkvsCore.a	KVS library providing particle format and visualization functions
Common	libpbvrCommon.a	Communication library providing protocol for socket communication
Daemon	pbvr_daemon	Daemon
FuctionParser	libpbvrFunc.a	Computer algebra library
InsituLib/struct	libInSituPBVR.a	Particle sampler
InsituLib/unstruct	libInSituPBVR.a	Particle sampler
InsituLib/AMR	libInSituPBVR.a	Particle sampler

- ⑤ The particle sampler coupled to the simulation code is coupled to the particle sampler, and the above libraries are linked to build the load module.

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

For building the user's simulation code, the reference as well as the link of the KVS library, computer algebra library, and particle sampler are required. Below is an example of a Makefile:

```

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

```

3.2. PBVR Client

The procedure followed to build the PBVR client from the source package is presented in this section. The installation of OpenGL and GLUT is required to build the PBVR client. The build procedure depends on the environment (Linux/Mac, Windows). The build of the PBVR client generates the load module shown in Table 3-4.

Table 3-4 Load module of the PBVR client

Directory	Load module	Explanation
KVS	libkvsCore.a	KVS library providing particle format and visualization functions
Common	libpbvrCommon.a	Communication library providing protocol for socket communication
FuctionParser	libpbvrFunc.a	Computer algebra library
glui	libglui.a	GLUI library providing GUI widget
Client	pbvr_client	PBVR client

3.2.1. Linux • Mac

Below are the steps to build a PBVR client on Linux and Mac. The file download destination in the following steps is \$ HOME / JAEA:

- ① Unzip pbvr_inSitu_1.00.tar.gz.

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

- ② Edit pbvr_inSitu_1.00/pbvr.conf and specify the build setting.

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

PBVR_MACHINE ⇒ Specify setting the file name in pbvr_inSitu_1.00/arch

PBVR_MAKE_CLIENT ⇒ Specify 1 (for the PBVR Client)

- ③ The KVS library is built using OpenGL and GLUT, which is disabled when PBVR_MAKE_CLIENT = 0, and using OpenGL and GLUT, which is enabled when PBVR_MAKE_CLIENT = 1. Therefore, if the daemon/particle sampler was built just before, the following command for rebuilding the KVS must be executed:

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

- ④ Build the whole code at the root of the source tree.

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

The PBVR client load module is created within the Client directory.

3.2.2. Windows

The following procedure is used to build the PBVR client in the Windows environment. The file download destination in the following procedure is \$HOME/JAEA:

- ① Unzip pbvr_inSitu_1.00.tar.gz/.

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

- ② Download [glut-3.7.6-bin_x64.zip](http://www.kitware.com/files/download/2010/02/glut-3.7.6-bin_x64.zip) from the following web page:

[About OpenGL/GLUT]

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

- ③ Unzip glut-3.7.6-bin_x64.zip and locate it on the following directory.

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

- ④ Open the following file with Microsoft Visual Studio 2017.

C:\pbvr\pbvr_inSitu_1.00\pbvr.sln

- ⑤ Choose **Release** and **x64** from the pull-down list, as shown in Figure 3-1.

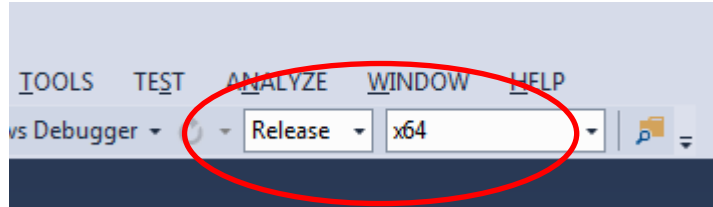


Figure 3-1 Build configuration of VisualStudio2017 for the PBVR client

- ⑥ Go to the menu **Build > Build Solution**.
- ⑦ Place glut32.dll, extracted in step 3 above, in the following directory:

C:\pbvr\pbvr_inSitu_1.00\x64\Release\glut32.dll

4 Setup of In-situ Visualization

The collaboration of the particle sampler coupled to the simulation, the daemon operating on the interactive node, and the PBVR client on the user PC interactively help visualize the batch-processed simulation. For this purpose, certain simple configurations and port forwarding connections are required.

4.1 . Setting Environment Variables

The daemon and the particle sampler that use the environment variables are shown in Table 4-1 below. At runtime, one must set the environment variables with the export command.

Table 4-1 Environment variables referenced by the daemon and the particle sampler

Env. var.	Explanation
VIS_PARAM_DIR	Directory placed the transfer function file (visualization param.)※1
PARTICLE_DIR	Destination directory for the particle files generated by particle sampler※1
TF_NAME	File name of the transfer function (without extension)※2

※1 If not specified, the daemon and the particle sampler will search the current directory in which each is running.

※2 If not specified, the daemon and the particle sampler adopt default.tf as the transfer function name.

4.2. Setting of Visualization Parameters

When the particle sampler converts volume data into visualization particle data, visualization parameters (e.g., transfer function, a range of physical values to be visualized, and a screen resolution) are required. Each item of the visualization parameters is described on a tag basis in the transfer function file. The user can use default.tf, stored in the example of the source package, as the transfer function file for the first startup.

The user needs to place the transfer function file in the directory specified by VIS_PARAM_DIR (shown in Table 4-1) with the file name specified by TF_NAME to execute the in-situ PBVR.

The user can edit the contents of the transfer function file by the GUI by activating the daemon and the PBVR client. The transfer function file specified by the environment variable is then read by the daemon, displayed on the PBVR client, and overwritten after editing.

4.3. Port Forwarding Connection

The particle data and visualization parameters are sent and received by socket communication between the daemon and the PBVR client. For performing socket communication between the interactive node on the remote supercomputer and the local PC, the user is required to connect both ports using ssh port forwarding.

4.3.1 . Remote Connection between Two Machines

The following example shows how to connect local machineA to a remote machine using port forwarding:

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

In the above command, portnumA is the port number of machineA; hostnameB is the host name of machineB; and portnumB is the port number of machineB. Often, hostnameB is displayed on the terminal of machine B and can be confirmed by the hostname command. If port forwarding is permitted on the login node of machine B, there is no need to enter a special host name for hostname B; however, a localhost can be used.

4.3.2. Remote Connection with Several Machines

This section provides an example of connecting the PBVR server and the PBVR client to two remote machines “machineA” and “machineB” via “machineC” for some reason (e.g., security). Once the SSH port forwarding is established, the launching method is basically the same as the stand-alone mode, similar to the two-point remote connection mentioned earlier.

Step 1 [SSH port forwarding A -> C]

```
machineA> ssh -L 60000:localhost:60000 username@machineC  
(Forwarding the 60000 port of machineA to the 60000 port of machineC)
```

Step 2 [SSH port forwarding C -> B]

```
machineC> ssh -L 60000:localhost:60000 username@machineB  
(Forwarding the 60000 port of machineC to the 60000 port of machineB)
```

4.4. Launching Daemon and Port Forwarding

The daemon is started at an interactive node or job and performs socket communication using the PBVR client. The user performs ssh port forwarding from the terminal on your PC to the interactive node, moves to the directory where the daemon load module is located, and starts the daemon as follows:

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

As described above, start the PBVR client from another terminal when waiting for the socket communication connection with the client. The default port number at the daemon startup is 60000. The port number can be changed using the command line option -p at startup, as shown below:

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

The daemon aggregates particle files and updates transfer function files with reference to the environment variables. The daemon then sends and receives data using the PBVR client through the port specified at startup.

5 Particle Sampler

The particle sampler can generate particles for in-situ visualization by inserting the `generate_particles` function into the simulation code. This function is defined in `kvs_wrapper.h` and can be used by referencing and linking particle generation libraries.

The particle sampler is executed using batch processing of the simulation and converts the calculated volume data to the particle data by referring to the environment variables (Section 4.1) and the transfer function file (Section 4.2). In the directory specified by `PARTICLE_DIR`, the particle sampler outputs a particle data file and a `t_pfi_coords_minmax.txt` file using the maximum and minimum coordinates of the area. The particle data file comprises a header file (`kvsml`), a coordinate file (`coord.dat`), a color file (`color.dat`), and a normal file (`normal.dat`) that outputs one set per node at each time step. At the same time, the particle sampler outputs the `$TF_NAME_timestep.tf` file recording history of the transfer function change, `history_timestep.txt` file recording the histogram, and the range of physical values (`state.txt`) that record the time step interval in `VIS_PARAM_DIR`.

5.1 . Particle Generation Function for Structured Grid

```
#include "kvs_wrapper.h"
void generate_particles(
    int time_step, domain_parameters dom, Typs** volume_data, int num_volume_data );
```

This function takes as arguments the time step of simulation, information on calculation area, and the volume data of the simulation result.

- `int time_step`: simulation time step
- `domain_parameters dom`: data structure defines the following calculation domain:

```
typedef struct
{
    float x_global_min; // Minimum of x coordinates in whole domain
    float y_global_min; // Minimum of y coordinates in whole domain
    float z_global_min; // Minimum of z coordinates in whole domain
    float x_global_max; // Maximum of x coordinates in whole domain
    float y_global_max; // Maximum of y coordinates in whole domain
    float z_global_max; // Maximum of z coordinates in whole domain
    float x_min;        // Minimum of x coordinates in subdomain
    float y_min;        // Minimum of y coordinates in subdomain
    float z_min;        // Minimum of z coordinates in subdomain
    int*  resolution;   // Pointer to int resolution[3]
    float cell_length;  // Length of a cell
} domain_parameters;
```

- `Typs ** volume_data`: a pointer to an array of volume data of simulation results. "Type" is a type of a user-specified physical value, while multi-variable volume data are defined as a two-dimensional

array. In the volume data of lattice resolution (X, Y, Z), the value of the position (i, j, k) of the n-th variable is referred to as volume_data [n] [i + j * X + k * X * Y].

- int num_volume_data: number of volume data

5.2. Particle Generation Function for Unstructured Grid

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

This function considers as arguments the time step of the simulation and information on the calculation area, the volume data of the simulation result, and the lattice information of the unstructured lattice.

- int time_step: simulation time step
- domain_parameters dom: data structure defines the following calculation domain:

```
typedef struct
{
    float x_global_min; // Minimum of x coordinates in whole domain
    float y_global_min; // Minimum of y coordinates in whole domain
    float z_global_min; // Minimum of z coordinates in whole domain
    float x_global_max; // Maximum of x coordinates in whole domain
    float y_global_max; // Maximum of y coordinates in whole domain
    float z_global_max; // Maximum of z coordinates in whole domain
} domain_parameters;
```

- Type** volume_data: a pointer to an array of volume data of simulation results. "Type" is a type of user-specified physical value. Multi-variable volume data are defined as a two-dimensional array. The value on the cell vertex of the nth variable is referred to as volume_data [n] [cell].
- float* coordinates: pointer to an array of vertex coordinates. The i-th vertex coordinates (x, y, z) are referenced by (coordinates [3 * i], coordinates [3 * i + 1], and coordinates [3 * i + 2]).
- int ncoords: number of coordinates
- unsigned int * connections: a pointer to the connection list of vertex IDs that form a hexahedral element. Figure 5-1 shows the hexahedral element configuration. The n-th vertex of the i-th hexahedral element is referred to by connections [6 * i + n].
- int ncells: number of elements

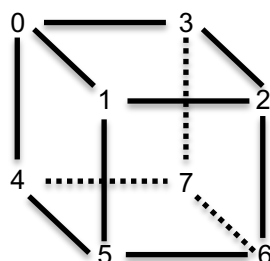


Figure 5-1 Connection of the vertices of a hexahedral element

5.3. Particle Generation Function for AMR Grid

In-situ PBVR supports block-structured AMR, which is optimized for the memory layout of many-core computers. In this type of a grid, an orthogonal grid of N^3 is defined as a unit of the minimum processing area, called “leaf,” and leaves of different sizes are connected in each layer. Therefore, the block-structured AMR is defined as a four-dimensional grid of $N \times N \times N \times L$ (L is the number of leaves). Figure 5-2 shows a two-dimensional example.

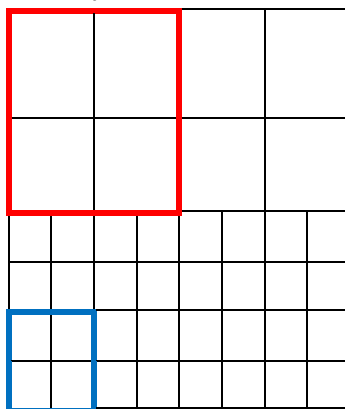


Figure 5-2 Example of a two-dimensional hierarchical grid. The upper part is the lattice of the hierarchy Lv. 1, while the lower part is the hierarchy Lv. 2. The leaf is defined by a 2×2 orthogonal grid. Red is a leaf of Lv. 1, while blue is a leaf of Lv. 2.

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

This function takes as arguments the time step of the simulation, the information of the calculation area, the configuration information of the hierarchical lattice, and the volume data of the simulation result.

- `int time_step`: simulation time step
- `domain_parameters dom`: data structure defines the following calculation domain:

```

typedef struct
{
    float x_global_min; // Minimum of x coordinates in whole domain
    float y_global_min; // Minimum of y coordinates in whole domain
    float z_global_min; // Minimum of z coordinates in whole domain
    float x_global_max; // Maximum of x coordinates in whole domain
    float y_global_max; // Maximum of y coordinates in whole domain
    float z_global_max; // Maximum of z coordinates in whole domain
    int* resolution; // Pointer to int resolution[4]
} domain_parameters;

```

-
- `std::vector<float>& leaf_length`: a reference to an array of leaf lengths; the length of the l -th leaf is referred to as `leaf_length [l]`.
- `std::vector<float>& leaf_min_coord`: a reference to an array of leaf minimum position coordinates; the coordinates of the l -th leaf are referred to as `leaf_min_coord [3 * l]`, `leaf_min_coord [3 * l + 1]`, and `leaf_min_coord [3 * l + 2]`.
- `float** values`: pointer to the array of resulting volume data; multivariate volume data are defined as a two-dimensional array; and in the volume data of the grid resolution (X, Y, Z, L), the value of the position (i, j, k, l) of the n th variable is values `[n] [i + j * X + k * X * Y Refer to + l * X * Y * Z]`.
- `int nvariables`: number of variables

5.4. Connection into Simulation Code

In-situ visualization is possible by compiling the `generate_particles` function into the simulation code of the user. This chapter shows the procedure for connecting the particle sampler to the test simulation code as an example. The test simulation code calculates the value of the charge density on the lattice vertex at each time step using the charge density equation of hydrogen. Therefore, the class name to be calculated is `Hydrogen`. The code without the particle sampler integration is as follows:

```

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

```

In the above source code, the Hydrogen class outputs the volume data of each time step in `hydro.values` for the loop.

5.4.1 . Domain Information and Particle Generation Function

The `generate_particles` function obtains domain information through the structure `domain_parameters`. The user needs to copy the area information obtained from the simulation code to the structure.

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

In the above code, the rank number of the MPI is used to calculate the area of the Hydrogen subdomain. The domain information, resolution, MPI rank, and volume data become the arguments of the `generate_particles` function. The `generate_particles` function is called after the simulation inside the time step loop. In the case of Hydrogen, it is inserted as follows:

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

6 PBVR Client

In the default mode, the PBVR client draws the particle data of the latest time step received from the daemon. The viewer of the PBVR client draws the particle data using OpenGL. Then, the PBVR client provides a GUI to edit the visualization parameters, including transfer functions, and sends the visualization parameters to the daemon. Note that sending and receiving data between the daemon and the PBVR client is performed through an arbitrary port number via socket communication.

6.1 . Launching PBVR Client

The following examples show how to launch the PBVR client:

```
$ pbvr_client [command line options]
```

Table 6-1 Command line options for the PBVR client

option	value	default	explanation
-p	Port number	60000	port number
-viewer	100 ~ 9999 × 100~9999	620×620	resolution of PBVR client
-shading	{L/P/B},ka,kd,ks,n	-	shading method ※1

※1. This argument specifies the shading parameters.

L: Lambert Shading

This method ignores specular reflection in the shading process.

Parameters “ka” and “kd” are the coefficients for ambient and diffusion, respectively, which 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 highlight.

Parameters “ka,” “kd,” and “ks” (coefficients for specular reflection lying between 0 and 1) and “n” (strength of highlight lying between 0 and 100) are used.

B : Blinn–Phong Shading

This is a shading model that simplifies Phong shading using the existing parameters, i.e., “ka,” “kd,” “ks,” and “n”.

6.2. Terminating of PBVR Client

The PBVR client is terminated by pressing Ctrl + c on the console where you started the program. When you press Ctrl + c, the PBVR client synchronizes with the daemon at the time of the time update, and both end. However, the Ctrl + c key input is ignored if communication is interrupted with the Stop button on the time step control panel. Note also that if you terminate the daemon with Ctrl + c, you will not be able to terminate the PBVR client with Ctrl + c.

6.2.1 . Forced Termination

If the PBVR client and the daemon are not terminated by pressing Ctrl + c, forced termination is required using the kill command with the process numbers of the client and the daemon by the ps command, as shown below:

【Forced termination of the PBVR client】

```
$ ps -C PBVRViewer
```

```
  PID TTY          TIME CMD
```

```
19582 pts/6    00:00:00 PBVRViewer
```

```
$ kill -9 19582
```

【Forced termination of the daemon】

```
$ ps -C CPUserver
```

```
  PID TTY          TIME CMD
```

```
19539 pts/5    00:00:00 CPUserver
```

```
$ kill -9 19539
```

6.3. Using PBVR Client GUI

6.3.1. Viewer

As shown in Figure 6-1, the **Viewer** displays the rendering result of the particle data.

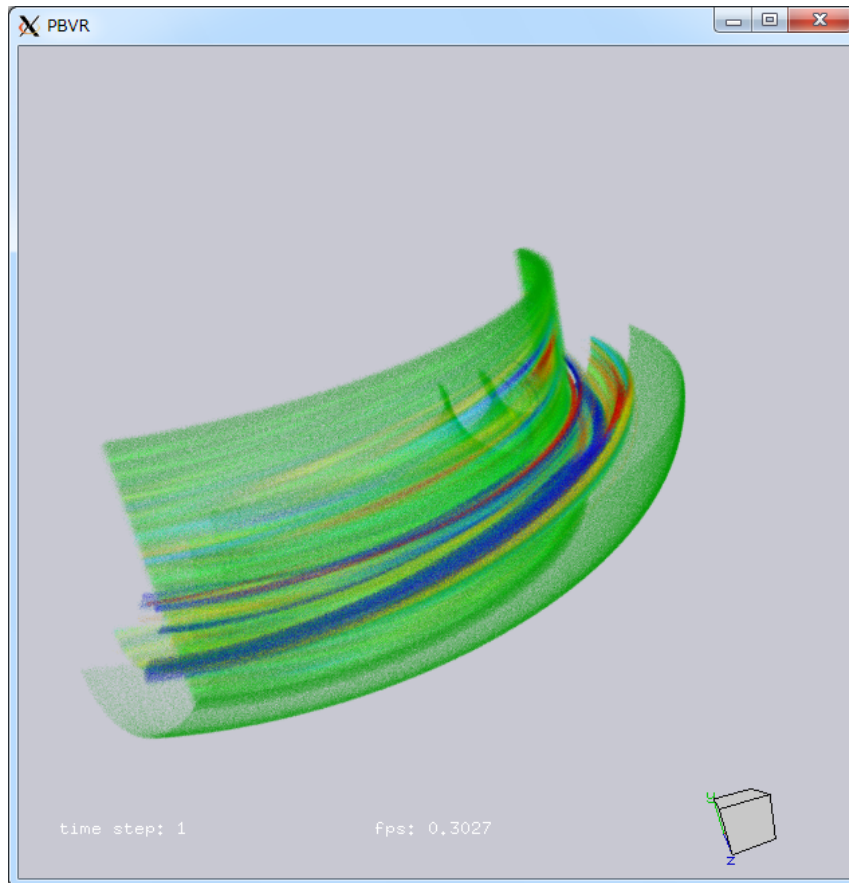


Figure 6-1 Viewer

[Operations]

Rotation: mouse left-dragging

Translation: mouse right-dragging

Zoom: Shift + left-dragging or dragging while pressing the mouse wheel

Reset: home button (fn + left arrow on Mac)

[Display]

time step: time step of the displayed data

fps: frame rate [frame/s]

6.3.2. Main Panel

Figure 6-2 shows the main panel of the PBVR client. The items in the panel are described below:

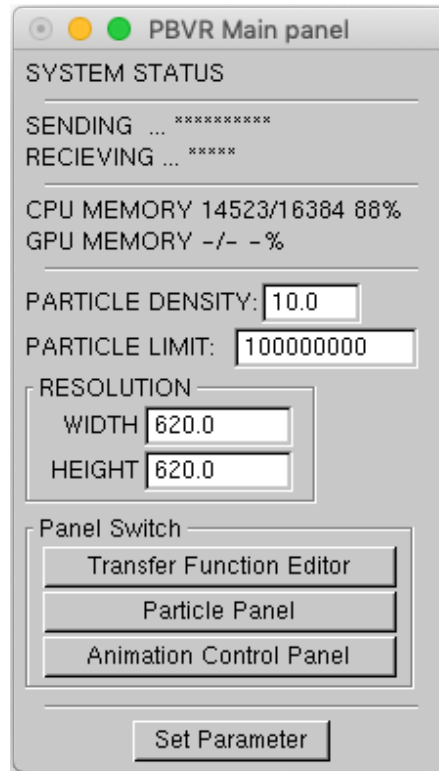


Figure 6-2 Main panel

- **PARTICLE DENSITY**

specifies the particle density related to the image depth

- **PARTICLE LIMIT**

specifies the maximum number of particles generated in the PBVR server. Use this to avoid the explosive increase of the number of particles (e.g., caused by the false settings of a transfer function). The number is multiplied by 10^6

- **RESOLUTION**

specifies the Viewer's resolution

- **Transfer Function Editor**

displays the Transfer Function Editor described in the next section

- **Particle Panel**

displays the Particle Panel described in the next section

- **Animation Control Panel**

displays the Animation Control Panel described in the next section

- **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

Set parameter

sends parameters specified in the “Main panel” to the server program

6.3.3. Transfer Function Editor

The **Transfer Function Editor** edits the transfer functions, which assigns color/opacity to each scalar value for volume rendering. It is activated by hitting the Transfer Function Editor button in the Main panel. In standard volume rendering, a transfer function is defined by only one physical quantity; however, the 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) Using an algebraic equation, synthesize a multidimensional transfer function from the one-dimensional transfer functions defined by color functions $C1, C2....$ and opacity functions $O1, O2, ...$

This new transfer function design adds significant flexibility to visualization. Figure 6-3 shows the Transfer Function Editor. Each item in the panel is explained below:

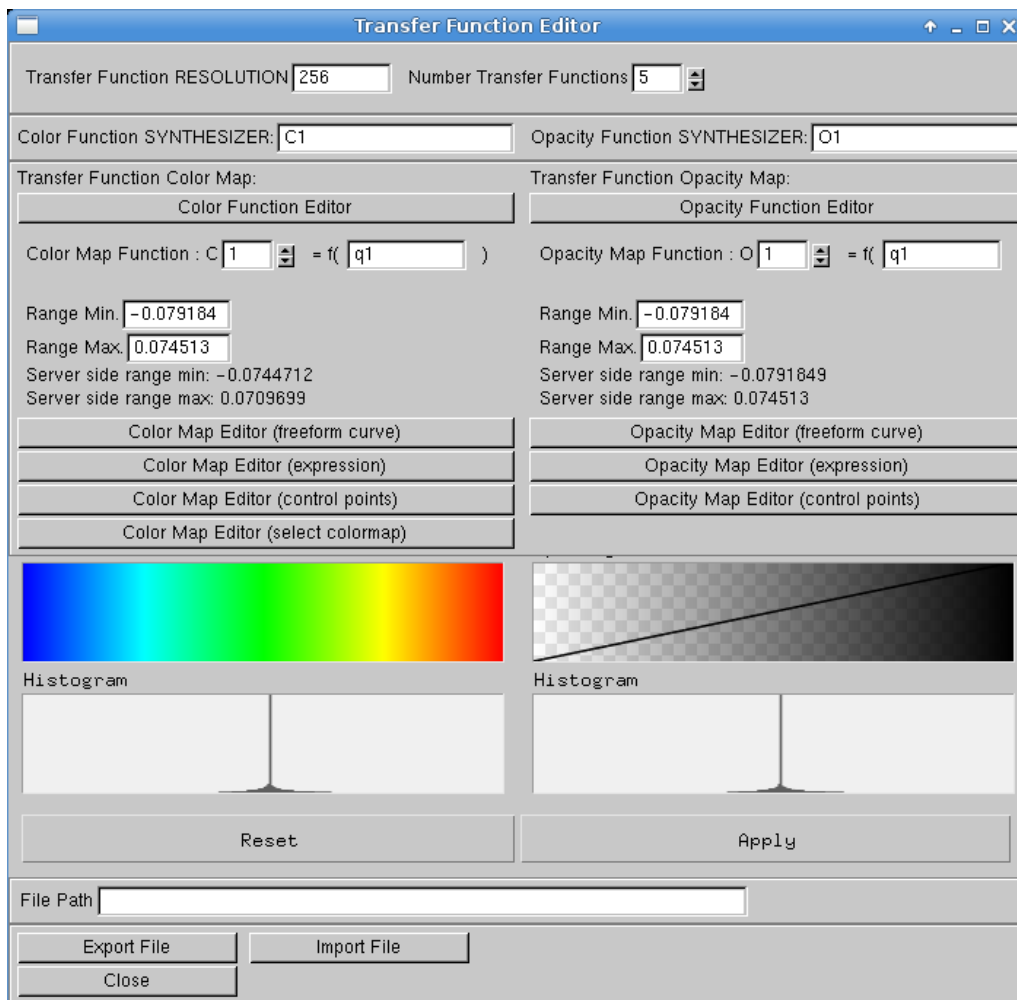


Figure 6-3 Transfer Function Editor

[Operations]

Scale change in histogram: drag the mouse up/down on the **Histogram**

- **Transfer Function RESOLUTION**

specifies the resolution of the transfer function

- **Number Transfer Functions**

specifies the limit number of the transfer functions that can be created

- **Color Function SYNTHESIZER**

specifies the composition formula of the transfer color functions created under the names C1 to C [N] *1

- **Opacity Function SYNTHESIZER**

specifies the composition formula of the transfer opacity functions created under the names O1 to O [N] *1

- **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

- **Close**

closes the Transfer Function Editor

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

6.3.3.1 Color Map Editor Panel

[Transfer Function Color Map category]

creates and displays the transfer function corresponding to the transfer color function name described in the transfer function synthesis formula of the Color Function SYNTHESIZER

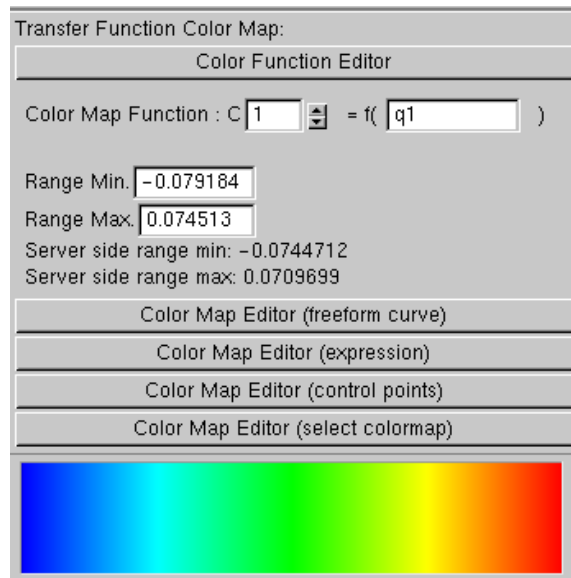


Figure 6-4 Transfer Function Color Map Category

- **Color Function Editor**

displays the Color Function Editor screen and creates and selects the transfer color function of C1 to C [N]. *1

- **Color Map Function**

creates and displays the transfer color function by the name of C1 to C [N]. *1 and defines the (synthesized) variable quantity used for the 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, and 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**

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

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

displays a sub-panel that specifies a transfer function with a freeform curve (use the mouse to edit the freeform curve)

- **Color Map Editor (expression) button**

displays a panel that creates the transfer functions of the variables and colors corresponding to the selected transfer function names by the numerical formula description

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

displays a panel that creates the transfer functions of the variables and colors corresponding to the selected transfer function names using control point designation.

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

displays a panel that creates the variables and color transfer functions corresponding to the selected transfer function name by selecting them from the color bars prepared in advance.

- **Color**

The color bar of the variable and color transfer function created by this editor is displayed.

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

6.3.3.1.1 . Color Function Editor

Using the Color Function Editor button, display the Color Function Editor screen and create and select the transfer color function of C1 to C [N].

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

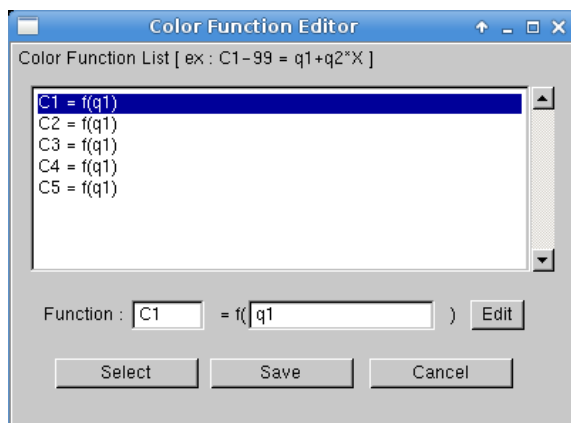


Figure 6-5 Color Function Editor screen

- **Color Function List**

display the created transfer function

- **Function**

enter the transfer function. (C [N] = f (variable))

- **Edit button**

reflect on the Color Function List

- **Save button**

apply the transfer function of the Color Function List

- **Select Button**

apply the transfer function of the Color Function List and select the transfer function of the list selection

6.3.3.1 .2. Color Map Editor

Use a color map editor (freedom curve) button to display a panel that creates a transfer function of the variable and color corresponding to the selected transfer function name with a free curved line input by mouse.

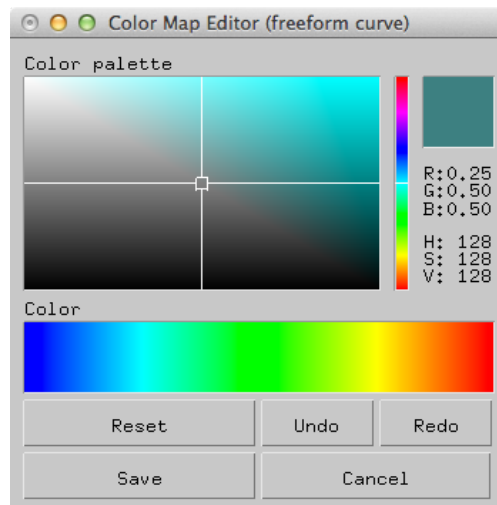


Figure 6-6Color Map Editor (freedom curve)

- **Color palette**

specifies the saturation, brightness, and hue of a color with the mouse cursor. On the left, the horizontal and vertical axes correspond to the saturation and brightness, respectively, while 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 the **Color palette** and the **RGB bar**.

- **Color**

blends the colors in the **Color** area with a color specified with the **Color palette** and the **RGB bar**. To specify the locations in the **Color** area, trace the locations by dragging the mouse cursor while pressing the left mouse button. The blending ratio of the original and overpainting colors 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 completely painted by the specified color rather than by the blended colors. Similarly, when the vertical center line of the **Color** bar is traced, the colors are replaced with the 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

6.3.3.1.3. Color Map Editor (expression)

Use a color map editor (expression) button to display a panel to create a transfer function by considering the equations as input.

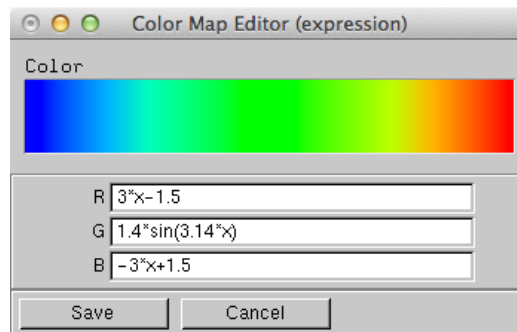


Figure 6-7 Color Map Editor (expression)

- **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

6.3.3.1.4. Color Map Editor (control points)

Use a color map editor (control points) button to display a panel for creating a transfer function. This editor considers control points as input.

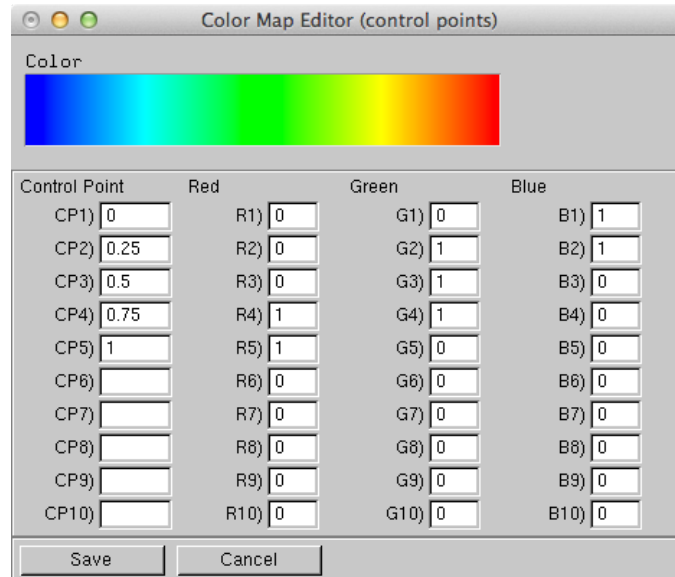


Figure 6-8 Color Map Editor (control points)

- **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

6.3.3.1.5. Color Map Editor (select colormap)

Use a color map editor (select colormap) button to display a panel to create a transfer function from the preset color bar templates.

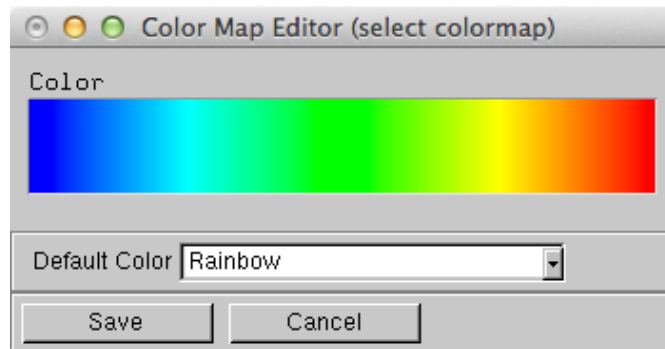


Figure 6-9 Color Map Editor (select colormap)

- **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
 - Black
 - White
- **Save**
 - saves the transfer function created with this panel
- **Cancel**
 - closes the panel

6.3.3.2 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.

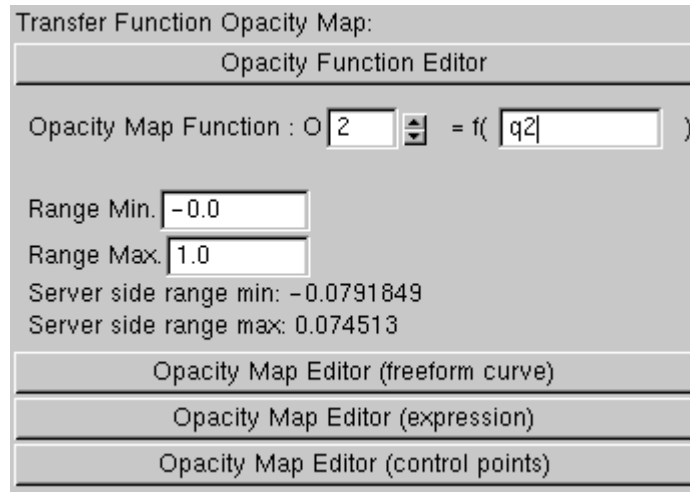


Figure 6-10 Transfer Fuction Opacity Map Category

- **Opacity Function Editor Button**

displays the Opacity Function Editor screen and creates and selects the transmission opacity function of O1 to O [N]. *1

- **Opacity Map Function**

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, and 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**

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

- **Opacity Map Editor (freeform curve) button**

displays a panel for creating a transfer function with a freeform curve (use the mouse to edit the freeform curve)

- **Opacity Map Editor (expression) button**

displays a panel for creating a transfer function of the variable and opacity corresponding to the selected transfer function name using a mathematical formula description

- **Opacity Map Editor (control point) button**

displays a panel for creating a transfer function of the variable and opacity corresponding to the selected transfer function name using a control point designation

- **Opacity**

displays the transfer function curve of the variable and opacity created with this editor

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

6.3.3.2.1 . Opacity Function Editor

Use the Opacity Function Editor button to display a panel to create and select the transmission opacity function of O1 to O [N].

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

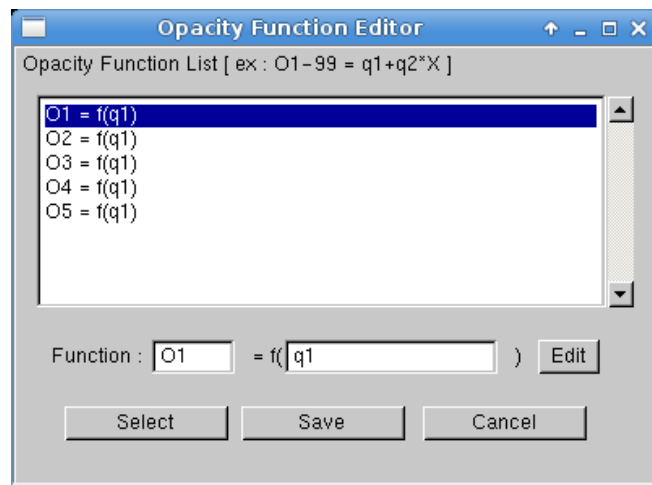


Figure 6-11 Opacity Function Editor

- **Opacity Function List**

displays the created transfer function

- **Function**

enter the transfer function (O [N] = f (variable))

- **Edit Button**

reflects on the Opacity Function List

- **Save button**

applies the transfer function of the Opacity Function List

- **Select button**

applies the transfer function of the Opacity Function List and selects the transfer function of the list selection

6.3.3.2.2. Opacity Map Editor (freedom curve)

Use the Opacity Map Editor button to display a panel to create a transfer function of the variable and opacity corresponding to the selected transfer function name with the free-form curve input using a mouse.

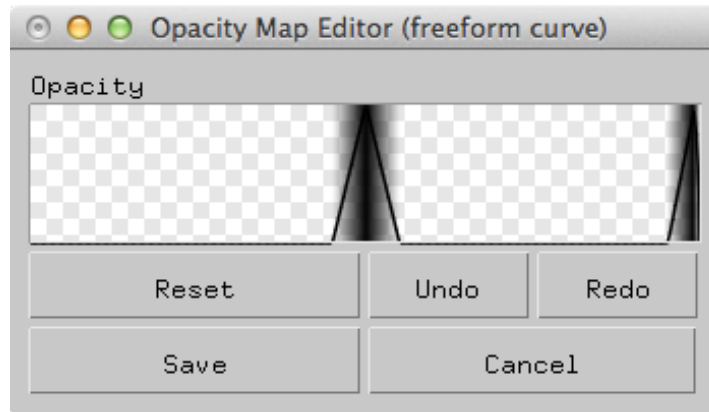


Figure 6-12 Opacity Map Editor (freeform curve)

- **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 the 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

6.3.3.2.3. Opacity Map Editor (expression)

Use the Opacity Map Editor (expression) button to display a panel to create a transfer function using equations.

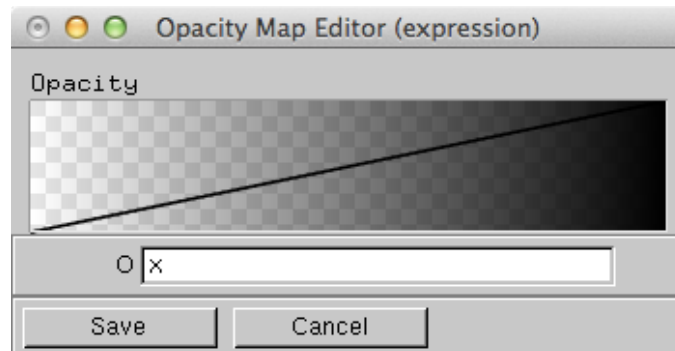


Figure 6-13 Opacity Map Editor (expression)

- **Opacity**
displays the transfer function for the opacity specified by the equation in 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

6.3.3.2.4. Opacity Map Editor (control point)

Use the Opacity Map Editor (control points) button to display a panel to create a transfer function by taking equations as input.

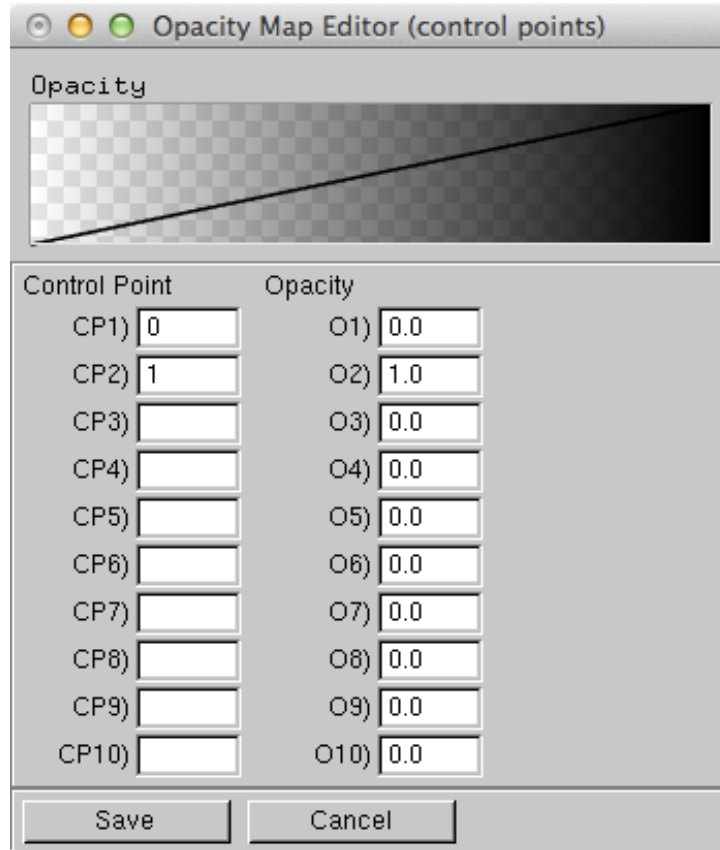


Figure 6-14 Opacity Map Editor (control points)

- **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

6.3.3.3 Function Editor

Table 6-2 lists the built-in math operations available in the function editor, which can be used to synthesize the transfer functions and variable quantities and define the colormap/opacity curves.

Table 6-2 Math operations in the function editors

Math operation	In function editors
+	+
-	-
×	*
/	/
Sin	sin(x)
Cosin	cos(x)
Tangent	tan(x)
Logarithm	log(x)
Exponential	exp(x)
Square root	sqrt(x)
Power	x^y

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

6.3.4. Time Panel

Figure 6-15 shows the **Time panel** that specifies the time steps for visualization. Each widget works as described in the following:

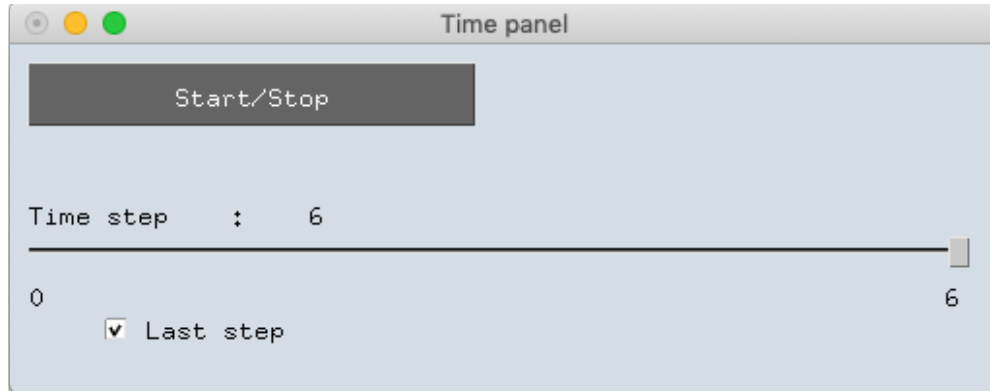


Figure 6-15 Time panel

- **Time step**
specifies the time step of the data to be rendered
- **Last step (check box)**
rendering the latest step particle
- **Start/Stop**
starts/stops the communication between the PBVR client and the PBVR server
When the button is red, communication is stopped, while when it is gray, communication is in progress.

6.3.5. Particle Panel

Figure 6-16 shows the **Particle panel** that integrates multiple particle datasets. The particle panel is activated by hitting the Particle panel button in the Main panel. Each widget works as described in the following:

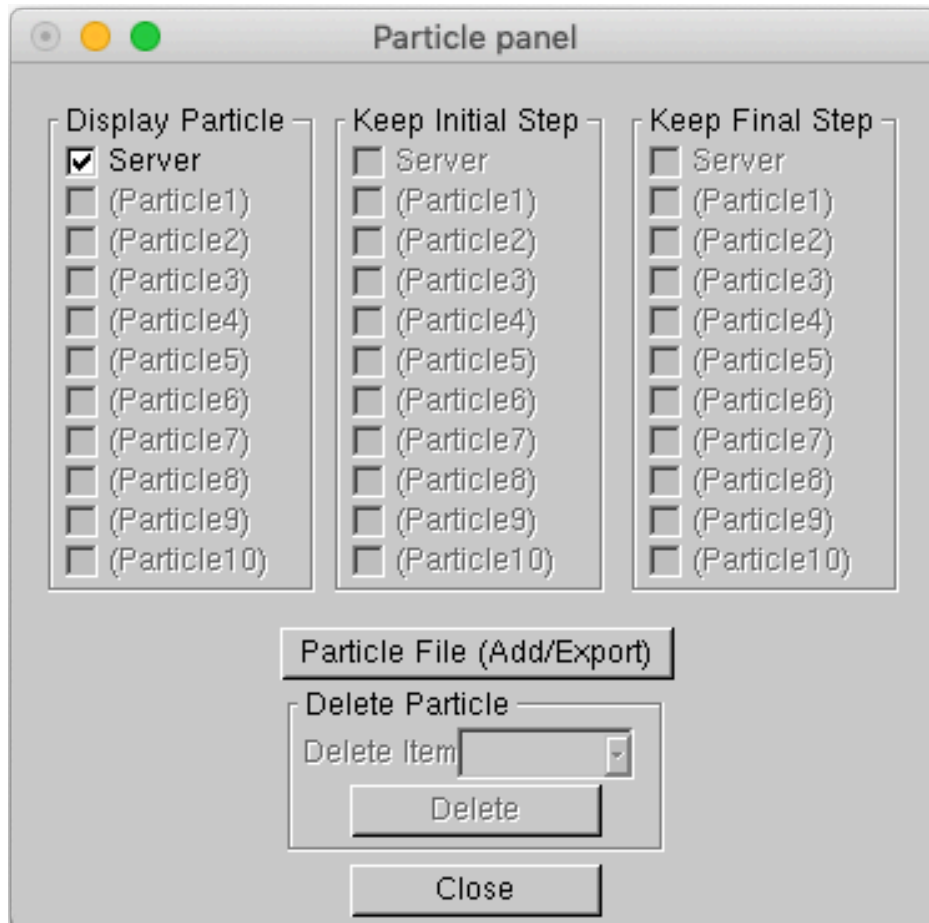


Figure 6-16 Particle panel

- **Display Particle**

shows a list of particle datasets sent from the PBVR server or are loaded from local files (maximum: 10 files)

- 1) Server check box

activated when a particle dataset from the PBVR server is integrated with the local particle data sets. This checkbox is not available in stand-alone mode.

- 2) (Particle1)–(Particle10) check box

activated when the particle data sets loaded from the local files are integrated. The checkbox is not available before the particle data sets are loaded via the Particle file panel or command line options `-pin1`, `-pin2`, ..., `-pin10`.

- **Keep Initial Step**

specify particle data sets, in which the initial step data are displayed before the time series starts, when the integrated particle datasets start from different time steps

- **Keep Final Step**

specify particle data sets, in which the final step data are displayed after the time series ends, when integrated particle datasets end at different time steps

- **Particle File (Add/Export)**

open the Particle File sub-panel

- **Delete Particle**

specify a particle data set to be deleted from a list in the Display Particle

- **Delete**

delete a particle data set

- **Close**

close Particle panel

The behavior of the particle integration is explained using the example listed in Table 6-3. All time steps are listed in Table 6-4 when both Server and Particle check box are checked. The first time step is kept displayed as shown in Table 6-5 in case of a checked Keep Initial Step in Client. In case of a checked Keep Final Step in Client, the last time step is kept displayed as shown in Table 6-6.

Table 6-3 List of server's and client's particles in five time steps.

	step0	step1	step2	step3	step4
Server		○	○	○	○
Client	○	○	○	○	

Table 6-4 Particle's time steps displayed in default.

	step0	step1	step2	step3	step4
Server	×	1	2	3	4
Client	0	1	2	3	×

Table 6-5 Particle's time steps displayed with Keep Initial Step.

	step0	step1	step2	step3	step4
Server	×	1	2	3	4
Client	0	0	0	0	0

Table 6-6 Particle's time steps displayed with Keep Final Step.

	step0	step1	step2	step3	step4
Server	×	1	2	3	4
Client	3	3	3	3	3

6.3.5.1 Particle File sub-panel

The **Particle File panel** is a panel for reading and writing particle data files. This panel is shown when the **Particle File** button is hit.

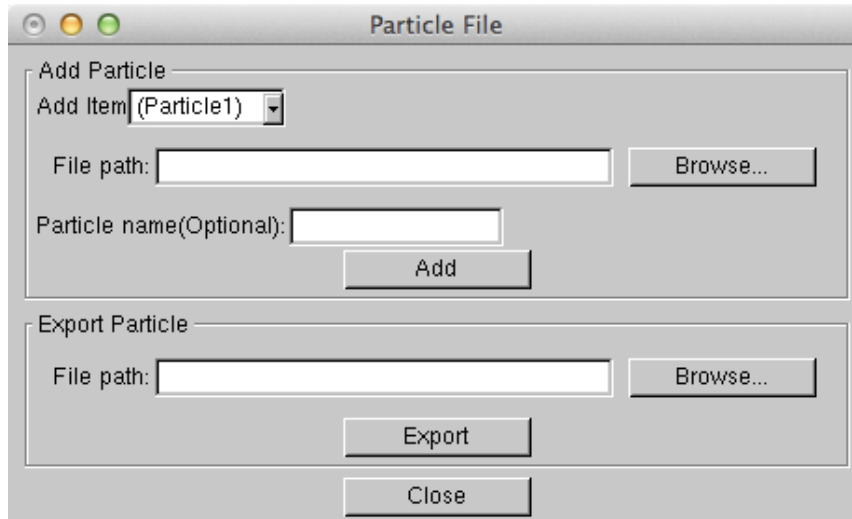


Figure 6-17 Particle File panel

- **Close**

close Particle File panel.

【Add Particle category】

- **Add Item**

specifies a particle data slot, in which a new particle dataset is loaded. When an old particle data set exists for the particle data slot, the slot is overwritten by a new particle data set

- **File path**

specifies a particle data file

- **Browse**

opens a file dialog for specifying the path to a visualization parameter file

- **Particle name (Optional)**

specifies the name of a particle dataset shown in the **Particle panel**

- **Add**

adds a particle dataset to the **Particle panel**

【Export Particle Category】

- **File path**

specifies a particle data file

- **Browse**

opens a file dialog for specifying the path to a visualization parameter file

- **Export**

outputs integrated particle data

6.3.5.2 Saving Particle Data

The particle data are stored using the Particle File panel in the Particle panel. The following is an example specification of the particle data file name, in which the prefix is p1, and the following files are generated:

```
./particle/p1_XXXXX_YYYYYYY_ZZZZZZZ.kvsm1  
./particle/p1_XXXXX_YYYYYYY_ZZZZZZZ_colors.dat  
./particle/p1_XXXXX_YYYYYYY_ZZZZZZZ_coords.dat  
./particle/p1_XXXXX_YYYYYYY_ZZZZZZZ_normals.dat
```

Here, XXXXX indicates time; YYYYYYY indicates subvolume number; ZZZZZZZ indicates the total number of subvolumes; and the colors, coords, and normals indicate the data of the color, coordinates, and normal vector, respectively. Enter the particle data file name and press the Export button to start particle storage. While the particle data storage process is in progress, the Export button becomes inactive, as shown in Figure 6-18. Save the particle data after saving all the time series particle data Ends, and then the Export button returns to the active state.

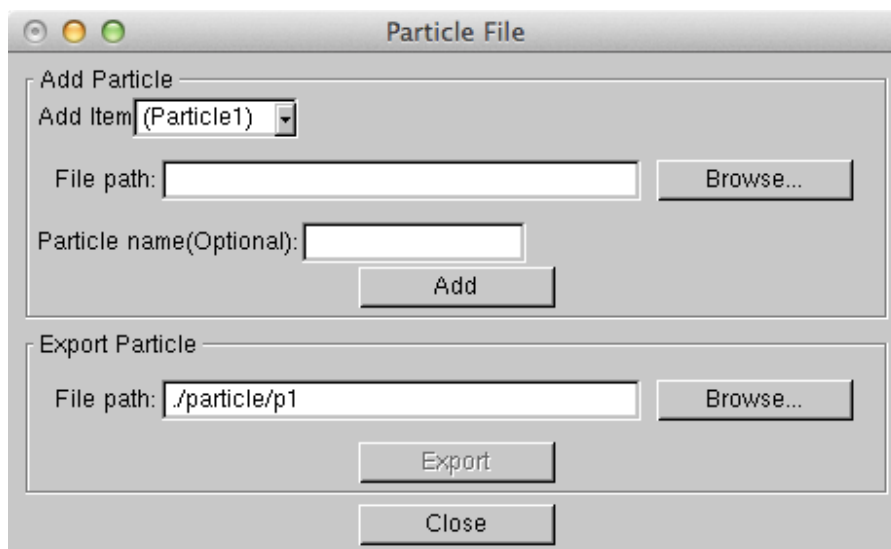


Figure 6-18 Particle File panel (saving particle data)

6.3.6. Image File Production

The PBVR client saves the image data on the Viewer in two modes and plays it as a movie. Image file production is activated by hitting the Animation Control Panel button in the Main panel.

- Time series data mode

Saves images of time series data as a series of image data files with the BMP format; the image data files are converted or compressed as a movie file via free software such as ImageMagic and ffmpeg

- Key frame animation mode

Keeps geometry information of the Viewer at an arbitrary point as a key frame and plays a series of key frames as a key frame animation

Figure 6-19 shows the **Animation Control Panel**. Each widget works as described as follows:

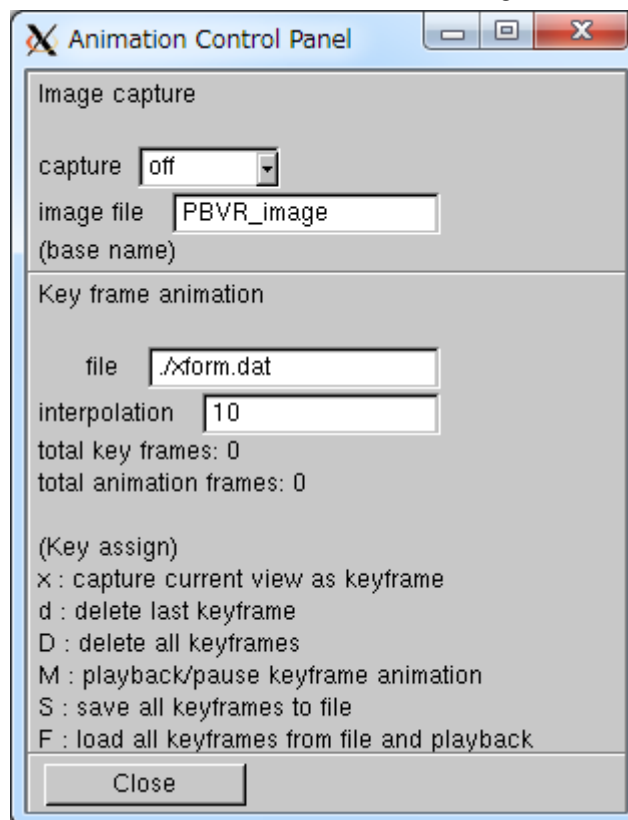


Figure 6-19 Animation Control Panel

- **capture**
controls on/off of image production
- **image file**
specifies a prefix of the image data files with the default name PBVR_image
- **file**

specifies a key frame file containing a series of geometry data with the default name `./xform.dat`

- **interpolation**

specifies the number of frames used for the 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; the value is initialized to 0 and incremented (or decremented) by pressing “x” (or “d”), and the value is initialized to 0 by pressing “D”

- **total animation frames**

shows the number of total frames stored in the current key frame animation, which is calculated as:

$$(\text{total key frames} - 1) \times \text{interpolation}$$

- **Close**

close Animation Control Panel

6.3.6.1 Image production

Image files are produced as follows:

- ① Specify prefix of 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.” When “-iout” option is not specified, they are saved in the current directory “./.” The following shows an example of the image data produced with the default prefix “PBVR_image:”

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

When the image files are produced from a key frame animation, which will be 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 obtained by pressing **Stop** in **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 Viewer by clicking it.
- ③ Adjust view and press “x” to store the geometry information of view on a memory.
- ④ Repeat ③.
- ⑤ Press “M (Shift+m)” to play the key frame animation.
- ⑥ If the contents of the key frame animation is OK, press “S (Shift+s)” to save a series of geometry information in the key frame file.

【Play a key frame file】

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

Table 6-7 Keys used for controlling the key frame animation

Key	Function
X	Add geometry information of the current viewer to the key frame data on a memory
D	Delete the last key frame
D	Delete all key frames
M	Play and pause key frame data on a memory
S	Save key frame data on a memory to a key frame file
F	Load a key frame file and play its key frame data

6.3.6.3 Key frame animation of the time series data

A key frame animation of the time series data is produced as follows:

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

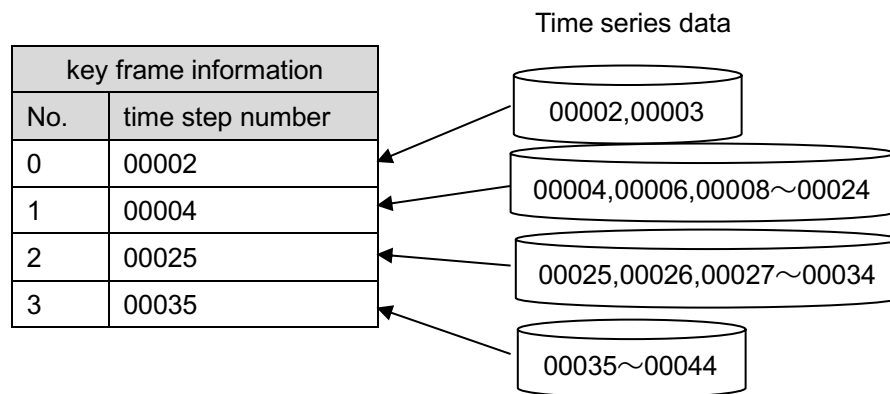


Figure 6-20 Key frame animation for the time series data

In an example in Figure 6-20, if one uses 10 interpolation frames between key frames, five interpolation frames are assigned to time steps 00002 and 00003 in between key frame nos. 0 and 1. In contrast, in between nos. 1 and 2, 10 interpolation frames are assigned to the time steps from 00004 to 00024. As a result, 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:

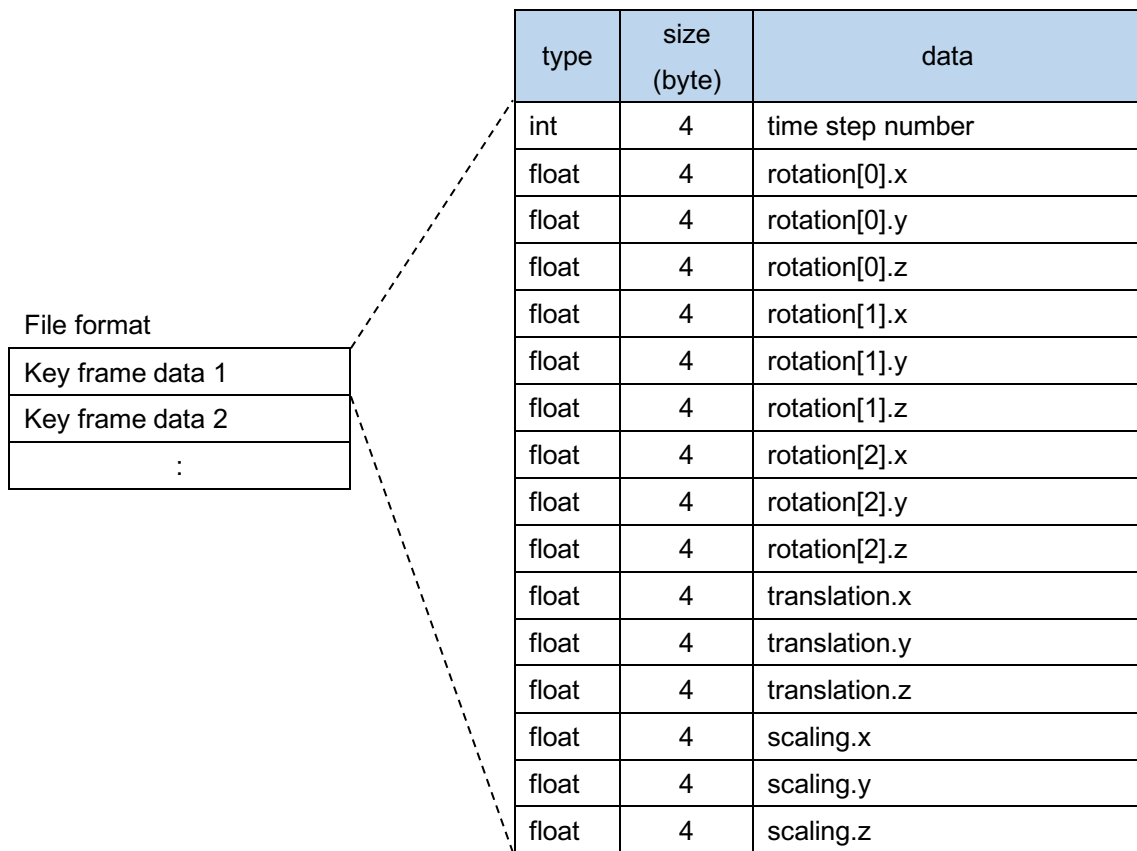


Figure 6-21 Key frame file format

7 Samples

This section shows an example of the port-forward connection from Linux/Mac to the many core cluster JKNL and supercomputer ICEX using Inline, included in the source code package to execute the in-situ PBVR. Moreover, this section shows an example of accessing a remote server from Windows and executing the in-situ PBVR.

7.1 . ICEX

The following shows the procedure for executing an in-situ PBVR for the unstructured grid in an interactive job on ICEX. In this example, the RSA public key and the ssh host name / user name are assumed to have been set. The source package is also assumed to be located at / home / g1 / userid. First, create a shell program that executes interactive jobs and place it in:

```
ICEX:/home/g1/UserID/pbvr_inSitu_1.00/Example/Hydrogen_unstruct/run.sh
```

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

Launch the first terminal and execute the following command:

[Terminal 1]

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

Launch the second terminal and execute the following command:

[Terminal 2]

```
$ ssh -L 61000:r16iOn0:61000 UserID@ICEX  
⇒ Specify hostname at Terminal 1 launching pbvr_daemon
```

Launch the third terminal and execute the following command:

[Terminal 3]

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

Launch the fourth terminal and execute the following command:

[Terminal 4]

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

7.2. JKNL

The command when executing the in-Situ PBVR for the hierarchical grid on JKNL is shown. In the following example, the RSA public key and the ssh host name/user name are assumed to have been set. The source package is also assumed to be located at / home / g1 / userid. First, create a shell program that executes interactive jobs and place it in:

```
JKNL:/home/g1/UserID/pbvr_inSitu_1.00/Example/Hydrogen_AMR/run.sh
```

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

Launch the first terminal and execute the following command:

[Terminal 1]

```
$ ssh ユーザ ID@jknlf.s.tokai-sc.jaea.go.jp
$ cd /home/g1/UserID/pbvr_inSitu_1.00/Example/Hydrogen_AMR
$ qsub run.sh
$ hostname
jkn13
$ export VIS_PARAM_DIR=/home/g1/UserID/pbvr_inSitu_1.00/Example/Hydrogen_AMR
$ export
PARTICLE_DIR=/home/g1/UserID/pbvr_inSitu_1.00/Example/Hydrogen_AMR/jupiter_particle_out
$ export TF_NAME=jupiter
$ cd $PBS_O_WORKDIR
$ ./pbvr_daemon -p 61000
```

Launch the second terminal and execute the following command:

[Terminal 2]

```
$ ssh -L 61000: jkn13:61000 UserID@jknlf.s.tokai-sc.jaea.go.jp
⇒ Specify hostname at Terminal 1 launching pbvr_daemon
```

Launch the third terminal and execute the following command:

[Terminal 3]

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

Launch the fourth terminal and execute the following command:

[Terminal 4]

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

7.3. Windows

The procedure for remotely connecting from a Windows client to a remote server is shown below. The following procedure assumes transfer from port 60000 of the client to port 60000 of the server. Set up port forwarding with the help of an SSH client software (e.g., TeraTerm or Putty) to connect the PBVR client in a Windows machine to the PBVR server in a remote machine. The following is an example for TeraTerm:

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

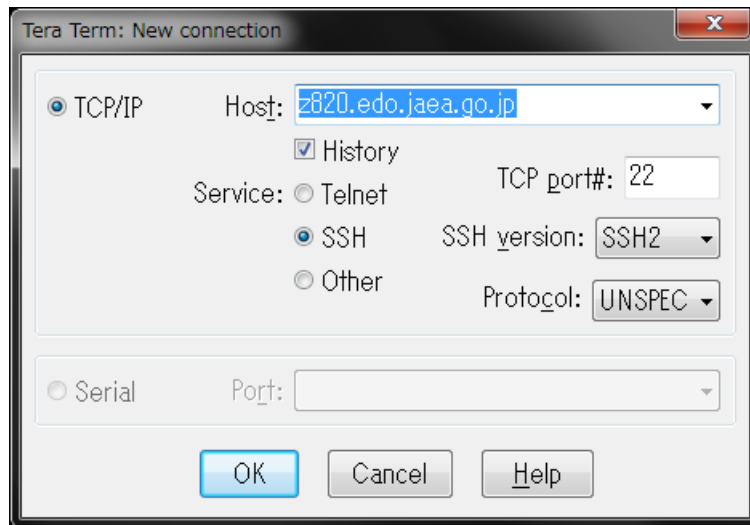


Figure 7-1 Tera Term dialog 1)

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

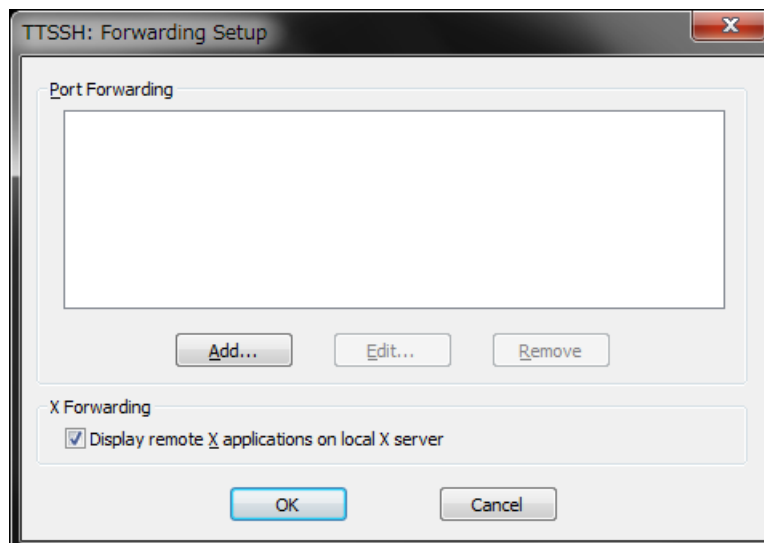


Figure 7-2 Tera Term dialog 2)

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

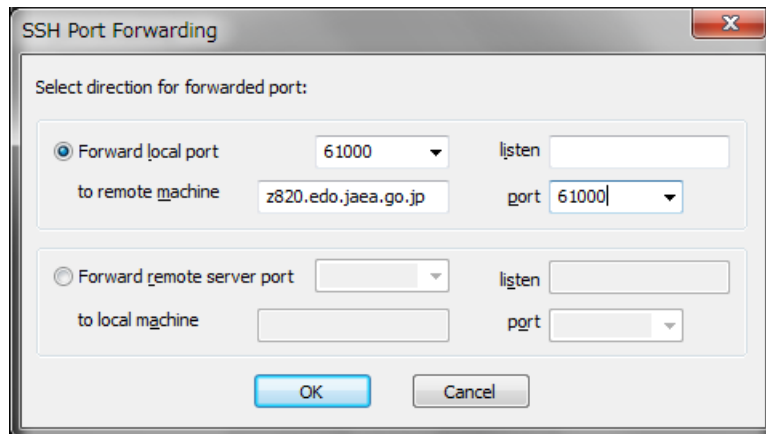


Figure 7-3 Tera Term dialog 3)

- 4) Connect to the server. Select **File > New Connection** from the menu bar. In the **New Connection** panel, enter the host name of the server 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 then click on **OK**.

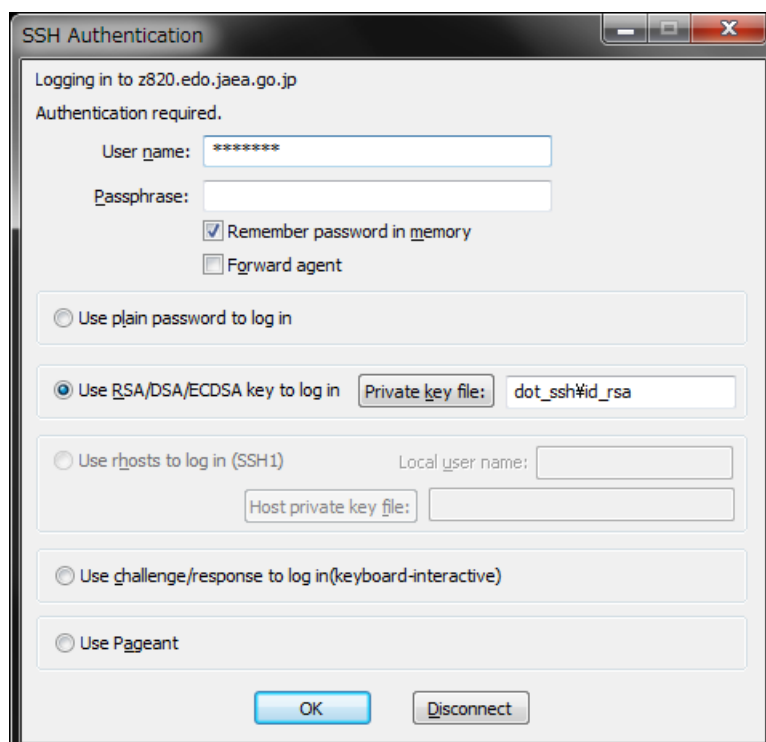


Figure 7-4 Tera Term dialog 4)

The procedures that follow show how to launch the 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 the PBVR client.

Step1 [Launch PBVR server]

```
Server> mpiexec -n 4 pbvr_server -p port_number
```

Step2 [Set a client parameter for Windows]

```
Windows> set TIMER_EVENT_INTERVAL=1000
```

Step3 [Launch PBVR client]

```
Windows> pbvr_client.exe -vin filename -p port_number
```

Note that the PBVR client on a Windows machine can be launched by executing a batch file using the following lines:

```
set TIMER_EVENT_INTERVAL=1000  
pbvr_client.exe -vin filename -p port_number
```