In-Situ PBVR (v.2.1)
User Guide

April. 2023

Japan Atomic Energy Agency

Center for Computational Science & e-Systems

Revision Record

Version number	Date revised	Revised chapter	Revised content
1.0	2019.03.28	-	Release
1.1	2021.05.01		Modified to common GUI with Client-Server PBVR.
1.2	2022.03.30		Added polygon composition function
2.0	2023.01.01		Source codes of CS/IS-PBVR are integrated. Pre-defined color maps are updated.
2.1	2023.04.20		Supported element type is added for unstructured grid liblary.

Contents

1 Introduction

This document is a user guide for In-Situ PBVR, which is an in-situ remote visualization system developed at the Center for Computational Science & e-Systems in Japan Atomic Energy Agency. In today's supercomputers, the computation speed greatly exceeds the I/O speed, making it difficult to output computation results. As a result, it has become difficult to apply the conventional visualization method, in which calculation results on a remote storage are transferred to a PC at hand for visualization. In-Situ visualization avoids large-scale data I/O by having the visualization program coupled to the simulation and processed simultaneously in the same environment as the simulation to reliably generate visualization images. In-Situ PBVR is a framework that enables interactive visualization through particle-based visualization and interactive in-situ control technology, whereas interactive in-situ visualization is difficult to achieve with conventional polygon-based methods. In-Situ PBVR is a C++ library built using the Particle Based Volume Rendering (PBVR) method, a particle-based volume rendering method developed by Koyamada Laboratory, Kyoto University, and the visualization library KVS. The framework consists of three components: a particle sampler, a daemon, and a PBVR client (Figure 1-1).

（1） Particle Sampler

The particle sampler is a visualization library that is coupled to a simulation code to generate particles in the same environment as the calculation. To build a simulation + in-situ visualization code, an array of calculation results is passed to the visualization functions provided by the particle sampler and inserted into the simulation code. The particle sampler refers to the visualization parameter file on the storage, converts the calculation results of each time step into compressed particles for visualization, and outputs them on the storage. The particle files are output from each process in a distributed manner.

（2） Daemon

The daemon runs on the login node or the interactive job and mediates between the particle files on the storage and the visualization parameters sent from the PBVR client. The daemon monitors the files on the storage, aggregates the distributed output particle files, and transfers them to the user PC via the network. The daemon also receives the visualization parameters sent from the PBVR client and outputs them as a visualization parameter file to be referenced by the particle sampler. Since this operation is executed asynchronously with the simulation, interactive control is possible without interfering with the simulation.

（3） PBVR Client

The PBVR Client runs on the user's PC and provides a viewer to display the visualization results and a GUI to edit the visualization parameters. PBVR Client communicates with the daemon using port forwarding. The PBVR client receives the particle data from the daemon, displays the volume rendered image on the viewer, and sends the visualization parameters edited by the user on the GUI to the daemon.

In-Situ PBVR implements a multivariate visualization function that can be used universally for various simulations. In 3D data visualization, colors and opacity are assigned to the physical values of the simulation results, and the function that describes the relationship between them is called the transfer function. In conventional visualization, 1D transfer functions that assign color and opacity to a single physical value are used. However, it is difficult to design a multidimensional transfer function for multivariate data using conventional visualization methods. The In-Situ PBVR provides a transfer function editor that allows users to design multidimensional transfer functions using algebraic expressions. The transfer function editor allows users to edit the 1D transfer function of each physical variable, and to write a multidimensional transfer function using arbitrary algebraic expressions with these 1D transfer functions as variables. In this algebraic expression, basic operators, elementary functions, and differential operators can be used.

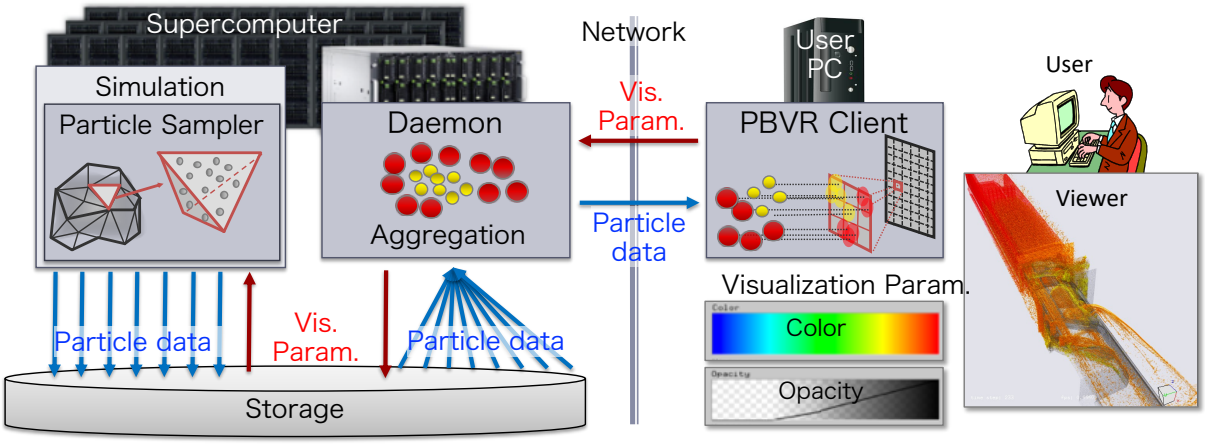


Figure 1-1 The configuration of In-Situ PBVR framework

1.1. Overview of Particle Sampler

The particle sampler is parallelized by hybrid of MPI/OMP programming model, and further accelerates the generation of particle data by using SIMD operation. The particle sampler is MPI parallelized without changing the domain decomposition of simulation, and generates particles in element parallel by OpenMP in each decomposed domain. The synthesis of physical values, the synthesis of transfer func tions, and the 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 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, multivariate data of simulation results, lattice data, and global coordinates of the region are input as arguments of the particle sampler. Also, the memory layout of multivariate data assumes an array structure (SOA).

1.2. Overview of Daemon

The daemon is executed on an interactive node or interactive job of the supercomputer, and is a key to realize interactive visualization during batch processing. The daemon constantly monitors files on

storage, and collects particle files output by the particle sampler. Because this collection is asynchronous with the particle sampler and the simulation, it does not impede the performance of the simulation. The collection of the particle files is parallelized by OpenMP, and the daemon aggregates the particle files into single particle data and sends it to the PBVR client. The daemon also receives the visualization parameters sent from the PBVR client and updates the visualization parameter file on the storage used by the particle sampler.

In the in-situ PBVR, since the daemon and the PBVR client send and receive data by socket communication via the Internet, the interactive node and the user PC need to be connected by port forwarding. Therefore, if the port forwarding is not permitted on a supercomputer, the daemon does not work and interactive visualization can not be used. In addition, interactive visualization can not be used even on a super computer that employs staging I/O where file output does not occur until the end of batch processing.

1.3. Overview of PBVR Client

The PBVR client is lunched on the user PC, and is comprised of a screen for displaying the visualization result, and a transfer function editor. In the transfer function editor, a “transfer function synthesizer” (TFS) is implemented in which is a function for realizing multivariate visualization. The TFS has a volume data synthesisis function which combines variables included in the result data to generate new volume data, and a transfer function synthesis function which combines multiple transfer functions. The user specifies the synthesis function by an algebraic expression on TFS. The algebraic expression is transferred to the PBVR sampler as a visualization parameter, and the volume data and transfer function synthesis are performed in real time by computer algebra system. Users can use initial mathematical functions and spatial derivatives of physical variables as algebraic expressions on TFS, and flexibly design the transfer function for the multivariate data by mathematical expressions.

1.4. Operating Environment

In-Situ PBVR is a cross-platform program and can be run on various supercomputers such as Fugaku, Oakforest PACS at the University of Tokyo/Tsukuba University, and SGI8600 owned by JAEA. The table below shows the operating environment.

Table 1-1 PBVR Client

Platform	OS	Compiler
Linux	Ubuntu18	g++
Mac	OSX12	clang
Windows	Windows10	MSVC

Table 1-2 Daemon

Platform	OS	Compiler
Linux	RedHat	g++

Table 1-3 Particle Sampler

Platform	OS	Compiler
FUGAKU	A64FX (ARM)	Fujitsu compiler
OakforestPACS	Intel Xeon Phi (Knights Landing, KNL)	Intel compiler
SGI8600	Intel Xeon Gold	Intel compiler

2 Package Configuration

The source code and load module (binary) packages of the In-Situ PBVR are available from the CCSE public page (https://ccse.jaea.go.jp/software/); the PBVR client, daemon, and particle sampler are built independently and work together on the user PC, login node, and computation node, respectively.

2.1. Load Module Package

For the load module, we provide PBVR clients built for Windows, Linux, and Mac, and the daemon and the particle sampler built for a Linux server. The particle sampler and daemon are built on the supercomputer SGI8600 and Fugaku (clang-mode). The particle sampler consists of the following three libraries.

1. Particle generation library that provides particle generation functions
2. KVS library that provides visualization functions
3. Mathematical expression processing library that provides algebraic expression processing functions

In addition, the particle generation library supports three types of simulation lattice structures (structured and unstructured lattice, and hierarchical lattice). The table below shows the list of load modules.

Table 2-1 Load modules for PBVR client

Machine type	Parallelization	Package name
Linux	OpenMP	pbvr_client_linux.tar.gz
Mac	OpenMP	pbvr_client_mac.tar.gz
Windows	OpenMP	pbvr_client_win.zip

Table 2-2 Daemon

Machine type	Parallelization	Package name
SGI8600	OpenMP	pbvr_daemon_s86.tar.gz
FUGAKU	OpenMP	pbvr_daemon_fugaku.tar.gz

Table 2-3 Load modules for particle sampler

Machine type	Parallelization	Package name
SGI8600	MPI+OpenMP	pbvr_sampler_s86.tar.gz
FUGAKU	MPI+OpenMP	pbvr_sampler_fugaku.tar.gz

2.2. Source Code Package

By compiling the source code package in user's environment, particle sampler, daemon, and PBVR client are generated. A source tree of In-Situ PBVR package are shown in Table 2-4. The source code package contains particle sampler coupled test simulation code.

Table 2-4 The source tree of In-Situ PBVR

Directory/File name	Explanation
is_pbvr/	Root directory for In-Situ PBVR
|-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 structured grid
|-Hydrogen_AMR/	for AMR grid
|-Hydrogen_unstruct_tetra/	for unstructured tetrahedral grid
|-Hydrogen_unstruct_hex/	for unstructured hexahedral grid
|-Hydrogen_unstruct_prism/	for unstructured prism grid
|-Hydrogen_unstruct_pyramid/	for unstructured pyramid grid
|-Hydrogen_unstruct_quadtetra/	for unstructured quadratic tetrahedral grid
|-Hydrogen_unstruct_quadhex/	for unstructured quadratic hexahedral grid
|-Fortran/	Fortran version
|-Hydrogen_struct/	for structured grid
|-Hydrogen_AMR/	for AMR grid
|-Hydrogen_unstruct	for unstructured grid
|-FunctionParser/	Computer algebra library
|-InSituLib/	Particle generation library
|-struct/	for structured grid
|-AMR/	for AMR grid
|-unstruct/	for unstructured grid
|-KVS/	KVS library

3 Build from Source Codes

The PBVR client is generated on the user PC, while the daemon and particle sampler are generated by building the source code on the supercomputer.

3.1. Daemon and Particle Sampler

This section shows the procedure to build the daemon and particle sampler libraries from the source package. In the following procedure, we assume that the source package has already been downloaded on the supercomputer and the file download path is $HOME.

① Unzip pbvr_inSitu_1.10.tar.gz

```
$ cd $HOME
$ tar xvfz pbvr_inSitu_1.10.tar.gz
```

② Edit a config file.

The compiler used for the build is controlled by editing pbvr.conf to suit the environment. Table 3 1 gives an overview of a variable specified in pbvr.conf, and Table 3 2 lists the compilation configuration files that can be used as variable values.

Table 3-1 Variable in pbvr.conf

Variable	Input	Explanation
PBVR_MACHINE	String	Specify a compile configuration file under arch

Table 3-2 Setting files for compilation

Filename	Explanation
Makefile_machine_gcc	Setting of sequential version compilation with gcc
Makefile_machine_gcc_omp	Setting of OpenMP version compilation with gcc
Makefile_machine_gcc_mpi_omp	Setting of MPI+OpenMP version compilation with gcc
Makefile_machine_intel	Setting of sequential version compilation with intel
Makefile_machine_intel_omp	Setting of OpenMP version compilation with intel
Makefile_machine_intel_mpi_omp	Setting of MPI+OpenMP version compilation with intel
Makefile_machine_fugaku_clang	Setting of compilation on FUGAKU (clang-mode)
Makefile_machine_fugaku_trad	Setting of compilation on FUGAKU (trad-mode)
Makefile_machine_s86_mpi_omp	Setting of compilation on SGI8600 Intel compiler and mpt library.

```
$ cd $HOME/is_pbvr
$ cat pbvr.conf
PBVR_MACHINE=Makefile_machine_gcc_mpi_omp
PBVR_MAKE_CLIENT=0
```

③ The daemon and the particle sampler libralies are built by make.

```
$ cd $HOME/is_pbvr
$ make all_clean
$ make
```

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

④ This completes the building of the daemon and particle sampler, and we will continue with the building of the test code under Example.

```
$ cd $HOME/is_pbvr/Example/C/Hydrogen_struct
$ make
```

3.2. PBVR Client

The PBVR client is built with the same code as the client-server PBVR (https://ccse.jaea.go.jp/software/PBVR/). To build the PBVR client for In-Situ, edit qtpbvr.conf as follows to enable PBVR_MODE=IS.

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

For details on how to build a PBVR client, please refer to the manual for the client-server PBVR.

4 Setting up of In-SituVisualization

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 visualizes the batch processed simulation. To accomplish this, some simple configuration and port forwarding connections are required.

4.1. Setting Environment Variables

The daemon and particle sampler use the environment variables shown in Table 41 below, and at runtime you need to set the environment variables with the export command.

Table 4-1 Environment variables referenced by daemon and 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 this, the daemon and particle sampler will search the current directory in which each is running.

※2 If not specified this, daemon and 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 such as a transfer function, a range of physical values to be visualized, and a screen resolution are required. Each item of the visualization parameters are 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 at first startup.

In order to execute In-Situ PBVR, the user needs to place the transfer function file in the directory specified by VIS_PARAM_DIR in Table 4-1 with the file name specified by TF_NAME.

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 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. In order to perform socket communication between the interactive node on the remote supercomputer and the local PC, the user needs to connect both ports by ssh port forwarding.

4.3.1. Remote Connection between Tow Machines

The following example shows how to connect local machineA to 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. hostnameB is often displayed on the terminal of machine B, and can also be confirmed by the hostname command. Also, 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, but localhost can be used.

4.3.2. Remote Connection with Sseveral 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.

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 with 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, when waiting for the socket communication connection with the client, start the PBVR client from another terminal. The default port number at daemon startup is 60000. The port number can be changed with 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. Then, daemon sends and receives data with the PBVR client through the port specified at startup.

5 Particle Sampler

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

The particle sampler is executed with batch processing of the simulation, and converts the calculated volume data to the particle data 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 which has the maximum and minimum coordinates of the area. The particle data file consists of a header file (kvsml), a coordinate file (coord.dat), a color file (color.dat), and a normal file (normal.dat), output one set per node at each time step. At the same time, the particle sampler outputs $TF_NAME_timestep.tf file recording history of transfer function change, history_timestep.txt file recording histogram and the range of physical values, state.txt recording 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_patameters dom, Typs** volume_data, int num_volume_data );
```

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

- int time_step：simulation time step
- domain_patameters dom：data structure defines 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 ycoordinates 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 user-specified physical value, and multi-variable volume data is defined as a two-dimensional array.

In 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 takes as arguments the time step of the simulation and information on the calculation area, volume data of the simulation result, and lattice information of the unstructured lattice.

- int time_step：simulation time step
- domain_patameters dom：data structure defines 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 ycoordinates in whole domain
    float z_global_max;    // Maximum of z coordinates in whole domain
} domain_parameters;
```

- Typs ** volume_data: A pointer to an array of volume data of simulation results. “Type” is a type of user-specified physical value, and multi-variable volume data is defined as a two-dimensional array. The value on the cell vertex of the nth variable is referred to by 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], coordinates [3 * i + 2]).
- int ncoords：number of coordinates
- unsigned int * connections: A pointer to the connection list of vertex IDs that make up a hexahedral element. The configuration of the hexahedral element is shown in Figure 5-1. The n-th vertex of the i-th hexahedral element is referred to by connections [8 * i + n].
- int ncells：number of elements.
- pbvr::VolumeObjectBase::CellType& celltype : element type. IS-PBVR supports tetrahedral, hexahedral, pyramid, prism, quadratic tetrahedral, and quadratic hexahedral elements. Each type is defined in VolumeObjectBase class written in following file.
  /DaemonAndSampler/InSituLib/unstruct/TFS/VolumeObjectBase.h

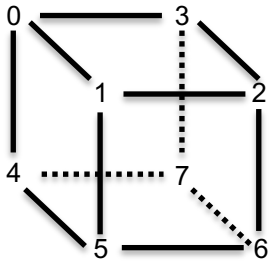


Figure 5-1 Connection of vetices of a hexahedral element

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 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, Block Structured AMR is defined as a four-dimensional grid of NxNxNxL (L is the number of leaves). Figure 5-2 shows a two-dimensional example.

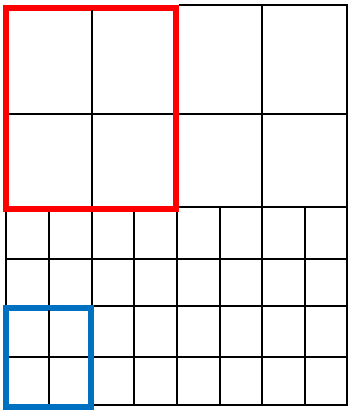


Figure 5-2 Example of a two-dimensional hierarchical grid. The upper part is the lattice of the hierarchy Lv. 1 and the lower part is the hierarchy Lv. 2. The leaf is defined by a 2 × 2 orthogonal grid, red is a leaf of Lv.1, blue is a leaf of Lv.2.

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

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

- int time_step：simulation time step
- domain_patameters dom：data structure defines 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 ycoordinates 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 by 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 referenced by (leaf_min_coord [3 * l], leaf_min_coord [3 * l + 1], leaf_min_coord [3 * l + 2]).
- float** values： Pointer to the array of resulting volume data. Multivariate volume data is defined as a two-dimensional array. In the volume data of the grid resolution (X, Y, Z, L), the value of the position (i, j, k, l) of the nth 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 particle sampler to test simulation code as an example. The test simulation code calculates the value of charge density on the lattice vertex at each time step according to the charge density equation of hydrogen. Therefore, the class name to be calculated is Hydrogen. The code without 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, Hydrogen class outputs volume data of each time step in hydro.values in for 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 MPI is used to calculate area of subdomain of Hydrogen. The domain information, resolution, MPI rank, and volume data becomes arguments of generate_particles function. The generate_particles function is called after 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 default mode, PBVR client draws particle data of the latest time step received from daemon. Viewer of PBVR client draws particle data using OpenGL. The PBVR client provides a GUI to edit visualization parameters, including transfer functions, and sends the visualization parameters to the daemon. Sending and receiving data between the daemon and the PBVR client is performed through an arbitrary port number by socket communication.

6.1. Launching PBVR Client

The following examples show how to launch PBVR client.

$ pbvr_client [command line options]

表 6-1 Command line options for PBVR client

option	value	default	explanation
-p	ポート番号	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 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.

6.2. Terminating of PBVR Client

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 ends. However, if communication is interrupted with the Stop button on the time step control panel, the Ctrl + c key input is ignored. Also note 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 daemon are not terminated by pressing the Ctrl + c key for some reason, forced termination is required using kill command with the process numbers of the client and daemon by ps command as shown below.

【Forced termination of PBVR client】

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

【Forced termination of daemon】

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

6.3. Using PBVR Client GUI

The PBVR client shows the visualization result and provides the GUI that can interactively control the visualization parameters.

6.3.1. Viewer

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

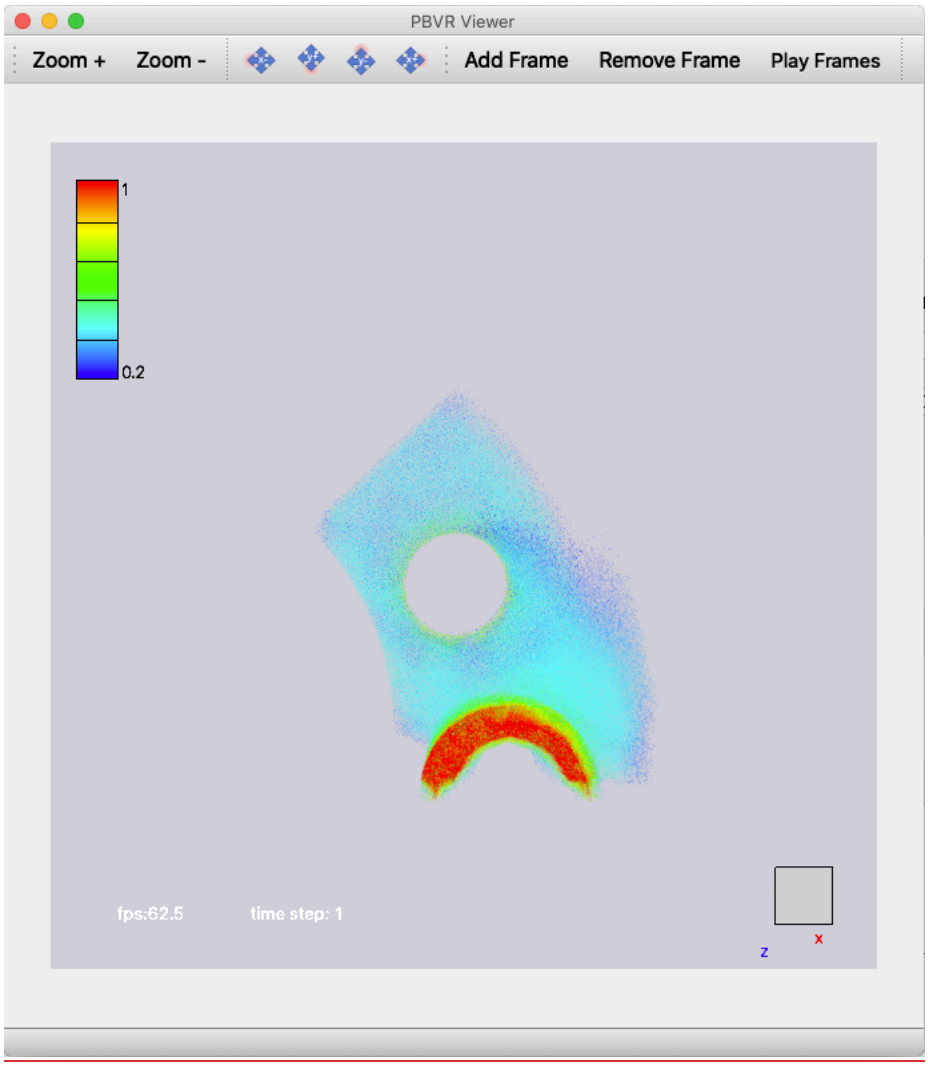


Figure6-1 Viewer

[Operations]

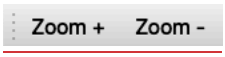
Rotation:	Dragging left with the mouse
Translation:	Dragging right with the mouse
Zoom:	Shift + left-dragging, or dragging while pressing the mouse wheel
Reset:	Home button (fn + left arrow on Macs)

[Display]

time step: Time step for the data displayed

fps: Frame rate (frame/sec)

[Tool bar]



Zoom in/out.


Translations.

Get keyframe/delete keyframe/play keyframe animation.

6.3.2. Tool Bar

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

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

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

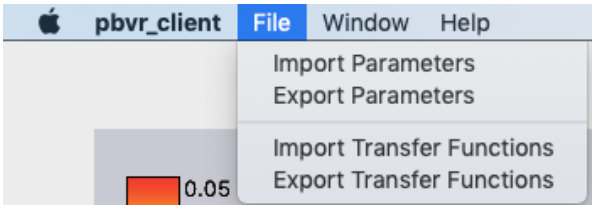


Figure 6-2 File tab

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

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

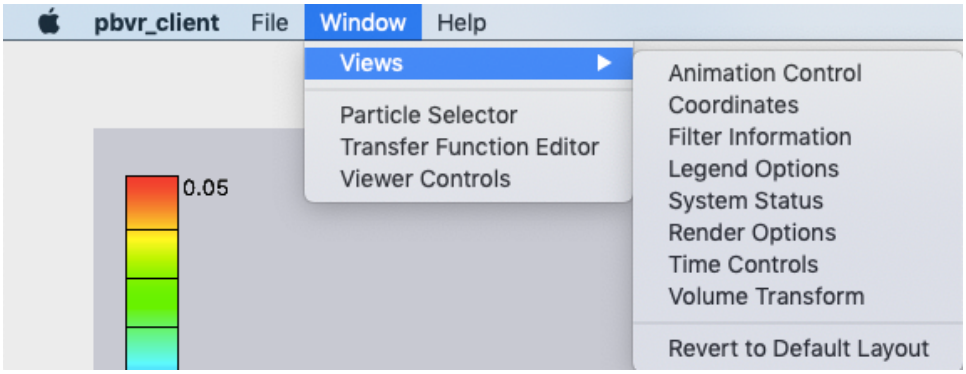


Figure 6-3 Window tab

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

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

6.3.2.1 FilterInfo

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

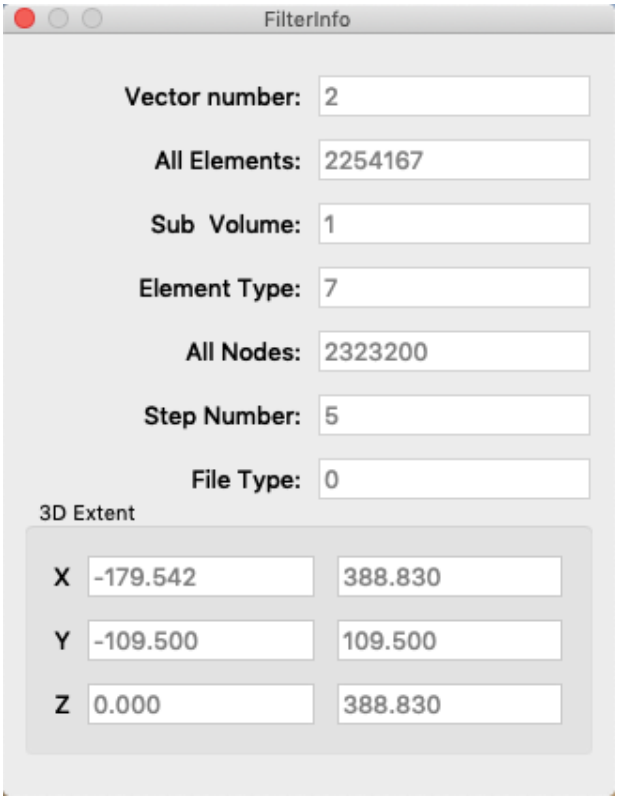


Figure 6-4 FilterInfo

- **Vector number** displays the number of variables consisting of the volume data.
- **All Elements** displays the number of elements consisting of the volume data.
- **Sub Volume** displays the number of sub-volume separated by the filter program.
- **Element Type** displays the type of element.
- **All Nodes** displays the number of nodes of the volume data.
- **Step Number** displays the number of time steps.
- **File Type** displays the volume decomposition type of a filter program used in client-sever type PBVR. It is unused in this in-situ PBVR.
- **3D Extent** displays the min-max X-Y-Z coordinates of the volume data.

6.3.2.2 System Status

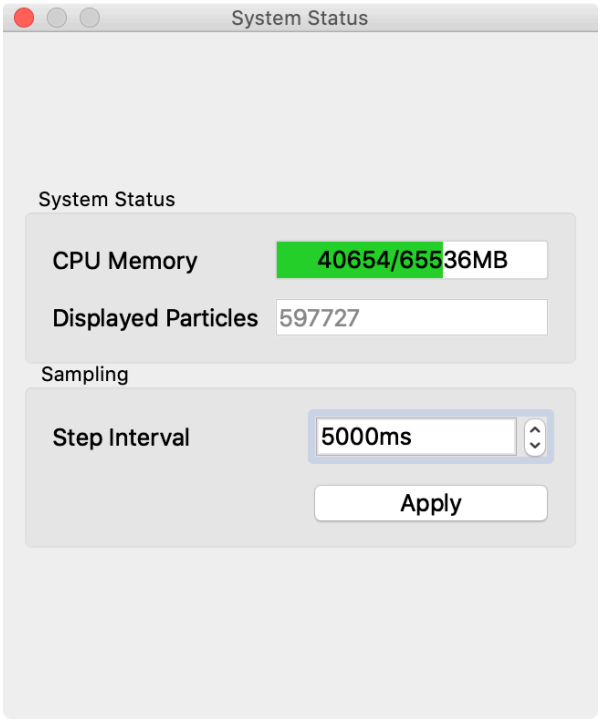


Figure 6-5 System status

- **CPU Memory** displays the system memory usage in megabytes.
- **Displayed Particles** displays the number of rendering particles.
- **Step Interval** specifies minimum of a time step update interval by [msec]. This is used to lengthen the update interval for time steps that are too short.

6.3.2.3 **Render Options**

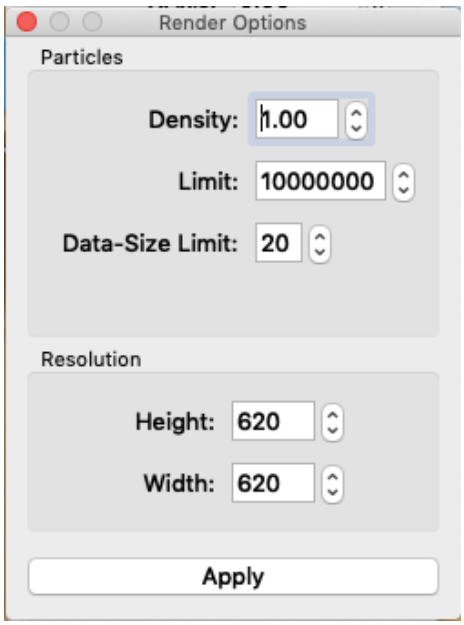


Figure 6-6 Render Options

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

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

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

Resolution specifies the viewer's resolution.

6.3.2.4 **Volume Transform**

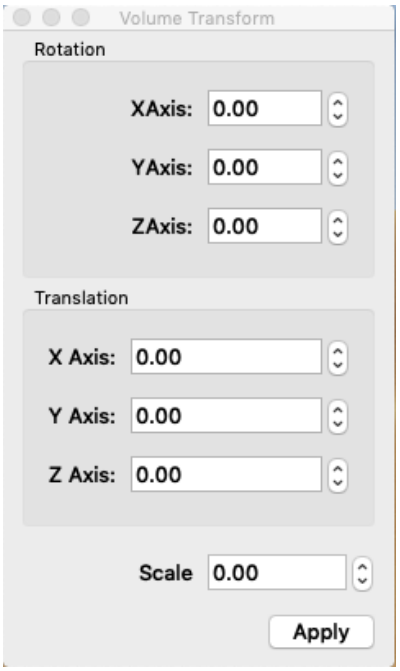


Figure 6-7 Volume Transform

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

6.3.3. Transfer Function Editor

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

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

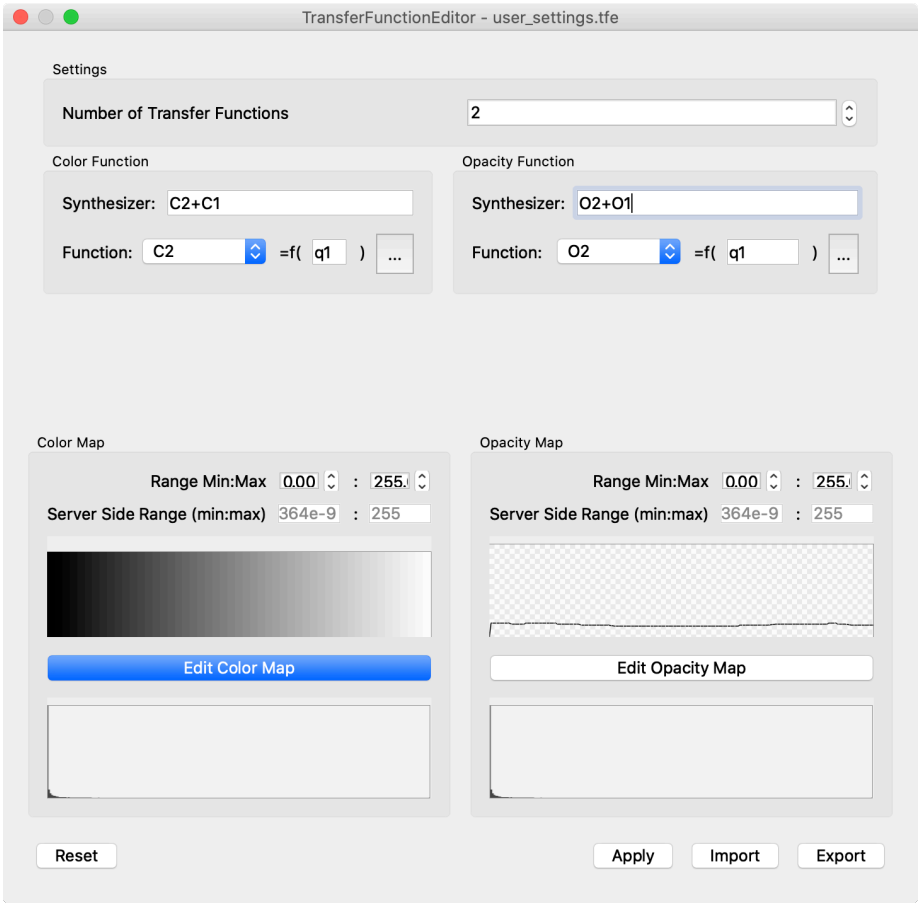


Figure 6-8 Transfer Function Editor

[Operations]

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

- **Number Transfer Functions** specifies the maximum number of transfer functions that can be created.
- **Color Function** category specifies the color functions and its argument which is synthesized by the physical values.
- **Opacity Function** category specifies the opacity functions and its argument which is synthesized by the physical values.
- **Color Map** category edits the color functions.
- **Opacity Map** category edits the opacity functions.
- **Reset** button returns the transfer functions to initial state.
- **Apply** button sends the transfer function to the server program.
- **Export** button saves the transfer function file with same format of "-pa" option.
- **Import** button reads the transfer function file.

[Available operatos for algebraic equations]
Binary operators +, -, *, /, ^
Functiobns sqrt(),cbrt(), log(), log10(), exp(), abs(), floor(), ceil(), sin(), cos(), tan, asin(), sinh(), cosh(), tanh(), asinh(), acosh(), atanh(), heavi(), rectfunc(),
Binary functions sigmoid(*,*), gauss(*,*)

6.3.3.1 Color Map Specification

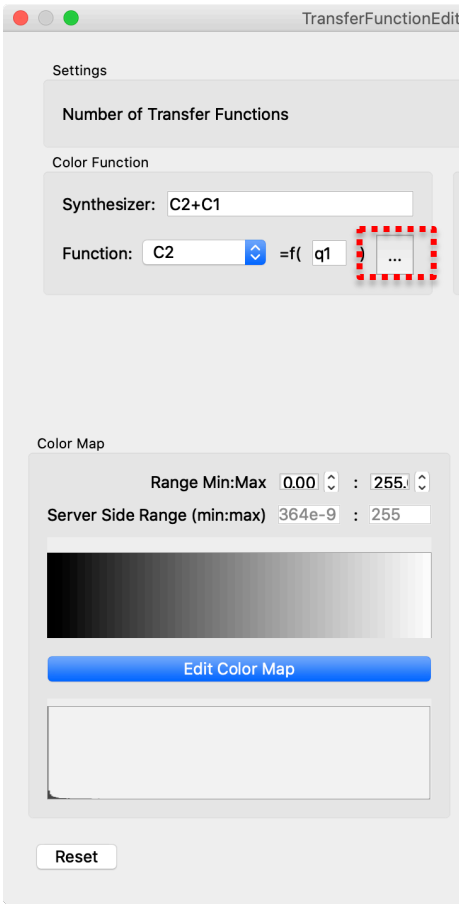


Figure 6-9 Top: Color Function category, bottom: Color Map category

[Color Function category]

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

[Color Map category]

- **User Defined Range Min:Max** specifies the minimum and maximum range of the color function which is assigned to the synthesized physical value.
- **Synth. Func. Range Min:Max** displays the minimum and maximum values of the synthesized physical value.
- **Edit Color Map** button opens Color Map Editor.
- **Histogram** displays a distribution of the synthesized physical value with the user defined range min:max.

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

6.3.3.1.1. Color Function Editor

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

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

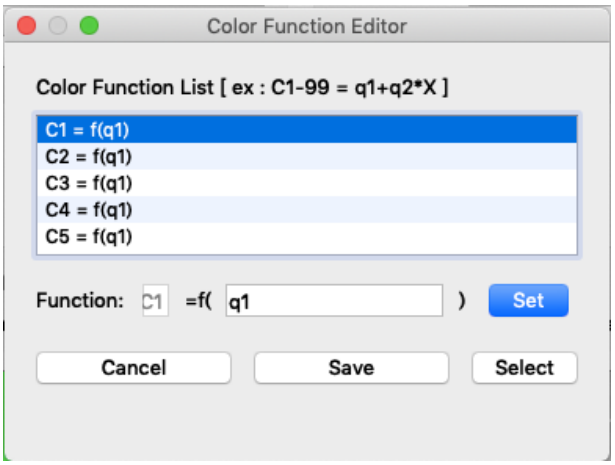


Figure 6-10 Color Function Editor

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

6.3.3.1.2. Color Function Editor: Freeform Curve Edit Tab

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

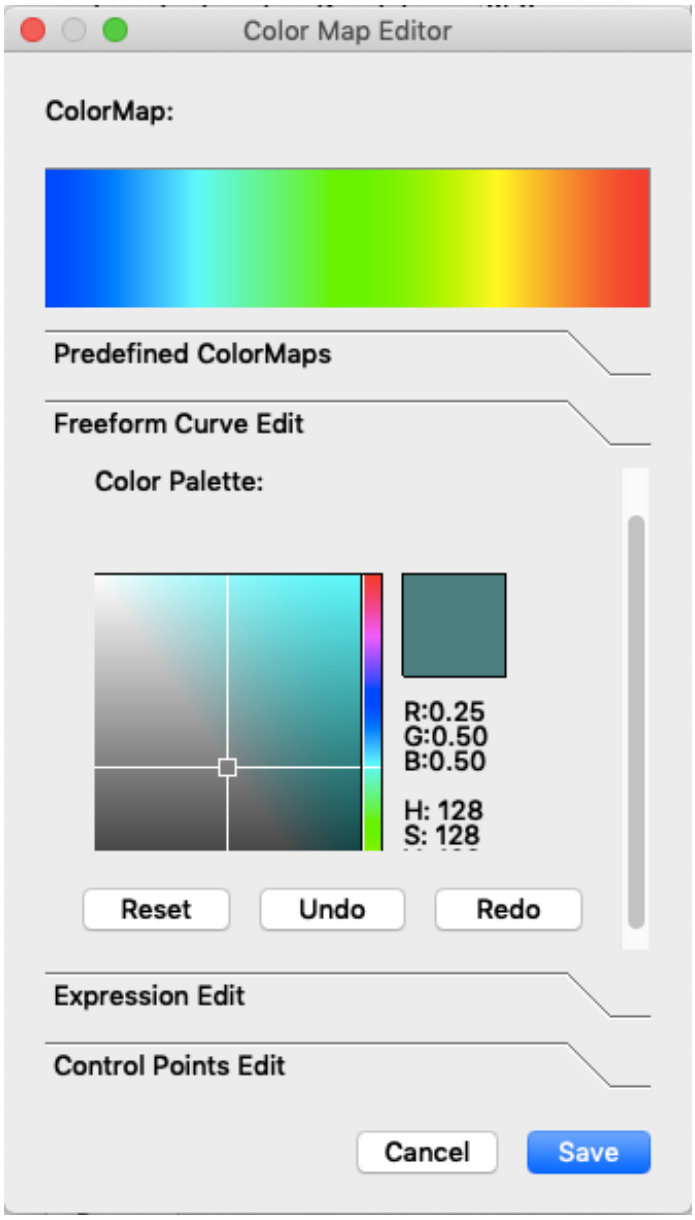


Figure 6-11 Color Map Editor: freeform curve tab

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

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

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

6.3.3.1.3. Color Map Editor: Expression Tab

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

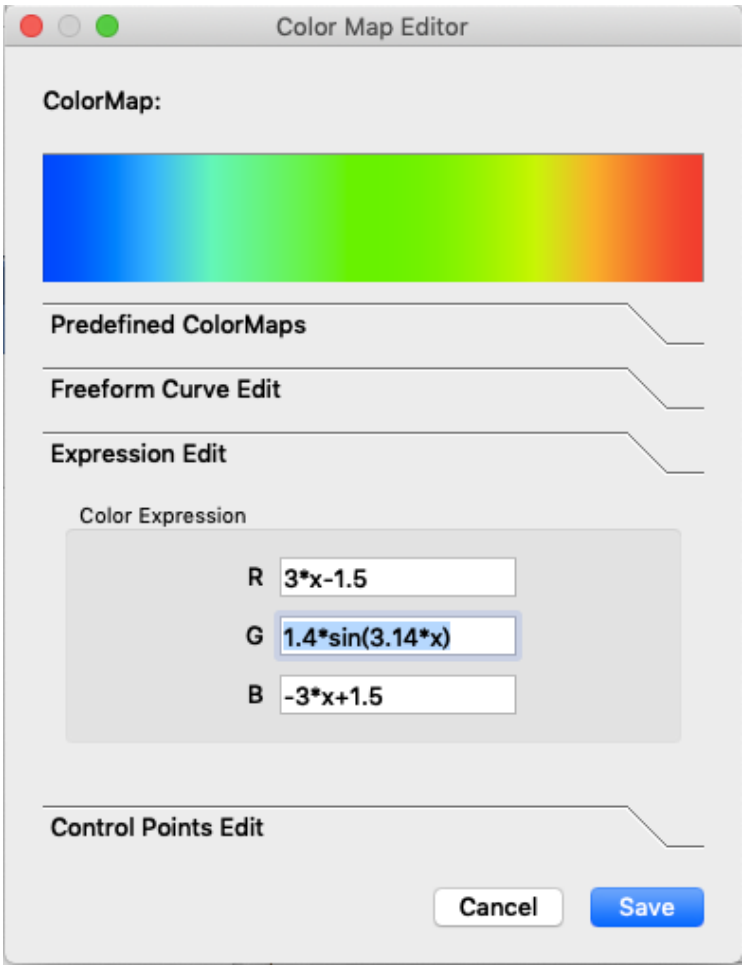


Figure 6-12 Color Map Editor: Expression tab

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

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

6.3.3.1.4. Color Map Editor: Control Points Edit Tab

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

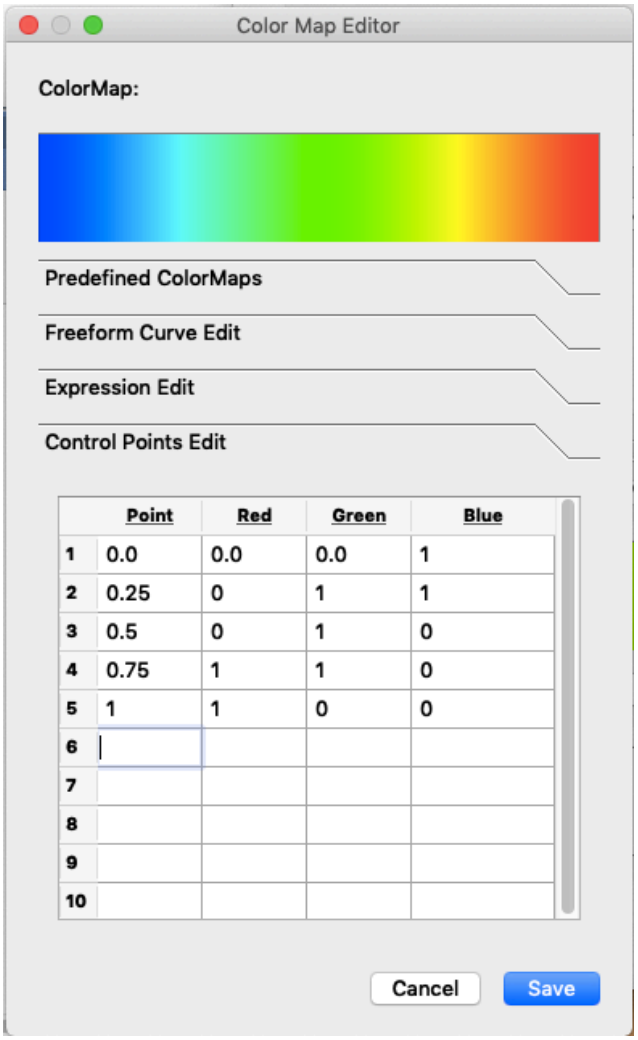


Figure 6-13 Color Map Editor: Control Points Edit Tab

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

Each control point is interpolated with a piecewise linear function.

6.3.3.1.5. Color Map Editor: Predefined ColorMaps Tab

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

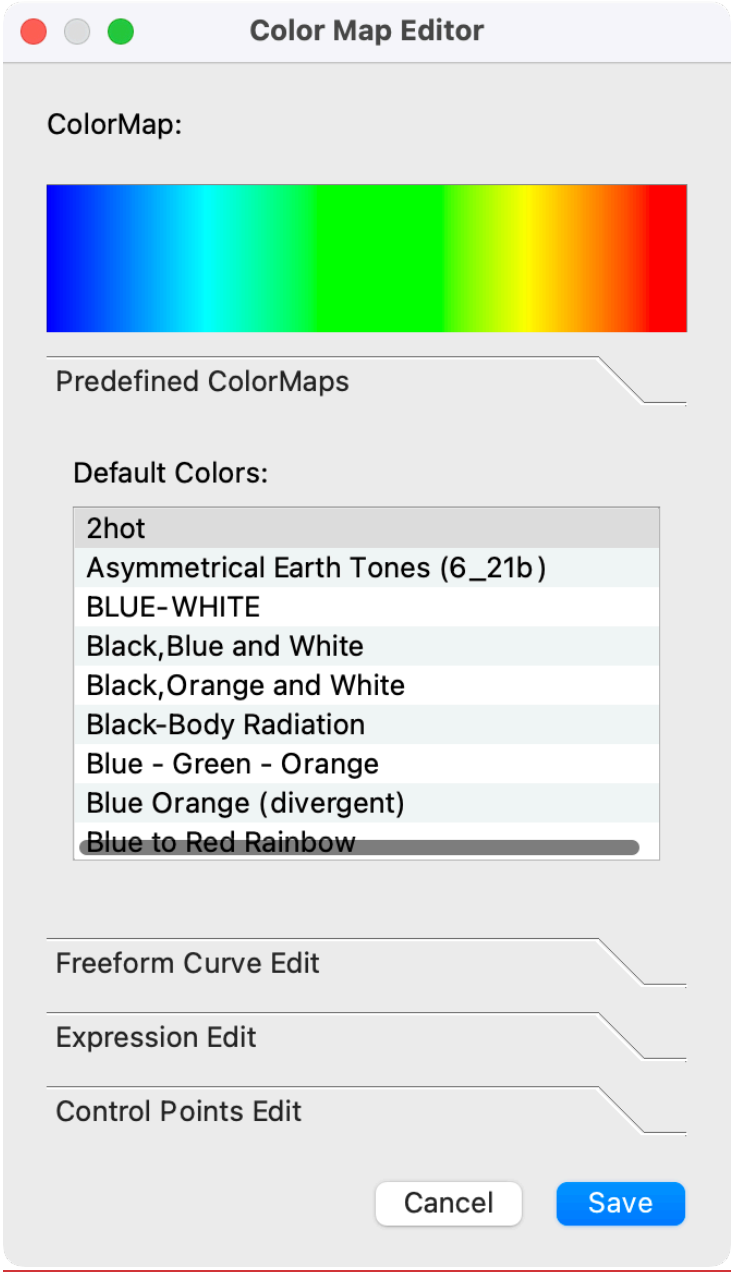


Figure 6-14 Color Map Editor: Predefined ColorMaps Tab

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

- Rainbow
- Blue-white-red
- Black-red-yellow-white
- Black-blue-violet--yellow-white
- Black-yellow-white

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

6.3.3.2 **Opacity Map Specification**

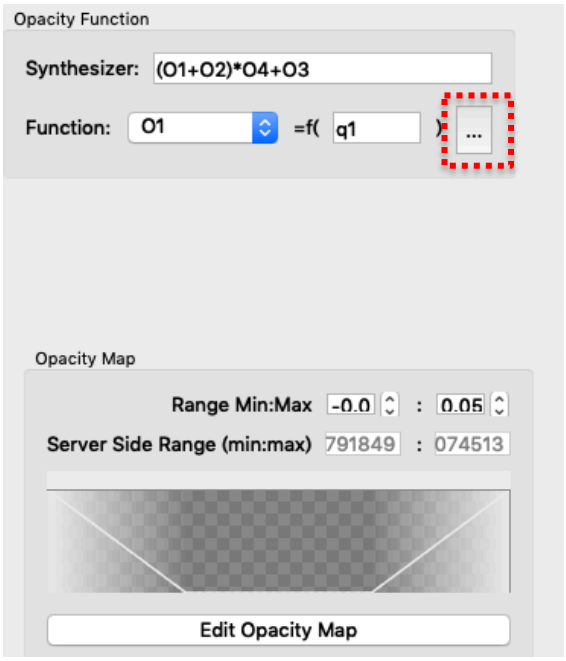


Figure 6-15 Top: Opacity Function category, bottom: Opacity Map category

[Opacity Function category]

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

[Opacity Map category]

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

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

6.3.3.2.1. Opacity Function Editor

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

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

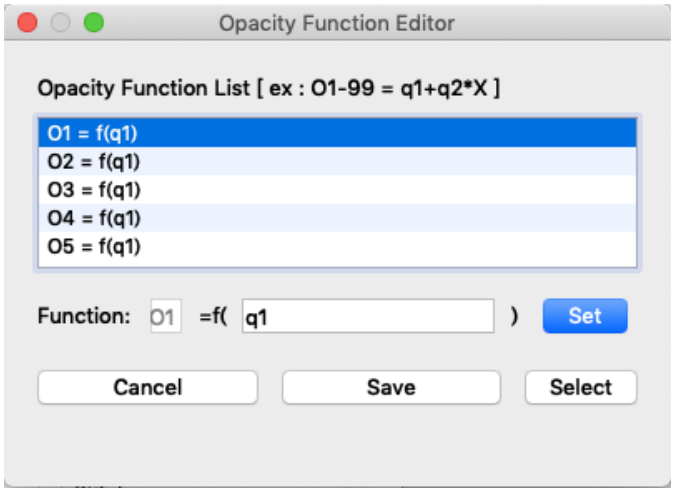


Figure 6-16 Opacity Function Editor

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

6.3.3.2.2. Opacity Map Editor: Freeform Curve Tab

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

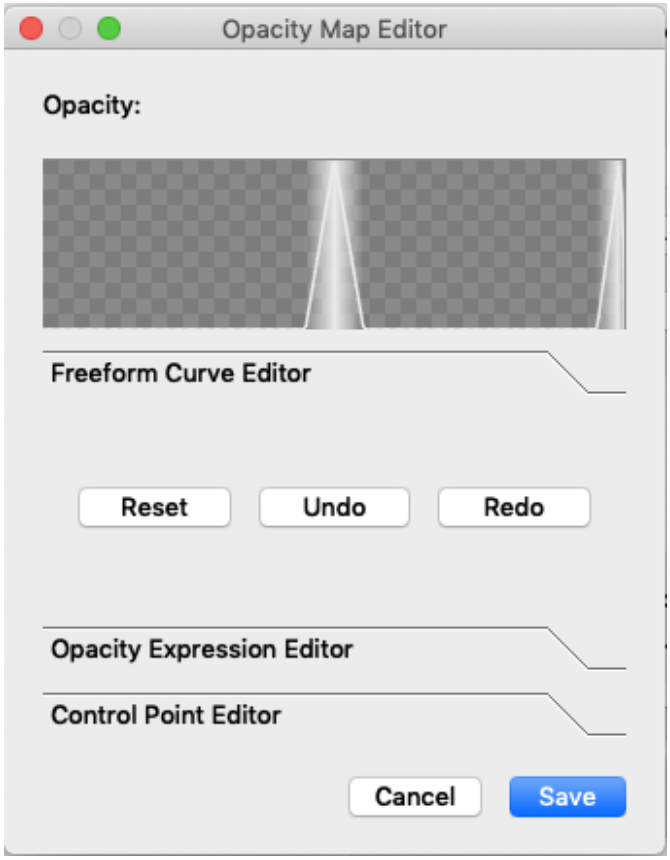


Figure 6-17 Opacity Map Editor: Freeform Curve Tab

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

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

6.3.3.2.3. Opacity Map Editor: Opacity Expression Tab

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

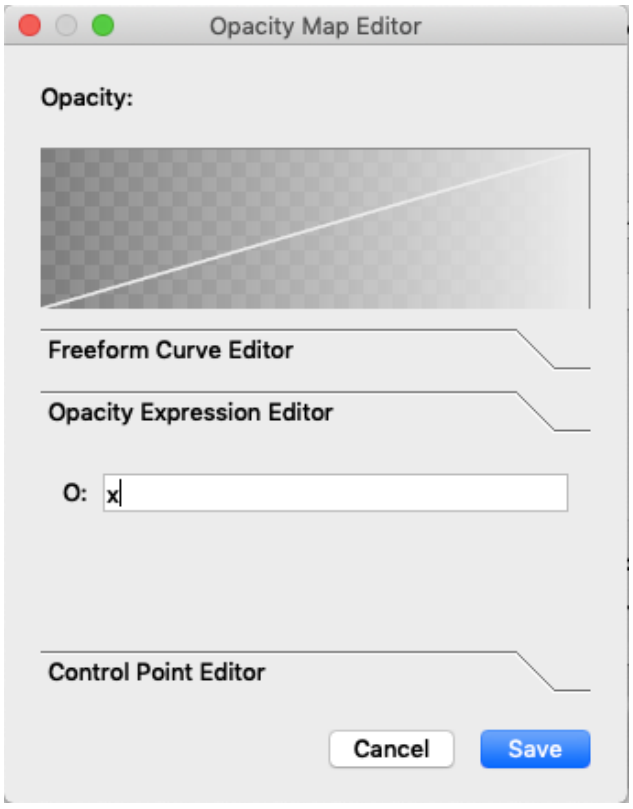


Figure 6-18 Opacity Map Editor: Opacity Expression Tab

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

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

6.3.3.2.4. Opacity Map Editor: Control Point Editor Tab

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

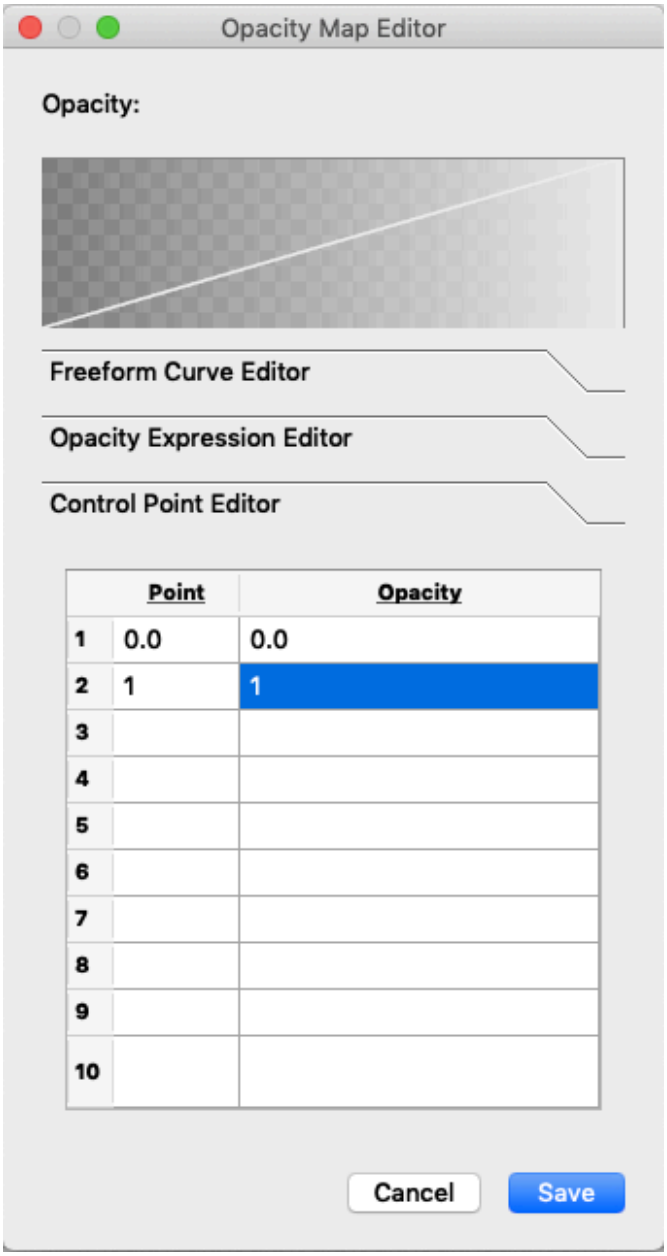


Figure 6-19 Opacity Map Editor: Control Point Editor Tab

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

Each control point is interpolated with a piecewise linear function.

6.3.3.3 **Function Editor**

Table 6-1 lists the built-in math operations available in the function editors. They can be used for the overall transfer functions, the parameters, and for the color and opacity curves.

Table 6-1 Math operations available in function editors

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

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

6.3.4. Time panel

Figure 6-20 shows **Time panel**, which specifies the time steps for the visualization. Each widget works as described below.

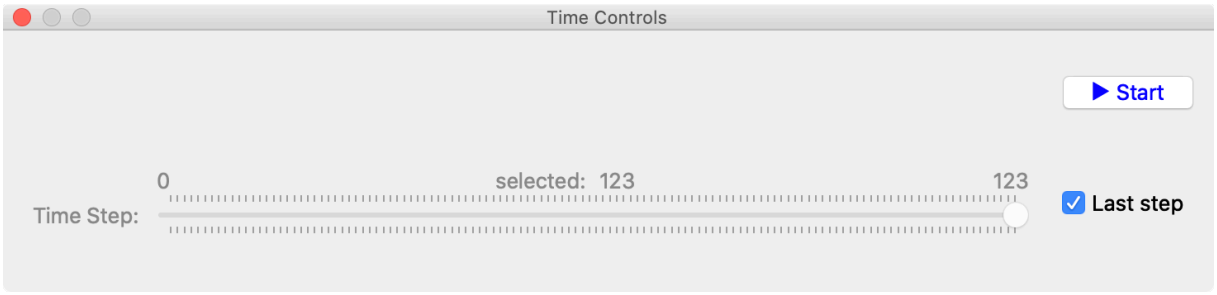


Figure 6-20 Time panel

- **Time step** specifies the time step to be rendered.
- If **Last step check box** is checked, the viewer displays the latest time step.
- **Start/Stop** starts/stops communications between PBVR Client and PBVR Server.

6.3.5. Particle and Polygon composition

CS-PBVR supports composition of the volume rendering and polygon rendering. Figure 6-21 shows **Particle panel**, which is used to integrate multiple particle datasets. Each widget works as described below.

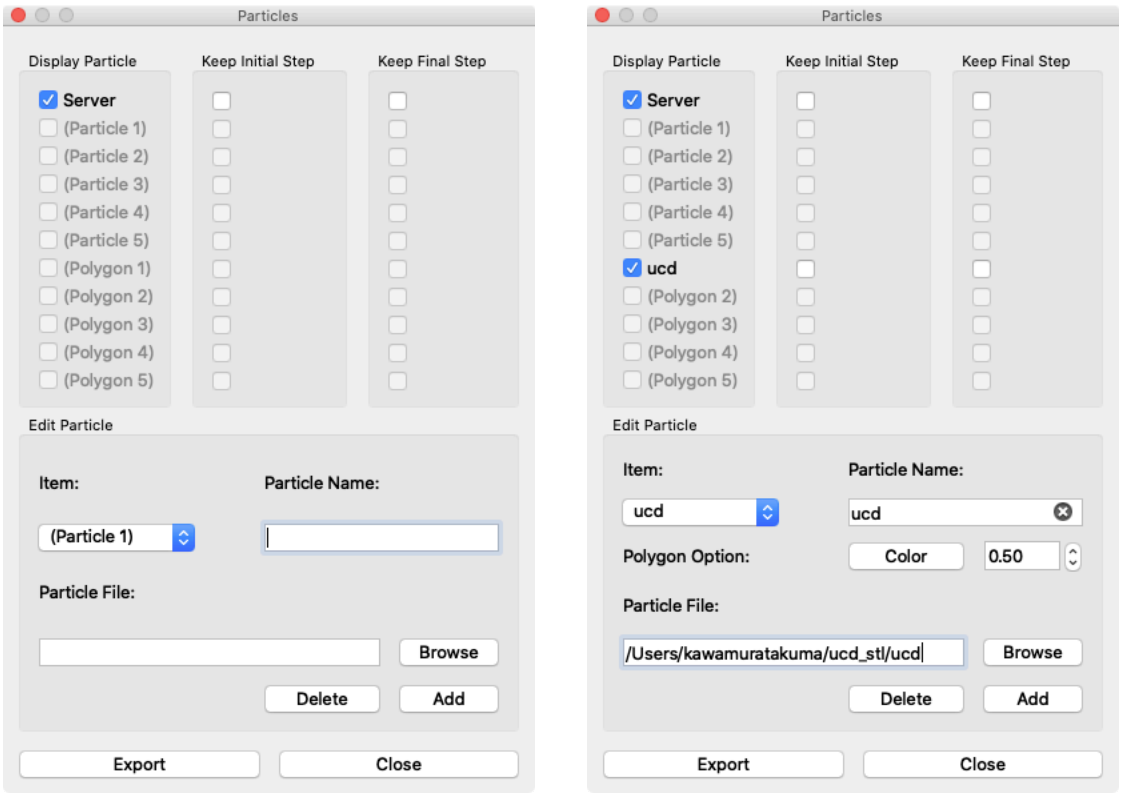


Figure 6-21 Particle panel. Left: particle selection mode, right: polygon selection mode.

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

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

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

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

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

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

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

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

[Edit Particle category]

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

6.3.5.1 Example of volume and polygon composition

The following is an example of the composition of the test program Hydrogen which is included in Example directory and the polygon hydrogen.stl of its boundary shape. After portforwarding client and server, the test program Hydrogen and the daemon is launched. Next, the volume rendering on the client is executed (Figure 6-22 left). Next, in the Edit Particle category of the Particle Integration panel, select

Polygon from the Item spin box, and specify the path of hydrogen.stl in Particle File. Next, specify the data name "hydrogen.stl" in Particle Name, and set the color and opacity. Then, check hydrogen.stl in the Display Particle category (right side of Figure 6-22), and the boundary shape will be synthesized (center of Figure 6-22).

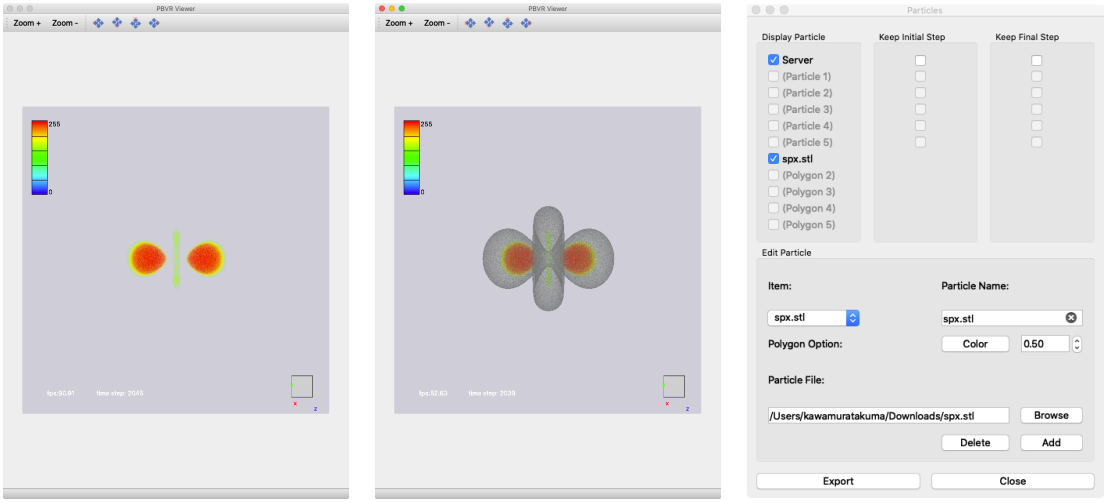


Figure 6-22 Left: volume rendering result of Hydrogen; Center: composite view of Hydrogen volume and its boundary shape; Right: settings of the Particle Integration panel in the composite view (Replace spx with hydrogen in the above figure.).

6.3.5.2 **Example of multi-time steps composition**

The behavior of the particle integration is explained using Figure 6-23 (sever side: 1~4 time steps, client side:0~3). If the server checkbox and the particle checkboxes are checked, all time steps are displayed as shown in Table 6-2. If the checkbox **Keep Initial Step** is checked for the client particles, only the first time step is displayed, as shown in Table 6-3. If **Keep Final Step** is checked for the client particles, only the final time step is displayed, as shown in Table 6-4.

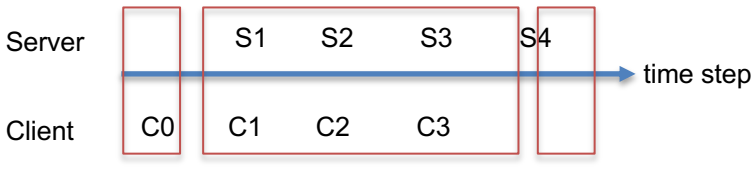


Figure 6-23 The behavior of the particle integration

Table 6-2 Particle datasets displayed by default.

	Step0	Step1	Step2	Step3	Step4

Server	-	S1	S2	S3	S4
Client	C0	C1	C2	C3	-

Table 6-3 Particle datasets displayed with Keep Initial Step for the server particles.

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

Table 6-4 Particle datasets displayed with Keep Final Step for the client particles.

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

6.3.6. Image file production

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

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

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

Figure 6-24 shows the **Animation Control Panel**. Each widget works as described below.

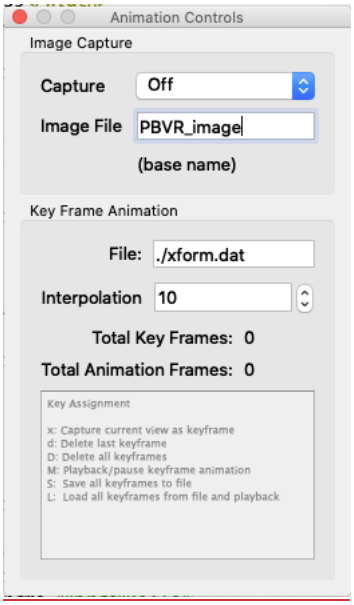


Figure 6-24 Animation Controls Panel

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

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

6.3.6.1 Image production

Image files are produced as follows.

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

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

PBVR_image.00001.bmp
PBVR_image.00002.bmp
…

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

PBVR_image_k.00001.bmp
PBVR_image_k.00002.bmp
…

6.3.6.2 Key frame animation of a still image

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

[Capture key frames and save them in a file]

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

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

[Play a key frame file]

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

Table6-5 Keys used for controlling key frame animation

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

6.3.6.3 **Key frame animation of time series data**

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

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

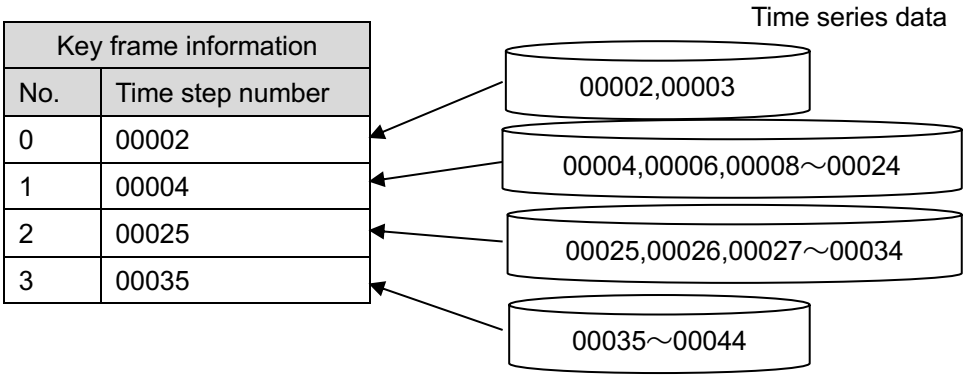


Figure6-25 Key frame animation for time series data

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

6.3.6.4 **Key frame file format**

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

File format

Key frame data 1
Key frame data 2
:

Type	Size (bytes)	data
int	4	Time step number
float	4	rotation[0].x
float	4	rotation[0].y
float	4	rotation[0].z
float	4	rotation[1].x
float	4	rotation[1].y
float	4	rotation[1].z
float	4	rotation[2].x
float	4	rotation[2].y
float	4	rotation[2].z
float	4	translation.x
float	4	translation.y
float	4	translation.z
float	4	scaling.x
float	4	scaling.y
float	4	scaling.z

Figure6-26 Key frame file format

6.3.7. Legend panel

Figure 6-27 shows the **Legend panel**. A legend is a bar showing how the values of a physical quantity are rendered as a color in the visualization. Each widget works as described below.

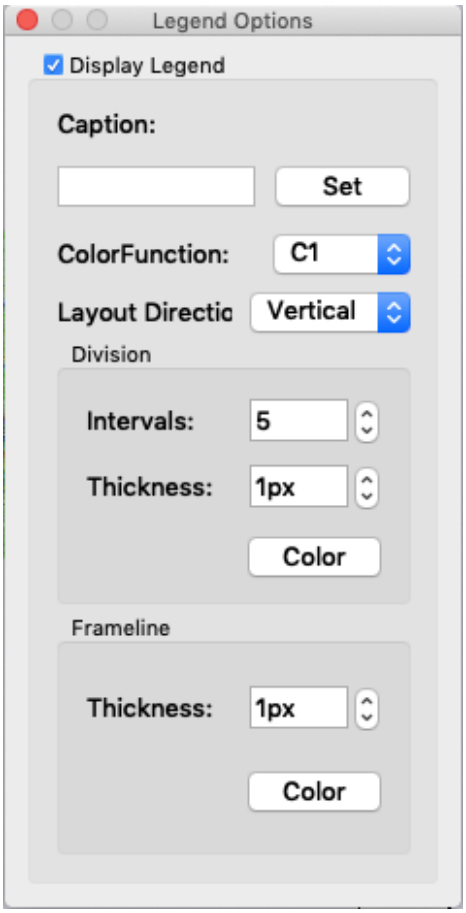


Figure 6-27 Legend panel

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

Figure 6-28 shows an example of legend.

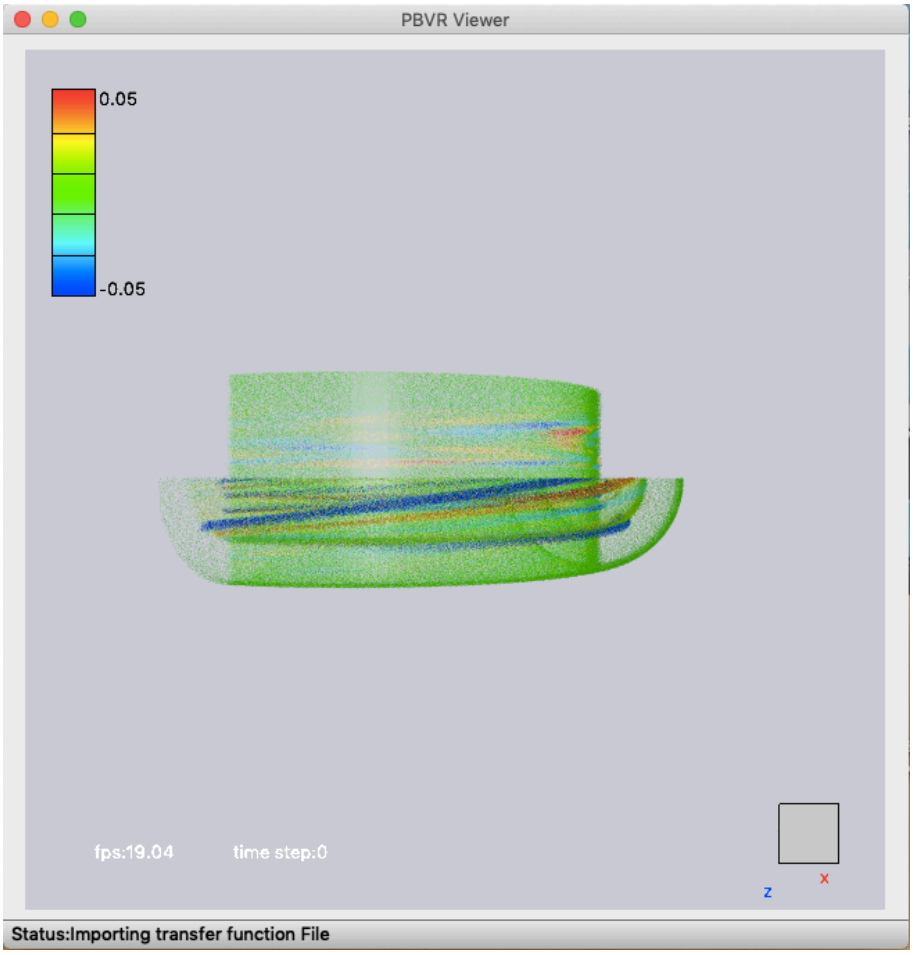


Figure 6-28 Example of a legend

6.3.8. Viewer Control Panel

Figure 6-29 shows the **Viewer Control Panel**, which specifies the properties of the viewer. Each widget works as described below.

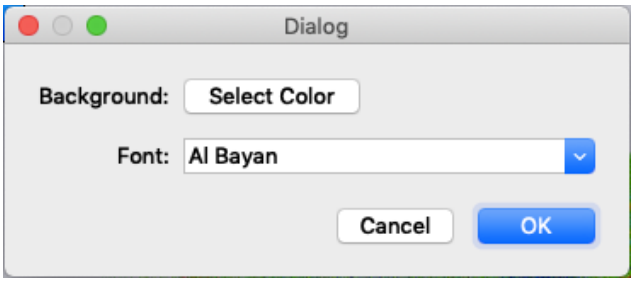


Figure 6-29 Viewer control panel

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

7 How to Execute Examples

This section shows an example of port-forward connection from Linux/Mac to FUGAKU using Examples included in the source code package to execute In-Situ PBVR. In this section, we will show you how to use a login node that are allowed to execute programs and how to use interactive jobs. This section also shows an example of accessing a remote server from Windows and executing In-Situ PBVR.

7.1. Login node

The following is the procedure for running the test program for the unstructured lattice (is_pbvr/Example/Hydrogen_unstruct) on the FUGAKU by using the login node that are allowed to execute programs. In this example, we assume that the RSA public key and the ssh host name and user name have been configured, the source package has been placed in /home/ on FUGAKU, and the build is complete. In this procedure, the test program combined with In-Situ PBVR is submitted to job, and the daemon is launched on the login node and the PBVR client is launched on the user PC.

Start up the first terminal, log in to FUGAKU, and submit the test program to the job using a script "run.sh".

[Terminal 1]

```
[userPC]$ ssh UseerID@login.fugaku.r-ccs.riken.jp
[fugaku]$ cd /home/is_pbvr/Example/Hydrogen_unstruct
[fugaku]$ mkdir particle_out
[fugaku]$ cat run.sh
#!/bin/sh
#PJM -L "node=1"
#PJM -L "rscunit=rscunit_ft01"
#PJM -L "rscgrp=small"
#PJM -L "elapse=0:60"
#PJM --mpi max-proc-per-node=4,proc=4
#PJM -s
export OMP_NUM_THREADS=12
export TF_NAME=default
export VIS_PARAM_DIR=$PJM_JOBDIR
export PARTICLE_DIR=$PJM_JOBDIR/particle_out
mpiexec -n 4 ./run qsub run.sh
[fugaku]$ pjsub run.sh
```

In the second terminal, make a port-forward connection between FUGAKU and the user PC and start the daemon.

[Terminal 2]

```
[userPC]$ ssh -L 61000:localhost:61000 UserID@loginX.fugaku.r-ccs.riken.jp
[fugaku]$ source /opt/intel/bin/compilervars.sh intel64
[fugaku]$ cd /home/is_pbvr/Daemon
[fugaku]$ export VIS_PARAM_DIR=/home/is_pbvr/Example/Hydrogen_unstruct
[fugaku]$ export PARTICLE_DIR=$VIS_PARAM_DIR/particle_out
[fugaku]$ ./pbvr_daemon -p 61000
```

X は 1~6

Activation of Intel compiler

In the third terminal, launche the PBVR client on the user PC.

[Terminal 3]

```
[usrPC]$ ./pbvr_client -p 61000
```

7.2. Interactive Job

Start up the first terminal, log in to Fugaku, and submit the test program to the job using a script "run.sh".

[Terminal 1]

```
[userPC]$ ssh FUGAKU
[fugaku]$ cd /home/is_pbvr/Example/Hydrogen_unstruct
[fugaku]$ mkdir particle_out
[fugaku]$ cat run.sh
#!/bin/sh
#PJM -L "node=1"
#PJM -L "rscunit=rscunit_ft01"
#PJM -L "rscgrp=small"
#PJM -L "elapse=0:60"
#PJM --mpi max-proc-per-node=4,proc=4
#PJM -s
export OMP_NUM_THREADS=12
export TF_NAME=default
export VIS_PARAM_DIR=$PJM_JOBDIR
export PARTICLE_DIR=$PJM_JOBDIR/particle_out
mpiexec -n 4 ./run qsub run.sh
```

Since Fugaku is not allowed to execute programs on the login node, the computation node and the user PC are connected port-forward and the daemon is executed in an interactive job.

Start the interactive job on terminal 2 and get the IP address of the computation node running the interactive job. “ip” command displays the network information, and a IP address after “inet” is the computation node’s IP.

[Terminal2]

```
[userPC]$ ssh FUGAKU
[fugaku]$ cd /home/is_pbvr/Example/Hydrogen_unstruct
[fugaku]$ ln -s /home/is_pbvr/Daemon/pbvr_daemon
[fugaku]$ pjsub --interact -L "node=1" -L "rscgrp=int" ¥
[fugaku]$ -L "elapse=0:10:00" --sparam "wait-time=600"
[intrct]$ ip address show tofu1
.....
    inet **.**.**.**/12 brd 10.255.255.255 scope global noprefixroute tofu1
.....
```

Using the IP address of the computation node, port forward connection is made between the user PC and the computation node at terminal 3.

[Terminal3]

```
[userPC]$ ssh -L 60000: **.**.**.**:60000 FUGAKU
```

Run the daemon at terminal 2.

[Terminal2]

```
[intrct]$ export TF_NAME=default
[intrct]$ export VIS_PARAM_DIR=$PJM_JOBDIR
[intrct]$ export PARTICLE_DIR=$PJM_JOBDIR/particle_out
[intrct]$ ./pbvr_daemon
```

Launch PBVR client on the user PC in terminal 4.

```
[userPC]$ ./pbvr_client -p 60000
```

7.3. **Windows**

The procedure for connecting remotely from a Windows client to a remote server is shown below. The following procedure assumes transfer from the port 60000 of the client to the port 60000 of the server. 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.

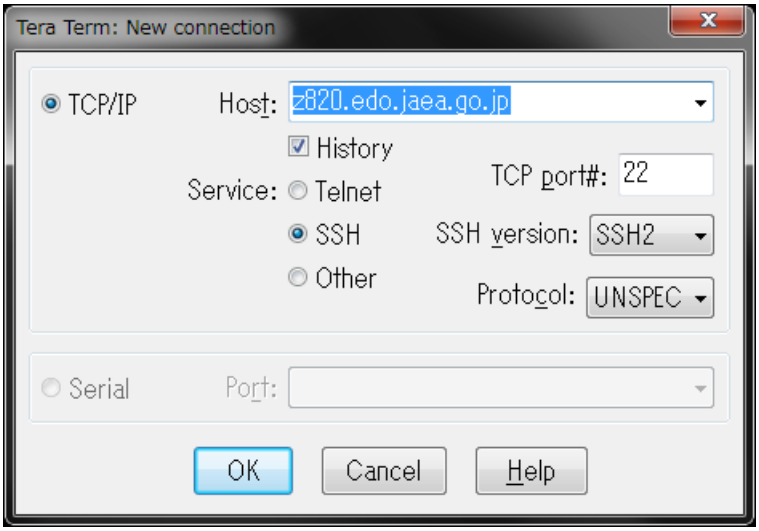


Figure 7-1 Tera Term dialog 1)

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

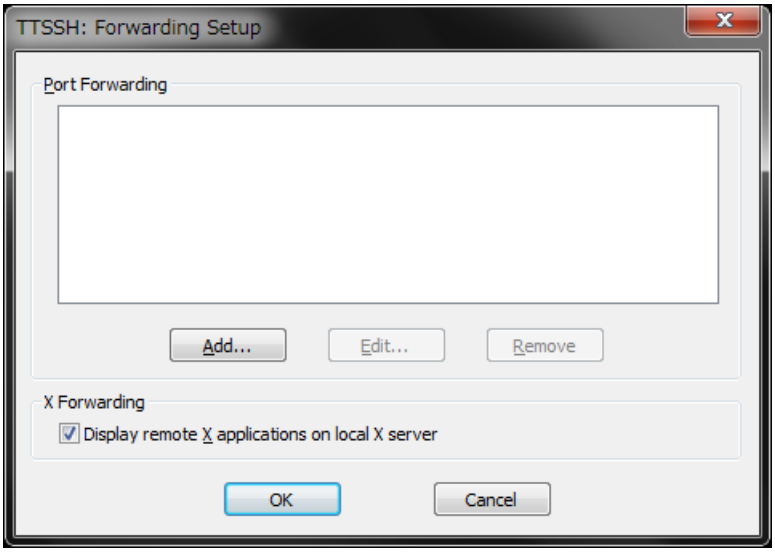


Figure 7-2 Tera Term dialog 2)

3) In the **Select Direction for Forwarded Port** dialog, select **Forward Local Port** and enter the port number to be used for 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.

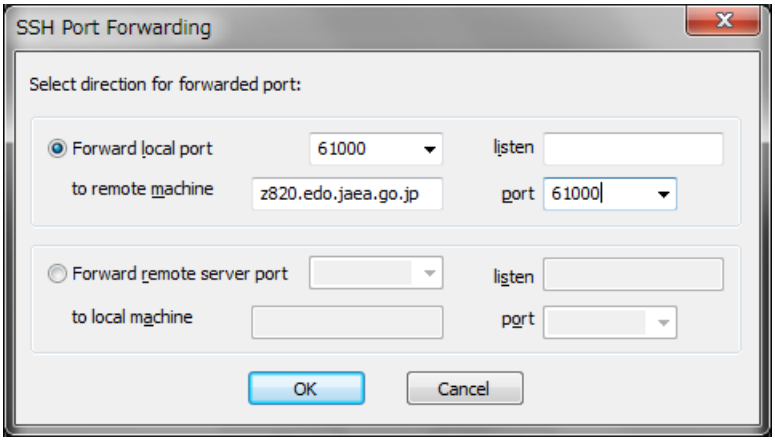


Figure 7-3 Tera Term dialog 3)

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

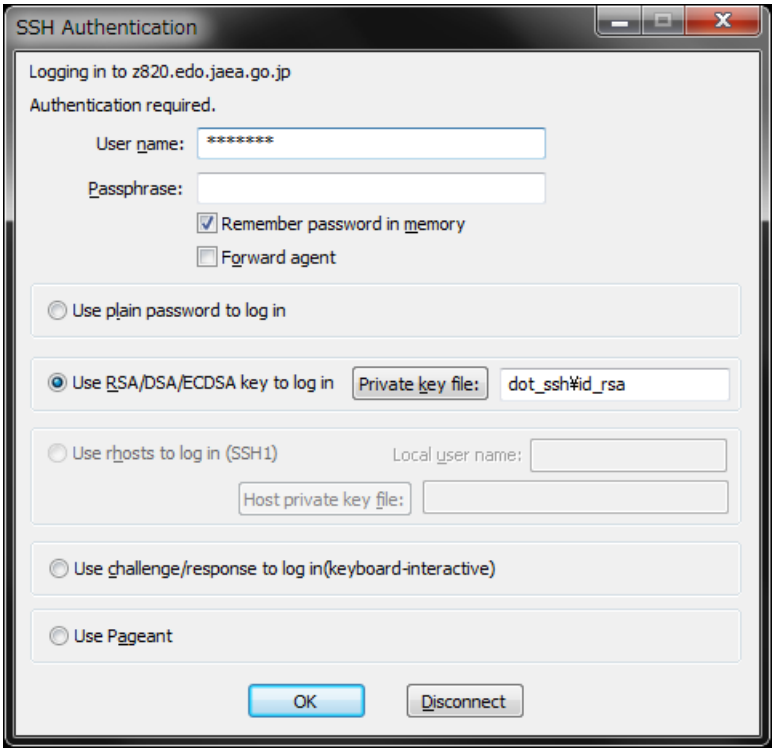


Figure 7-4 Tera Term dialog 4)

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

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