DVRP Library Documentation

A Parallel Library for 3D-Visualization

RRZN (http://www.rrzn.uni-hannover.de) Universität Hannover

DVRP Library Documentation: A Parallel Library for 3D-Visualization

by RRZN (http://www.rrzn.uni-hannover.de) Universität Hannover

Published 13-01-2003

The DVRP Library is a high-performance 3D-visualization library running on parallel computers and is primarily intended for resource intensive scientific simulation applications. It is a component of an innovative concept of distributed computation and visualization which has been developed at the computer center (RRZN (http://www.rrzn.uni-hannover.de)) of the University of Hanover (http://www.uni-hannover.de).

Please report any errors or inconsistencies found in either the library or this documentation to `<Karsten.Chmielewski@rvs.uni-hannover.de>` along with any suggestions for improvements.

Revision History

Revision 1.0	April 2002	Revised by: KC
Initial release.		
Revision 1.1	August 2002	Revised by: KC
Revision 1.2	January 2003	Revised by: KC

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction and Tutorial

This chapter introduces the DVRP Library. First, we provide background information on the DVRP visualization concept in general and how the DVRP Library fits into this scheme (Section 1.1). This is followed by a outline of the DVRP Library user interface (Section 1.2) where certain peculiarities of mixed-language programming are mentioned. It is recommended reading.

The main DVRP Library features are explained in a step-by-step tutorial based on various sample programs starting in Section 1.4. The tutorial includes screenshots of all outputs and web-links to explore the corresponding 3D-scenes.

1.1. Background

An innovative concept of distributed computation and visualization has been developed at the RRZN (http://www.rrzn.uni-hannover.de) (see Stephan Olbrich, 2000). It consists of three components:

- DVRP Library
- DVRP Streaming-Server
- DocShow-VR Viewer

The application interface of the DVRP Library provides the functionality to create 3D-scene descriptions from scientific data. The DVRP Library can directly exploit any parallel processing features of the underlying simulation by performing data-extraction and post-processing on the same processor as the data-generation. For output the 3D-scenes are represented in the "DVR" (*D*ocShow-*V*irtual *R*eality) binary format (see Appendix C for a detailed specification).

The DVR 3D-scenes are transferred to a DVRP Streaming-Server where they are stored. Typically, the data-size of the 3D-scene descriptions, e.g. an iso-surface, is much smaller than the corresponding dataset. Moreover, the DVR binary format has been specially designed to minimize network traffic.

Finally, the generated DVR 3D-scenes are retrieved from the DVRP Streaming-Server and presented by the "DocShow-VR" Viewer. This viewing software uses OpenGL for rendering and runs as a plugin on internet-browsers. The presented scenes can be fully navigated with various input devices. Therefore, users can easily visualize results of their calculations with standard internet technology. The DVRP Streaming-Server allows to access and visualize the 3D-data as soon as they are created. In addition, an integrated back-channel optionally allows to control ("steer") parameters of the simulation through the DocShow-VR Viewer during the calculation.

The DocShow-VR plugin can be obtained from the DVRP Library Homepage (http://www.rvs.uni-hannover.de/people/chmielewski/dvrp/).

1.2. DVRP Library Interface

The DVRP Library is implemented in the C-programming language but can be called from other languages, the most important being Fortran (77/90/95), as well. For use with Fortran 90/95 an DVRP interface module (see Chapter 2) has been developed and enables the user to profit from the advanced language features of Fortran 90/95.

1.2.1. Platform Dependencies

Originally used on the Cray T3E platform the DVRP Library has been ported to various architectures. The complete list of supported platforms is shown in Table 1-1.

Table 1-1. Supported Platforms

Architecture	C-Compiler	Fortran 90/95-Compiler
IBM Regatta	IBM xlc 5	IBM xlf 7/8
Sun Enterprise	Sun WorkShop 6	Sun Workshop 6
	Sun WorkShop 6	NAGWare 4.2
Linux (PC)	gcc	NAGWare 4.2
Cray T3E	Cray Standard C 6.5	Cray Fortran 3.5
HP Risc	HP C compiler	HP F90 v.2.4
SGI	MIPSpro Compilers	MIPSpro Compilers

A few implementation details of the DVRP Library are platform/compiler dependent. For instance, no standardized interface from C to Fortran exists to date. The points in question are:

1. Internal representations of the integer and floating-point types

   The correspondence between the various integer and floating point types in C and Fortran is not unique. In this documentation we denote the floating-point types as “REAL” and the integral types as “INTEGER” both in capital letters. Their interpretation is given in Table 1-2 and Table 1-3.

2. Argument passing between subprograms

   In the C-programming language parameters can be passed by reference or by value whereas in Fortran only the call by reference exists. In order to enable calls to the DVRP Library from Fortran parameters are passed by reference into the library functions.

3. Internal naming conventions of subprograms

   The internal name by functions defined in C are seen from the Fortran compiler is platform/compiler dependent. (One reason is that Fortran names are not case sensitive.) In this documentation we will give all the function/subprogram names in all capital letters. This is the name convention used by various compilers. Generally, the internal names of the DVRP library routines have been adapted to match those of the Fortran compiler for each supported platform.

Note: Items 2 and 3 are not relevant when the DVRP Fortran 95 module is used.

Table 1-2. Type Correspondences in C and Fortran on the T3E

Architecture	C	Fortran	Documentation
T3E	char*	character(len=*)	CHARACTER
	int	integer	INTEGER
	float	—	—
	double	real	REAL

Table 1-3. Type Correspondences in C and Fortran on Intel-x86

Architecture	C	Fortran	Documentation
PC (x86)	char*	character(len=*)	CHARACTER
	int	integer	INTEGER
	float	real	REAL
	double	real(kind(0d0))	—

Table 1-4. Type Correspondences in C and Fortran on Sun Ultra-Enterprise

Architecture	C	Fortran	Documentation
Sun	char*	character(len=*)	CHARACTER
	int	integer	INTEGER
	float	real	REAL
	double	real(kind(0d0))	—

Table 1-5. Type Correspondences in C and Fortran on the HLRN IBM-Regatta

Architecture	C	Fortran	Documentation
HLRN IBM-Regatta	char*	character(len=*)	CHARACTER
	int	integer	INTEGER
	float	real	—
	double	real(kind(0d0))	REAL

More information on the Fortran type parameters is given in Appendix A.

Note: For convenience, the DVRP Library interface module defines the `TYPE cstring`. A variable of that type can be assigned to with an ordinary Fortran `character` expression. All DVRP Library routines in the DVRP Fortran 90/95 interface expect `cstring` arguments.

1.2.2. Global Scope of Variables

Especially when interfacing from Fortran it is important to remember that in fact pointers are passed into the DVRP Library routines and for most routines the data are accessed only through these pointers, i.e., no local copies are made. As a consequence the user has to ensure that variables which have been passed as arguments remain in scope for all DVRP Library routines using those data even if the use is only made implicitly. As a rule of thumb all data that is passed into DVRP Library subprograms *after* a DVRP_INIT call for a specific stream should remain in scope until the final DVRP_EXIT.

For example, registering an array for the use with the DVRP Library in a particular subprogram and later visualizing the scene in another subprogram where the array is not accessible through host association results in undefined behavior and has been a source of error in the past.

1.3. DVRP Library Organization

The DVRP Library routines are organized as follows.

- Initialization and Clean-up
- Logging
- Virtual Camera and 3D-Scene Generation
- Material Properties and Color Management
- Geometric Primitives and Compound Objects
- Visualization of Volume-Data
- Steering

In the following sections we will present sample programs which demonstrate the general usage of the DVRP Library. The detailed description of all the library functions is given in Chapter 2.

> **Note:** All examples have been run through a code beautifier and are available as PDF documents as well as plain text via their respective links.

1.4. Sample Program 1

The goal of the first sample program (PDF (samples/demo1.pdf), text (samples/demo1.F90)) is to generate a 3D-scene showing two static particles (spheres).

- Initialization and Clean-up

  The access to the DVRP Library is established by importing the DVRP Fortran module via the `use DVRP` statement. All programs accessing the DVRP Library need to call the DVRP_INIT initialization function before any other library function. Here, a new output stream (number "`0`") is created.

  Calling the DVRP_OUTPUT_LOCAL function associates a local filename with this stream which will be used for output.

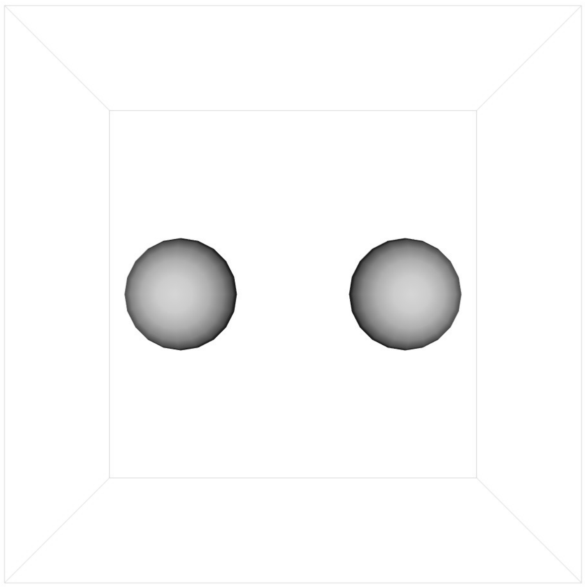
  The following 2 function calls define the bounding box to be displayed (DVRP_BOUNDINGBOX) along with properties of the virtual camera (DVRP_CAMERA).

The call to DVRP_VISUALIZE sends the DVR 3D-scene description corresponding to the previous definitions to the output stream. Since the virtual camera setup is finished the DVRP_EXIT function for that stream is invoked.

- Visualization of Volume-Data

  Another output stream is created and associated to another local file in the following two calls to DVRP_INIT and DVRP_OUTPUT_LOCAL. The file is intended to hold the DVR 3D-data description. The coordinates of the two spherical particles are setup. Their visual appearance is controlled by the call to DVRP_MATERIAL_RGB. (Strictly, this call belongs into the "Material Properties and Color Management" section.) In this example all parameters are zero, using default properties for the material finish. The DVRP_PARTICLES function generates the introduces the particles at the indicated positions into the scene and sets their radius. As before, DVRP_VISUALIZE is used to send the DVR 3D-scene description to the output stream with a parameter identifying the data as particles. The program is finalized with another call to DVRP_EXIT.

Figure 1-1. Output of Demo 1

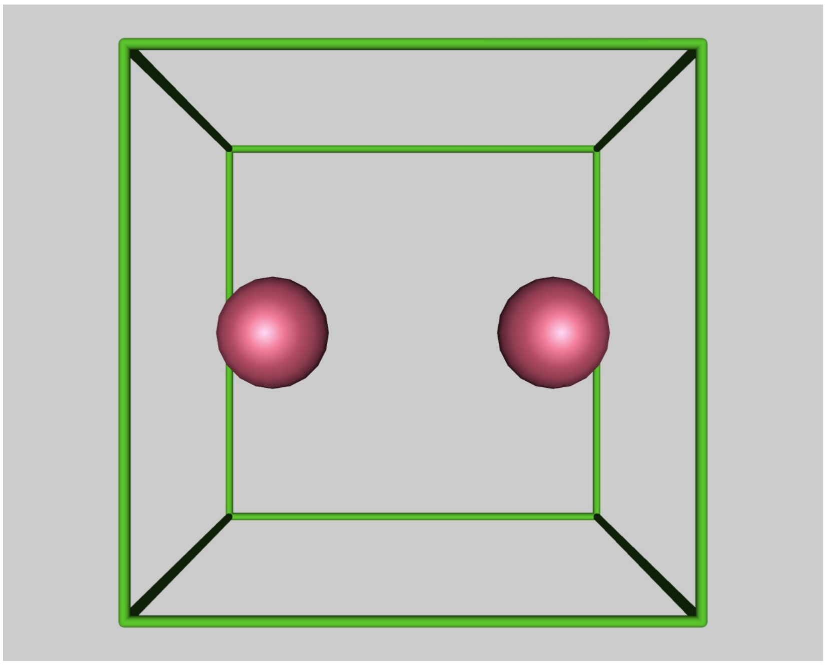


1.5. Sample Program 2

The second sample program differs from the Demo 2 (PDF (samples/demo2.pdf), text (samples/demo2.f90)) previous one by sending the output to the streaming server (the SGI Origin) and by generating a whole sequence of 3D-scenes depicting the two particles in rotation.

These differences are achieved by calling DVRP_OUTPUT_RTSP instead of DVRP_OUTPUT_LOCAL which requires host and user data to be specified. The animation sequence is produced by the `do nr = 1, 40` loop. Inside the loop the respective particle positions are generated, passed to the DVRP Library by the DVRP_PARTICLES call and finally send to the output stream with the DVRP_VISUALIZE.

Figure 1-2. Output of Demo 2



1.6. Sample Program 3

This example Demo 3 (PDF (samples/demo3.pdf), text (samples/demo3.F90)) introduces steering, i.e., the possibility for a user to interact with the running simulation program through a feedback channel. Variables can be made available for steering through the DVRP_STEERING_INIT function. It requires to pass information about the variable name, its type and the valid range. A user connected to the streaming server during program execution can access those variables through a popup steering-control window on the DocShow plugin and change its value in the valid range through sliders.

The simulation can make repeated calls to DVRP_STEERING_UPDATE to synchronize any user-made changes with the corresponding variables. In the example Demo 3, showing particles in motion, the user can interactively change their color.

Warning

The variables intended for steering have to be in global scope. No local variables should be used. It is recommended to declare those variables with the SAVE attribute in a module file which is then accessed from the main program.

Figure 1-3. Output of Demo 3 (a)

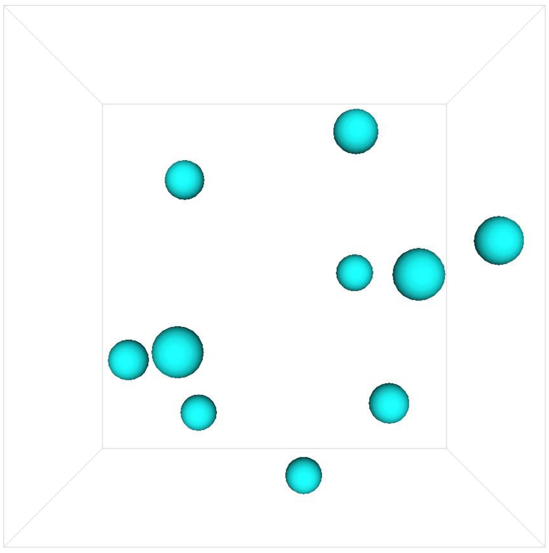


Figure 1-4. Output of Demo 3 (b)

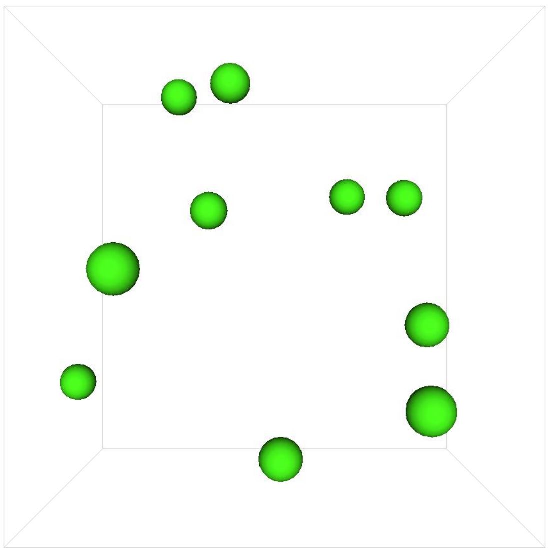
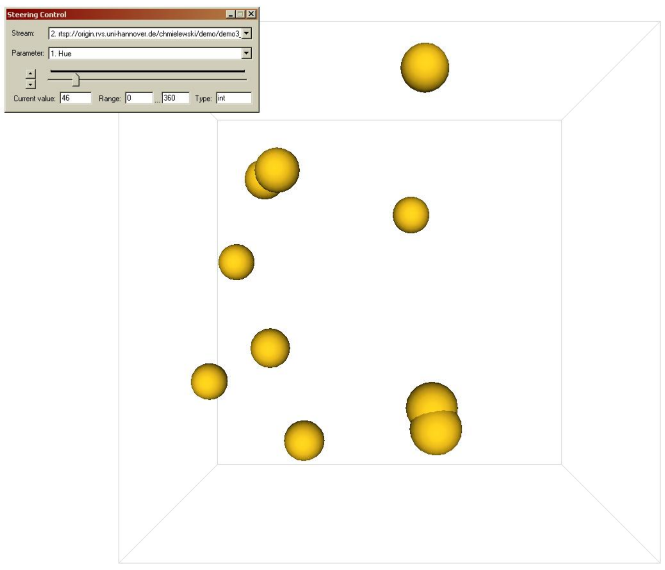


Figure 1-5. Output of Demo 3 (c)



Stream: 2. rtsp://origin.rvs.uni-hannover.de/chmielewski/demo/demo3_

Parameter: 1. Hue

Current value: 46 Range: 0 ... 360 Type: int

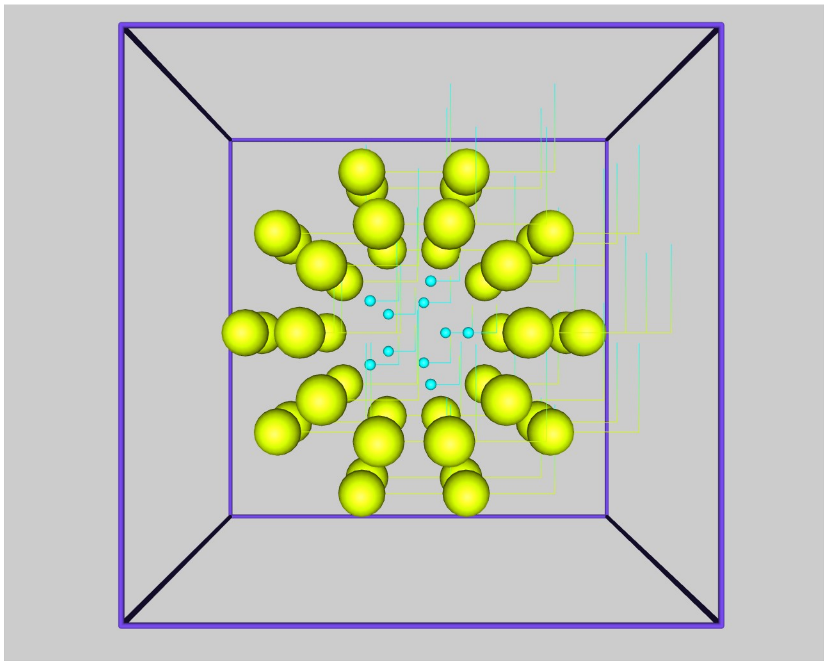
1.7. Sample Program 4

This sample program Demo 4 (PDF (samples/demo4.pdf), text (samples/demo4.f90)) visualizes particles with DVRP Library functions of higher complexity.

First, it makes use of parallel computing features. At program start call to the MPI parallel library initialize the parallel environment in the usual way. In this particular example the program will run on two processors.

Second, the function DVRP_PARTICLES allows refined control over the particle properties. For instance, radii and color can be assigned on a per particle basis. Optionally, line segments indicating the particle motion can be drawn.

Figure 1-6. Output of Demo 4



1.8. Sample Program 5

This sample program Demo 5 (PDF (samples/demo5.pdf), text (samples/demo5.f90) visualizes iso-surfaces and slicers. This is accomplished by defining 3D-volume data on a grid, setting up iso-surface and slicer properties and finally calling the DVRP_VISUALIZE function with the appropriate mode parameter.

Figure 1-7. Output of Demo 5 (iso-surface)

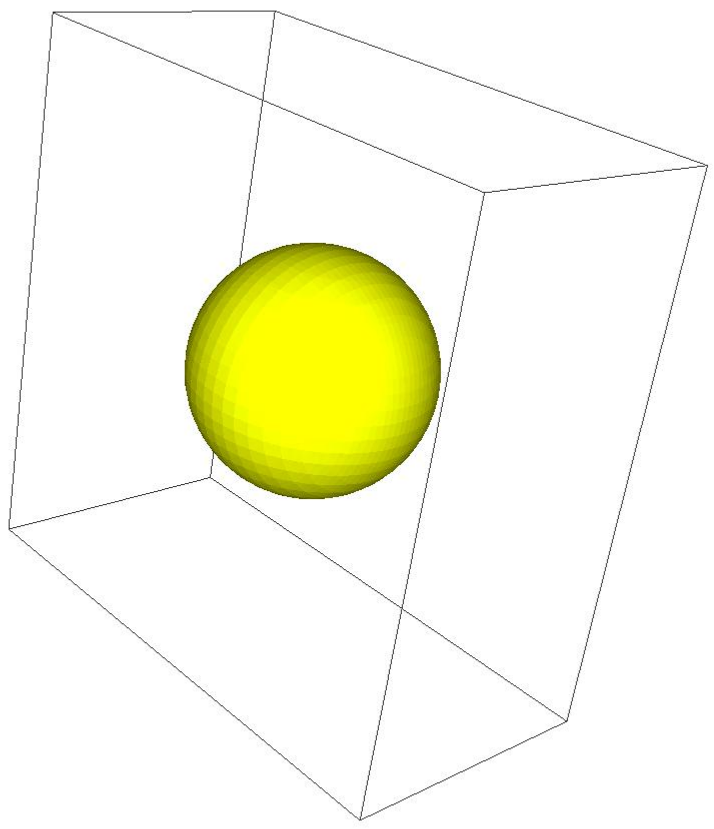
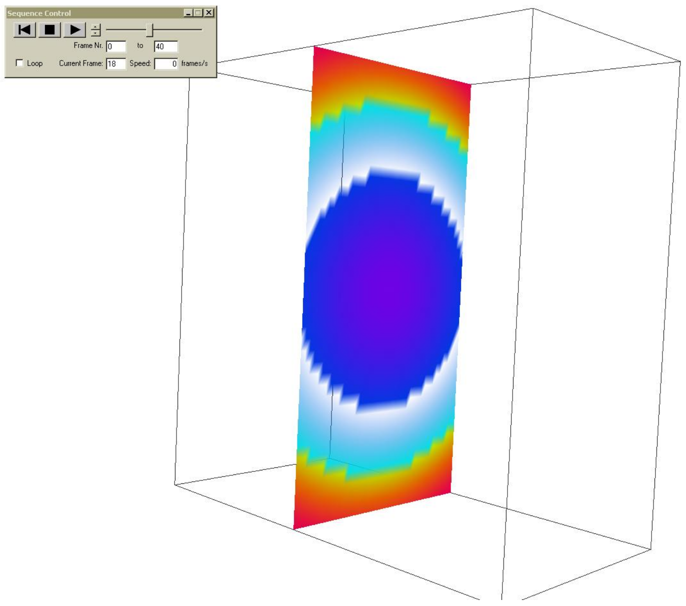


Figure 1-8. Output of Demo 5 (slicer)



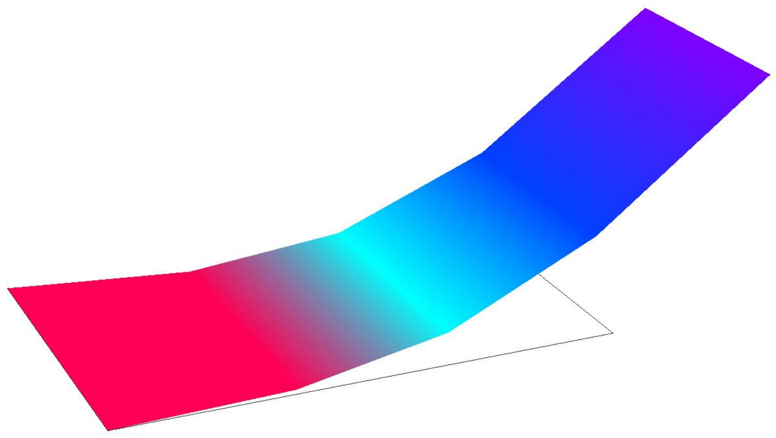
Frame Nr.: 0 to 40

Loop Current Frame: 18 Speed: 0 frames/s

1.9. Sample Program 6

This sample program Demo 6 (PDF (samples/demo6.pdf), text (samples/demo6.f90)) visualizes polygons. This is accomplished by defining 3D-volume data on a set of vertices by calling DVRP_VERTICES. A polygon mesh is defined through the DVRP_POLYGONS library call. Finally, the DVRP_RESULTS declares the array holding the 3D-volume data.

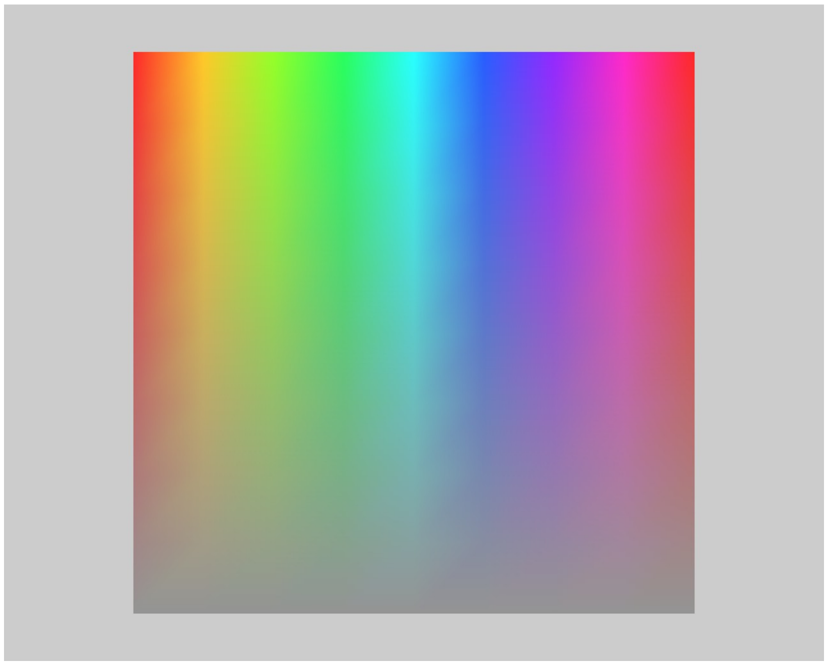
Figure 1-9. Output of Demo 6



1.10. Sample Program 7

This sample program Demo 7 (PDF (samples/demo7.pdf), text (samples/demo7.f90)) animates the HLS color model. Since the color space is three-dimensional it cannot be visualized directly. Here, the color-dependence on the values of hue and saturation is shown in a two-dimensional plot. The dependence on the lightness is visualized by varying its value with time starting from light values. The main DVRP Library call is **DVRP_COLORTABLE_HLS** .

Figure 1-10. Output of Demo 7

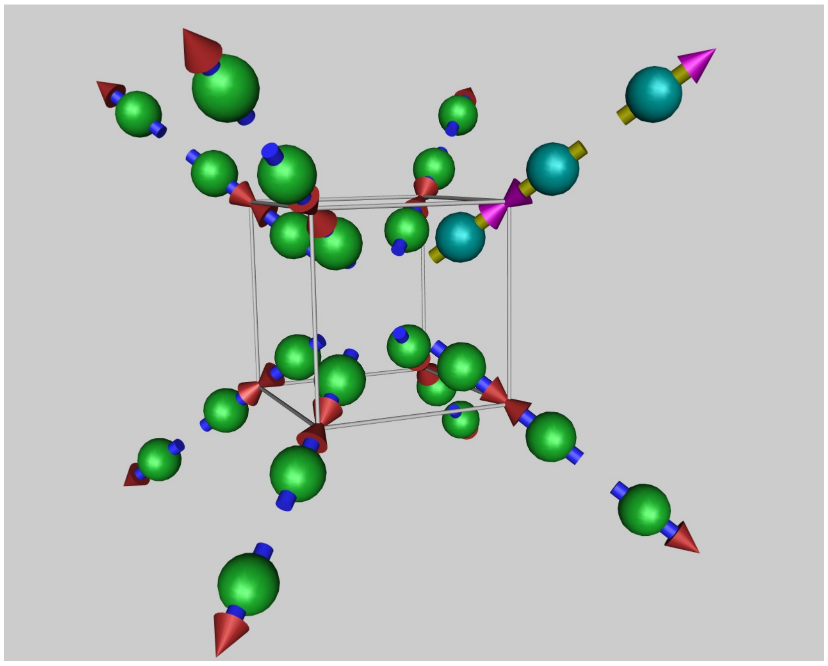


1.11. Sample Program 8

The sample program Demo 8 (PDF (samples/demo8.pdf), text (samples/demo8.f90)) introduces particles with spin, i.e., spheres with an additional axis of orientation. Particles with spin are easily created through another overloaded version of the DVRP_PARTICLES subprogram.

Internally, a particle with spin is composed of a sphere, cylinder and cone with an arrow-like appearance of cylinder and cone along the orientation axis. The subprogram DVRP_MATERIAL allows to specify the individual material properties of the primitives in a single call.

Figure 1-11. Output of Demo 8



Chapter 2. Fortran 90/95-Interface

This section describes the interface to the DVRP Library from Fortran 90/95.

2.1. The DVRP module

To simplify the access to the DVRP Library from Fortran 90/95 the module **DVRP** is provided. It defines the interface to the DVRP Library subprograms and ensures that all actual arguments are passed with the correct type and rank. It can be accessed via a

```
use DVRP
```

statement.

The DVRP Library Module simplifies the usage of the DVRP Library and minimizes the chance of inadvertent errors by

- overloading subprogram names
- defining the type `cstring` which encapsulates string/character handling in a transparent way
- freeing the user to pass array sizes as separate arguments

> **Note:** The `use`-Statement is mandatory if the Fortran 90/95 interface shall be used and has to be included in all program units which access the DVRP Library. We will suppress the `use`-Statement in the detailed routine descriptions that follow.

2.2. Initialization and Clean-up

This section covers those DVRP Library functions that are needed to open and close output streams and to specify their properties. Various different output channels can be open at the same time. The generated DVR 3D-scene description data can be output to a local file, a remote host via FTP or to a streaming server.

2.2.1. DVRP_INIT

Opens and initializes a DVRP Library output stream.

Synopsis

```
interface
   subroutine DVRP_INIT(stream, verbose)
     INTEGER, intent(in) :: stream, verbose
   end subroutine DVRP_INIT
end interface
```

Arguments

`INTEGER stream` (input)

Number of the output stream to be initialized.

`INTEGER verbose` (input, optional)

Selects whether verbose messages should be output (verbose=1) or not (verbose=0). Default: verbose=0.

Description

The DVRP_INIT function initializes the DVRP Library for the selected stream. It has to be the first DVRP Library function to be invoked for that stream.

2.2.2. DVRP_EXIT

Closes an DVRP output stream.

Synopsis

```
interface
   subroutine DVRP_EXIT (stream)
     INTEGER, intent(in) :: stream
   end subroutine DVRP_EXIT
end interface
```

Arguments

`INTEGER stream` (input)

Number of the output stream to be closed.

2.2.3. DVRP_OUTPUT_FTP

Prepares a given stream for output via FTP.

Synopsis

```
interface
   subroutine DVRP_OUTPUT_FTP (stream, append, host, username, password, &
        directory, filename)
     INTEGER, intent(in) :: stream, append
     type(cstring), intent(in) :: host, username, password, directory, filename
   end subroutine DVRP_OUTPUT_FTP
end interface
```

Arguments

`INTEGER stream` (input)

Number of the output stream to be initialized.

`INTEGER append` (input)

Specifies whether output will overwrite (append=0) or append to (append=1) the associated remote file.

`CHARACTER host` (input)

Name of the host.

`CHARACTER username` (input)

User name on the remote host.

`CHARACTER password` (input)

Password on the remote host.

`CHARACTER directory` (input)

Directory on the remote host.

`CHARACTER filename` (input)

Filename pattern on the remote host to be used for DVR output associated with the specified stream.

Description

Filenames for DVRP Library output should generally have the suffix "dvr." The part of the filename specification before the file suffix acts as a pattern for the actual generation of filenames. The pattern is given as a C-language print format. This is useful when a sequence of output files is generated. The pattern string determines the length of the filename part that will hold the sequence number.

Example: `filename = 'myfile_%02d.dvr'` will generate filenames starting with "`myfile_`" followed by a two-digit sequence number and the "`.dvr`" suffix.

2.2.4. DVRP_OUTPUT_LOCAL

Prepares a given stream for output into a local file.

Synopsis

```
interface
   subroutine DVRP_OUTPUT_LOCAL (stream, append, filename)
     INTEGER, intent(in) :: stream, append
     type(cstring), intent(in) :: filename
   end subroutine DVRP_OUTPUT_LOCAL
end interface
```

Arguments

`INTEGER stream` (input)

Number of the output stream to be initialized.

`INTEGER append` (input)

Specifies whether output will overwrite (append=0) or append to (append=1) the associated remote file. The **DVRP** module provides mnemonic names for these constants: DVRP_local_replace, DVRP_local_append.

`CHARACTER filename` (input)

Filename pattern on the local host to be used for DVR output associated with the specified stream.

Description

Filenames for DVRP Library output should generally have the suffix "dvr." The part of the filename specification before the file suffix acts as a pattern for the actual generation of filenames. The pattern is given as a C-language print format. This is useful when a sequence of output files is generated. The pattern string determines the length of the filename part that will hold the sequence number.

Example: `filename = 'myfile_%02d.dvr'` will generate filenames starting with "`myfile_`" followed by a two-digit sequence number and the "`.dvr`" suffix.

2.2.5. DVRP_OUTPUT_RTSP

Prepares a given stream for output to a DVRP streaming server.

Synopsis

```
interface
   subroutine DVRP_OUTPUT_RTSP (stream, host, username, password, directory, filename)
     INTEGER, intent(in) :: stream
     type(cstring), intent(in) :: host, username, password, directory, filename
   end subroutine DVRP_OUTPUT_RTSP
end interface
```

Arguments

`INTEGER stream` (input)

Number of the output stream to be initialized.

`CHARACTER host` (input)

Name of the host.

`CHARACTER username` (input)

User name on the remote host.

`CHARACTER password` (input)

User name on the remote host.

`CHARACTER directory` (input)

Directory on the remote host.

`CHARACTER filename` (input)

Filename pattern on the remote host to be used for DVR output associated with the specified stream.

Description

Filenames for DVRP Library output should generally have the suffix “dvr.” The part of the filename specification before the file suffix acts as a pattern for the actual generation of filenames. The pattern is given as a sequence of asterisks. This is useful when a sequence of output files is generated. The pattern string determines the length of the filename part that will hold the sequence number.

Example: `filename = 'myfile_**.dvr'` will generate filenames starting with “`myfile_`” followed by a two-digit sequence number and the “`.dvr`” suffix.

2.3. Logging

In this section routines for logging are covered. They allow to track DVRP Library actions and events to be traced and access this information after program execution from a log-file.

2.3.1. DVRP_LOG

General driver routine for the logging facility. For easy access to the logging facilities specialized functions with a simplified interface are provided.

Synopsis

```
interface
   subroutine DVRP_LOG (mode, event, parameter, name)
     INTEGER, intent(in) :: mode, event, parameter
     type(cstring), intent(in) :: name
   end subroutine DVRP_LOG
end interface
```

Arguments

`INTEGER mode` (input)

Specifies the action to be taken: 0 - initialization, 1 - an event is logged, 2 - an event name is defined, 3 - finalize the logging and exit

`INTEGER event` (input)

Number of the event for which the function is called.

`INTEGER parameter` (input)

The interpretation of this argument depends on the selected mode.

mode = 0: Specifies whether the logging will overwrite (parameter = 0) or append to (parameter = 1) the specified file.

mode = 1: Number of the event to be logged.

mode = 2: Number of the event to be logged.

`CHARACTER name` (input)

Specifies the filename used for the logging.

Description

It is recommended to use the specialized driver routines.

2.3.2. DVRP_LOG_DISABLE

Disable logging for a given event number.

Synopsis

```
interface
   subroutine DVRP_LOG_DISABLE (event)
     INTEGER, intent(in) :: event
   end subroutine DVRP_LOG_DISABLE
end interface
```

Arguments

`INTEGER event` (input)

Number of the event whose logging is to be disabled.

2.3.3. DVRP_LOG_ENABLE

Disable logging for a given event number.

Synopsis

```
interface
   subroutine DVRP_LOG_ENABLE (event)
     INTEGER, intent(in) :: event
   end subroutine DVRP_LOG_ENABLE
end interface
```

Arguments

`INTEGER event` (input)

Number of the event whose logging is to be disabled.

2.3.4. DVRP_LOG_EVENT

Synopsis

```
interface
   subroutine DVRP_LOG_EVENT (event, parameter)
     INTEGER, intent(in) :: event, parameter
   end subroutine DVRP_LOG_EVENT
end interface
```

Arguments

`INTEGER event` (input)

Event number to be logged.

`INTEGER parameter` (input)

Parameter written into the log-file.

2.3.5. DVRP_LOG_EXIT

Finalizes the logging.

Synopsis

```
interface
   subroutine DVRP_LOG_EXIT (allevents)
     INTEGER, intent(out) :: allevents
   end subroutine DVRP_LOG_EXIT
end interface
```

Arguments

`INTEGER allevents` (output)

Total number of events that have been logged.

2.3.6. DVRP_LOG_INIT

Initializes the logging facility.

Synopsis

```
interface
   subroutine DVRP_LOG_INIT (name, append)
     INTEGER, intent(in) :: append
     type(cstring), intent(in) :: name
   end subroutine DVRP_LOG_INIT
end interface
```

Arguments

`CHARACTER name` (input)

Specifies the file used for the logging.

`INTEGER append` (input)

Specifies whether the logging will overwrite (append = 0) or append to (append = 1) the specified file.

2.3.7. DVRP_LOG_SYMBOL

Makes a new symbol available for logging.

Synopsis

```
interface
   subroutine DVRP_LOG_SYMBOL (event, name)
     INTEGER, intent(in) :: event
     type(cstring), intent(in) :: name
   end subroutine DVRP_LOG_SYMBOL
end interface
```

Arguments

`INTEGER event` (input)

Event number of the new symbol which will be logged.

`CHARACTER name` (input)

Name of the event.

2.3.8. DVRP_VERBOSE

Specifies whether verbose output message are requested.

Synopsis

```
interface
   subroutine DVRP_VERBOSE (stream, verbose)
     INTEGER, intent(in) :: stream, verbose
   end subroutine DVRP_VERBOSE
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER verbose` (input)

Specifies whether verbose output message are requested for the given output stream.

2.4. Virtual Camera and 3D-Scene Generation

This section explains how to set-up a virtual camera and generate the 3D-scene description DVR files. The virtual camera describes the perspective taken by the fictitious observer. Optionally, a bounding box can be drawn into the 3D-scene in order to guide the eye.

After the virtual camera and the 3D-objects, using routines to be covered in Section 2.6 and Section 2.7, have been specified the DVR 3D-scene data is finally created with a Section 2.4.3 call.

2.4.1. DVRP_CAMERA

Specifies properties of the virtual camera

Synopsis

```
interface
   subroutine DVRP_CAMERA (stream, center, distance)
     INTEGER, intent(in) :: stream
     REAL, intent(in) :: center(:), distance
   end subroutine DVRP_CAMERA
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`REAL center(3)` (input)

One dimensional array of extent 3 defining the position of the virtual camera.

`REAL distance` (input)

Focal distance of the virtual camera.

2.4.2. DVRP_BOUNDINGBOX

Defines the bounding box to be shown along with the 3d-data. The rectangular box, the edges of which are oriented along the coordinate axes, is specified by the coordinates of two of its vertices which share no adjacent surface.

Synopsis

```
interface
   subroutine DVRP_BOUNDINGBOX (stream, mode, r, x0, y0, z0, x1, y1, z1)
     INTEGER, intent(in) :: stream, mode
     REAL, intent(in) :: r, x0, y0, z0, x1, y1, z1
   end subroutine DVRP_BOUNDINGBOX
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER mode` (input)

Specifies the visual representation of the bounding box edges: 0 - None, 1 - Lines, 2 - Cylinders. The **DVRP** module provides mnemonic names for these constants: DVRP_BB_None, DVRP_BB_Lines, DVRP_BB_Cyl.

`REAL r` (input)

Radius of the cylinders to be displayed when argument mode equals 2.

`REAL x0` (input)

The x-coordinate of the first vertex.

`REAL y0` (input)

The y-coordinate of the first vertex.

`REAL z0` (input)

The z-coordinate of the first vertex.

`REAL x1` (input)

The x-coordinate of the second vertex.

`REAL y1` (input)

The y-coordinate of the second vertex.

`REAL z1` (input)

The z-coordinate of the second vertex.

Description

The material properties for the cylindrical representation can be specified by a preceding call to one of the DVRP material subprograms: DVRP_MATERIAL_RGB, DVRP_MATERIAL_HLS.

2.4.3. DVRP_VISUALIZE

Generates the DVR 3D-scene description and sends it to the DVRP Library output stream.

Synopsis

```
interface
  subroutine DVRP_VISUALIZE (stream, mode, nr)
    INTEGER, intent(in) :: stream, mode, nr
  end subroutine DVRP_VISUALIZE
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER mode` (input)

Possible values and their meaning are summarized in the Table 2-1.

Table 2-1. Mode Parameters (Visualize)

Value	Object	Mnemonic Name

Value	Object	Mnemonic Name
0	None	DVRP_Vis_None
1	Iso-Surface	DVRP_Vis_Isosurface
2	Slicer	DVRP_Vis_Slicer
3	Particles	DVRP_Vis_Particles
4	Polygons	DVRP_Vis_Polygons
5	Coloured Polygons	DVRP_Vis_ColouredPolygons
6	Cones	DVRP_Vis_Cones
7	Cylinder	DVRP_Vis_Cylinder
8	Particles with Spin (Dipole)	DVRP_Vis_Dipole
9	Molecules	DVRP_Vis_Molecule

`INTEGER nr` (input)

Specifies the sequence number.

2.5. Material Properties and Color Management

This section covers those DVRP Library functions that specify the visual properties of the 3D-objects to be visualized. Among the properties that can be defined are material and color characteristics. Material properties describe the optical behavior of the surfaces exposed by the 3D-objects.

Color specifications are based on the RGB (red, green, blue) color model or the HLS (hue, lightness, saturation) color model. Details of the HLS color model are presented in Appendix B. See also sample program 7 which gives an impression of the three dimensional color space of HLS values by animating a surface with varying color values.

2.5.1. DVRP_COLORTABLE_HLS

Specifies the colors to be used in the visualization according to the HLS color model.

Synopsis

```
interface
   subroutine DVRP_COLORTABLE_HLS (stream, mode, values, h, l, s, transparency)
    INTEGER, intent(in) :: stream, mode
    REAL, intent(in), dimension(:,:) :: values, h, l, s, transparency
  end subroutine DVRP_COLORTABLE_HLS
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER mode` (input)

Specifies the visual surface properties: 0 - diffuse and ambient color material, 1 - emissive color material. The **DVRP** module provides mnemonic names for these constants: DVRP_Col_Diffuse, DVRP_Col_Emissive.

`INTEGER intervals` (input)

Number of color intervals.

`REAL values` (input)

Array of `intervals` lower interval end points.

`REAL h` (input)

Array of HUE values.

`REAL l` (input)

Array of lightness values.

`REAL s` (input)

Array of saturation values.

`REAL transparency` (input)

Array of transparency values.

Description

The colors to be used in the visualization are specified by first defining arrays of color attributes and then associating with each color-table element a range of real values out of a contiguous sequence of real-valued intervals. In this way any real value of the application can be uniquely mapped into a color table entry which will be used for the visualization.

2.5.2. DVRP_COLORTABLE_RGB

Specifies the colors to be used in the visualization according to the RGB color model.

Synopsis

```
interface
   subroutine DVRP_COLORTABLE_RGB (stream, mode, values, r, g, b, transparency)
    INTEGER, intent(in :: stream, mode
    REAL, intent(in), dimension(:,:) :: values, r, g, b, transparency
  end subroutine DVRP_COLORTABLE_RGB
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER mode` (input)

Specifies the visual surface properties: 1 - diffuse and ambient color material, 2 - emissive color material.

`INTEGER intervals` (input)

Number of color intervals.

`REAL values` (input)

Array of `intervals` lower interval end points.

`REAL r` (input)

Array of Red color values.

`REAL g` (input)

Array of Green color values.

`REAL b` (input)

Array of Blue color values.

`REAL transparency` (input)

Array of transparency values.

Description

The colors to be used in the visualization are specified by first defining arrays of color attributes and then associating with each color-table element a range of real values out of a contiguous sequence of real-valued intervals. In this way any real value of the application can be uniquely mapped into a color table entry which will be used for the visualization.

2.5.3. DVRP_MATERIAL

Specifies the material properties used for the visualization. At present this call is only used for the material specification for particles with spin. Its interface is subject to future change.

Synopsis

```
interface
   subroutine DVRP_MATERIAL(stream, modes, colors)
     INTEGER, intent(in) :: stream, modes(:)
     REAL, intent(in) :: colors(:,:)
   end subroutine DVRP_MATERIAL
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER modes(3)` (input)

Specifies the material mode for each of the underlying primitives. The order of primitives is: Sphere, cylinder, cone.

`REAL colors(3,4)` (input)

Color values of the underlying primitives. The order of primitives is: Sphere, cylinder, cone. The first dimension of **colors** respectively refers to sphere, cylinder and cone; the second dimension to the red, green, blue and transparency color attributes.

Description

The **DVRP_MATERIAL** subprogram is intended to provide a unique interface for the material properties. At present its use is limited to specify the material properties for particles with spin.

2.5.4. DVRP_MATERIAL_HLS

Specifies the material properties used for the visualization according to the HLS color model.

Synopsis

```
interface
   subroutine DVRP_MATERIAL_HLS (stream, mode, hue, l, s, transparency)
     INTEGER, intent(in) :: stream, mode
     REAL, intent(in) :: hue, l, s, transparency
   end subroutine DVRP_MATERIAL_HLS
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER mode` (input)

Possible values and their meaning are summarized in the Table 2-2.

Table 2-2. Mode Parameters (DVRP_MATERIAL_HLS)

Value	Object	Mnemonic Name
0	None	DVRP_Mat_None
1	Ambient and Diffuse	DVRP_Mat_AmbientDiffuse
2	Emissive	DVRP_Mat_Emissive
3	Specular	DVRP_Mat_Specular
4	Diffuse	DVRP_Mat_Diffuse
5	Ambient	DVRP_Mat_Ambient

`REAL hue` (input)

HUE value of the surface.

`REAL l` (input)

Lightness value of the surface.

`REAL s` (input)

Saturation value of the surface.

`REAL transparency` (input)

Transparency value of the surface.

Description

Further information about the HLS color model is provided in Appendix B.

2.5.5. DVRP_MATERIAL_RGB

Specifies the material properties used for the visualization according to the RGB color model.

Synopsis

```
interface
   subroutine DVRP_MATERIAL_RGB (stream, mode, red, green, blue, transparency)
     INTEGER, intent(in) :: stream, mode
```

```
      REAL, intent(in) :: red, green, blue, transparency
    end subroutine DVRP_MATERIAL_RGB
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER mode` (input)

See Table 2-2

`REAL red` (input)

Red proportion of the color. (Argument range: [0, 1])

`REAL green` (input)

Green proportion of the color. (Argument range: [0, 1])

`REAL blue` (input)

Blue proportion of the color. (Argument range: [0, 1])

`REAL transparency` (input)

Degree of transparency. (Argument range: [0, 1])

2.6. Geometric Primitives and Compound Objects

This section describes the usage of objects either made of geometrical primitives or compounds thereof.

2.6.1. DVRP_CONE

Defines cone primitives to be visualized. Each cone is characterized by its center-of-mass position and its orientation axis.

Synopsis

```
interface
  subroutine DVRP_CONE (stream, mode, vertices, orientation, height, radius)
    INTEGER, intent(in) :: stream, mode
    REAL, intent(in) :: vertices(:,:), orientation(:,:), height, radius
  end subroutine DVRP_CONE
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER mode` (input)

Specifies which parts of the cone are to be displayed. Its value is the sum of all the desired components. Possible components are: 1 - sides, 2 - bottom. For instance, mode = 3 selects the sides and the bottom to be drawn.

`REAL vertices(:,:)` (input)

Center-of-mass positions of the cones. The extent of the first dimension is equal to the number of cones. The extent of the second dimension has to be equal to three with vertices(i,:) holding the x-, y-, and z-coordinates of the i-th cone.

`REAL orientation(:,:)` (input)

Orientations of the cones. The extent of the first dimension is equal to the number of cones. The extent of the second dimension has to be equal to three with orientation(i,:) holding the x-, y-, and z-direction of the i-th cone's orientation. The orientation vector does not have to be normalized.

`REAL height` (input)

The height of the cone.

`REAL radius` (input)

The radius of the cone.

Description

2.6.2. DVRP_CYLINDER

Defines cylinder primitives to be visualized. Each cylinder is characterized by its center-of-mass position and its orientation axis.

Synopsis

```
interface
  subroutine DVRP_CYLINDER (stream, mode, vertices, orientation, height, radius)
    INTEGER, intent(in) :: stream, mode
    REAL, intent(in) :: vertices(:,:), orientation(:,:), height, radius
  end subroutine DVRP_CYLINDER
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER mode` (input)

Specifies which parts of the cylinder are to be displayed. Its value is the sum of all the desired components. Possible components are: 1 - sides, 2 - top, 4 - bottom. For instance, mode = 5 selects the sides and the bottom to be drawn.

`REAL vertices(:,:)` (input)

Center-of-mass positions of the cylinders. The extent of the first dimension is equal to the number of cylinders. The extent of the second dimension has to be equal to three with vertices(i,:) holding the x-, y-, and z-coordinates of the i-th cylinder.

`REAL orientation(:,:)` (input)

Orientations of the cylinders. The extent of the first dimension is equal to the number of cylinders. The extent of the second dimension has to be equal to three with orientation(i,:) holding the x-, y-, and z-direction of the i-th cylinder's orientation. The orientation vector does not have to be normalized.

`REAL height` (input)

The height of the cylinder.

`REAL radius` (input)

The radius of the cylinder.

Description

2.6.3. DVRP_PARTICLES

Defines 3D-particles to be visualized. This subprogram is overloaded and can be called in four ways.

Synopsis

```
interface
  subroutine DVRP_PARTICLES(stream, xp, yp, zp, radius)
    INTEGER, intent(in) :: stream
    REAL, intent(in) :: xp(:), yp(:), zp(:), radius
  end subroutine DVRP_PARTICLES
end interface
```

Synopsis

```
interface
  subroutine DVRP_PARTICLES(stream, xp, yp, zp, radius)
    INTEGER, intent(in) :: stream
    REAL, intent(in) :: xp(:), yp(:), zp(:), radius(:)
  end subroutine DVRP_PARTICLES
end interface
```

Synopsis

```
interface
   subroutine DVRP_PARTICLES(stream, npart, xp, yp, zp, mode, &
       radius, col, seglen)
     INTEGER, intent(in) :: stream, npart, mode, col(:), seglen(:)
     REAL, intent(in) :: xp(:), yp(:), zp(:), radius(:)
   end subroutine DVRP_PARTICLES
end interface
```

Synopsis

```
interface
   subroutine DVRP_PARTICLES(stream, location, orientation, radii, &
        offset, groups);
      INTEGER, intent(in) :: stream
      REAL, intent(in) :: center(:,:), orientation(:,:), radii(:)
      REAL, intent(in), optional :: offset(2)
      INTEGER, intent(in), optional :: groups(:)
   end subroutine DVRP_PARTICLES
end interface
```

This version of the DVRP_PARTICLES subroutine defines particles with spin. Each particle is characterized by its center-of-mass position and an orientation axis. Internally, a particle with spin is composed of a sphere, cylinder and cone with an arrow-like appearance of cylinder and cone along the orientation axis. The size of sphere, cylinder and cone can be specified independently. Further parameters can be changed from the default settings via optional arguments.

The location argument defines the center-of-mass of the sphere. Cylinder and cone are placed with their center-of-mass along the axis of orientation at at distance from the sphere which is either set by default or via the optional offset parameter.

The optional groups argument is used in correspondence with refers to a can be used to define different is a set of integers n_i which partition the total number of particles N, i.e., N = \sum_i n_i. n_i = k means: k particles will be colored according to the color specification of color group i.

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`REAL xp` (input)

One dimensional array holding the x-coordinates of the particles.

`REAL yp` (input)

One dimensional array holding the x-coordinates of the particles.

`REAL zp` (input)

One dimensional array holding the x-coordinates of the particles.

`REAL radius` (input)

Specifying the radius of the particles. It can either be a scalar assigning the same radius to all particles or a one dimensional array describing individual particle radii.

`INTEGER mode` (input)

Specifies the particle visualization mode. Its value is interpreted as a bit field where individual bits can be switched on and off. Their meaning is as follows:

1 - Radii are displayed.

2 - Colors are displayed on a per-particles basis.

4 - Lines are displayed. This mode is a means of visualizing the particle motion by showing a part of the arc traveled.

8 - Colors are displayed per vertex.

`INTEGER color` (input)

One dimensional array of indices. Each entry holds the index-pointer into a color-table object which will be used for that particle.

`INTEGER seglen` (input)

One dimensional array describing the length of the particle arcs.

`REAL location(:,:)` (input)

Two dimensional array specifying the center-of-mass location of the particles with spin. The extent of the first dimension is equal to the number of particles. The extent of the second dimension has to be equal to three with location(i,:) holding the x-, y-, and z-coordinates of the i-th particle.

`REAL orientation(:,:)` (input)

Two dimensional array specifying the orientation of the particles with spin. The extent of the first dimension is equal to the number of particles. The extent of the second dimension has to be equal to three with orientation(i,:) holding the x-, y-, and z-direction of the i-th particle's orientation. The orientation vector does not have to be normalized.

`REAL radii(3)` (input)

> One dimensional array of extent three holding the radii the three primitives making up a particle with spin: Sphere, cylinder and cone.

`REAL offset` (input, optional)

> Optional one dimensional array of extent two. If present, specifies the center-of-mass offset of cylinder and cone, respectively, with respect to the center-of-mass of the sphere. Otherwise, default values are used. The offset is measured in units of the sphere radius.

Note

Depending on the value of `mode` some of the actual arguments might not be relevant but still have to be supplied.

2.6.4. DVRP_POLYGONS

Sets up a polygon mesh. The mesh consisting of triangles or quads is mapped to an underlying set of vertices.

Synopsis

```
interface
   subroutine DVRP_POLYGONS (stream, indices, n, pcount)
     INTEGER, intent(in) :: stream, indices, n, pcount
   end subroutine DVRP_POLYGONS
end interface
```

Arguments

`INTEGER stream` (input)

> Number of the DVRP Library output stream.

`INTEGER indices` (input)

> Array of polygon (triangle or quad) vertex indices.

`INTEGER n` (input)

> The array establishes a correspondence between the polygon (triangle/quad) vertices and an underlying set of vertices by having each polygonal vertex point to a unique vertex of the set.

`INTEGER pcount` (input)

> Number of polygons.

Description

Each polygon vertex is identified by its vertex number from the set of all vertices. In total the mapping between all vertices and the polygons provides a polygon mesh.

2.7. Visualization of Volume-Data

This section introduces DVRP Library functions related to the visualization of 3D-volume data.

2.7.1. DVRP_DATA

Defines 3D-volume data.

Synopsis

```
interface
   subroutine DVRP_DATA (stream, field, vdim, xdim, ydim, zdim, xnext, ynext, znext)
     INTEGER, intent(in) :: stream, vdim, xdim, ydim, zdim, xnext, ynext, znext
     REAL, intent(in) :: field(:,:,:)
   end subroutine DVRP_DATA
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`REAL field` (input)

Array holding the 3D-volume data at grid points which have been defined by the DVRP_GRID_INIT function.

`INTEGER vdim` (input)

`INTEGER xdim` (input)

Number of data points along the x-axes.

`INTEGER ydim` (input)

Number of data points along the y-axes.

`INTEGER zdim` (input)

Number of data points along the z-axes.

`INTEGER xnext` (input)

> This argument is used by the MPI environment and specifies whether another processor element holds an adjacent grid cell in x-direction. (0 - false, 1 - true)

`INTEGER ynext` (input)

> This argument is used by the MPI environment and specifies whether another processor element holds an adjacent grid cell in y-direction. (0 - false, 1 - true)

`INTEGER znext` (input)

> This argument is used by the MPI environment and specifies whether another processor element holds an adjacent grid cell in z-direction. (0 - false, 1 - true)

Description

This function defines 3D-volume data on a set of rectangular grid points which have been defined previously through a call to DVRP_GRID. The data can thereafter be visualized by the slicers or iso-surfaces.

2.7.2. DVRP_GRID

Specifies a rectangular grid of 3D-coordinates which will be used for the definition of 3D-volume data.

Synopsis

```
interface
   subroutine DVRP_GRID (stream, xsteps, ysteps, zsteps, xcoord, ycoord, zcoord)
     INTEGER, intent(in) :: stream, xsteps, ysteps, zsteps
     REAL, dimension(:), intent(in) :: xcoord, ycoord, zcoord
   end subroutine DVRP_GRID
end interface
```

Arguments

`INTEGER stream` (input)

> Number of the DVRP Library output stream.

`INTEGER xsteps` (input)

> Number of grid points along x-axes.

`INTEGER ysteps` (input)

> Number of grid points along y-axes.

`INTEGER zsteps` (input)

Number of grid points along z-axes.

`REAL xcoord(:)` (input)

One dimensional array of the x-coordinates.

`REAL ycoord` (input)

One dimensional array of the y-coordinates.

`REAL zcoord(:)` (input)

One dimensional array of the z-coordinates.

2.7.3. DVRP_RESULTS

Specifies the 3D-volume data field to be used for the visualization.

Synopsis

```
interface
   subroutine DVRP_RESULTS (stream, results, vcount)
     INTEGER, intent(in) :: stream, vcount
     REAL, intent(in) :: results(:)
   end subroutine DVRP_RESULTS
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`REAL results` (input)

Array holding the 3D-volume data.

`INTEGER vcount` (input)

Number of elements passed. (Extent of the array.)

Description

The 3D-volume data field is connected to a set of vertices. It is assumed that the vertices have been mapped to a polygon mesh through the DVRP_POLYGONS library call. Visualization is performed by coloring the polygons according to the values of the data field along with material and color properties.

2.7.4. DVRP_SLICER

Specifies a plane to be used as slicer for 3D-volume data visualization.

Synopsis

```
interface
   subroutine DVRP_SLICER (stream, axis, position)
     INTEGER, intent(in) :: stream, axis
     REAL, intent(in) :: position
   end subroutine DVRP_SLICER
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`INTEGER axis` (input)

Specifies the orientation of the plane to be used as a slicer. 0: y-z plane, 1: x-z plane, 2: x-y plane

`REAL position` (input)

Specifies the position of the slicer by defining the point of intersection with the coordinate axis perpendicular to the slicer plane.

2.7.5. DVRP_THRESHOLD

Specifies the iso-value used for the iso-surface visualization.

Synopsis

```
interface
   subroutine DVRP_THRESHOLD (stream, value)
     INTEGER, intent(in) :: stream
     REAL, intent(in) :: value
   end subroutine DVRP_THRESHOLD
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`REAL value` (input)

Specifies the iso-value used for the iso-surface visualization.

2.7.6. DVRP_VERTICES

Specifies the array which holds the coordinates of all vertices.

Synopsis

```
interface
   subroutine DVRP_VERTICES (stream, vertices, vcount)
     INTEGER :: stream, vcount
     REAL :: vertices(:,:)
   end subroutine DVRP_VERTICES
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`REAL vertices` (input)

Array holding the 3*N coordinates of all N vertices. Its first dimension has the extent 3.

`INTEGER vcount` (input)

Number of vertices.

2.8. Steering

This section describes the steering facilities, i.e., the way a user can interact with an online calculation.

2.8.1. DVRP_STEERING_INIT

Makes a user-defined variable available for steering. Three overloaded versions of this subprogram exist: One is intended for integer variables, one for single-precision real variables and one for double-precision real variables.

> **Note:** The availability of these routines is platform dependent. Only interfaces for supported floating-point types exist.

Synopsis (a)

```
interface
  subroutine DVRP_STEERING_INIT (stream, vname, from, to, var)
    INTEGER, intent(in) :: stream
    type(cstring), intent(in) :: vname
    INTEGER, intent(in) :: from, to
    INTEGER, intent(inout) :: var
  end subroutine DVRP_STEERING_INIT
end interface
```

Synopsis (b)

```
interface
  subroutine DVRP_STEERING_INIT (stream, vname, from, to, var)
    INTEGER, intent(in) :: stream
    type(cstring), intent(in) :: vname
    real(kind(0.0)), intent(in) :: from, to
    real(kind(0.0)), intent(inout) :: var
  end subroutine DVRP_STEERING_INIT
end interface
```

Synopsis (c)

```
interface
  subroutine DVRP_STEERING_INIT (stream, vname, from, to, var)
    INTEGER, intent(in) :: stream
    type(cstring), intent(in) :: vname
    real(kind(0d0)), intent(in) :: from, to
    real(kind(0d0)), intent(inout) :: var
  end subroutine DVRP_STEERING_INIT
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

`CHARACTER name` (input)

Name of the variable. This name will be used in the steering control window.

`INTEGER/REAL from` (input)

Lower bound of the allowed range for the steering variable.

`INTEGER/REAL to` (input)

Upper bound of the allowed range for the steering variable.

`INTEGER/REAL value` (input)

The variable associated with steering. The current value of `var` will be used as the initial value for this variable.

Description

Depending on the type of variable intended for steering the arguments `from`, `to` and `var` can either all be of INTEGER or all of REAL kind.

2.8.2. DVRP_STEERING_UPDATE

Synchronizes all variables in the simulation program available for steering with the currently selected values via the steering control window.

Synopsis

```
interface
   subroutine DVRP_STEERING_UPDATE (stream)
     INTEGER, intent(in) :: stream
   end subroutine DVRP_STEERING_UPDATE
end interface
```

Arguments

`INTEGER stream` (input)

Number of the DVRP Library output stream.

Appendix A. Fortran Math Model

This section gives details of the valid integer and real kind parameters of the platforms supported by the DVRP Library.

A.1. T3E Kind Specifications

Table A-1. T3E Integer Kind Specifications

	Default	8 Bit	16 Bit	32 Bit	64 Bit
Kind Number	8	1	2	4	8
Digits	63	7	15	31	63
Range	18	2	4	9	18
Huge	9223372036854775807	127	32767	2147483647	9223372036854775807

Table A-2. T3E Real Kind Specifications

	Default
Kind Number	8
Digits	53
Maxexponent	1024
Minexponent	-1021
Precision	15
Radix	2
Range	307
Epsilon	0.22204460E-15
Tiny	0.22250739-307
Huge	0.17976931+309

Note: There is only one real kind on the T3E.

A.2. Intel-x86 Kind Specifications (NAGWare f95)

The compiler is the NAGWare f95 4.x compiler.

Table A-3. Intel-x86 Integer Kind Specifications

	Default	8 Bit	16 Bit	32 Bit	64 Bit
Kind Number	3	1	2	3	4
Digits	31	7	15	31	63
Range	9	2	4	9	18
Huge	2147483647	127	32767	2147483647	9223372036854775807

Table A-4. Intel-x86 Real Kind Specifications

	Default	32 Bit	64 Bit
Kind Number	1	1	2
Digits	24	24	53
Maxexponent	128	128	1024
Minexponent	-125	-125	-1021
Precision	6	6	15
Radix	2	2	2
Range	37	37	307
Epsilon	0.11920929E-06	0.11920929E-06	0.22204460E-15
Tiny	0.11754944E-37	0.11754944E-37	0.22250739-307
Huge	0.34028235E+39	0.34028235E+39	0.17976931+309

A.3. Sun Ultra-Enterprise Kind Specifications

The compiler is the Sun WorkShop 6 Fortran 95.

Table A-5. Sun Ultra-Enterprise Integer Kind Specifications

	Default	8 Bit	16 Bit	32 Bit	64 Bit
Kind Number	4	1	2	4	8
Digits	31	7	15	31	63
Range	9	2	4	9	18
Huge	2147483647	127	32767	2147483647	9223372036854775807

Table A-6. Sun Ultra-Enterprise Real Kind Specifications

	Default	32 Bit	64 Bit	128 Bit
Kind Number	4	4	8	16
Digits	24	24	53	113
Maxexponent	128	128	1024	16384
Minexponent	-125	-125	-1021	-16381
Precision	6	6	15	33
Radix	2	2	2	2
Range	37	37	307	4931
Epsilon	0.11920929E-06	0.11920929E-06	0.22204460E-15	0.19259299E-33
Tiny	0.11754944E-37	0.11754944E-37	0.22250739-307	0.33621031-4931
Huge	0.34028235E+39	0.34028235E+39	0.17976931+309	0.11897315+4933

A.4. HLRN IBM-Regatta Kind Specifications

The compiler is XL Fortran for AIX Version 7 or higher.

Table A-7. HLRN IBM-Regatta Integer Kind Specifications

	Default	8 Bit	16 Bit	32 Bit	64 Bit
Kind Number	4	1	2	4	8
Digits	31	7	15	31	63
Range	9	2	4	9	18
Huge	2147483647	127	32767	2147483647	9223372036854775807

Table A-8. HLRN IBM-Regatta Real Kind Specifications

	Default	32 Bit	64 Bit	128 Bit
Kind Number	4	4	8	16
Digits	24	24	53	113
Maxexponent	128	128	1024	16384
Minexponent	-125	-125	-1021	-16381
Precision	6	6	15	33
Radix	2	2	2	2
Range	37	37	307	4931
Epsilon	0.11920929E-06	0.11920929E-06	0.22204460E-15	0.19259299E-33
Tiny	0.11754944E-37	0.11754944E-37	0.22250739-307	0.33621031-4931

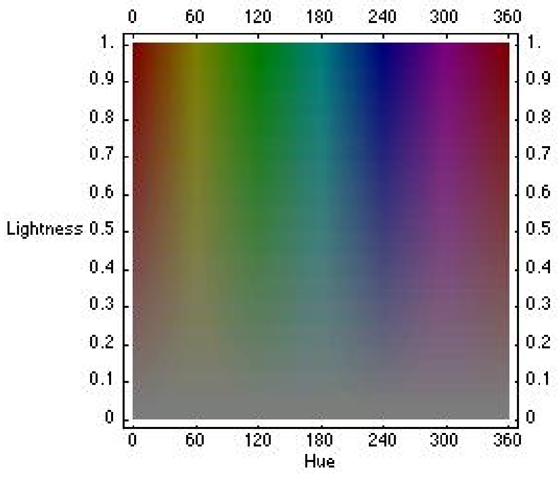
	Default	**32 Bit**	**64 Bit**	**128 Bit**
Huge	0.34028235E+39	0.34028235E+39	0.17976931+309	0.11897315+4933

Appendix B. HLS Color Model

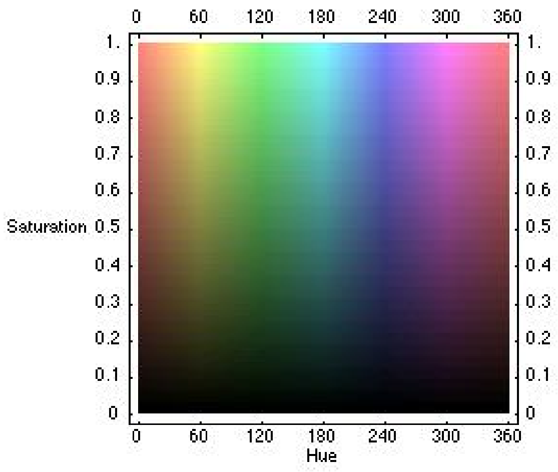
Color is specified by a triplet of hue, lightness and saturation numbers in the HLS color model. The hue value is typically interpreted as an angle coordinate and ranges from 0 degrees to 360 degrees. The values 0 and 360 are identified and correspond to the color red. Generally, colors differing by 180 degrees are complementary colors. The meaning of lightness and saturation values can best be understood from the examples given in Figure B-1.

Further information can be found in [FOLEY].

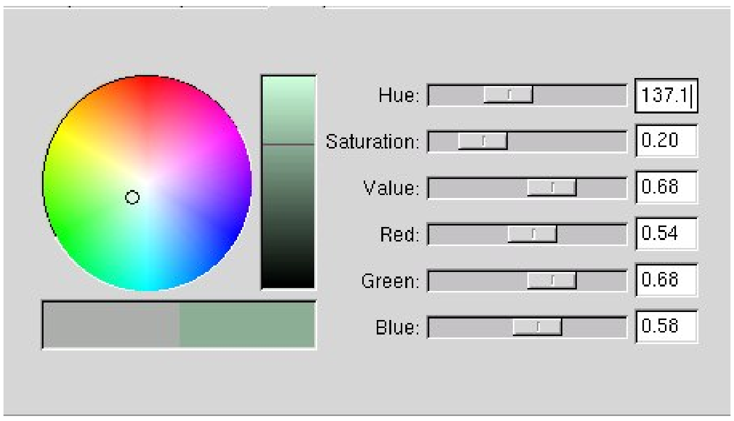
Figure B-1. HLS Color Model



Hue-Lightness for a fixed saturation value (s=0.5).



Hue-Saturation for a fixed saturation value (l=0.5).



Hue: 137.1
Saturation: 0.20
Value: 0.68
Red: 0.54
Green: 0.68
Blue: 0.58

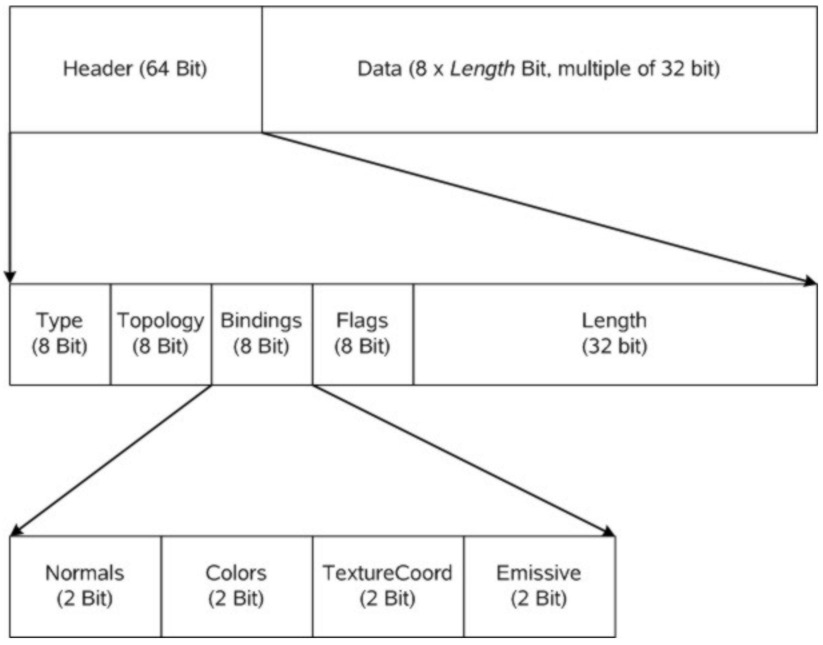
Color manipulation in the GIMP program.

Tip: See also sample program 7 which gives an impression of the three dimensional color space of HLS values by animating a surface with varying color values.

Appendix C. DVR Binary Format Specification

The detailed specification of the DVR binary format is given.

Figure C-1. DVR Binary Format Specification



Structure of the DVR Data Block

References

J.D. Foley, A. van Dam, and S.K. Feiner, *Computer Graphics—Principles and Practice*, Addison-Wesley, 2nd edition, 1996.

S. Olbrich, H. Pralle, and S. Raasch, *Efficient volume visualization using parallelization and 3D streaming techniques in a high performance computing and networking scenario*, RRZN Universität Hannover, 2000.

S. Olbrich and H. Pralle, *Multimedia- und Virtual-Reality-Technologie für die wissenschafliche Visualisierung* , RRZN Universität Hannover, Edited by G. Buziek et al, 2000.

[Stephan Olbrich, 2000] S. Olbrich, *Ein leistungsfähiges System zur Online-Präsentation von Sequenzen komplexer virtueller 3D-Szenen* , RRZN Universität Hannover, 2000.

Alphabetical Listing of DVRP Library Functions

DVRP_BOUNDINGBOX
DVRP_CAMERA
DVRP_CONE
DVRP_CYLINDER
DVRP_COLORTABLE_HLS
DVRP_COLORTABLE_RGB
DVRP_DATA
DVRP_EXIT
DVRP_GRID
DVRP_INIT
DVRP_LOG
DVRP_LOG_DISABLE
DVRP_LOG_ENABLE
DVRP_LOG_EVENT
DVRP_LOG_EXIT
DVRP_LOG_INIT
DVRP_LOG_SYMBOL
DVRP_MATERIAL
DVRP_MATERIAL_HLS
DVRP_MATERIAL_RGB
DVRP_OUTPUT_FTP
DVRP_OUTPUT_LOCAL
DVRP_OUTPUT_RTSP
DVRP_PARTICLES
DVRP_POLYGONS
DVRP_RESULTS
DVRP_SLICER
DVRP_STEERING_INIT
DVRP_STEERING_UPDATE
DVRP_THRESHOLD
DVRP_VERBOSE
DVRP_VERTICES
DVRP_VISUALIZE