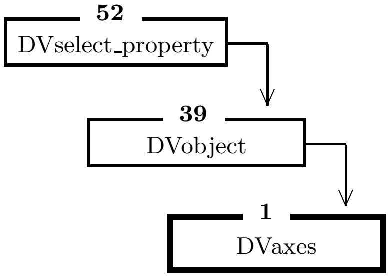
Contents

1

class **DVaxes** : public DVobject

A class for creating axes for use in DataViewer scenes.

Inheritance



Public Members

Constructor. 8

1.9 **DVaxes** (LCVMarray_2D<float> coords)
Constructor. 8

1.10 void **reset_data** (LCVMarray<float> center,
LCVMarray<float> x_dir,
LCVMarray<float> y_dir,
LCVMarray<float> z_dir)
Change the position and orientation of the axes. 9

virtual **~DVaxes** () *The destructor*

virtual void
draw (LCVMstack<DVproperty*>&)
The draw routine.

virtual DVobject*
copy () *We also want a routine that returns a pointer to a copy of the object created with new.*

DVaxes (const DVaxes &data)
A copy constructor

const DVaxes&
operator= (const DVaxes &data)
An assignment operator

DVaxes (FileOption a_file_op)
constructor from the global file with a file format option

Protected Members

LCVMarray_2D<float>
points

void **set_data** ()

void **set_data** (float x, float y, float z)

void **set_data** (float *center)

void **set_data** (float x, float y, float z, float d1x,
float d1y, float d1z, float d2x, float d2y,
float d2z, float d3x, float d3y, float d3z)

void **set_data** (float *center, float *directions)

void **set_data** (LCVMarray<float> center)

void **set_data** (LCVMarray<float> center,
LCVMarray<float> x_dir,
LCVMarray<float> y_dir,
LCVMarray<float> z_dir)

void **set_data** (LCVMarray<float> center,
LCVMarray_2D<float> directions)

void **set_data** (LCVMarray_2D<float> coords)

void **set_data** (const DVaxes &data)

A class for creating axes for use in DataViewer scenes.

1.1

DVaxes ()

Default constructor.

Default constructor. Axes will be centered at (0,0,0)

1.2

DVaxes (float x, float y, float z)

Constructor.

Constructor. Axes will be centered at (x,y,z)

1.3

DVaxes (float *center)

Constructor.

Constructor. Axes will be centered at (center[0],center[1],center[2])

1.4

DVaxes (float x, float y, float z, float d1x, float d1y, float d1z, float d2x, float d2y, float d2z, float d3x, float d3y, float d3z)

Constructor.

Constructor. Axes will be centered at (x,y,z) and will have directors d1,d2, and d3.

1.5

DVaxes (float *center, float *directions)

Constructor.

Constructor. Axes will be centered at (x,y,z) and will have directors d1=(directions[0],...), d2=(directions[3],...), and d3=(directions[6],...).

1.6

DVaxes (LCVMarray<float> center)

Constructor.

Constructor. Axes will be centered at (center[0],center[1],center[2])

1.7

DVaxes (LCVMarray<float> center, LCVMarray<float> x_dir, LCVMarray<float> y_dir, LCVMarray<float> z_dir)

Constructor.

Constructor. Axes will be centered at (x,y,z) and will have directors x_dir,y_dir, and z_dir.

1.8

DVaxes (LCVMarray<float> center, LCVMarray_2D<float> directions)

Constructor.

Constructor. Axes will be centered at (x,y,z) and will have directors d1=(directions[0][0],...), d2=(directions[1][0],...), and d3=(directions[2][0],...).

1.9

DVaxes (LCVMarray_2D<float> coords)

Constructor.

Constructor. Axes will be centered at (0,0,0) and will have directors d1=(coords[0][0],...), d2=(directions[1][0],...), and d3=(coords[2][0],...).

1.10

void **reset_data** (LCVMarray<float> center, LCVMarray<float> x_dir, LCVMarray<float> y_dir, LCVMarray<float> z_dir)

Change the position and orientation of the axes.

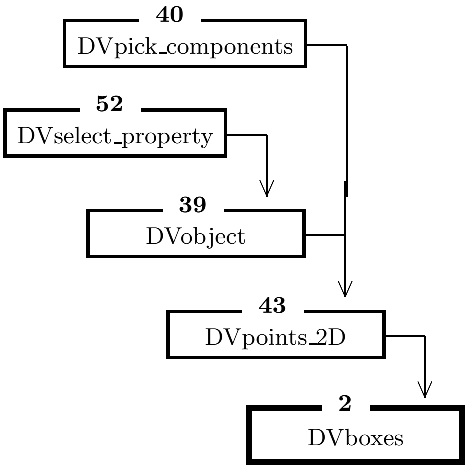
Change the position and orientation of the axes.

2

class **DVboxes** : public DVpoints_2D

A class for creating boxes

Inheritance



Public Members

n strings of m points in one 3D array m can be different for each string. 13

virtual **~DVboxes** () *Destructor*

DVboxes& **set_data** () *Set data routines*

DVboxes& **set_data** (const float x0, const float y0, const float z0, const float x1, const float y1, const float z1)
2 points given by 3 float values

2.5 DVboxes& **set_data** (LCVMarray<float> &point0, LCVMarray<float> &point1)
2 points given in 2 arrays. 13

2.6 DVboxes& **set_data** (LCVMarray_2D<float> &points)
n points given in one 2D LCVMarray. 13

2.7 DVboxes& **set_data** (LCVMarray_3D<float> &coords)
n strings of m points in one 3D array m can be different for each string. 13

virtual void
draw (LCVMstack<DVproperty*>&)
draw routine

DVboxes (const DVboxes &data)
We also want a copy constructor as well as an assignment operator

const DVboxes&
operator= (const DVboxes &data)

virtual void
print (ostream&) *a virtual print function*

DVboxes (FileOption a_file_op)
Constructor from a file

2.8 virtual int**ReadMyProp** () *Specific properties in the scene file.* 14

Protected Members

2.9 virtual DVobject*

A class for creating boxes

2.1

DVboxes ()

Default box from 0, 0, 0 to 1, 1, 1

Default box from 0, 0, 0 to 1, 1, 1

2.2

DVboxes (LCVMarray<float> &point0, LCVMarray<float> &point1)

2 points given in 2 arrays The two points must have the same number of coordinates.

2 points given in 2 arrays The two points must have the same number of coordinates.

2.3

DVboxes (LCVMarray_2D<float> &points)

n points given in one 2D LCVMarray.

n points given in one 2D LCVMarray. A bos is drawn between each point in the array and the next one.

2.4

DVboxes (LCVMarray_3D<float> &coords)

n strings of m points in one 3D array m can be different for each string.

n strings of m points in one 3D array m can be different for each string. Each string will be drawn as in the LCVMarray_2D case.

2.5

DVboxes& **set_data** (LCVMarray<float> &point0, LCVMarray<float> &point1)

2 points given in 2 arrays.

2 points given in 2 arrays. The two points must have the same number of coordinates.

2.6

DVboxes& **set_data** (LCVMarray_2D<float> &points)

n points given in one 2D LCVMarray.

n points given in one 2D LCVMarray. A bos is drawn between each point in the array and the next one.

2.7

DVboxes& **set_data** (LCVMarray_3D<float> &coords)

n strings of m points in one 3D array m can be different for each string.

n strings of m points in one 3D array m can be different for each string. Each string will be drawn as in the LCVMarray_2D case.

2.8

virtual int **ReadMyProp** ()

Specific properties in the scene file.

Specific properties in the scene file. Must be specified even if there is none.

2.9

virtual DVobject* **copy** ()

We also want a routine that returns a pointer to a copy of the object created with new

We also want a routine that returns a pointer to a copy of the object created with new

3 class **DVcolor**

A class which encapsulates a color.

Public Members

DVcolor_float **R** — *The red value.*

DVcolor_float **G** — *The green value.*

DVcolor_float **B** — *The blue value.*

DVcolor_float **A** — *The transparency value.*

DVcolor (const DVcolor_float R_val=1.0,
const DVcolor_float G_val=1.0,
const DVcolor_float B_val=1.0,
const DVcolor_float A_val=1.0)
Constructor.

~DVcolor () *Distructor*

DVcolor (const DVcolor&)
Copy constructor

const DVcolor&
operator= (const DVcolor&)
Assignment operator

int **operator==** (const DVcolor&) const
Check equality

int **operator!=** (const DVcolor&) const
Check inequality

void **set_data** (const DVcolor_float R_val=1.0,
const DVcolor_float G_val=1.0,
const DVcolor_float B_val=1.0,
const DVcolor_float A_val=1.0)
Set_data routine to match contructor

void **set_data** (const DVcolor&)

void **set_data** (const DVcolor_float*) void **get_data** (DVcolor_float&, DVcolor_float&, DVcolor_float&, DVcolor_float&) const

void **get_data** (DVcolor_float*)

void **get_data** (DVcolor&) const

DVcolor_float&
operator[] (const int)
We want foo[0] to be foo.R, ... foo[3] to be foo.A

DVcolor (FileOption)
Methods concrening the reading from a file

DVcolor (FileOption, int &)
Methods concrening the reading from a file

void **print** (ostream &, FileOption)
Methods concrening the reading from a file

void **print** (ostream &) const
Methods concrening the reading from a file

inline **DVcolor** (const DVcolor_float R_val, const DVcolor_float G_val, const DVcolor_float B_val, const DVcolor_float A_val)
constructor

inline **DVcolor** (const DVcolor &data)
copy constructor

inline void
get_data (DVcolor_float &R_val, DVcolor_float &G_val, DVcolor_float &B_val, DVcolor_float &A_val) const

inline void
get_data (DVcolor_float *rgba_vals)

inline void
get_data (DVcolor &r) const

inline int **operator!=** (const DVcolor &data) const
check not equal

inline const DVcolor&

		operator= (const DVcolor &data)	*assignment operator*
inline	int	**operator==** (const DVcolor &data) const	*check equal*
inline	void	**set_data** (const DVcolor_float R_val, const DVcolor_float G_val, const DVcolor_float B_val, const DVcolor_float A_val)	*set_data routine*
inline	void	**set_data** (const DVcolor_float *rgba_vals)	
inline	void	**set_data** (const DVcolor &pt)	
inline		**~DVcolor** ()	*destructor (does nothing)*

A class which encapsulates a color.

4 class **DVcolor_by_data**

A class which allows an object to be colored based upon a color map and values of its data

Public Members

		DVcolor_by_data ()	*Default constructor*	
		~DVcolor_by_data ()	*Destructor*	
		DVcolor_by_data (const DVcolor_by_data &data)	*Copy constructor*	
	const DVcolor_by_data&	**operator=** (const DVcolor_by_data &data)	*Assignement operator*	
	int	**operator==** (const DVcolor_by_data &data) const	*Check for equality*	
	int	**operator!=** (const DVcolor_by_data &data) const	*Check for inequality*	
	int	**min_set** () const	*Check to see is the minimum has been explicitly set*	
	int	**max_set** () const	*Check to see is the maximum has been explicitly set*	
	void	**set_min** (const float)	*Explicitly set the minimum*	
	void	**set_max** (const float)	*Explicitly set the maximum*	
4.2	void	**unset_min** ()	*Unset the minimum.*	19
4.3	void	**unset_max** ()	*Unset the maximum.*	20
	void	**set_color_map_function** (color_map_function_pointer)	*Set the colormap function to use to change data values into RGB values.*	
	void	**set_color_map** (DVcolor_map &)		

Explicitly set the colormap

void **print** (ostream &) const
Print function

void **get_color** (const float, DVcolor&)
Given a floating point value return its color.

void **get_color** (const float, const float, const float, DVcolor&)
Given a minimum, maximum, and value, return its color.

Protected Members

float **min**

float **max**

int **min_set_flag** *1 if min has been explicitly set*

int **max_set_flag** *1 if max has been explicitly set*

DVcolor_map* **color_map**

A class which allows an object to be colored based upon a color map and values of its data

4.2

void **unset_min** ()

Unset the minimum.

Unset the minimum. This allows the minimum to computed from the data

4.3

void **unset_max** ()

Unset the maximum.

Unset the maximum. This allows the maximum to computed from the data

4.1

color_map_function_pointer **color_map_function**

Ppointer to function to call to compute the R,G,B values– Function needs to take as arguments: min, max, val, DVcolor

Ppointer to function to call to compute the R,G,B values– Function needs to take as arguments: min, max, val, DVcolor

5 class **DVcolor_map**

A class which maps values to colors

Public Members

Protected Members

LCVMarray<float>
points

LCVMarray<DVcolor>
colors

A class which maps values to colors

5.1

DVcolor_map (LCVMarray<float> &, LCVMarray<DVcolor> &)

Constructor with an array of points and colors.

Constructor with an array of points and colors. The color map will interpolate between the given colors based upon the values given in the float array.

5.2

DVcolor_map (DVcolor &, DVcolor &, DVcolor &)

Constructor with 3 colors.

Constructor with 3 colors. The color map will interpolate between the given colors.

5.3

DVcolor_map (LCVMarray<DVcolor> &)

Constructor with an array of colors.

Constructor with an array of colors. The color map will interpolate between the given colors using even spacing.

5.4

DVcolor_map (DVcolor &, DVcolor &)

Constructor with 2 colors.

Constructor with 2 colors. The color map will interpolate between then two given colors.

5.5

void **set_data** (DVcolor &, DVcolor &)

Set_data with 2 colors The color map will interpolate between then two given colors

Set_data with 2 colors The color map will interpolate between then two given colors

5.6

void **set_data** (DVcolor &, DVcolor &, DVcolor &)

Set_data with 3 colors The color map will interpolate between the given colors

Set_data with 3 colors The color map will interpolate between the given colors

5.7

void **set_data** (LCVMarray<float> &, LCVMarray<DVcolor> &)

Set_data with an array of points and colors The color map will interpolate between the given colors based upon the values given in the float array

Set_data with an array of points and colors The color map will interpolate between the given colors based upon the values given in the float array

5.8

void **set_data** (LCVMarray<DVcolor> &)

Set_data with an array of colors The color map will interpolate between the given colors using even spacing

Set_data with an array of colors The color map will interpolate between the given colors using even spacing

5.9

void **get_color** (const float, const float, const float, DVcolor&)

Get the color which corresponds to the given min, max, and value

Get the color which corresponds to the given min, max, and value

6

void **color_map_standard** (const float min, const float max, const float val, DVcolor &color)

A standard "rainbow" blue to red colormap

A standard "rainbow" blue to red colormap

7

void **color_map_reverse_standard** (const float min, const float max, const float val, DVcolor &color)

A standard "rainbow" colormap except it goes from red to blue

A standard "rainbow" colormap except it goes from red to blue

8

void **color_map_gray_scale** (const float min, const float max, const float val, DVcolor &color)

A colormap which runs from black to white

A colormap which runs from black to white

9

void **color_map_reverse_gray_scale** (const float min, const float max, const float val, DVcolor &color)

A colormap which runs from white to black

A colormap which runs from white to black

10

void **color_map_discrete** (const float min, const float max, const float val, DVcolor &color)

A colormap with discrete bands of color

A colormap with discrete bands of color

11

void **old_markers_color_map** (const float min, const float max, const float val, DVcolor &color)

A colormap used for old style markers, this should go away

A colormap used for old style markers, this should go away

12

float **ABS** (const float a)

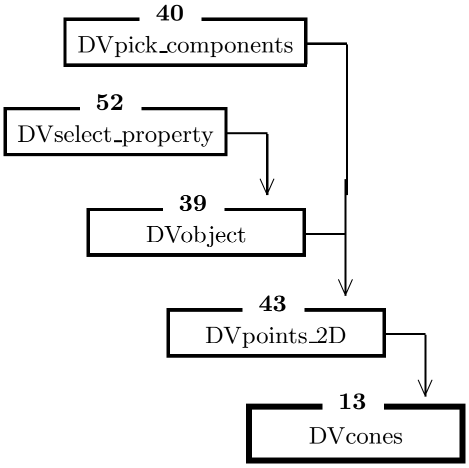
A convenience routine for computing absolute values.

13

class **DVcones** : public DVpoints_2D

A class for drawing cones

Inheritance



Public Members

13.2 **DVcones** (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0, const float r1=1.0, const float r2=1.0)
A simple cone, with two extremities and two radius values given. 35

13.3 **DVcones** (LCVMarray_2D<float> &coords)
The coordinates of the points for one string of cones are stored in the array coords. 36

13.4 **DVcones** (LCVMarray_3D<float> &coords)
The coordinates of the points for several strings of cones are in the array coords. 36

virtual **~DVcones** () *destructor*

13.5 DVcones& **set_data** (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0, const float r1=1.0, const float r2=1.0)
A simple cone, with two extremities and two radius values given. 36

13.6 DVcones& **set_data** (LCVMarray_2D<float> &coords)
The coordinates of the points for one string of cones are stored in the array coords. 37

13.7 DVcones& **set_data** (LCVMarray_3D<float> &coords)
The coordinates of the points for several strings of cones are in the array coords. 37

13.8 void **get_used_radius** (LCVMarray_2D<float> &, DVproperty *)
It retrieves an array of values for the radius in each point, using the radius_coord variable and the scaling from the DVproperty. 37

virtual void
draw (LCVMstack<DVproperty*>&)
draw routine

DVcones (const DVcones &data)
We also want a copy constructor as well as an assignment operator

const DVcones&
operator= (const DVcones &data)

int **operator==** (const DVcones &data)
check for equality and not equal

int **operator!=** (const DVcones &data)

DVcones& **set_radius_coord** (const int)
Set the coordinate for the radius

int **get_radius_coord** ()
Get the coordinate for the radius

virtual void
print (ostream&) *a virtual print function*

friend ostream&

		operator<< (ostream &, DVcones &)	*A friend routine to overload the* **<<** *operator*	
		DVcones (FileOption a_file_op)	*Constructor from a scene file.*	
	DVcones&	**set_data** (FileOption a_file_op)	*Set the data from a scene file.*	
13.9	virtual int	**ReadMyProp** ()	*Specific properties in the scene file.*	38

Protected Members

	int	**radius_coord**	*The coordinate that will be the radius of the cones.*	
13.1	virtual DVobject*	**copy** ()	*We also want a routine that returns a pointer to a copy of the object created with new*	38

A class for drawing cones

13.2

DVcones (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0, const float r1=1.0, const float r2=1.0)

A simple cone, with two extremities and two radius values given.

A simple cone, with two extremities and two radius values given. The line between the two points is the center line of the cone. By default it is a cylinder of radius 1.0, starting at the origin, parallel to the x-axis and of length one.

13.3

DVcones (LCVMarray_2D<float> &coords)

The coordinates of the points for one string of cones are stored in the array coords.

The coordinates of the points for one string of cones are stored in the array coords. A cone will be drawn (see the description above) between each point in the string and the next one.

13.4

DVcones (LCVMarray_3D<float> &coords)

The coordinates of the points for several strings of cones are in the array coords.

The coordinates of the points for several strings of cones are in the array coords. The strings may not have the same length. Each string of cones is drawn as before.

13.5

DVcones& **set_data** (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0, const float r1=1.0, const float r2=1.0)

A simple cone, with two extremities and two radius values given.

A simple cone, with two extremities and two radius values given. The line between the two points is the center line of the cone. By default it is a cylinder of radius 1.0, starting at the origin, parallel to the x-axis and of length one.

13.6

DVcones& **set_data** (LCVMarray_2D<float> &coords)

The coordinates of the points for one string of cones are stored in the array coords.

The coordinates of the points for one string of cones are stored in the array coords. A cone will be drawn (see the description above) between each point in the string and the next one.

13.7

DVcones& **set_data** (LCVMarray_3D<float> &coords)

The coordinates of the points for several strings of cones are in the array coords.

The coordinates of the points for several strings of cones are in the array coords. The strings may not have the same length. Each string of cones is drawn as before.

13.8

void **get_used_radius** (LCVMarray_2D<float> &, DVproperty *)

It retrieves an array of values for the radius in each point, using the radius_coord variable and the scaling from the DVproperty.

It retrieves an array of values for the radius in each point, using the radius_coord variable and the scaling from the DVproperty.

13.9

virtual int **ReadMyProp** ()

Specific properties in the scene file.

Specific properties in the scene file. For this class, the radius coordinate.

13.1

virtual DVobject* **copy** ()

We also want a routine that returns a pointer to a copy of the object created with new

We also want a routine that returns a pointer to a copy of the object created with new

14

typedef DVcontainerbase **DVcontainer**

After moving the select_properties to be a base class for DVobject instead of a base class for this class, there was no difference between this class and DVcontainerbase

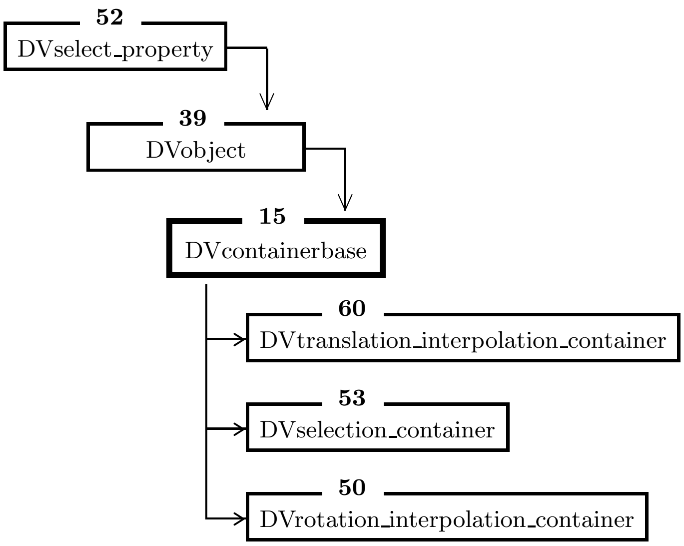
After moving the select_properties to be a base class for DVobject instead of a base class for this class, there was no difference between this class and DVcontainerbase

15

class **DVcontainerbase** : public DVobject

Containers group children together which share properties.

Inheritance



Public Members

DVcontainerbase ()
Constructors for this class

15.1 virtual **~DVcontainerbase** ()
Default destructor. 42

15.2 DVobject& **copy_child** (DVobject &child)
Copy a DVobject into the list of children for this container 42

15.3 DVobject& **delete_current_child** ()
Delete the "current" child from the list (current child is always the last child in the list or null) 42

15.4 DVobject* **operator**[] (const int) *this lets us refer to children by number.* 42

int **get_size** () const *return the number of children the container has*

15.5 virtual void
draw (LCVMstack<DVproperty*>&)
The draw routine. 43

DVcontainerbase (DVcontainerbase &data)

const DVcontainerbase&
operator= (DVcontainerbase &data)

virtual void
print (ostream &) *print the containerbase – make it virtual.*

void **erase** () *erase all children*

friend ostream&
operator<< (ostream &, DVcontainerbase &)
A friend routine to overload the << *operator*

DVcontainerbase (FileOption a_file_op)
A constructor for the scene file parse.

void **ReadNode** () *A helper routine for the scene file parse.*

Protected Members

virtual DVobject*
copy () *A routine that returns a pointer to a copy of the object created with new.*

LCVMdoubly_linked_list<DVobject *>
list *The list of children*

Containers group children together which share properties. The various container classed group together DVobjects in a hierarchy. Lower levels of the hierarchy inherit properties from higher levels of the hierarchy.

15.1

virtual **~DVcontainerbase** ()

Default destructor.

Default destructor. It is declared virtual so that the correct decendant destructor will get called.

15.2

DVobject& **copy_child** (DVobject &child)

Copy a DVobject into the list of children for this container

Copy a DVobject into the list of children for this container

15.3

DVobject& **delete_current_child** ()

Delete the "current" child from the list (current child is always the last child in the list or null)

Delete the "current" child from the list (current child is always the last child in the list or null)

15.4

DVobject* **operator**[] (const int)

this lets us refer to children by number.

this lets us refer to children by number. i.e. foo[n] gives the nth element from the front of the list if n > 0, and the -nth element from the back of the list if n < 0.

15.5

virtual void **draw** (LCVMstack<DVproperty*>&)

The draw routine.

The draw routine. We again want it to be virtual so that the correct decendant routine gets called.

16 class **DVcreate_image_arguments**

This class is used as the argument to the function which creates the scene graph.

Public Members

This class is used as the argument to the function which creates the scene graph. All of the data are references so the uses of this class are limited.

16.1

DVtips_and_hints& **dvhints**

Various hints for the DVgeometry object to aid in its rendering.

Various hints for the DVgeometry object to aid in its rendering. Entries here include background color and animation parameters.

16.2

DVnamed_properties& **named_properties**

The "named_properties".

The "named_properties". NOTE: This is not commonly used.

16.3

DVgeometry& **scene**

A reference to the DVgeometry which called the function

A reference to the DVgeometry which called the function

16.4

DVcreate_image_arguments (LCVMstring&, LCVMstring&, DVcontainer&, DVtips_and_hints&, DVnamed_properties&, DVgeometry&)

Constructor

Constructor

17

void **normalize** (LCVMarray<float>&)

This function normalizes a vector of floats–we make it do any size vector since it is easy

This function normalizes a vector of floats–we make it do any size vector since it is easy

18

void **normcrossprod** (LCVMarray<float>&, LCVMarray<float>&, LCVMarray<float>&)

This function computes the cross product of two vectors.

This function computes the cross product of two vectors. It is assumed that all three vectors have length three (no checking is done).

19

void **compute_normal_vector** (LCVMarray<float>&,
LCVMarray<float>&,
LCVMarray<float>&,
LCVMarray<float>&)

This function computes the normal to the plane determined by the three points v0,v1,v2 (3 vectors).

This function computes the normal to the plane determined by the three points v0,v1,v2 (3 vectors). I.e. it computes the cross product of of (v0-v1) and (v1-v2). It is assumed that all four vectors have length three (no checking is done).

20

float **norm** (LCVMarray<float>&)

This function computes the norm of a vector of floats–we make it do any size vector since it is easy

This function computes the norm of a vector of floats–we make it do any size vector since it is easy

21

float **scalarprod** (LCVMarray<float>&, LCVMarray<float>&)

This function computes the scalar product of two vectors–we make it do any size vector since it is easy

This function computes the scalar product of two vectors–we make it do any size vector since it is easy

22

enum **DVframed_lines_frame_status**

This defines whether the frame is drawn or not

This defines whether the frame is drawn or not

23

enum **DVribbon_form_status**

This defines whether the ribbon is drawn as a solid surface or as a collection of vectors

This defines whether the ribbon is drawn as a solid surface or as a collection of vectors

24

enum **DVribbon_mirror_status**

This defines whether the ribbon is drawn one only one side of the curve or on both sized of the curve

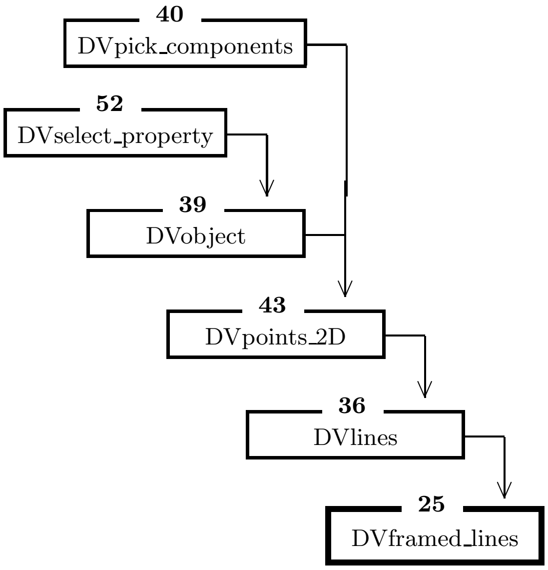
This defines whether the ribbon is drawn one only one side of the curve or on both sized of the curve

25

class **DVframed_lines** : public DVlines

A class for rendering curves in space.

Inheritance



Public Members

25.2 **DVframed_lines** (const float x0=0.0,
const float y0=0.0,
const float z0=0.0,
const float x1=1.0,
const float y1=0.0,
const float z1=0.0)

Draws one ribbon as a vector field.

DVframed_lines (DVframed_lines &data)
We also want a copy constructor as well as an assignment operator

int **copy_ribbons** (DVframed_lines &data)
Copies the list of ribbons.

const DVframed_lines& **operator=** (DVframed_lines &data)
Assignment operator

int **operator==** (DVframed_lines &data)
check for equality

int **operator!=** (DVframed_lines &data)
Check for not equality

DVframed_lines& **set_frame_status** (const DVframed_lines_frame_status)
Set the frame status

DVframed_lines& **set_ribbon_form_status** (const DVribbon_form_status)
Choose between surface or vector field ribbon.

DVframed_lines& **set_ribbon_mirror_status** (const DVribbon_mirror_status)
Choose between one sided and mirrored ribbon.

int **add_ribbon** (int, int, int)
Adds a ribbon to the list, x, y and z coordinates given.

int **add_ribbon** (int, int, int, int)
Adds a ribbon to the list, x, y and z and color coordinates given.

int **add_ribbon** (LCVMarray<int> *new_ribbon)
Adds a ribbon from an array of coordinates.

int **add_ribbon** (LCVMarray<int> &)

Adds a ribbon from an array of coordinates

int **set_ribbon** (int, int, int, int)
Sets the current ribbon coordinates

int **set_ribbon** (LCVMarray<int> &)
Sets the current ribbon coordinates

25.7 void **default_frame_coords** ()
Sets the default frame with the coordinates: 3 to 9 for the x, y, z coordinates, 10 to 12 for the color

62

void **set_frame_coords** (int d1x, int d1y, int d1z, int d2x, int d2y, int d2z, int d3x, int d3y, int d3z)
Set the frame coordinates, color not given.

void **set_frame_coords** (int d1x, int d1y, int d1z, int d2x, int d2y, int d2z, int d3x, int d3y, int d3z, int color1, int color2, int color3)
Set the frame coordinates, including the color.

void **set_frame_coords** (LCVMarray_2D<int> &coords)
Set the frame coordinates from an array.

DVframed_lines_frame_status
get_frame_status ()
Get the frame status

int **get_ribbon_number** ()
Get the number of ribbons in the list.

void **set_coord_names** (LCVMarray<LCVMstring> &the_names)
Set the coordinates names from an LCVMarray, no size needed.

void **set_coord_names** (char **the_names, int coord_nr)

Set the coordinates names from a usual array, its size is needed.

LCVMstring **get_coord_name** (int coord_nr)
Get the name of the coordinate specified by a number.

const char* **get_char_coord_name** (int coord_nr)
Get the name of the coordinate specified by a number.

int **get_coords_number** (LCVMstring &a_name)
Get the number of the coordinate with the given name.

int **get_coords_number** (char *a_name)
Get the number of the coordinate with the given name.

virtual void **print** (ostream&) *a virtual print function*

friend ostream& **operator<<** (ostream &, DVframed_lines &)
A friend routine to overload the << *operator*

DVframed_lines (FileOption a_file_op)
constructor from the global file with a file format option

void **ReadVBMoptions** ()
Read the options generated by VBM and in the VBM style

void **ReadRibbonDef** ()
Read the ribbon coordinates in the VBM style

void **ReadSomeCoords** (int *where, const int n, const int def_val = 0)
Read the coordinates in the VBM style

void **ReadDefaults** () *Read the default coordinates in the VBM style*

void **ReadCoordDef** ()
Read the default coordinates in the VBM style, no color given

void **ReadFrameDef** ()

			Read the frame in the VBM style	
	void	**ReadFrameColorDef** ()	*Read the frame plus its color in the VBM style*	
25.8	virtual int	**ReadMyProp** ()	*Specific properties for the scene file format.*	63

Protected Members

	DVframed_lines_frame_status	**frame_status**	*Tells if the frame is drawn or not.*	
	int	**ribbon_number**	*The number of ribbons to be drawn.*	
	LCVMarray_2D<int>	**frame_coords**	*The coordinates of the frame.*	
	LCVMarray<LCVMstring>	**coord_names**	*The names of all used coordinates.*	
	LCVMdoubly_linked_list<LCVMarray<int> *>	**ribbons**	*A list of coordinate vectors (3) for the ribbons*	
	DVribbon_form_status	**form_status**	*Tells if the ribbon is rendered as a surface or as a vector.*	
	DVribbon_mirror_status	**mirror_status**	*Tells if the ribbon is on one or on two sides of the line.*	
25.1	virtual DVobject*	**copy** ()	*We also want a routine that returns a pointer to a copy of the object created with new*	63

A class for rendering curves in space. It can render a group of space curves, using either lines or cylinders. Compared to the DVlines class from which it is derived, this class can place several director fields or ribbons along the center line. The ribbons can be either vectors or surfaces, and on one or both sides of the line.

25.2

DVframed_lines (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0)

the end points of the line segment are given default line starts at the origin, proceeds along the positive x-axis and has length one.

the end points of the line segment are given default line starts at the origin, proceeds along the positive x-axis and has length one. the end points of the line segment are given.

25.3

DVframed_lines (const float x0, const float y0, const float z0, const float d0, const float d1, const float d2, const float x1, const float y1, const float z1, const float e0, const float e1, const float e2)

for giving a ribbon segment.

for giving a ribbon segment. the end points of the ribbon segment are given–3 coordinates for the end point and 3 coordinates for the direction.

25.4

DVframed_lines (LCVMarray_2D<float> &coords)

The coordinates of the points for one line are stored in the array coords.

The coordinates of the points for one line are stored in the array coords. An OpenGL line is drawn between each point and the next in the array.

25.5

DVframed_lines (LCVMarray_3D<float> &coords)

The coordinates of the points for several lines are in the array coords.

The coordinates of the points for several lines are in the array coords. Each line is drawn as before, and can have its own length.

25.6

void **set_defaults** ()

Set what has to be rendered by default: no frame, no ribbon, mirrored ribbon, and surface ribbon

Set what has to be rendered by default: no frame, no ribbon, mirrored ribbon, and surface ribbon

25.7

void **default_frame_coords** ()

Sets the default frame with the coordinates: 3 to 9 for the x,y,z coordinates, 10 to 12 for the color

Sets the default frame with the coordinates: 3 to 9 for the x,y,z coordinates, 10 to 12 for the color

25.8

virtual int **ReadMyProp** ()

Specific properties for the scene file format.

Specific properties for the scene file format. For this class: the frame coordinates, any number of ribbon coordinates, one-sided or mirrored ribbon, vector field or surface ribbon.

25.1

virtual DVobject* **copy** ()

We also want a routine that returns a pointer to a copy of the object created with new

We also want a routine that returns a pointer to a copy of the object created with new

26

int **create_image_function** (DVcreate_image_arguments &dv)

Here is the prototype for the user defined "func" routine.

Here is the prototype for the user defined "func" routine. This is what actually defines what gets drawn! It should return 0 on failure and non-0 on success.

27

enum **DVexport_state_type**

enum for the various export options

28

enum **DVrecord_state_type**

enum for the various record states

29 class **DVgeometry**

A toplevel object which provided services such as animation and display list creation.

Public Members

		set_animation_frames (const int, const int, const int)	*Set the animation first frame, last frame and step sizes.*	76
	int	**get_animation_first_frame_number** ()	*get the animation first frame*	
	int	**get_animation_last_frame_number** ()	*get the animation last frame*	
	int	**get_animation_step_frame_number** ()	*get the animation step size*	
	DVgeometry&	**set_current_frame_number** (const int)	*set the current frame*	
	int	**get_current_frame_number** ()	*get the current frame number*	
	DVgeometry&	**set_frames_per_second** (const float)	*get and set the frame rate*	
	float	**get_frames_per_second** () const		
	int	**compute_animation_size** ()	*determine the size of the animation*	
29.13	int	**valid_draw_frame** (const int frame_num) const	*given a frame number, it returns the frame number closest to it that we are allowed to display*	76
	int	**create_display_lists** ()	*generate display lists*	
	void	**animate_run_forward** ()	*run the animation forward starting from the beginning*	
	void	**animate_run_backward** ()	*run the animation backward starting from the end*	
	void	**animate_play_forward** ()	*run the animation forward starting from the current frame*	
	void	**animate_play_backward** ()		

run the animation backward starting from the current frame

friend void **continuous_animate_next** (ClientData)
continuously animate frames going forward

friend void **continuous_animate_next** (XtPointer, XtIntervalId *)
continuously animate frames going forward

friend void **continuous_animate_previous** (ClientData)
continuously animate frames going backward

friend void **continuous_animate_previous** (XtPointer, XtIntervalId *)
continuously animate frames going backward

int **animate_next** ()
display the next frame–return zero if there is no next frame

int **animate_previous** ()
display the previous frame–return zero if there is no previous frame

void **animation_to_start** ()
display the first frame of the animation set

void **animation_to_end** ()
display the last frame of the animation set

void **animation_to_frame** (const int frame_num)
display the frame given by frame_num.

void **animate_stop** ()

void **show_front** ()
Set the view matrix to show the "front" of the scene.

void **show_back** ()
Set the view matrix to show the "back" of the scene.

void **show_top** ()
Set the view matrix to show the "top" of the scene.

void **show_bottom** ()
Set the view matrix to show the "bottom" of the scene.

void **show_left** () *Set the view matrix to show the "left" of the scene.*

void **show_right** () *Set the view matrix to show the "right" of the scene.*

DVexport_state_type
set_export_state (const LCVMstring&)

DVexport_state_type
get_export_state (LCVMstring&)

DVexport_state_type
get_export_state ()

DVrecord_state_type
export_start_record (const LCVMstring&, const LCVMstring&)

DVrecord_state_type
export_start_record (const DVexport_state_type, const LCVMstring&)

DVrecord_state_type
export_pause_record ()

DVrecord_state_type
export_continue_record ()

DVrecord_state_type
export_stop_record ()

DVrecord_state_type
get_record_state ()

int **animation_draw** ()

int **export_draw** ()

int **export_dataviewer** (LCVMstring&)
Export the scene in the DataViewer file format.

int **export_povary** (LCVMstring&)
Export the scene as a Povray file (NOT SUPPORTED).

int **export_inventor** (LCVMstring&)
Export the scene as an Inventor file (NOT SUPPORTED).

int **export_postscript** (LCVMstring&)

Export the scene as postscript (NOT SUPPORTED).

int **export_ppm** (LCVMstring&)
Export the scene as a ppm bitmap file.

29.14 void **set_geometry_pointer** ()
Set the global dv_current_scene_pointer. 77

29.15 void **set_geometry_pointer** (DVgeometry*) const
Set the global dv_current_scene_pointer. 77

29.16 void **get_name_stack** (int x, int y, LCVMstring &name_stack)
Return the name_stack of the vertex designated by the give x and y coordinates in the viewport. 77

void **set_perspective** (float fov, float ratio, float front_clip, float back_clip)
Set the perspective matrix

void **get_perspective** (float &fov, float &ratio, float &front_clip, float &back_clip)
Set the perspective matrix

void **set_user_function** (int (*func)(DVcreate_image_arguments &))
Set the function which the DVgeometry will use to create the scene graph.

29.17 void **new_top_level_container** (DVcontainer&)
Set the top level container to the given container and generate display lists. 78

DVcontainer& **get_top_level_container** ()
return a reference to the top level container

void **reset_named_properties** ()
reset the named properties so that there are no pre-existing names

29.18 void **set_default_input_string** (LCVMstring&)

A toplevel object which provided services such as animation and display list creation. It is the main entry point for DataViewer if you want to use it in retained mode.

29.4

DVgeometry (int (*)(DVcreate_image_arguments &)=NULL)

Default constructor.

Default constructor. It takes as an arguement a function which will create the scene graph which we wish to plot. Note that if a DVgeometry object is created with the function pointer NULL then load_file and read_file and load_file don't actually do anything.

29.5

int **draw** ()

Draw the scene to the current OpenGL drawable.

Draw the scene to the current OpenGL drawable. Bascially it calles the display list(s) which contains the compiled version of scene graph. It does not do a swapbuffer on double buffered displays you you need to wrap it in some OpenGL code if you have a double buffered display.

29.6

void **useTimer** ()

Turn on the timer.

Turn on the timer. When the timer is on DVgeometry will print out frames per-second for the current scene

29.7

int **get_animation_optimize_flag** () const

Return the value of the animation_optimize_flag.

Return the value of the animation_optimize_flag. When this flag is true, the DV geometry object will optimized the animation by precomputing a display list for each frame of the animation. This can lead to a LARGE amount of memory usage.

29.8

int **get_first_frame_number** () const

Gets the first frame number.

Gets the first frame number. This number is NOT used internally by DVgeometry, it is only provided as information to whatever external program is using the DVgeometry. It should be considered as a hint from the user defined scene graph creation function. Its main use to to allow scene graph creation function to communicate which subset of the animation to draw.

29.9

int **get_last_frame_number** () const

Gets the last frame number.

Gets the last frame number. This number is NOT used internally by DVgeometry, it is only provided as information to whatever external program is using the DVgeometry. It should be considered as a hint from the user defined scene graph creation function. Its main use to to allow scene graph creation function to communicate which subset of the animation to draw.

29.10

void **set_first_frame_number** (int)

Sets the first frame number.

Sets the first frame number. This number is NOT used internally by DVgeometry, it is only provided as information to whatever external program is using the DVgeometry. It should be considered as a hint from the user defined scene graph creation function. Its main use to to allow scene graph creation function to communicate which subset of the animation to draw.

29.11

void **set_last_frame_number** (int)

Sets the last frame number.

Sets the last frame number. This number is NOT used internally by DVgeometry, it is only provided as information to whatever external program is using the DVgeometry. It should be considered as a hint from the user defined scene graph creation function. Its main use to to allow scene graph creation function to communicate which subset of the animation to draw.

29.12

DVgeometry& **set_animation_frames** (const int, const int, const int)

Set the animation first frame, last frame and step sizes.

Set the animation first frame, last frame and step sizes. These are the numbers used internally by DVgeometry to compute the bounds of the animation. If animation_optimize_flag is true, this information is used to pre-compute display lists.

29.13

int **valid_draw_frame** (const int frame_num) const

given a frame number, it returns the frame number closest to it that we are allowed to display

given a frame number, it returns the frame number closest to it that we are allowed to display

29.14

void **set_geometry_pointer** ()

Set the global dv_current_scene_pointer.

Set the global dv_current_scene_pointer. This is a hack for the Python interface. If there are several DVgeometry objects it points at the last one used.

29.15

void **set_geometry_pointer** (DVgeometry*) const

Set the global dv_current_scene_pointer.

Set the global dv_current_scene_pointer. This is a hack for the Python interface. If there are several DVgeometry objects it points at the last one used.

29.16

void **get_name_stack** (int x, int y, LCVMstring &name_stack)

Return the name_stack of the vertex designated by the give x and y coordinates in the viewport.

Return the name_stack of the vertex designated by the give x and y coordinates in the viewport. In general, each DVcontainer (or other grouping node) will push a new name onto the stack and increment it for each of its children. Also, some compsite nodes (such as DVlines) will push their own names onto the stack. Unfortunately, there is no one place to look for behavior of each class in this regard.

29.17

void **new_top_level_container** (DVcontainer&)

Set the top level container to the given container and generate display lists.

Set the top level container to the given container and generate display lists. NOTE: This routine makes a copy of the given container to create a DVcontainer. So, any special properties of the given container (such as a DVselection_container) will be lost!

29.18

void **set_default_input_string** (LCVMstring&)

Set the default_filename.

Set the default_filename. This is a string which is sent to the function which creates the scene graph.

29.19

void **get_default_input_string** (LCVMstring&) const

Get the default_filename.

Get the default_filename. This is a string which is sent to the function which creates the scene graph.

29.20

void **set_default_output_string** (LCVMstring&)

Set the default output string.

Set the default output string. This is a string returned by the function which creats the scene graph. Note: I am not really sure why there is a set routine for this.

29.21

void **get_default_output_string** (LCVMstring&) const

Get the default output string.

Get the default output string. This is a string returned by the function which creats the scene graph

29.22

void **set_export_filename** (LCVMstring&)

Set the export_filename.

Set the export_filename. This is the filename which is used by the various export commands.

29.23

void **get_export_filename** (LCVMstring&) const

Get the export_filename.

Get the export_filename. This is the filename which is used by the various export commands.

29.24

void **set_pick_size** (double val)

Set the size of pick region.

Set the size of pick region. This is the size of the region in which vertices will be returned by the get_name_stack function. If it is small the user must be very careful about entering coordinates which lay exactly above the vertex (for example with the mouse). If it is large then the coordinates can be farther the actual location of a vertex, but multiple vertices may be selected.

29.25

void **set_persp_angle** (double val)

Set the size of perspective angle.

Set the size of perspective angle. This is a subset of the set_perspective command.

29.26

double **get_persp_angle** ()

Get the size of perspective angle

Get the size of perspective angle

30

class **DVgeometry_pointer**

A small class for setting pointers to DVgeometry objects.

Public Members

DVgeometry_pointer (DVgeometry *ptr=0)
Constructor

~DVgeometry_pointer ()
Destructor.

DVgeometry_pointer (const DVgeometry_pointer &data)
Copy constructor

const DVgeometry_pointer& **operator=** (const DVgeometry_pointer &data)
Assignment operator

int **operator==** (const DVgeometry_pointer &data) const
Check for equality

int **operator!=** (const DVgeometry_pointer &data) const
Not equality

operator DVgeometry* () const
Get routines

DVgeometry* **get_geometry_pointer** () const
Get routines

inline **DVgeometry_pointer** (DVgeometry *ptr)

inline **DVgeometry_pointer** (const DVgeometry_pointer &data)

inline DVgeometry* **get_geometry_pointer** () const

inline **operator DVgeometry*** () const

inline int **operator!=** (const DVgeometry_pointer &data) const

inline const DVgeometry_pointer&

		operator= (const DVgeometry_pointer &data)
inline	int	**operator==** (const DVgeometry_pointer &data) const
inline	void	**set_geometry_pointer** (DVgeometry *ptr)
inline	void	**set_geometry_pointer** (const DVgeometry_pointer &data)
inline		**~DVgeometry_pointer** ()

A small class for setting pointers to DVgeometry objects.

31 class **DVlight_position**

A class which defines the position of a light.

Public Members

	float	**x**	*This is the x position of the light in homogeneous coordinates*	
	float	**y**	*This is the y position of the light in homogeneous coordinates*	
	float	**z**	*This is the z position of the light in homogeneous coordinates*	
31.1	float	**w**	*This is the w position of the light in homogeneous coordinates.* . . .	85

DVlight_position (const float x_val=0.0, const float y_val=0.0, const float z_val=1.0, const float w_val=0.0)
Constructor

~DVlight_position ()
Destructor

DVlight_position (const DVlight_position&)
Copy constructor

const DVlight_position& **operator=** (const DVlight_position&)
Sssignment operator

int **operator==** (const DVlight_position&) const
Check equality

int **operator!=** (const DVlight_position&) const
Check inequality

void **set_data** (const float x_val=0.0, const float y_val=0.0, const float z_val=1.0, const float w_val=0.0)
Set_data routine which matches the constructor

void **set_data** (const DVlight_position&)

Set the data based upon another DVlight_position

void **get_data** (float&, float&, float&, float&) const
Get the current light position

void **get_data** (DVlight_position&) const
Copy the current light position into the give DVlight_position instance.

float& **operator**[] (const int)
We want foo[0] to be foo.x, ... foo[3] to be foo.w

inline **DVlight_position** (const float x_val, const float y_val, const float z_val, const float w_val)
constructor

inline **DVlight_position** (const DVlight_position &data)
copy constructor

inline void
get_data (float &x_val, float &y_val, float &z_val, float &w_val) const

inline void
get_data (DVlight_position &r) const

inline int **operator!=** (const DVlight_position &data) const
check not equal

inline const DVlight_position&
operator= (const DVlight_position &data)
assignment operator

inline int **operator==** (const DVlight_position &data) const
check equal

inline void
set_data (const float x_val, const float y_val, const float z_val, const float w_val)
set_data routine

inline void
set_data (const DVlight_position &pt)

inline **~DVlight_position** ()
destructor (does nothing)

A class which defines the position of a light.

31.1

float **w**

This is the w position of the light in homogeneous coordinates.

This is the w position of the light in homogeneous coordinates. If it is 0, then the light is directional, if it is 1, then it is a point source

32

class **DVlight_properties**

The properties (ambient, diffuse, position, etc) of a light

Public Members

DVlight_properties ()
Constructors

DVlight_properties (const DVlight_properties&)
Copy constructor from a DVlight_properties

DVlight_properties (const DVrich_light_properties&)
Copy constructor from a DVrich_light_properties

~DVlight_properties ()
Destructor

const DVlight_properties&
operator= (const DVlight_properties &)
Assignment operator from a DVlight_properties

const DVlight_properties&
operator= (const DVrich_light_properties &)
Assignment operator from a DVrich_light_properties

int **operator==** (const DVlight_properties&) const
Check equaliity

int **operator!=** (const DVlight_properties&) const
Check inequaliity

int **is_spotlight** () const
Return true if the light is a spotlight, false otherwise

32.1 int **is_positional** () const
Return true is the light is positional. 87

void **print** (ostream&) const

int **active** *Flag as to whether the light is active or not*

DVcolor	**ambient**	*The ambient color of the light*
DVcolor	**diffuse**	*The diffuse color of the light*
DVcolor	**specular**	*The specular color of the light*
DVlight_position	**position**	*The position of the light (in homogenous coordinates)*
DVspotlight_direction	**spot_direction**	*The direction of the light, if it is a spotlight*
DVlighting_exponent_float	**spot_exponent**	*The exponent of the light, if it is a spotlight*
DVspotlight_cutoff_float	**spot_cutoff**	*The cutoff value of the light, if it is a spotlight*
DVnon_negative_float	**constant_attenuation**	*The constant attenuation of the light*
DVnon_negative_float	**linear_attenuation**	*The linear attenuation of the light*
DVnon_negative_float	**quadratic_attenuation**	*The quadratic attenuation of the light*

The properties (ambient, diffuse, position, etc) of a light

32.1

int **is_positional** () const

Return true is the light is positional.

Return true is the light is positional. This functino returns true if the light is a point light. In other words, if position.w != 0. Otherwise, the light is directional and it returns false.

33 class **DVlights**

A list of lights

Public Members

	DVlights (const int numlights=0)	*Default constructor*
	DVlights (const DVlights&)	*Copy constructor*
	~DVlights ()	*Destructor*
DVlights&	**operator=** (const DVlights&)	*Assignment operator*
int	**operator==** (const DVlights&) const	*Check equality*
int	**operator!=** (const DVlights&) const	*Check inequality*

Protected Members

A list of lights

33.1 LCVMarray<DVlight_properties> **lights**

This is the main part of this simple class.

This is the main part of this simple class. This class is only a wrapper around an LCVMarray of DVlight_properties.

34

enum **DVlines_line_status**

This defines whether the line is not drawn at all, drawn a a thin line, or drawn as a cylinder

This defines whether the line is not drawn at all, drawn a a thin line, or drawn as a cylinder

35

enum **DVlines_ribbon_status**

This defines whether the ribbon is drawn or not

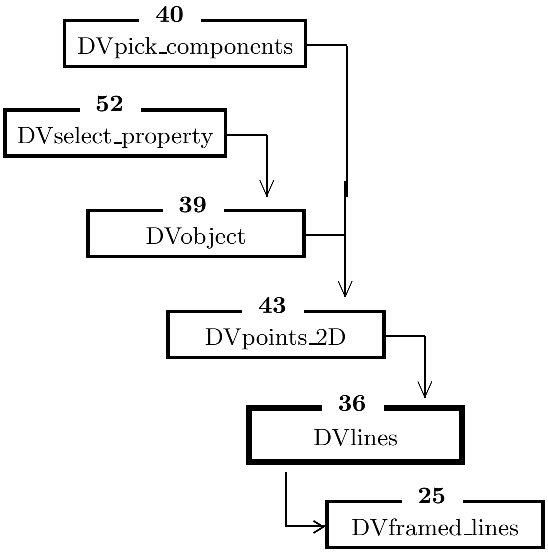
This defines whether the ribbon is drawn or not

36

class **DVlines** : public DVpoints_2D

A class for rendering one dimensional manifolds (curves in space).

Inheritance



Public Members

36.3 **DVlines** (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0)

Constructor the end points of the line segment are given default line starts at the origin, proceeds along the positive x-axis and has length one. 94

36.4 **DVlines** (const float x0, const float y0, const float z0, const float d0, const float d1, const float d2, const float x1, const float y1, const float z1, const float e0, const float e1, const float e2)

Constructor for giving a ribbon segment. 95

36.5 **DVlines** (LCVMarray_2D<float> &coords)
Constructor The coordinates of the points for one line are stored in the array coords 95

36.6 **DVlines** (LCVMarray_3D<float> &coords)
Constructor The coordinates of the points for the lines are in the array coords 95

36.7 virtual **~DVlines** () *Destructor* 95

36.8 DVlines& **set_data** (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0)
Set data routine the end points of the line segment are given default line starts at the origin, proceeds along the positive x-axis and has length one. 96

36.9 DVlines& **set_data** (const float x0, const float y0, const float z0, const float d0, const float d1, const float d2, const float x1, const float y1, const float z1, const float e0, const float e1, const float e2)
Set data routine for giving a ribbon segment. 96

36.10 DVlines& **set_data** (LCVMarray_2D<float> &coords)
Set data routine The coordinates of the points for one line are stored in the array coords 96

36.11 DVlines& **set_data** (LCVMarray_3D<float> &coords)
Set data routine The coordinates of the points for the lines are in the array coords 97

virtual void **draw** (LCVMstack<DVproperty*>&)
Main draw routine

void **draw_lines** (LCVMstack<DVproperty*>&)
Draw routine just for the lines

void **draw_cylinders** (LCVMstack<DVproperty*>&)

Draw routine just for the cylinders

void **draw_ribbons** (LCVMstack<DVproperty*>&)
Draw routine just for the ribbons

DVlines (const DVlines &data)
Copy constructor

const DVlines&
operator= (const DVlines &data)
Assignment operator

int **operator==** (const DVlines &data)
Check for equality

int **operator!=** (const DVlines &data)
Check for not equality

36.12 DVlines& **set_line_status** (const DVlines_line_status)
Sets the line status. 97

36.13 DVlines& **set_ribbon_status** (const DVlines_ribbon_status)
Sets the ribbon status. 97

DVlines_line_status
get_line_status () *Get the line status*

DVlines_ribbon_status
get_ribbon_status ()
Get the ribbon status

virtual void
print (ostream&) *A virtual print function*

friend ostream&
operator<< (ostream &, DVlines &)
A friend routine to overload the << operator

DVlines (FileOption a_file_op)
Methods concerning the scene file format

DVlines& **set_data** (FileOption a_file_op)
Methods concerning the scene file format

virtual int **ReadMyProp** () *Methods concerning the scene file format*

Protected Members

A class for rendering one dimensional manifolds (curves in space). It can render a group a space curves, using either lines or cylinders. It also includes the ability to place a director field, or "ribbon", along the curve.

36.3

DVlines (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0)

Constructor the end points of the line segment are given default line starts at the origin, proceeds along the positive x-axis and has length one.

Constructor the end points of the line segment are given default line starts at the origin, proceeds along the positive x-axis and has length one. the end points of the line segment are given.

36.4

DVlines (const float x0, const float y0, const float z0, const float d0, const float d1, const float d2, const float x1, const float y1, const float z1, const float e0, const float e1, const float e2)

Constructor for giving a ribbon segment.

Constructor for giving a ribbon segment. the end points of the ribbon segment are given–3 coordinates for the end point and 3 coordinates for the direction

36.5

DVlines (LCVMarray_2D<float> &coords)

Constructor The coordinates of the points for one line are stored in the array coords

Constructor The coordinates of the points for one line are stored in the array coords

36.6

DVlines (LCVMarray_3D<float> &coords)

Constructor The coordinates of the points for the lines are in the array coords

Constructor The coordinates of the points for the lines are in the array coords

36.7

virtual **~DVlines** ()

Destructor

Destructor

36.8

DVlines& **set_data** (const float x0=0.0, const float y0=0.0, const float z0=0.0, const float x1=1.0, const float y1=0.0, const float z1=0.0)

Set data routine the end points of the line segment are given default line starts at the origin, proceeds along the positive x-axis and has length one.

Set data routine the end points of the line segment are given default line starts at the origin, proceeds along the positive x-axis and has length one. the end points of the line segment are given

36.9

DVlines& **set_data** (const float x0, const float y0, const float z0, const float d0, const float d1, const float d2, const float x1, const float y1, const float z1, const float e0, const float e1, const float e2)

Set data routine for giving a ribbon segment.

Set data routine for giving a ribbon segment. the end points of the ribbon segment are given–3 coordinates for the end point and 3 coordinates for the direction

36.10

DVlines& **set_data** (LCVMarray_2D<float> &coords)

Set data routine The coordinates of the points for one line are stored in the array coords

Set data routine The coordinates of the points for one line are stored in the array coords

36.11

DVlines& **set_data** (LCVMarray_3D<float> &coords)

Set data routine The coordinates of the points for the lines are in the array coords

Set data routine The coordinates of the points for the lines are in the array coords

36.12

DVlines& **set_line_status** (const DVlines_line_status)

Sets the line status.

Sets the line status. Can be line, cylinder, or none.

36.13

DVlines& **set_ribbon_status** (const DVlines_ribbon_status)

Sets the ribbon status.

Sets the ribbon status. Can be ribbon, or none.

36.1

DVlines_line_status **line_status**

Defines how the lines will be drawn.

Defines how the lines will be drawn. Options are thin lines, tubes, or not at all.

36.2

DVlines_ribbon_status **ribbon_status**

Defines how the ribbons will be drawn.

Defines how the ribbons will be drawn. Options are as flat surfaces or not at all.

37

class **DVmaterial_properties**

A class which encapsulates the properties that may be modified for an object

Public Members

	DVmaterial_properties ()	*Default constructor*
	DVmaterial_properties (const DVmaterial_properties&)	*Copy constructor from a DVmaterial_properties*
	DVmaterial_properties (const DVrich_material_properties&)	*Copy constructor from a DVrich_material_properties*
	~DVmaterial_properties ()	*Destructor*
const DVmaterial_properties&	**operator=** (const DVmaterial_properties &)	*Assignment operator from a DVmaterial_properties*
const DVmaterial_properties&	**operator=** (const DVrich_material_properties &)	*Assignement operator from a DVrich_material_properties*
int	**operator==** (const DVmaterial_properties&) const	*Check equality*
int	**operator!=** (const DVmaterial_properties&) const	*Check inequality*
void	**print** (ostream &output) const	*Print routine*
DVcolor	**ambient_factors**	*A color to use for the ambient lighting of the object*
DVcolor	**diffuse_factors**	*A color to use for the diffuse lighting of the object*
DVcolor	**specular_factors**	*A color to use for the specular highlights of the object*

DVcolor	**emission_factors**	*A color to use for the emissivity of the object*
DVlighting_exponent_float	**shininess**	*A float used for the shininess of the object*

A class which encapsulates the properties that may be modified for an object

38

class **DVnamed_properties**

A class which contains the named versions of all of the properties an object can have.

Public Members

DVnamed_properties ()
The default constructor

DVnamed_properties (DVnamed_properties&)
Copy constructor

~DVnamed_properties ()
Destructor

const DVnamed_properties&
operator= (DVnamed_properties&)
Assignment operator

void **erase** () *Delete the contents of the object*

void **print** (ostream &output)
Print the object

DVnamed_color_table
named_color_table
The colors

DVnamed_positive_float_table
named_positive_float_table
Not 100 percent sure why this is here

DVnamed_non_negative_float_table
named_non_negative_float_table
Not 100 percent sure why this is here

DVnamed_non_negative_int_table
named_non_negative_int_table
Not 100 percent sure why this is here

DVnamed_complexity_int_table
named_complexity_int_table
The geometry complexity

DVnamed_lighting_exponent_float_table

named_lighting_exponent_float_table
The spotlight exponent

DVnamed_spotlight_cutoff_float_table
named_spotlight_cutoff_float_table
The spotlight cutoff angle

DVnamed_spotlight_direction_table
named_spotlight_direction_table
The spotlight direction

DVnamed_light_position_table
named_light_position_table
The lights position

DVnamed_scale_table
named_scale_table
Not 100 percent sure why this is here

DVnamed_color_by_data_table
named_color_by_data_table
Color by data objects that can be accessed by name.

DVnamed_rich_light_properties_table
named_rich_light_properties_table
Light properties

DVnamed_rich_material_properties_table
named_rich_material_properties_table
Material properties

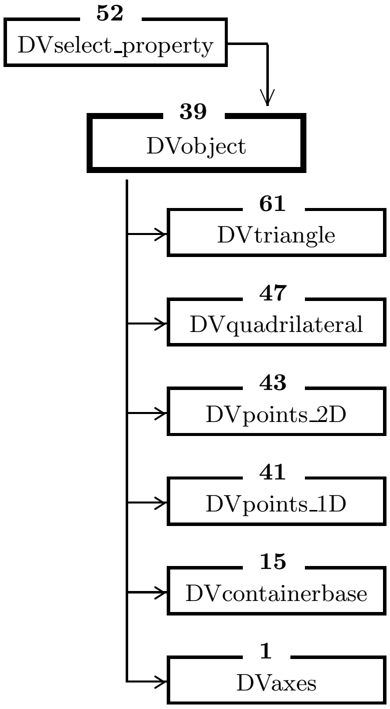
A class which contains the named versions of all of the properties an object can have. It is probably best not to use this, since the named version of the properties will probably go away at some point.

39

class **DVobject** : public DVselect_property

The top of the DataViewer class hierarchy.

Inheritance



Public Members

Add the given translation to the translation already there. 108

39.10 DVobject& **add_translate** (const float x, const float y, const float z)
Add the given translation to the translation already there. 108

Set the rotation with an axis (data), an angle and specifying if it

DVobject& **set_rotate** (LCVMarray<float> &data, const float angle, const char type='d')
is in radians ('r') or degrees ('d'- by default)

Set the rotation with an axis (x, y, z), an angle and specifying if it

DVobject& **set_rotate** (const float x, const float y, const float z, const float angle, const char type='d')
is in radians ('r') or degrees ('d'- by default)

39.11 DVobject& **add_rotate** (LCVMarray<float> &data, const float angle, const char type='d')
Add the given rotation to the rotation already there. 109

39.12 DVobject& **add_rotate** (const float x, const float y, const float z, const float angle, const char type='d')
Add the given rotation to the rotation already there. 109

void **get_translation** (LCVMarray<float>&)
Get the translation

void **get_rotation** (LCVMarray<float>&, float&)
Get the rotation

39.13 virtual void **draw** (LCVMstack<DVproperty*>&)
The draw routine. 109

void **set_OpenGL_lighting_properties** (DV-light_properties_property_item_array &lights)

A routine to set lighting properties for OpenGL

void **set_OpenGL_material_properties** (DVmaterial_properties_property_item &material, DVcolor &color)

A routine to set material properties for OpenGL

void **set_OpenGL_color** (DVcolor &color)

A routine to set color for OpenGL

virtual DVobject& **copy_child** (DVobject &child)

Function to be declared in derived classes. This is only a place holder.

virtual DVobject& **delete_current_child** ()

Function to be declared in derived classes. This is only a place holder.

DVobject (const DVobject &data)

Copy constructor

const DVobject& **operator=** (const DVobject &data)

Assignment operator

int **operator==** (const DVobject &data)

Check for equality

int **operator!=** (const DVobject &data)

Check for not equality

virtual void **print** (ostream &)

Virtual print function so that we can print instantiations of the various classes derived from DVobject

friend ostream&

		operator<< (ostream &, DVobject &)	*A friend routine to overload the << operator*	
39.14	int	**ReadProp** ()	*Read the object properties from a scene file.*	110
39.15	void	**ReadDVrotate** ()	*Read the object rotation from a scene file.*	110
39.16	void	**ReadDVtranslation** ()	*Read the object translation from a scene file.*	110
39.17	void	**ReadSimpleProps** ()	*Read the object properties from a scene file A simple loop on <tt/ReadProp()/ : while it detects a simple property, it reads it.*	111

Protected Members

39.1	class DVcontainerbase LCVMarray<float>	**translation**	*The tranlation of the object.*	111
39.2	LCVMarray<float>	**rotation_axis**	*Set the axis about which the orientation of the object is rotated*	111
39.3	float	**rotation_angle**	*The angle about the axis by which the orientation of the object is changed*	111
39.4	LCVMarray<float>	**rotation_quaterions**	*The orientation may also be set by entering quaterions.*	112
	virtual DVobject*	**copy** ()	*We also want a routine that returns a pointer to a copy of the object created with new.*	
	void	**axis_to_quat** (LCVMarray<float> axis, float angle, LCVMarray<float> &quat)		

A convenience routine to change vector angle format to quaternions

void **quat_to_axis** (LCVMarray<float> quat, LCVMarray<float> &axis, float &angle)
A convenience routine to change quaterions to vector angle format

void **add_quat** (LCVMarray<float> a, LCVMarray<float> b, LCVMarray<float> &c)
Add two quaterions a+b = c

The top of the DataViewer class hierarchy. All renderable objects in DataViewer inherit from this class. It contains attributes which apply to a wide range of geometrical objects, position, rotation, etc.

39.5

DVobject ()

Contructor.

Contructor. Set translation to zero and rotation angle to zero.

39.6

virtual **~DVobject** ()

Default destructor.

Default destructor. It is declared virtual so that the correct decendant destructor will get called.

39.7

DVobject& **set_translate** (LCVMarray<float> &data)

Set the translation

Set the translation

39.8

DVobject& **set_translate** (const float x, const float y, const float z)

Set the translation

Set the translation

39.9

DVobject& **add_translate** (LCVMarray<float> &data)

Add the given translation to the translation already there.

Add the given translation to the translation already there.

39.10

DVobject& **add_translate** (const float x, const float y, const float z)

Add the given translation to the translation already there.

Add the given translation to the translation already there.

39.11

DVobject& **add_rotate** (LCVMarray<float> &data, const float angle, const char type='d')

Add the given rotation to the rotation already there.

Add the given rotation to the rotation already there. It composes the existing rotation with the new one.

39.12

DVobject& **add_rotate** (const float x, const float y, const float z, const float angle, const char type='d')

Add the given rotation to the rotation already there.

Add the given rotation to the rotation already there. It composes the existing rotation with the new one.

39.13

virtual void **draw** (LCVMstack<DVproperty*>&)

The draw routine.

The draw routine. It is given a reference to the overall property stack.

39.14

int **ReadProp** ()

Read the object properties from a scene file.

Read the object properties from a scene file. If it encounters a simple property, like a rotation, a translation or a property inherited from the <tt/DVselect_property/ class, it reads it, positions the file pointer imediatly after it, and retruns 1. If there is no such property following the pointer in the file, it returns 0.

39.15

void **ReadDVrotate** ()

Read the object rotation from a scene file.

Read the object rotation from a scene file. It assumes that the existence of a rotation property has been detected before.

39.16

void **ReadDVtranslation** ()

Read the object translation from a scene file.

Read the object translation from a scene file. It assumes that the existence of a rotation property has been detected before.

39.17

void **ReadSimpleProps** ()

Read the object properties from a scene file A simple loop on <tt/ReadProp()/ : while it detects a simple property, it reads it.

Read the object properties from a scene file A simple loop on <tt/ReadProp()/ : while it detects a simple property, it reads it. This function applies to geometrical classes that only inherit from DVobject, and not from any other class.

39.1

class DVcontainerbase LCVMarray<float> **translation**

The tranlation of the object.

The tranlation of the object. Normally interpreted as the center of the object.

39.2

LCVMarray<float> **rotation_axis**

Set the axis about which the orientation of the object is rotated

Set the axis about which the orientation of the object is rotated

39.3

float **rotation_angle**

The angle about the axis by which the orientation of the object is changed

The angle about the axis by which the orientation of the object is changed

39.4

LCVMarray<float> **rotation_quaterions**

The orientation may also be set by entering quaterions.

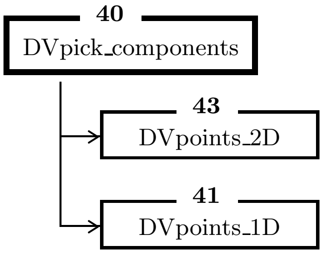
The orientation may also be set by entering quaterions.

40

class **DVpick_components**

A class which allows the user to select x,y, and z components from a list of values.

Inheritance



Public Members

	DVpick_components ()	*Default constructor*
	˜DVpick_components ()	*Destructor*
void	**set_x_plot_coordinate** (const int i, const int select=0)	*Set the X coordinate*
void	**set_y_plot_coordinate** (const int j, const int select=0)	*Set the Y coordinate*
void	**set_z_plot_coordinate** (const int k, const int select=0)	*Set the Z coordinate*
void	**set_plot_coordinate** (const int i, const int coord, const int select=0)	*Set all the 'coord'th coordinate to be i*
void	**set_xyz_plot_coordinates** (const int i, const int j, const int k, const int select=0)	

Set all 3 coordinates at once

void **set_xyz_plot_coordinates** (LCVMarray<int> &i, const int select=0)
Set all 3 coordinates at once from an array

void **set_xyz_plot_coordinates** (LCVMarray_2D<int> &i)
Set all 3 coordinates at once from an array. Not sure why this is 2D.

void **set_color_coordinate** (const int i, const int select=0)
Set color coordinate

void **set_color_coordinates** (LCVMarray<int> &i)
Set color coordinate from array

DVpick_components (const DVpick_components &data)
Copy constructor

const DVpick_components&
operator= (const DVpick_components &data)
Assignment operator

int **operator==** (const DVpick_components &data)
Check for equality

int **operator!=** (const DVpick_components &data)
Check for inequality

void **print** (ostream &) *Print function*

int **check_color_coordinates** (int select = 0)
Check if the color coordinate has been set

int **ReadPickCompProp** ()
Methods concrening the scene file format

void **ReadCoords** () *Methods concrening the scene file format*

void **ReadColorCoord** ()
Methods concrening the scene file format

Protected Members

int	**num_elts**	*The number of components in each point*
LCVMarray_2D<int>	**use_xyz**	*Which coordinates to use for x, y, and z*
LCVMarray<int>	**use_color**	*Which coordinates to use for color (if any)*
void	**default_use_coords** ()	*Assign valid use_coordinates*
void	**default_use_colors** ()	*Assign default use_colors*
void	**check_use_coords** ()	*Check for valid use_coordinates*
void	**check_use_colors** ()	*Check for valid use_colors*

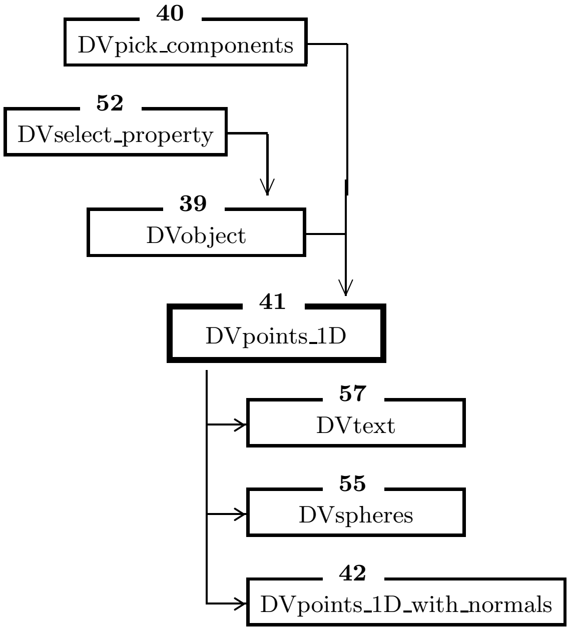
A class which allows the user to select x,y, and z components from a list of values.

41

class **DVpoints_1D** : public DVobject, public DVpick_components

A class which encapsulated a one dimensional array of points.

Inheritance



Public Members

41.1 **DVpoints_1D** () *Constructor Default point is no points* 119

41.2 **DVpoints_1D** (const float x, const float y, const float z)
Constructor Give three points to define a point with 3 components 119

41.3 **DVpoints_1D** (const float x0, const float y0, const float z0, const float x1, const float y1, const float z1)

Constructor Defines two points, each with 3 components 120

41.4 **DVpoints_1D** (const float x0, const float y0,
const float z0, const float d0,
const float d1, const float d2,
const float x1, const float y1,
const float z1, const float e0,
const float e1, const float e2)
Constructor Defines a two points, each with 6 components. 120

41.5 **DVpoints_1D** (LCVMarray<float> &point)
Constructor Defines a point with components in the array point .. 120

41.6 **DVpoints_1D** (LCVMarray_2D<float> &coords)
Constructor Defines an array of points with components in the array "coords" 120

virtual **~DVpoints_1D** () *Destructor*

virtual void
draw (LCVMstack<DVproperty*>&)
Draw routine

DVpoints_1D (const DVpoints_1D &data)
Copy constructor

const DVpoints_1D&
operator= (const DVpoints_1D &data)
Assignment operator

int **operator==** (const DVpoints_1D &data)
Check for equality

int **operator!=** (const DVpoints_1D &data)
Check for inequality

void **get_used_coordinates** (LCVMarray_2D<float>&,
const int select=0)
Make smaller array with just the components that are being used

void **get_used_coordinates** (LCVMarray_2D<float>&,
DVscale &,
const int select=0)

Make smaller array with just the components that are being used and scale them

void **get_used_colors** (LCVMarray<float>&, const int select=0)
Make smaller array with just the component that is being used for color

virtual void **print** (ostream&) *A virtual print function*

virtual int **ReadMyProp** () *Methods concerning the scene file format*

void **ReadDVProps** () *Methods concerning the scene file format*

void **print** (ostream &, FileOption)
Methods concerning the scene file format

Protected Members

virtual DVobject* **copy** () *of the object created with new.*

void **set_data** () *The default point is no points.*

void **set_data** (const float, const float, const float)
makes a point with 3 components

void **set_data** (const float x0, const float y0, const float z0, const float x1, const float y1, const float z1)
defines two points, each with 3 components

void **set_data** (const float x0, const float y0, const float z0, const float d0, const float d1, const float d2, const float x1, const float y1, const float z1, const float e0, const float e1, const float e2)
defines two points, each with 6 components

void **set_data** (LCVMarray<float> &point)

makes a point with components in the array point

void **set_data** (LCVMarray_2D<float> &coords)
makes an array of points with components in the array "coords"

LCVMarray_2D<float>
points

A class which encapsulated a one dimensional array of points. It is not meant to be used itself, but as a basis from which other classes inherit.

41.1

DVpoints_1D ()

Constructor Default point is no points

Constructor Default point is no points

41.2

DVpoints_1D (const float x, const float y, const float z)

Constructor Give three points to define a point with 3 components

Constructor Give three points to define a point with 3 components

41.3

DVpoints_1D (const float x0, const float y0, const float z0, const float x1, const float y1, const float z1)

Constructor Defines two points, each with 3 components

Constructor Defines two points, each with 3 components

41.4

DVpoints_1D (const float x0, const float y0, const float z0, const float d0, const float d1, const float d2, const float x1, const float y1, const float z1, const float e0, const float e1, const float e2)

Constructor Defines a two points, each with 6 components.

Constructor Defines a two points, each with 6 components. The "d" components define a director field along the curve.

41.5

DVpoints_1D (LCVMarray<float> &point)

Constructor Defines a point with components in the array point

Constructor Defines a point with components in the array point

41.6

DVpoints_1D (LCVMarray_2D<float> &coords)

Constructor Defines an array of points with components in the array "coords"

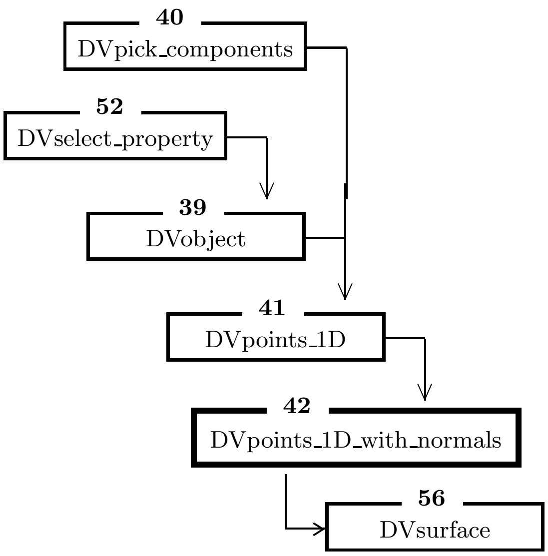
Constructor Defines an array of points with components in the array "coords"

42

class **DVpoints_1D_with_normals** : public DVpoints_1D

A list of points with an attached vector field or normals

Inheritance



Public Members

DVpoints_1D_with_normals ()
Default constructor – default is no points, no normals

42.5 **DVpoints_1D_with_normals** (LCVMarray_2D<float> &point)
Defines points with components in the array point. 123

42.6 **DVpoints_1D_with_normals** (LCVMarray_2D<float> &point, LCVMarray_2D<float> &n)

Defines points with components in the array point normals given – point and n must have the same leading size each vector in n must have 3 components. 124

virtual **~DVpoints_1D_with_normals** ()
Destructor

virtual void
draw (LCVMstack<DVproperty*>&)
Draw routine

DVpoints_1D_with_normals (const DVpoints_1D_with_normals &data)
Copy constructor

const DVpoints_1D_with_normals&
operator= (const DVpoints_1D_with_normals &data)
Assignment operator

int **operator==** (const DVpoints_1D_with_normals &data)
Check for equality

int **operator!=** (const DVpoints_1D_with_normals &data)
Check for inequality

virtual void
print (ostream&) *A virtual print function*

int **user_supplied_normals** ()
Returns 1 if the user supplied normal vectors

Protected Members

virtual DVobject*
copy () *A routine that returns a pointer to a copy of the object created with new.*

LCVMarray_2D<float>
normals *normals[i] is a normal vector. normals[i] must have size 3*

int **user_supplied_normals_flag**

			1 if user supplied normals	
42.1	void	**set_data** ()	*Functions to set the nodes and normals–these mimic the constructors.*	124
42.2	void	**set_data** (LCVMarray_2D<float> &point)	*Makes points with components in the array point.*	124
42.3	void	**set_data** (LCVMarray_2D<float> &point, LCVMarray_2D<float> &n)	*Makes points with components in the array point.*	125
42.4	void	**set_nodes** ()	*Protected functions to set the nodes and normals.*	125
	void	**set_nodes** (LCVMarray_2D<float> &v)	*Array of nodes given*	
	void	**set_normals** ()	*No normals given*	
	void	**set_normals** (LCVMarray_2D<float> &n)	*Array of normals given.*	
	void	**set_user_supplied_normals** ()	*The normals are given by the user.*	
	void	**set_compute_normals** ()	*The class should compute normals itself.*	

A list of points with an attached vector field or normals

42.5

DVpoints_1D_with_normals (LCVMarray_2D<float> &point)

Defines points with components in the array point.

Defines points with components in the array point. no normals given.

42.6

DVpoints_1D_with_normals (LCVMarray_2D<float> &point, LCVMarray_2D<float> &n)

Defines points with components in the array point normals given – point and n must have the same leading size each vector in n must have 3 components.

Defines points with components in the array point normals given – point and n must have the same leading size each vector in n must have 3 components. (usually the vectors are normalized.)

42.1

void **set_data** ()

Functions to set the nodes and normals–these mimic the constructors.

Functions to set the nodes and normals–these mimic the constructors. The default is no points and no normals.

42.2

void **set_data** (LCVMarray_2D<float> &point)

Makes points with components in the array point.

Makes points with components in the array point. No normals.

42.3

void **set_data** (LCVMarray_2D<float> &point, LCVMarray_2D<float> &n)

Makes points with components in the array point.

Makes points with components in the array point. Normals given in n.

42.4

void **set_nodes** ()

Protected functions to set the nodes and normals.

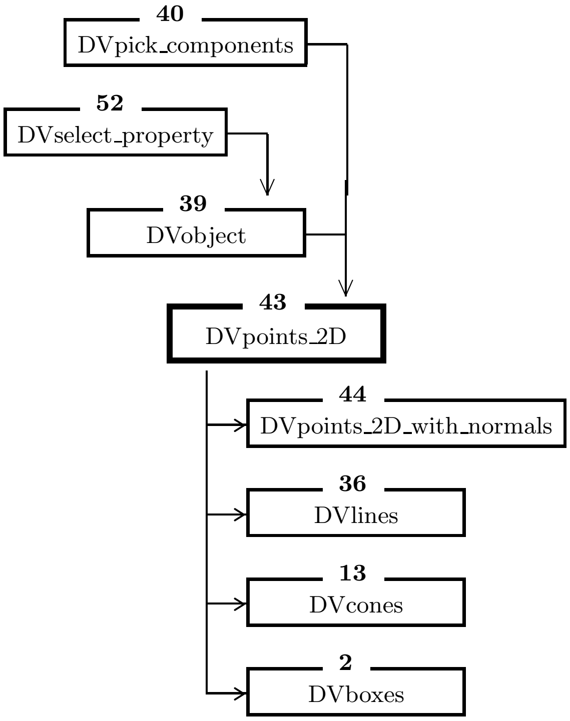
Protected functions to set the nodes and normals. This one makes a default list of points (empty).

43

class **DVpoints_2D** : public DVobject, public DVpick_components

A class which encapsulated a two dimensional array of points.

Inheritance



Public Members

virtual void
draw (LCVMstack<DVproperty*>&)
Draw routine

DVpoints_2D (const DVpoints_2D &data)
Copy constructor

const DVpoints_2D&
operator= (const DVpoints_2D &data)
Assignment operator

int **operator==** (const DVpoints_2D &data)
Check for equality

int **operator!=** (const DVpoints_2D &data)
Check for inequality

void **get_used_coordinates** (LCVMarray_3D<float>&, const int select=0)
Make smaller array with just the components that are being used

void **get_used_coordinates** (LCVMarray_3D<float>&, DVscale &, const int select=0)
Make smaller array with just the components that are being used and scale them

void **get_used_colors** (LCVMarray_2D<float>&, const int select=0)
Make smaller array with just the component that is being used for color

virtual void
print (ostream&) *A virtual print function*

virtual int **ReadMyProp** () *Methods concrening the scene file format*

void **ReadDVProps** () *Methods concrening the scene file format*

Protected Members

virtual DVobject*
copy () *of the object created with new.*

void **set_data** () *The default is no points.*

void **set_data** (LCVMarray_2D<float> &point)

		makes one group of points with components in the array point
void	**set_data** (LCVMarray_3D<float> &coords)	*makes an array of groups of points with components in the array "coords"*
LCVMarray_3D<float>	**points**	*points[i][j] is a "point" points[i] is a group of points*

A class which encapsulated a two dimensional array of points. It is not meant to be used itself, but as a basis from which other classes inherit.

43.1

DVpoints_2D ()

Constructor.

Constructor. Default constructor – default is no points

43.2

DVpoints_2D (LCVMarray_2D<float> &point)

Constructor.

Constructor. Defines one group of points with components in the array point

43.3

DVpoints_2D (LCVMarray_3D<float> &coords)

Constructor.

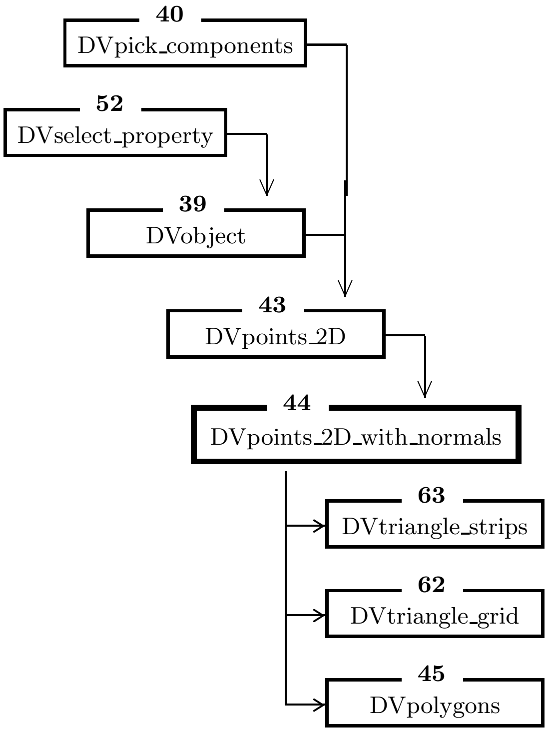
Constructor. Defines an array of groups of points with components in array "coords"

44

class **DVpoints_2D_with_normals** : public DVpoints_2D

A class which encapsulates a two dimensional array of points with an attach vector field.

Inheritance



Public Members

DVpoints_2D_with_normals ()
Default constructor – default is no points, no normals

44.7 **DVpoints_2D_with_normals** (LCVMarray_2D<float> &point)

virtual **~DVpoints_2D_with_normals** ()
Destructor

virtual void
draw (LCVMstack<DVproperty*>&)
Draw routine

DVpoints_2D_with_normals (const DVpoints_2D_with_normals &data)
Copy constructor

const DVpoints_2D_with_normals&
operator= (const DVpoints_2D_with_normals &data)
Assignment operator

int **operator==** (const DVpoints_2D_with_normals &data)
Check for equality

int **operator!=** (const DVpoints_2D_with_normals &data)

Check for inequality

virtual void
print (ostream&) *A virtual print function*

int **user_supplied_normals** ()
Returns 1 if the user supplied normal vectors

Protected Members

virtual DVobject*
copy () *A routine that returns a pointer to a copy of the object created with new.*

44.1 LCVMarray_3D<float>
normals *Normals[i][j] is a normal vector.* 133

int **user_supplied_normals_flag**
1 if user supplied normals

void **set_data** () *The default is no points and no normals*

44.2 void **set_data** (LCVMarray_2D<float> &point)
Makes one group of points with components in the array point.
134

44.3 void **set_data** (LCVMarray_3D<float> &coords)
Makes an array of groups of points with components in the array "coords". 134

44.4 void **set_data** (LCVMarray_2D<float> &point, LCVMarray_2D<float> &n)
Makes one group of points with components in the array point.
134

44.5 void **set_data** (LCVMarray_3D<float> &coords, LCVMarray_3D<float> &n)
Makes an array of groups of points with components in the array "coords". 135

44.6 void **set_nodes** () *Protected functions to set the nodes and normals.* 135

void **set_nodes** (LCVMarray_3D<float> &v)

		2_D array of nodes given
void	**set_normals** ()	*No normals given*
void	**set_normals** (LCVMarray_3D<float> &n)	*2_D array of normals given.*
void	**set_user_supplied_normals** ()	*The user has supplied normals*
void	**set_compute_normals** ()	*The class should compute the normals itself*

A class which encapsulates a two dimensional array of points with an attach vector field.

44.7

DVpoints_2D_with_normals (LCVMarray_2D<float> &point)

Defines one group of points with components in the array point.

Defines one group of points with components in the array point. No normals given.

44.8

DVpoints_2D_with_normals (LCVMarray_3D<float> &coords)

Defines an array of groups of points with components in array "coords".

Defines an array of groups of points with components in array "coords". No normals given.

44.9

DVpoints_2D_with_normals (LCVMarray_2D<float> &point, LCVMarray_2D<float> &n)

Defines one group of points with components in the array point.

Defines one group of points with components in the array point. Normals given – point and n must have the same leading size each vector in n must have 3 components. (usually the vectors are normalized.).

44.10

DVpoints_2D_with_normals (LCVMarray_3D<float> &coords, LCVMarray_3D<float> &n)

Defines an array of groups of points with components in array "coords".

Defines an array of groups of points with components in array "coords". Normals given – coords and n must have the same structure each vector in n must have 3 components. (usually the vectors are normalized.).

44.1

LCVMarray_3D<float> **normals**

Normals[i][j] is a normal vector.

Normals[i][j] is a normal vector. Normals[i] is a group of normals. normals[i][j] must have size 3

44.2

void **set_data** (LCVMarray_2D<float> &point)

Makes one group of points with components in the array point.

Makes one group of points with components in the array point. No normals.

44.3

void **set_data** (LCVMarray_3D<float> &coords)

Makes an array of groups of points with components in the array "coords".

Makes an array of groups of points with components in the array "coords". No normals.

44.4

void **set_data** (LCVMarray_2D<float> &point, LCVMarray_2D<float> &n)

Makes one group of points with components in the array point.

Makes one group of points with components in the array point. Normals given in n.

44.5

void **set_data** (LCVMarray_3D<float> &coords, LCVMarray_3D<float> &n)

Makes an array of groups of points with components in the array "coords".

Makes an array of groups of points with components in the array "coords". Normals given in n.

44.6

void **set_nodes** ()

Protected functions to set the nodes and normals.

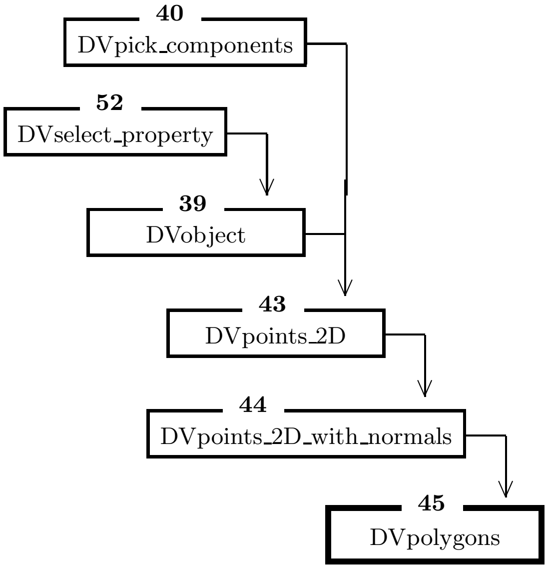
Protected functions to set the nodes and normals. This one makes a default list of points (empty).

45

class **DVpolygons** : public DVpoints_2D_with_normals

A class which describes a list of polygons.

Inheritance



Public Members

DVpolygons () *Constructors. This one makes a default list of polygons (empty)*

45.1 **DVpolygons** (LCVMarray<float> &v0,
LCVMarray<float> &v1,
LCVMarray<float> &v2)
Three nodes given–creates one triangle. 140

45.2 **DVpolygons** (LCVMarray<float> &v0,
LCVMarray<float> &v1,
LCVMarray<float> &v2,
LCVMarray<float> &n0,
LCVMarray<float> &n1,
LCVMarray<float> &n2)

2_D array of nodes given–v[i] is a polygon. 143

45.10 **DVpolygons** (LCVMarray_3D<float> &v, LCVMarray_3D<float> &n)
2_D array of nodes given and 2_D array of normals as well. 144

virtual **~DVpolygons** () *Destructor*

DVpolygons&
set_data ()

45.11 DVpolygons&
set_data (LCVMarray<float> &v0,
LCVMarray<float> &v1,
LCVMarray<float> &v2)
Three nodes given–creates one triangle. 144

45.12 DVpolygons&
set_data (LCVMarray<float> &v0,
LCVMarray<float> &v1,
LCVMarray<float> &v2,
LCVMarray<float> &n0,
LCVMarray<float> &n1,
LCVMarray<float> &n2)
Three nodes given and 3 normals given–creates one triangle. 145

45.13 DVpolygons&
set_data (LCVMarray<float> &v0,
LCVMarray<float> &v1,
LCVMarray<float> &v2,
LCVMarray_2D<float> &n)
Three nodes given and an array of 3 normals given–creates one triangle. 145

45.14 DVpolygons&
set_data (LCVMarray<float> &v0,
LCVMarray<float> &v1,
LCVMarray<float> &v2,
LCVMarray<float> &v3)
Four nodes given–creates one quadrilateral. 145

45.15 DVpolygons&

set_data (LCVMarray<float> &v0,
LCVMarray<float> &v1,
LCVMarray<float> &v2,
LCVMarray<float> &v3,
LCVMarray<float> &n0,
LCVMarray<float> &n1,
LCVMarray<float> &n2,
LCVMarray<float> &n3)
Four nodes given and 4 normals given–creates one quadrilateral. 146

45.16 DVpolygons&
set_data (LCVMarray<float> &v0,
LCVMarray<float> &v1,
LCVMarray<float> &v2,
LCVMarray<float> &v3,
LCVMarray_2D<float> &n)
Four nodes given and an array of 4 normals given–creates one quadrilateral. 146

45.17 DVpolygons&
set_data (LCVMarray_2D<float> &v)
Array of nodes given–v[i] is a vertex. 147

45.18 DVpolygons&
set_data (LCVMarray_2D<float> &v,
LCVMarray_2D<float> &n)
Array of nodes given and array of normals given–v[i] is a vertex. . 147

45.19 DVpolygons&
set_data (LCVMarray_3D<float> &v)
2_D array of nodes given–v[i] is a polygon. 147

45.20 DVpolygons&
set_data (LCVMarray_3D<float> &v,
LCVMarray_3D<float> &n)
2_D array of nodes given and 2_D array of normals as well. 147

virtual void
draw (LCVMstack<DVproperty*>&)
Virtual draw routine

DVpolygons (const DVpolygons &data)

Copy constructor

const DVpolygons&
operator= (const DVpolygons &data) *Assignment operator*

virtual void
print (ostream&) *A virtual print function*

DVpolygons (FileOption a_file_op) *Methods concerning the scene file format*

virtual int **ReadMyProp** () *Methods concerning the scene file format*

Protected Members

virtual DVobject*
copy () *A routine that returns a pointer to a copy of the object created with new.*

A class which describes a list of polygons. Polygons can have 3 or more vertices. They must (it is up to the user to make sure) be convex. They must (it is up to the user to make sure) be planar. If any one of the above two properties are not satisfied, then "things could go wrong".

45.1

DVpolygons (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2)

Three nodes given–creates one triangle.

Three nodes given–creates one triangle. Each LCVMarray v_i must have the same size.

45.2

DVpolygons (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray<float> &n0, LCVMarray<float> &n1, LCVMarray<float> &n2)

Three nodes given and 3 normals given–creates one triangle.

Three nodes given and 3 normals given–creates one triangle. Each LCVMarray v_i must have the same size and each LCVMarray n_i must have size 3. (Usually n_i is a unit vector.).

45.3

DVpolygons (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray_2D<float> &n)

Three nodes given and an array of 3 normals given–creates one triangle.

Three nodes given and an array of 3 normals given–creates one triangle. Each LCVMarray v_i must have the same size and n must be 3x3. n[i] is the normal for vertex i. (Usually n[i] is a unit vector.)

45.4

DVpolygons (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray<float> &v3)

Four nodes given–creates one quadrilateral.

Four nodes given–creates one quadrilateral. Each LCVMarray must have the same size.

45.5

DVpolygons (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray<float> &v3, LCVMarray<float> &n0, LCVMarray<float> &n1, LCVMarray<float> &n2, LCVMarray<float> &n3)

Four nodes given and 4 normals given–creates one quadrilateral.

Four nodes given and 4 normals given–creates one quadrilateral. each LCVMarray v_i must have the same size and each LCVMarray n_i must have size 3. (Usually n_i is a unit vector.)

45.6

DVpolygons (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray<float> &v3, LCVMarray_2D<float> &n)

Four nodes given and an array of 4 normals given–creates one quadrilateral.

Four nodes given and an array of 4 normals given–creates one quadrilateral. each LCVMarray v_i must have the same size and n must be 4x3. n[i] is the normal for vertex i. (Usually n[i] is a unit vector.)

45.7

DVpolygons (LCVMarray_2D<float> &v)

Array of nodes given–v[i] is a vertex.

Array of nodes given–v[i] is a vertex. v must be rectangular. v[i] must have size at least 3.

45.8

DVpolygons (LCVMarray_2D<float> &v, LCVMarray_2D<float> &n)

Array of nodes given and array of normals given–v[i] is a vertex.

Array of nodes given and array of normals given–v[i] is a vertex. v must be rectangular. The (leading) size of v must be the same as the (leading) size of n. n[i] must have size 3. v[i] must have size at least 3.

45.9

DVpolygons (LCVMarray_3D<float> &v)

2_D array of nodes given–v[i] is a polygon.

2_D array of nodes given–v[i] is a polygon. v[i][j] is a vertex. Each vertex must have the same length. Each polygon must have at least three vertices.

45.10

DVpolygons (LCVMarray_3D<float> &v, LCVMarray_3D<float> &n)

2_D array of nodes given and 2_D array of normals as well.

2_D array of nodes given and 2_D array of normals as well. v[i] is a polygon. v[i][j] is a vertex. Each vertex must have the same length. Each polygon must have at least three vertices. There must be a normal for each vertex. Each normal must have 3 components. (Usually the normals are unit vectors.)

45.11

DVpolygons& **set_data** (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2)

Three nodes given–creates one triangle.

Three nodes given–creates one triangle. Each LCVMarray v_i must have the same size.

45.12

DVpolygons& **set_data** (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray<float> &n0, LCVMarray<float> &n1, LCVMarray<float> &n2)

Three nodes given and 3 normals given–creates one triangle.

Three nodes given and 3 normals given–creates one triangle. Each LCVMarray v_i must have the same size and each LCVMarray n_i must have size 3. (Usually n_i is a unit vector.).

45.13

DVpolygons& **set_data** (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray_2D<float> &n)

Three nodes given and an array of 3 normals given–creates one triangle.

Three nodes given and an array of 3 normals given–creates one triangle. Each LCVMarray v_i must have the same size and n must be 3x3. n[i] is the normal for vertex i. (Usually n[i] is a unit vector.)

45.14

DVpolygons& **set_data** (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray<float> &v3)

Four nodes given–creates one quadrilateral.

Four nodes given–creates one quadrilateral. Each LCVMarray must have the same size.

45.15

DVpolygons& **set_data** (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray<float> &v3, LCVMarray<float> &n0, LCVMarray<float> &n1, LCVMarray<float> &n2, LCVMarray<float> &n3)

Four nodes given and 4 normals given–creates one quadrilateral.

Four nodes given and 4 normals given–creates one quadrilateral. each LCVMarray v_i must have the same size and each LCVMarray n_i must have size 3. (Usually n_i is a unit vector.)

45.16

DVpolygons& **set_data** (LCVMarray<float> &v0, LCVMarray<float> &v1, LCVMarray<float> &v2, LCVMarray<float> &v3, LCVMarray_2D<float> &n)

Four nodes given and an array of 4 normals given–creates one quadrilateral.

Four nodes given and an array of 4 normals given–creates one quadrilateral. each LCVMarray v_i must have the same size and n must be 4x3. n[i] is the normal for vertex i. (Usually n[i] is a unit vector.)

45.17

DVpolygons& **set_data** (LCVMarray_2D<float> &v)

Array of nodes given–v[i] is a vertex.

Array of nodes given–v[i] is a vertex. v must be rectangular. v[i] must have size at least 3.

45.18

DVpolygons& **set_data** (LCVMarray_2D<float> &v, LCVMarray_2D<float> &n)

Array of nodes given and array of normals given–v[i] is a vertex.

Array of nodes given and array of normals given–v[i] is a vertex. v must be rectangular. The (leading) size of v must be the same as the (leading) size of n. n[i] must have size 3. v[i] must have size at least 3.

45.19

DVpolygons& **set_data** (LCVMarray_3D<float> &v)

2_D array of nodes given–v[i] is a polygon.

2_D array of nodes given–v[i] is a polygon. v[i][j] is a vertex. Each vertex must have the same length. Each polygon must have at least three vertices.

45.20

DVpolygons& **set_data** (LCVMarray_3D<float> &v, LCVMarray_3D<float> &n)

2_D array of nodes given and 2_D array of normals as well.

2_D array of nodes given and 2_D array of normals as well. v[i] is a polygon. v[i][j] is a vertex. Each vertex must have the same length. Each polygon must have at least three vertices. There must be a normal for each vertex. Each normal must have 3 components. (Usually the normals are unit vectors.)

46

class **DVproperty**

A class to contain the properties of an object.

Public Members

		DVproperty ()	*Default constructor*	
		DVproperty (const DVproperty&)	*Copy constructor*	
		~DVproperty ()	*Destructor*	
	const DVproperty&	**operator=** (const DVproperty&)	*Assignment operator*	
	int	**operator==** (const DVproperty&) const	*Check equality*	
	int	**operator!=** (const DVproperty&) const	*Check inequality*	
	void	**print** (ostream&)	*Print routine*	
46.1	int	**frame_number**	*Used to keep track of which frames to draw for selection containers.*	149
46.2	DVcolor_property_item_array	**color**	*The array of colors.*	149
46.3	DVpositive_float_property_item_array	**width**	*The array of widhts.*	150
46.4	DVcomplexity_int_property_item	**geometry_complexity**	*The geometry complexity.*	150
46.5	DVcomplexity_int_property_item	**light_complexity**	*The light complexity.*	150
46.6	DVscale_property_item	**scale**	*The scale.*	150
46.7	DVcolor_by_data_property_item_array	**color_by_data**	*The array of color_by_data objects.*	151
46.8	DVlight_properties_property_item_array	**lights**	*The array of light properties.* ...	151
46.9	DVmaterial_properties_property_item_array			

material *The array of material properties.*

151

A class to contain the properties of an object. The DVproperty class is used for the data that gets put on the stack at each level as we traverse the tree structure. we also make (almost) everything public since this is basically a struct... Also, since in general, only the various draw routines and the routine to make this "stack entry" from the properties of the DVobject will have access to this, we can get by without being so protective. There is not a lot of stuff to mess up though–the colors take care of themselves for example.

we have a constructor and a destructor. Also since we will most often copy the current top of the stack to make a new top, we will need a copy constructor.

46.1

int **frame_number**

Used to keep track of which frames to draw for selection containers.

Used to keep track of which frames to draw for selection containers. It does not come from DVselect_property.

46.2

DVcolor_property_item_array **color**

The array of colors.

The array of colors. It is an array since some types of objects use more then one. It comes from DVselect_property.

46.3

DVpositive_float_property_item_array **width**

The array of widhts.

The array of widhts. It is an array since some types of objects use more then one. It comes from DVselect_property.

46.4

DVcomplexity_int_property_item **geometry_complexity**

The geometry complexity.

The geometry complexity. It comes from DVselect_property.

46.5

DVcomplexity_int_property_item **light_complexity**

The light complexity.

The light complexity. It comes from DVselect_property.

46.6

DVscale_property_item **scale**

The scale.

The scale. It comes from DVselect_property.

46.7

DVcolor_by_data_property_item_array **color_by_data**

The array of color_by_data objects.

The array of color_by_data objects. It is an array since some types of objects use more then one. It comes from DVselect_property.

46.8

DVlight_properties_property_item_array **lights**

The array of light properties.

The array of light properties. It is an array since some types of objects use more then one. It comes from DVselect_property.

46.9

DVmaterial_properties_property_item_array **material**

The array of material properties.

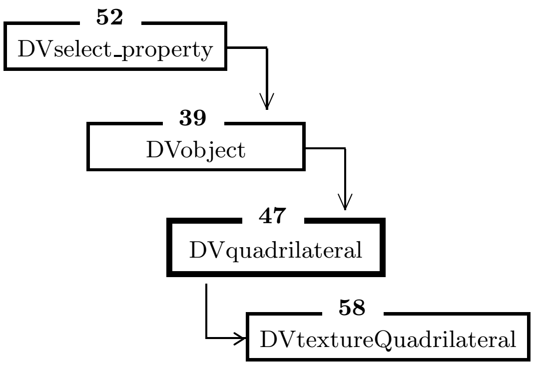
The array of material properties. It is an array since some types of objects use more then one. It comes from DVselect_property.

47

class **DVquadrilateral** : public DVobject

A simple class for a quadrilateral.

Inheritance



Public Members

47.1 **DVquadrilateral** ()
Constructor. 158

47.2 **DVquadrilateral** (const float *const,
const float *const,
const float *const,
const float *const)
Constructor. 158

47.3 **DVquadrilateral** (const float[][3])
Constructor. 159

47.4 **DVquadrilateral** (const float *const,
const float *const,
const float *const,
const float *const,
const float *const,
const float *const,
const float *const,
const float *const)

virtual **~DVquadrilateral** ()
Destructor

DVquadrilateral&
set_data () *Functions to set the nodes and normals – no data given*

DVquadrilateral&
set_data (const float *const, const float *const,
const float *const, const float *const)
Four nodes given, no normals given

DVquadrilateral&
set_data (const float[][3])
An array of four nodes given, no normals given

DVquadrilateral&

set_data (const float *const, const float *const,
const float *const, const float *const,
const float *const, const float *const,
const float *const, const float *const)
Four nodes and four normal vectors given

DVquadrilateral&
set_data (const float *const, const float *const,
const float *const, const float *const,
const float *const)
Four nodes and one normal vector given

DVquadrilateral&
set_data (const float *const, const float *const,
const float *const, const float *const,
const float[][3])
Four nodes and an array of four normal vectors given

DVquadrilateral&
set_data (const float[][3], const float *const,
const float *const, const float *const,
const float *const)
An array with the four nodes and four normal vectors given

DVquadrilateral&
set_data (const float[][3], const float *const)
An array with the four nodes and one normal vector given

DVquadrilateral&
set_data (const float[][3], const float[][3])
An array with the four nodes and an array with four normal vectors

virtual void
draw (LCVMstack<DVproperty*>&)
Virtual draw routine

DVquadrilateral (const DVquadrilateral &data)
Copy constructor

DVquadrilateral&
operator= (const DVquadrilateral &data)
Assignment operator

virtual void

print (ostream&) *A virtual print function*

friend ostream&
operator<< (ostream &, DVquadrilateral &)
A friend routine to overload the << *operator*

inline **DVquadrilateral** ()
first one makes a "default" quadrilateral. (no nodes or normals)

inline **DVquadrilateral** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3)
four nodes and no normals given

inline **DVquadrilateral** (const float p[][3])
an array of four nodes but no normals given

inline **DVquadrilateral** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3,
const float *const n0,
const float *const n1,
const float *const n2,
const float *const n3)
four nodes and four normal vectors given

inline **DVquadrilateral** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3,
const float *const n)
four nodes and one normal vector given

inline **DVquadrilateral** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3,
const float n[][3])

four nodes and an array of four normal vectors given

inline **DVquadrilateral** (const float p[][3],
const float *const n0,
const float *const n1,
const float *const n2,
const float *const n3)
an array with the four nodes and four normal vectors given

inline **DVquadrilateral** (const float p[][3],
const float *const n)
an array with the four nodes and one normal vector given

inline **DVquadrilateral** (const float p[][3],
const float n[][3])
an array with the four nodes and an array with four normal vectors

inline DVquadrilateral&
set_data () *no data given*

inline DVquadrilateral&
set_data (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3)
four nodes given, no normals given

inline DVquadrilateral&
set_data (const float p[][3])
an array of four nodes given, no normals given

inline DVquadrilateral&

set_data (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3,
const float *const n0,
const float *const n1,
const float *const n2,
const float *const n3)
four nodes and four normal vectors given

inline DVquadrilateral&
set_data (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3,
const float *const n)
four nodes and one normal vector given

inline DVquadrilateral&
set_data (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3, const float n[][3])
four nodes and an array of four normal vectors given

inline DVquadrilateral&
set_data (const float p[][3], const float *const n0,
const float *const n1,
const float *const n2,
const float *const n3)
an array with the four nodes and four normal vectors given

inline DVquadrilateral&
set_data (const float p[][3], const float *const n)
an array with the four nodes and one normal vector given

inline DVquadrilateral&
set_data (const float p[][3], const float n[][3])
an array with the four nodes and an array with four normal vectors

inline **~DVquadrilateral** ()
destructor

Protected Members

float	**nodes** [4][3]	*First row is node 0 – third column is z0, z1, z2, z4*
float	**normals** [4][3]	*First row is normal 0 – third column is nz0, ..., nz4*
virtual DVobject*	**copy** ()	*A routine that returns a pointer to a copy of the object created with new.*

A simple class for a quadrilateral. This is a simple class which draws a single quadrilateral. It is meant more as an example then as a class to use for large applications.

47.1

DVquadrilateral ()

Constructor.

Constructor. Default constructor, no data given

47.2

DVquadrilateral (const float *const, const float *const, const float *const, const float *const)

Constructor.

Constructor. Four nodes given

47.3

DVquadrilateral (const float[][3])

Constructor.

Constructor. Array of four nodes given

47.4

DVquadrilateral (const float *const, const float *const, const float *const, const float *const, const float *const, const float *const, const float *const, const float *const)

Constructor.

Constructor. Four nodes and four normal vectors given

47.5

DVquadrilateral (const float *const, const float *const, const float *const, const float *const, const float *const)

Constructor.

Constructor. Four nodes and one normal vector given

47.6

DVquadrilateral (const float *const, const float *const, const float *const, const float *const, const float[][3])

Constructor.

Constructor. Four nodes and an array of four normal vectors given

47.7

DVquadrilateral (const float[][3], const float *const, const float *const, const float *const, const float *const)

Constructor.

Constructor. An array with the four nodes and four normal vectors given

47.8

DVquadrilateral (const float[][3], const float *const)

Constructor.

Constructor. An array with the four nodes and one normal vector given

47.9

DVquadrilateral (const float[][3], const float[][3])

Constructor.

Constructor. An array with the four nodes and an array with four normal vectors

48 class **DVrich_light_properties**

A class for a rich light property.

Public Members

DVrich_light_properties ()
The default constructor

DVrich_light_properties (const DVrich_light_properties&)
Copy constructor from another DVrich_light_properties

DVrich_light_properties (const DVlight_properties&)
Copy constructor from a DVlight_properties

~DVrich_light_properties ()
Destructor

const DVrich_light_properties& **operator=** (const DVrich_light_properties&)
Assignment operator from another DVrich_light_properties

const DVrich_light_properties& **operator=** (const DVlight_properties&)
Assignment operator from a DVlight_properties

int **operator==** (const DVrich_light_properties&) const
Check equality

int **operator!=** (const DVrich_light_properties&) const
Check inequality

int **is_spotlight** () const
Return 1 if it is a spotlight

int **is_positional** () const
Return 1 if it is a position (i.e. point or directional) light

void **print** (ostream&) const

		Print routine
int	**active**	*Contains 1 if the light is active*
DVcolor_type	**ambient**	*Ambient color of light*
DVcolor_type	**diffuse**	*Diffuse color of light*
DVcolor_type	**specular**	*Specular color of light*
DVlight_position_type	**position**	*Light position*
DVspotlight_direction_type	**spot_direction**	*Spotlight direction*
DVlighting_exponent_float_type	**spot_exponent**	*Spoltlight exponent*
DVspotlight_cutoff_float_type	**spot_cutoff**	*Spotlight cutoff angle*
DVnon_negative_float_type	**constant_attenuation**	*Spotlight constant attenuation*
DVnon_negative_float_type	**linear_attenuation**	*Spotlight linear attenuation*
DVnon_negative_float_type	**quadratic_attenuation**	*Spotlight quadratic attenuation*

A class for a rich light property. The difference between a rich property and a normal on is that the rich property may either be "explicit" in which case the data for the property is stored locally, or "named" in which case the data for the property is stored in a global object. The idea is that when the global object is modified, then so are all of the objects which point at that table.

49

class **DVrich_material_properties**

A class to encapsulate a set of rich material properties.

Public Members

DVrich_material_properties ()
Constructor

DVrich_material_properties (const DVrich_material_properties&)
Copy constructor from another DVrich_material_properties

DVrich_material_properties (const DVmaterial_properties&)
Copy constructor from a DVmaterial_properties

~DVrich_material_properties ()
Destructor

const DVrich_material_properties& **operator=** (const DVrich_material_properties&)
Assignment operator from another DVrich_material_properties

const DVrich_material_properties& **operator=** (const DVmaterial_properties&)
Assignment operator from a DVmaterial_properties

int **operator==** (const DVrich_material_properties&) const
Check equality

int **operator!=** (const DVrich_material_properties&) const
Check inequality

void **print** (ostream &output) const
Print function

DVcolor_type **ambient_factors** *The ambient color*

DVcolor_type

diffuse_factors *The diffuse color*

DVcolor_type
specular_factors *The specular color*

DVcolor_type
emission_factors *The emission color*

DVlighting_exponent_float_type
shininess *The shininess exponent*

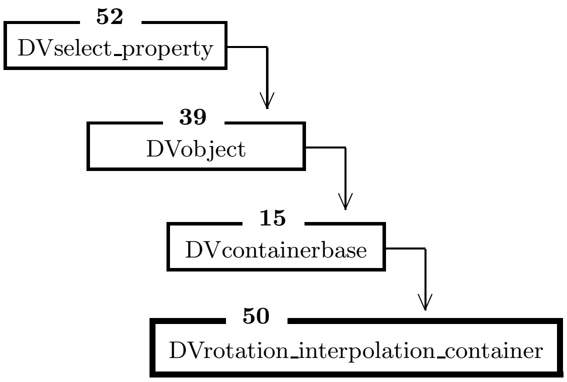
A class to encapsulate a set of rich material properties. Rich properities are those that can either be set explicitly (i.e. the data for the property resides within the object itself) or named (i.e. the data for the property resides in a global table whose keys are strings).

50

class **DVrotation_interpolation_container** : public DVcontainerbase

DEVELOPER: A container which can animate over a range of rotations

Inheritance



Public Members

DVrotation_interpolation_container (const int steps=1)
Constructors for this class

virtual **~DVrotation_interpolation_container** ()
correct decendant destructor will get called.

virtual void **draw** (LCVMstack<DVproperty*>&)
correct decendant routine gets called.

DVrotation_interpolation_container (DVrotation_interpolation_container &data)
We also want a copy constructor as well as an assignment operator

const DVrotation_interpolation_container&

operator= (DVrotation_interpolation_container &data)

virtual void
print (ostream &) *print the rotation_interpolation_container – make it virtual.*

friend ostream&
operator<< (ostream &, DVrotation_interpolation_container &)
A friend routine to overload the << *operator*

int **ReadAnimateProp** ()
Routines concerning the scene file format

DVrotation_interpolation_container (FileOption a_file_op)

Protected Members

virtual DVobject*
copy () *of the object created with new.*

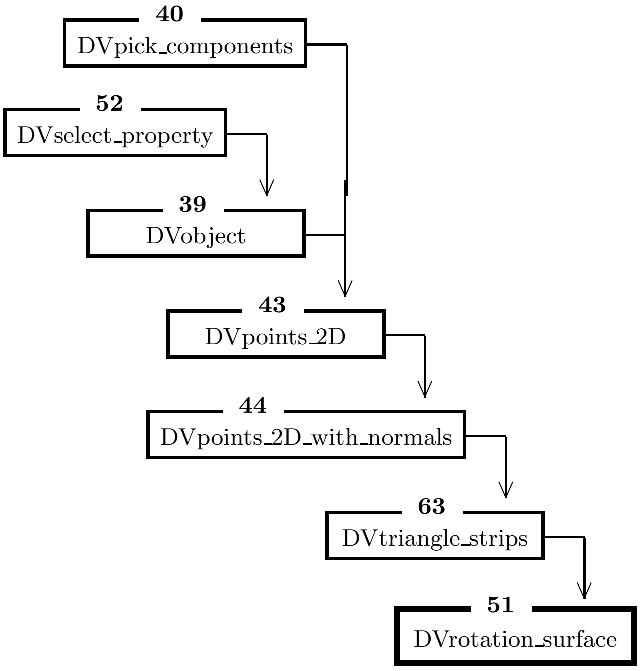
DEVELOPER: A container which can animate over a range of rotations

51

class **DVrotation_surface** : public DVtriangle_strips

A class to transform curves into rotation surfaces

Inheritance



Public Members

51.1 **DVrotation_surface** (DVlines &data,
const float end_angle = 360.0,
char degrees='d')

Constructor with a curve as input, and with an angle in degrees (by default) or in radians (last parameter equal to 'r') 171

51.2 **DVrotation_surface** (LCVMarray_2D<float>
&line_points,
LCVMarray<float>
&the_colors,
const float end_angle = 360.0,
char degrees = 'd')
Similar to the DVlines constructor, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r') 171

51.3 **DVrotation_surface** (LCVMarray_2D<float>
&line_points,
const float end_angle = 360.0,
char degrees = 'd')
Similar to the DVlines constructor, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r') 171

51.4 void **set_data** (DVlines &data,
const float end_angle = 360.0,
char degrees = 'd')
Set data starting with the curve (data) that has to be turned into a rotation surface. 172

51.5 void **set_data** (LCVMarray_2D<float> &line_points,
LCVMarray<float> &the_colors,
const float end_angle = 360.0,
char degrees = 'd')
Similar to the DVlines set_data, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r') 172

51.6 void **set_data** (LCVMarray_2D<float> &line_points,
const float end_angle = 360.0,
char degrees = 'd')

Similar to the DVlines set_data, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r') 172

virtual **~DVrotation_surface** ()
the destructor

virtual void
draw (LCVMstack<DVproperty*>&)
The draw function

51.7 **DVrotation_surface** (FileOption a_file_op)
The constructor from the input scene file 173

51.8 virtual int**ReadMyProp** () *Specific properties for this class: none, but it has to be specified as it is virtual* 173

Protected Members

51.9 virtual DVobject*
copy () *We also want a routine that returns a pointer to a copy of the object created with new* 173

DVrotation_surface (const DVrotation_surface &data)
The copy constructor

const DVrotation_surface&
operator= (const DVrotation_surface &data)
The assignment operator

A class to transform curves into rotation surfaces

51.1

DVrotation_surface (DVlines &data, const float end_angle = 360.0, char degrees='d')

Constructor with a curve as input, and with an angle in degrees (by default) or in radians (last parameter equal to 'r')

Constructor with a curve as input, and with an angle in degrees (by default) or in radians (last parameter equal to 'r')

51.2

DVrotation_surface (LCVMarray_2D<float> &line_points, LCVMarray<float> &the_colors, const float end_angle = 360.0, char degrees = 'd')

Similar to the DVlines constructor, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r')

Similar to the DVlines constructor, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r')

51.3

DVrotation_surface (LCVMarray_2D<float> &line_points, const float end_angle = 360.0, char degrees = 'd')

Similar to the DVlines constructor, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r')

Similar to the DVlines constructor, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r')

51.4

void **set_data** (DVlines &data, const float end_angle = 360.0, char degrees = 'd')

Set data starting with the curve (data) that has to be turned into a rotation surface.

Set data starting with the curve (data) that has to be turned into a rotation surface. Needs an angle in degrees (by default) or in radians (last parameter equal to 'r').

51.5

void **set_data** (LCVMarray_2D<float> &line_points, LCVMarray<float> &the_colors, const float end_angle = 360.0, char degrees = 'd')

Similar to the DVlines set_data, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r')

Similar to the DVlines set_data, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r')

51.6

void **set_data** (LCVMarray_2D<float> &line_points, const float end_angle = 360.0, char degrees = 'd')

Similar to the DVlines set_data, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r')

Similar to the DVlines set_data, but needs an angle in degrees (by default) or in radians (last parameter equal to 'r')

51.7

DVrotation_surface (FileOption a_file_op)

The constructor from the input scene file

The constructor from the input scene file

51.8

virtual int **ReadMyProp** ()

Specific properties for this class: none, but it has to be specified as it is virtual

Specific properties for this class: none, but it has to be specified as it is virtual

51.9

virtual DVobject* **copy** ()

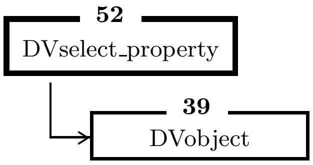
We also want a routine that returns a pointer to a copy of the object created with new

We also want a routine that returns a pointer to a copy of the object created with new

52 class **DVselect_property**

A class which contains a set of properties and can differeniate those that have been explicitly from those that haven't.

Inheritance



Public Members

52.1 DVcolor_select_property_item_array
color *An array of colors.* 176

52.2 DVpositive_float_select_property_item_array
width *An array of widths.* 176

52.3 DVcomplexity_int_select_property_item
geometry_complexity
The geometry complexity of the object 176

52.4 DVcomplexity_int_select_property_item
light_complexity *The light complexity of the object*
177

52.5 DVscale_select_property_item
scale *The scale of the object.* 177

52.6 DVcolor_by_data_select_property_item_array
color_by_data *An array of color_by_data objects.*
.... 177

52.7 DVrich_light_properties_select_property_item_array
lights *An array of light_properties.* 177

52.8 DVrich_material_properties_select_property_item_array
material *An array of material_properties.* 178

DVselect_property ()
Default constructor

DVselect_property (const DVselect_property&)

Copy constructor

~DVselect_property ()
Destructor

const DVselect_property&
operator= (const DVselect_property&)
Assignment operator

int **operator==** (const DVselect_property&) const
Check equality

int **operator!=** (const DVselect_property&) const
Check inequality

52.9 void **extract_properties** (DVproperty&)
Given a DVproperty this function modifies those properties in the DVproperty that have not been explicitly set in the DVselect_property. 178

void **print** (ostream&) *Print function*

52.10 void **default_lighting** ()
Turn on the default lighting. . . . 178

int **ReadProp** () *Read the object properties from a scene file*

void **ReadDVColor** () *Read the object properties from a scene file*

void **ReadDVlightComp** ()
Read the object properties from a scene file

void **ReadDVgeomComp** ()
Read the object properties from a scene file

void **ReadDVscale** () *Read the object properties from a scene file*

void **ReadDVwidth** () *Read the object properties from a scene file*

void **ReadColorMap** ()
Read the object properties from a scene file

A class which contains a set of properties and can differeniate those that have been explicitly from those that haven't. All the data is public since this should act like a struct. Each of the data does have its own checks on the data though.

52.1

DVcolor_select_property_item_array **color**

An array of colors.

An array of colors. The exact use of the colors is up to the class which inherits from this class. For example, some classes might only use the first entry, while others might use more.

52.2

DVpositive_float_select_property_item_array **width**

An array of widths.

An array of widths. The exact use of the widths is up to the class which inherits from this class. For example, some classes might only use the first entry, while others might use more.

52.3

DVcomplexity_int_select_property_item **geometry_complexity**

The geometry complexity of the object

The geometry complexity of the object

52.4

DVcomplexity_int_select_property_item **light_complexity**

The light complexity of the object

The light complexity of the object

52.5

DVscale_select_property_item **scale**

The scale of the object.

The scale of the object.

52.6

DVcolor_by_data_select_property_item_array
color_by_data

An array of color_by_data objects.

An array of color_by_data objects. They allow the object to be colored by some component of its data. The exact use of the colors_by_data objects is up to the class which inherits from this class. For example, some classes might only use the first entry, while others might use more.

52.7

DVrich_light_properties_select_property_item_array **lights**

An array of light_properties.

An array of light_properties. The exact use of the light_properties is up to the class which inherits from this class. For example, some classes might only use the first entry, while others might use more.

52.8

DVrich_material_properties_select_property_item_array **material**

An array of material_properties.

An array of material_properties. The exact use of the material_properties is up to the class which inherits from this class. For example, some classes might only use the first entry, while others might use more.

52.9

void **extract_properties** (DVproperty&)

Given a DVproperty this function modifies those properties in the DVproperty that have not been explicitly set in the DVselect_property.

Given a DVproperty this function modifies those properties in the DVproperty that have not been explicitly set in the DVselect_property. For example, a DVselect_property which has had its color member modified will only modify the color member of the DVproperty, and leave the other members of the DVproperty unchanged.

52.10

void **default_lighting** ()

Turn on the default lighting.

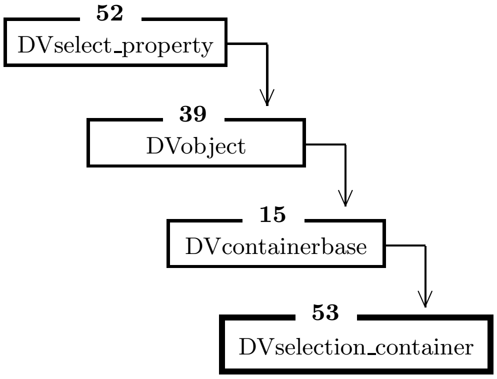
Turn on the default lighting. The default lighting consists of light 0 being turned on, at (0,0,0) with ambient(0.2,0.2,0.2,1.0), diffuse(0.8,0.8,0.8,1.0), and specular(0.0,0.0,0.0,1.0).

53

class **DVselection_container** : public DVcontainerbase

A contatiner object which can be used to create animations.

Inheritance



Public Members

53.5 **DVselection_container** (const int first=0, const int step=1, const int do_not_animate=0)
Constructor. 181

53.6 virtual **~DVselection_container** ()
Default destructor. 181

void **set_first_frame_number** (const int first=0)

void **set_step_frame_number** (const int step=1)

void **set_do_not_animate_flag** (const int do_not_animate=0)

int **get_first_frame_number** () const

int **get_step_frame_number** () const

int **get_do_not_animate_flag** () const

53.7 virtual void

draw (LCVMstack<DVproperty*>&)
The draw routine. 181

DVselection_container (DVselection_container &data)
Copy constructor

const DVselection_container&
operator= (DVselection_container &data)
An assignment operator

virtual void
print (ostream &) *Print the selection_container – make it virtual.*

friend ostream&
operator<< (ostream &, DVselection_container &)
A friend routine to overload the << *operator*

int **ReadAnimateProp** ()
Routines concerning the scene file format

DVselection_container (FileOption a_file_op)
Routines concerning the scene file format

Protected Members

53.2 int **first_frame_number**
The frame number of the first child. 182

53.3 int **step_frame_number**
The number of frames to stay on each child. 182

53.4 int **do_not_animate_flag**
If this is non-zero draw all children, just like a normal DVcontainer 182

virtual DVobject*
copy () *A routine that returns a pointer to a copy of the object created with new.*

A contatiner object which can be used to create animations. A DVselection_container draws one of its children at a time, based upon the frame_number give in the DVproperty item in the draw() function.

53.5

DVselection_container (const int first=0, const int step=1, const int do_not_animate=0)

Constructor.

Constructor. It allows you to get the first_frame_number, step_frame_number, and do_not_animate_flag.

53.6

virtual **~DVselection_container** ()

Default destructor.

Default destructor. It is declared virtual so that the correct decendant destructor will get called.

53.7

virtual void **draw** (LCVMstack<DVproperty*>&)

The draw routine.

The draw routine. We again want it to be virtual so that the correct decendant routine gets called.

53.2

int **first_frame_number**

The frame number of the first child.

The frame number of the first child. This frame number, and all lower frame numbers, will map into the first child

53.3

int **step_frame_number**

The number of frames to stay on each child.

The number of frames to stay on each child.

53.4

int **do_not_animate_flag**

If this is non-zero draw all children, just like a normal DVcontainer

If this is non-zero draw all children, just like a normal DVcontainer

54

class **DVsimple_point**

A class which defines point in three dimensional space

Public Members

float	**x**	*The x component*
float	**y**	*The y component*
float	**z**	*The z component*
	DVsimple_point (const float x_val=+0.0, const float y_val=+0.0, const float z_val=+0.0)	*Constructor*
	~DVsimple_point ()	*Destructor*
	DVsimple_point (const DVsimple_point&)	*Copy constructor*
const DVsimple_point&	**operator=** (const DVsimple_point&)	*Assignment operator*
int	**operator==** (const DVsimple_point&) const	*Check equality*
int	**operator!=** (const DVsimple_point&) const	*Check inequality*
void	**set_data** (const float x_val=+0.0, const float y_val=+0.0, const float z_val=+0.0)	*Set the data from 3 floating point values*
void	**set_data** (const DVsimple_point&)	*Set the data from another DVsimple_point*
void	**get_data** (float&, float&, float&) const	*Get the data into 3 floating point variables*
void	**get_data** (DVsimple_point&) const	

Copy the data into a DVsimple_point

float& **operator**[] (const int)
We want foo[0] to be foo.x, ... foo[2] to be foo.z

inline **DVsimple_point** (const float x_val,
const float y_val,
const float z_val)
constructor

inline **DVsimple_point** (const DVsimple_point &data)
copy constructor

inline void
get_data (float &x_val, float &y_val,
float &z_val) const

inline void
get_data (DVsimple_point &r) const

inline int **operator!=** (const DVsimple_point &data) const
check not equal

inline const DVsimple_point&
operator= (const DVsimple_point &data)
assignment operator

inline int **operator==** (const DVsimple_point &data) const
check equal

inline void
set_data (const float x_val, const float y_val,
const float z_val)
set_data routine

inline void
set_data (const DVsimple_point &pt)

inline **~DVsimple_point** ()
destructor (does nothing)

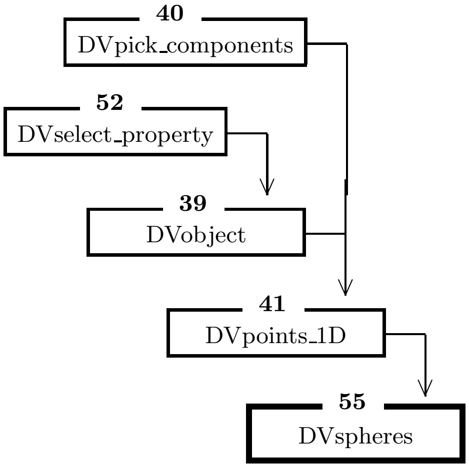
A class which defines point in three dimensional space

55

class **DVspheres** : public DVpoints_1D

A collection of spheres.

Inheritance



Public Members

55.1 **DVspheres** (const float r=1.0)
Constructor. 187

55.2 **DVspheres** (const float x, const float y, const float z, const float r=1.0)
Constructor. 187

55.3 **DVspheres** (LCVMarray<float> point, const float r=1.0)
Constructor. 188

55.4 **DVspheres** (LCVMarray_2D<float> coords, const float r=1.0)
Constructor. 188

virtual **~DVspheres** () *Destructor*

DVspheres&

set_radius (const float r=1.0)
Set the radius for all of the spheres

float **get_radius** () *Get the radius*

virtual void
draw (LCVMstack<DVproperty*>&)
Draw routine

DVspheres (const DVspheres &data)
Copy constructor

const DVspheres&
operator= (const DVspheres &data)
Assignment operator

55.5 void **set_data** (const float r=1.0)
Change the data in the DVsphere object. 188

55.6 void **set_data** (const float x, const float y, const float z, const float r=1.0)
Change the data in the DVsphere object. 189

55.7 void **set_data** (LCVMarray<float> point, const float r=1.0)
Change the data in the DVsphere object. 189

55.8 void **set_data** (LCVMarray_2D<float> coords, const float r=1.0)
Change the data in the DVsphere object. 189

int **operator==** (const DVspheres &data)
Check for equality

int **operator!=** (const DVspheres &data)
Check for inequality

virtual void
print (ostream&) *A virtual print function*

friend ostream&
operator<< (ostream &, DVspheres &)
A friend routine to overload the << operator

DVspheres (FileOption a_file_op)

		Methods concerning the scene file format
virtual int	**ReadMyProp** ()	*Methods concerning the scene file format*
void	**print** (ostream &, FileOption)	*Methods concerning the scene file format*

Protected Members

virtual DVobject*	**copy** ()	*A routine that returns a pointer to a copy of the object created with new.*

A collection of spheres. This class provides a collection of spheres. They can all have different centers, but they all share the same properties (radius, explicit color, etc.).

55.1

DVspheres (const float r=1.0)

Constructor.

Constructor. A single sphere, centered at the origin, with the given radius

55.2

DVspheres (const float x, const float y, const float z, const float r=1.0)

Constructor.

Constructor. A single sphere, centered at (x,y,z), with the given radius

55.3

DVspheres (LCVMarray<float> point, const float r=1.0)

Constructor.

Constructor. A single sphere, centered at the coordinates given in the array, with the given radius

55.4

DVspheres (LCVMarray_2D<float> coords, const float r=1.0)

Constructor.

Constructor. Multiple spheres, each centered at one of the sets of coordinates given in the array, with the given radius

55.5

void **set_data** (const float r=1.0)

Change the data in the DVsphere object.

Change the data in the DVsphere object. A single sphere, centered at the origin, with the given radius

55.6

void **set_data** (const float x, const float y, const float z, const float r=1.0)

Change the data in the DVsphere object.

Change the data in the DVsphere object. A single sphere, centered at (x,y,z), with the given radius

55.7

void **set_data** (LCVMarray<float> point, const float r=1.0)

Change the data in the DVsphere object.

Change the data in the DVsphere object. A single sphere, centered at the coordinates given in the array, with the given radius

55.8

void **set_data** (LCVMarray_2D<float> coords, const float r=1.0)

Change the data in the DVsphere object.

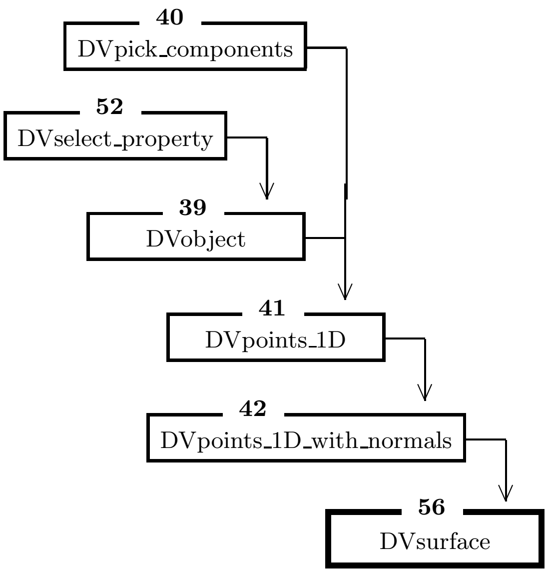
Change the data in the DVsphere object. Multiple spheres, each centered at one of the sets of coordinates given in the array, with the given radius

56

class **DVsurface** : public DVpoints_1D_with_normals

This is a class of simple surfaces.

Inheritance



Public Members

DVsurface () *Default constructor, makes an empty surface.*

56.1 **DVsurface** (LCVMarray<int> &nn, LCVMarray_2D<float> &v, LCVMarray<int> &en, LCVMarray_2D<int> &p)
Constructor with 4 LCVM arrays. 192

56.2 **DVsurface** (LCVMarray<int> &nn, LCVMarray_2D<float> &v, LCVMarray_2D<float> &nv, LCVMarray<int> &en, LCVMarray_2D<int> &p)

Constructor with 5 LCVM arrays. 193

virtual **~DVsurface** () *Destructor*

DVsurface&
set_data () *Set data to empty surface.*

56.3 DVsurface&
set_data (LCVMarray<int> &nn,
LCVMarray_2D<float> &v,
LCVMarray<int> &en,
LCVMarray_2D<int> &p)
Set data with 4 LCVM arrays. . 193

56.4 DVsurface&
set_data (LCVMarray<int> &nn,
LCVMarray_2D<float> &v,
LCVMarray_2D<float> &nv,
LCVMarray<int> &en,
LCVMarray_2D<int> &p)
Set data with 5 LCVM arrays. . 193

virtual void
draw (LCVMstack<DVproperty*>&)
Draw routine

void **compute_normals** (DVscale &scale)
Compute smooth normals

DVsurface (const DVsurface &data)
Copy constructor

const DVsurface&
operator= (const DVsurface &data)
Assignment operator

virtual void
print (ostream&) *a virtual print function*

int **is_faceted** () const
Returns 1 is the surface has smooth normals, and 0 otherwise

void **set_faceted** () *Do not use smooth normals*

void **set_smooth** () *Use smooth normals if available*

DVsurface (FileOption a_file_op)

		Methods concrening the scene file format
virtual	int**ReadMyProp** ()	*Methods concrening the scene file format*

Protected Members

virtual	DVobject* **copy** ()	*A routine that returns a pointer to a copy of the object created with new.*

This is a class of simple surfaces. The surface is made up polygons.

56.1

DVsurface (LCVMarray<int> &nn, LCVMarray_2D<float> &v, LCVMarray<int> &en, LCVMarray_2D<int> &p)

Constructor with 4 LCVM arrays.

Constructor with 4 LCVM arrays. nn – array of node numbers v – array of nodes en – array of element numbers p – array of polygons nn and v must have the same size. Each node in v must have the same number of components. en and p must have the same size. nn must be sorted (increasing).

56.2

DVsurface (LCVMarray<int> &nn, LCVMarray_2D<float> &v, LCVMarray_2D<float> &nv, LCVMarray<int> &en, LCVMarray_2D<int> &p)

Constructor with 5 LCVM arrays.

Constructor with 5 LCVM arrays. nn – array of node numbers v – array of nodes nv – array of normal vectors en – array of element numbers p – array of polygons nn, v, and nv must have the same size. Each node in v must have the same number of components. each normal vector in nv must have 3 components. Usually nv are unit vectors. en and p must have the same size. nn must be sorted (increasing).

56.3

DVsurface& **set_data** (LCVMarray<int> &nn, LCVMarray_2D<float> &v, LCVMarray<int> &en, LCVMarray_2D<int> &p)

Set data with 4 LCVM arrays.

Set data with 4 LCVM arrays. nn – array of node numbers v – array of nodes en – array of element numbers p – array of polygons nn and v must have the same size. Each node in v must have the same number of components. en and p must have the same size. nn must be sorted (increasing).

56.4

DVsurface& **set_data** (LCVMarray<int> &nn, LCVMarray_2D<float> &v, LCVMarray_2D<float> &nv, LCVMarray<int> &en, LCVMarray_2D<int> &p)

Set data with 5 LCVM arrays.

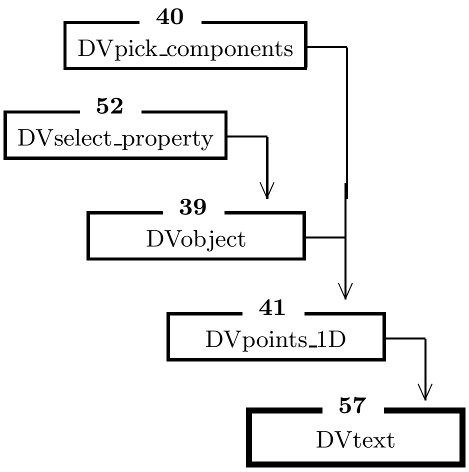
Set data with 5 LCVM arrays. nn – array of node numbers v – array of nodes nv – array of normal vectors en – array of element numbers p – array of polygons nn, v, and nv must have the same size. Each node in v must have the same number of components. each normal vector in nv must have 3 components. Usually nv are unit vectors. en and p must have the same size. nn must be sorted (increasing).

57

class **DVtext** : public DVpoints_1D

DEVELOPER: A object which implements 3D text

Inheritance



Public Members

	DVtext (const LCVMstring input_text="") *Since all arguements are optional this is the default constructor*
void	**set_data** (const LCVMstring input_text="")
	DVtext (const float x, const float y, const float z, const LCVMstring input_text="") *The string is given and it starts at the given x, y, and z values*
void	**set_data** (const float x, const float y, const float z, const LCVMstring input_text="")
	DVtext (LCVMarray<float> point, const LCVMstring input_text="") *The string is given and it starts at the point given in the arrray*
void	**set_data** (LCVMarray<float> point, const LCVMstring input_text="")

virtual **~DVtext** () *destructor*

virtual void
draw (LCVMstack<DVproperty*>&) *virtual draw routine*

DVtext (const DVtext &data) *We also want a copy constructor as well as an assignment operator*

const DVtext&
operator= (const DVtext &data)

virtual void
print (ostream&) *a virtual print function*

DVtext (FileOption a_file_op)

friend ostream&
operator<< (ostream &, DVtext &) *A friend routine to overload the << operator*

inline **~DVtext** () *destructor*

Protected Members

LCVMstring
text

virtual DVobject*
copy () *of the object created with new.*

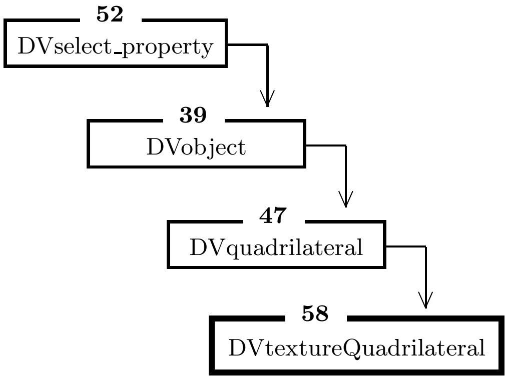
DEVELOPER: A object which implements 3D text

58

class **DVtextureQuadrilateral** : public DVquadrilateral

A class which reads a texture from a file and maps it onto a quadrilateral.

Inheritance



Public Members

DVtextureQuadrilateral (char *filname=NULL)
Default constructor

DVtextureQuadrilateral (const float *const,
const float *const,
const float *const,
const float *const,
char *filname=NULL)
Constructor with four nodes given as float arrays

DVtextureQuadrilateral (const float[][3],
char *filname=NULL)

Constructor with four nodes given as one 2D float array and four normals given as float arrays. .. 202

58.5 **DVtextureQuadrilateral** (const float[][3], const float *const, char *filname=NULL)
Constructor with four nodes given as one 2D float array and one normal given as a float array. 203

58.6 **DVtextureQuadrilateral** (const float[][3], const float[][3], char *filname=NULL)
Constructor with four nodes given as one 2D float array and four normals given as one 2D float array. 203

~DVtextureQuadrilateral ()
Destructor

DVtextureQuadrilateral&
set_filename (char *filname=NULL)
Set the filename from which to read the texture

virtual void
draw (LCVMstack<DVproperty*>&)
Virtual draw routine

DVtextureQuadrilateral (const DVtextureQuadrilateral &data)
Copy constructor

DVtextureQuadrilateral&
operator= (const DVtextureQuadrilateral &data)
Assignment operator

virtual DVobject*
copy ()
A routine that returns a pointer to a copy of the object created with new.

virtual void
print (ostream&)
A virtual print function

friend ostream&

operator<< (ostream &, DVtextureQuadrilateral &)
A friend routine to overload the **<<** *operator*

DVtextureQuadrilateral (FileOption a_file_op)
Methods concerning the file formats, read and write

inline **DVtextureQuadrilateral** (char* filename)
first one makes a "default" textureQuadrilateral.

inline **DVtextureQuadrilateral** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3,
char *filename)
is the coordinates (x, y, z) of one node.

inline **DVtextureQuadrilateral** (const float p[][3],
char* filename)
we just call set_nodes and compute_normals.

inline **DVtextureQuadrilateral** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3,
const float *const n0,
const float *const n1,
const float *const n2,
const float *const n3,
char *filename)
four nodes and four normal vectors given

inline **DVtextureQuadrilateral** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const p3,
const float *const n,
char *filename)

four nodes and one normal vector given

inline **DVtextureQuadrilateral** (const float *const p0, const float *const p1, const float *const p2, const float *const p3, const float n[][3], char *filename)
four nodes and an array of four normal vectors given

inline **DVtextureQuadrilateral** (const float p[][3], const float *const n0, const float *const n1, const float *const n2, const float *const n3, char *filename)
an array with the four nodes and four normal vectors given

inline **DVtextureQuadrilateral** (const float p[][3], const float *const n, char *filename)
an array with the four nodes and one normal vector given

inline **DVtextureQuadrilateral** (const float p[][3], const float n[][3], char *filename)
an array with the four nodes and an array with four normal vectors

inline **~DVtextureQuadrilateral** ()
destructor

Protected Members

char* **texture_filename** *The filename from which the texture is read*

A class which reads a texture from a file and maps it onto a quadrilateral. This class can only read textures from ASCII ppm files.

58.1

DVtextureQuadrilateral (const float *const, const float *const, const float *const, const float *const, const float *const, const float *const, const float *const, const float *const, char *filname=NULL)

Constructor with four nodes given as float arrays and four normals given as float arrays.

Constructor with four nodes given as float arrays and four normals given as float arrays.

58.2

DVtextureQuadrilateral (const float *const, const float *const, const float *const, const float *const, const float *const, char *filname=NULL)

Constructor with four nodes given as float arrays and one normal given as a float array.

Constructor with four nodes given as float arrays and one normal given as a float array.

58.3

DVtextureQuadrilateral (const float *const, const float *const, const float *const, const float *const, const float[][3], char *filname=NULL)

Constructor with four nodes given as float arrays and four normals given as one 2D float array.

Constructor with four nodes given as float arrays and four normals given as one 2D float array.

58.4

DVtextureQuadrilateral (const float[][3], const float *const, const float *const, const float *const, const float *const, char *filname=NULL)

Constructor with four nodes given as one 2D float array and four normals given as float arrays.

Constructor with four nodes given as one 2D float array and four normals given as float arrays.

58.5

DVtextureQuadrilateral (const float[][3], const float *const, char *filname=NULL)

Constructor with four nodes given as one 2D float array and one normal given as a float array.

Constructor with four nodes given as one 2D float array and one normal given as a float array.

58.6

DVtextureQuadrilateral (const float[][3], const float[][3], char *filname=NULL)

Constructor with four nodes given as one 2D float array and four normals given as one 2D float array.

Constructor with four nodes given as one 2D float array and four normals given as one 2D float array.

59 class **DVtips_and_hints**

A class which is used to encapsulate the information sent from DVgeomtry to the user defined scene graph creatation routine.

Public Members

int **get_first_frame_number** () const

int **get_last_frame_number** () const

int **get_animation_optimize_flag** () const

int **get_animation_first_frame_number** () const

int **get_animation_last_frame_number** () const

int **get_animation_step_frame_number** () const

void **get_background_color** (DVcolor&) const

void **get_background_color** (DVcolor_type&) const

float **get_frames_per_second** () const

A class which is used to encapsulate the information sent from DVgeomtry to the user defined scene graph creatation routine.

59.1

DVtips_and_hints& **set_first_frame_number** (const int)

The first frame number of the animation.

The first frame number of the animation. This number is NOT used internally by DVgeometry, it is only provided as information to whatever external program is using the DVgeometry. It should be considered as a hint from the user defined scene graph creation function. Its main use to to allow scene graph creation function to communicate which subset of the animation to draw.

59.2

DVtips_and_hints& **set_last_frame_number** (const int)

The first frame number of the animation.

The first frame number of the animation. This number is NOT used internally by DVgeometry, it is only provided as information to whatever external program is using the DVgeometry. It should be considered as a hint from the user defined scene graph creation function. Its main use to to allow scene graph creation function to communicate which subset of the animation to draw.

59.3

DVtips_and_hints& **set_animation_optimize_flag** (const int)

Turn on optimization of the animation.

Turn on optimization of the animation. This is performed by precomputing a display list for each frame of the animation.

59.4

DVtips_and_hints& **set_animation_first_frame_number** (const int)

Set the animation first frame number.

Set the animation first frame number. This information is used internally by DVgeometry to compute the bounds of the animation.

59.5

DVtips_and_hints& **set_animation_last_frame_number** (const int)

Set the animation last frame number.

Set the animation last frame number. This information is used internally by DVgeometry to compute the bounds of the animation.

59.6

DVtips_and_hints& **set_animation_step_frame_number** (const int)

Set the animation step frame number.

Set the animation step frame number. For example, if this number is 2, then only every other frame will be shown. This information is used internally by DVgeometry to compute the bounds of the animation.

59.7

DVtips_and_hints& **set_background_color** (const DVcolor&)

Set the background color

Set the background color

59.8

DVtips_and_hints& **set_background_color** (const DVcolor_type&)

Set the background color

Set the background color

59.9

DVtips_and_hints& **set_background_color** (const LCVMstring&)

Set the background color

Set the background color

59.10

DVtips_and_hints& **set_background_color** (const DVcolor_float r = 0.0, const DVcolor_float g = 0.0, const DVcolor_float b = 0.0, const DVcolor_float a = 1.0)

Set the background color

Set the background color

59.11

DVtips_and_hints& **set_frames_per_second** (const float)

Set the number of frames to display per second

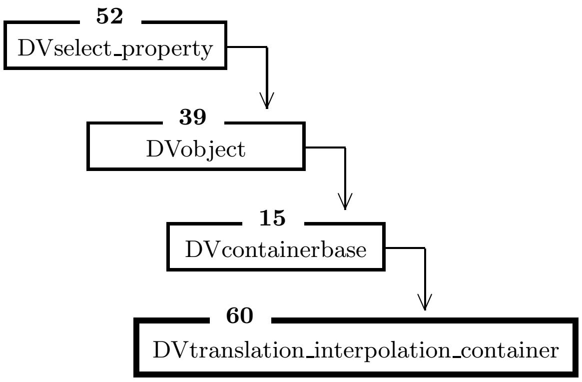
Set the number of frames to display per second

60

class **DVtranslation_interpolation_container** : public DVcontainerbase

DEVELOPER: A container which can animate over a range of translations

Inheritance



Public Members

DVtranslation_interpolation_container (const int steps=1)
Constructors for this class

virtual **~DVtranslation_interpolation_container** ()
correct decendant destructor will get called.

virtual void
draw (LCVMstack<DVproperty*>&)
correct decendant routine gets called.

DVtranslation_interpolation_container (DVtranslation_interpolation_container &data)
We also want a copy constructor as well as an assignment operator

const DVtranslation_interpolation_container&

operator= (DVtranslation_interpolation_container &data)

virtual void
print (ostream &) *print the translation_interpolation_container – make it virtual.*

friend ostream&
operator<< (ostream &, DVtranslation_interpolation_container &)
A friend routine to overload the << *operator*

int **ReadAnimateProp** ()
Routines concerning the scene file format

DVtranslation_interpolation_container (FileOption a_file_op)

Protected Members

virtual DVobject*
copy () *of the object created with new.*

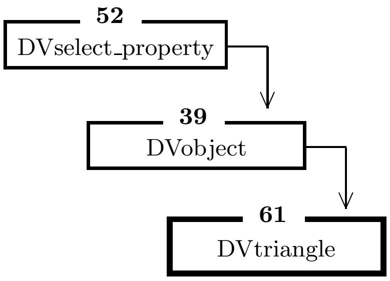
DEVELOPER: A container which can animate over a range of translations

61

class **DVtriangle** : public DVobject

A simple class that displays one triangle.

Inheritance



Public Members

DVtriangle () — *Default constructor, no data given*

DVtriangle (const float *const, const float *const, const float *const)
Three nodes given

DVtriangle (const float[][3])
Array of three nodes given

DVtriangle (const float *const, const float *const, const float *const, const float *const, const float *const, const float *const)
Three nodes and three normal vectors given

DVtriangle (const float *const, const float *const, const float *const, const float *const)
Three nodes and one normal vector given

DVtriangle (const float *const, const float *const, const float *const, const float[][3])
Three nodes and an array of three normal vectors given

DVtriangle (const float[][3], const float *const, const float *const, const float *const)

An array with the three nodes and three normal vectors given

DVtriangle (const float[][3], const float *const)
An array with the three nodes and one normal vector given

DVtriangle (const float[][3], const float[][3])
An array with the three nodes and an array with three normal vectors

virtual **~DVtriangle** () *Destructor*

DVtriangle&
set_data () *Functions to set the nodes and normals, no data given.*

DVtriangle&
set_data (const float *const, const float *const, const float *const)
Three nodes given, no normals given.

DVtriangle&
set_data (const float[][3])
An array of three nodes given, no normals given

DVtriangle&
set_data (const float *const, const float *const, const float *const, const float *const, const float *const, const float *const)
Three nodes and three normal vectors given

DVtriangle&
set_data (const float *const, const float *const, const float *const, const float *const)
Three nodes and one normal vector given

DVtriangle&
set_data (const float *const, const float *const, const float *const, const float[][3])
Three nodes and an array of three normal vectors given

DVtriangle&

set_data (const float[][3], const float *const, const float *const, const float *const)
An array with the three nodes and three normal vectors given

DVtriangle&
set_data (const float[][3], const float *const)
An array with the three nodes and one normal vector given

DVtriangle&
set_data (const float[][3], const float[][3])
An array with the three nodes and an array with three normal vectors

virtual void
draw (LCVMstack<DVproperty*>&)
Virtual draw routine

DVtriangle (const DVtriangle &data)
Copy constructor

DVtriangle&
operator= (const DVtriangle &data)
Assignment operator

virtual void
print (ostream&) *A virtual print function*

DVtriangle (FileOption a_file_op)
A function to read from a scenefile

friend ostream&
operator<< (ostream &, DVtriangle &)
A friend routine to overload the << operator

inline **DVtriangle** () *first one makes a "default" triangle. (no nodes or normals given)*

inline **DVtriangle** (const float *const p0, const float *const p1, const float *const p2)
three nodes and no normals given

inline **DVtriangle** (const float p[][3])

an array of three nodes but no normals given

inline **DVtriangle** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const n0,
const float *const n1,
const float *const n2)
three nodes and three normal vectors given

inline **DVtriangle** (const float *const p0,
const float *const p1,
const float *const p2,
const float *const n)
three nodes and one normal vector given

inline **DVtriangle** (const float *const p0,
const float *const p1,
const float *const p2,
const float n[][3])
three nodes and an array of three normal vectors given

inline **DVtriangle** (const float p[][3], const float *const n0,
const float *const n1,
const float *const n2)
an array with the three nodes and three normal vectors given

inline **DVtriangle** (const float p[][3], const float *const n)
an array with the three nodes and one normal vector given

inline **DVtriangle** (const float p[][3], const float n[][3])
an array with the three nodes and an array with three normal vectors

inline DVtriangle&
set_data () *no data given*

inline DVtriangle&

set_data (const float *const p0,
const float *const p1,
const float *const p2)
three nodes given, no normals given

inline DVtriangle&
set_data (const float p[][3])
an array of three nodes given, no normals given

inline DVtriangle&
set_data (const float *const p0,
const float *const p1,
const float *const p2,
const float *const n0,
const float *const n1,
const float *const n2)
three nodes and three normal vectors given

inline DVtriangle&
set_data (const float *const p0,
const float *const p1,
const float *const p2,
const float *const n)
three nodes and one normal vector given

inline DVtriangle&
set_data (const float *const p0,
const float *const p1,
const float *const p2, const float n[][3])
three nodes and an array of three normal vectors given

inline DVtriangle&
set_data (const float p[][3], const float *const n0,
const float *const n1,
const float *const n2)
an array with the three nodes and three normal vectors given

inline DVtriangle&
set_data (const float p[][3], const float *const n)
an array with the three nodes and one normal vector given

inline DVtriangle&

	set_data (const float p[][3], const float n[][3])	*an array with the three nodes and an array with three normal vectors*
inline	**~DVtriangle** ()	*destructor*

Protected Members

float	**nodes** [3][3]	*First row is node 0 – third column is z0, z1, z2*
float	**normals** [3][3]	*First row is normal 0 – third column is nz0, nz1, nz2*
virtual	DVobject* **copy** ()	*A routine that returns a pointer to a copy of the object created with new.*

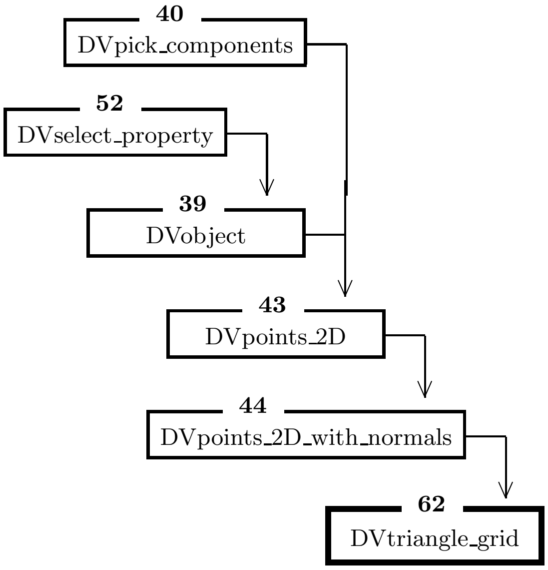
A simple class that displays one triangle.

62

class **DVtriangle_grid** : public DVpoints_2D_with_normals

This is a class of regular grid made up of triangles.

Inheritance



Public Members

	DVtriangle_grid ()	
	Constructor– no points–no triangles	
62.1	**DVtriangle_grid** (LCVMarray_3D<float> &nodes)	
	2D array of nodes.	220
62.2	**DVtriangle_grid** (LCVMarray_3D<float> &nodes, LCVMarray_3D<float> &nv)	
	2D array of nodes.	220
virtual	**~DVtriangle_grid** ()	
	Destructor	
62.3	DVtriangle_grid&	

Protected Members

This is a class of regular grid made up of triangles. It is an example of what can be done with the DVpoints_2D_with_normals class.

62.1

DVtriangle_grid (LCVMarray_3D<float> &nodes)

2D array of nodes.

2D array of nodes. must be rectangular. each nodes[i][j] must must have the same number of coordinates.

62.2

DVtriangle_grid (LCVMarray_3D<float> &nodes, LCVMarray_3D<float> &nv)

2D array of nodes.

2D array of nodes. must be rectangular. each nodes[i][j] must must have the same number of coordinates. normals given in the array nv, must have the same leading two size as nodes (one normal vector for each point). Each vector must have length 3.

62.3

DVtriangle_grid& **set_data** ()

No points–no triangles

No points–no triangles

62.4

DVtriangle_grid& **set_data** (LCVMarray_3D<float> &nodes)

2D array of nodes.

2D array of nodes. must be rectangular. each nodes[i][j] must must have the same number of coordinates.

62.5

DVtriangle_grid& **set_data** (LCVMarray_3D<float> &nodes, LCVMarray_3D<float> &nv)

2D array of nodes.

2D array of nodes. must be rectangular. each nodes[i][j] must must have the same number of coordinates. normals given in the array nv, must have the same leading two size as nodes (one normal vector for each point). Each vector must have length 3.

62.6

virtual DVobject* **copy** ()

A routine that returns a pointer to a copy of the object created with new

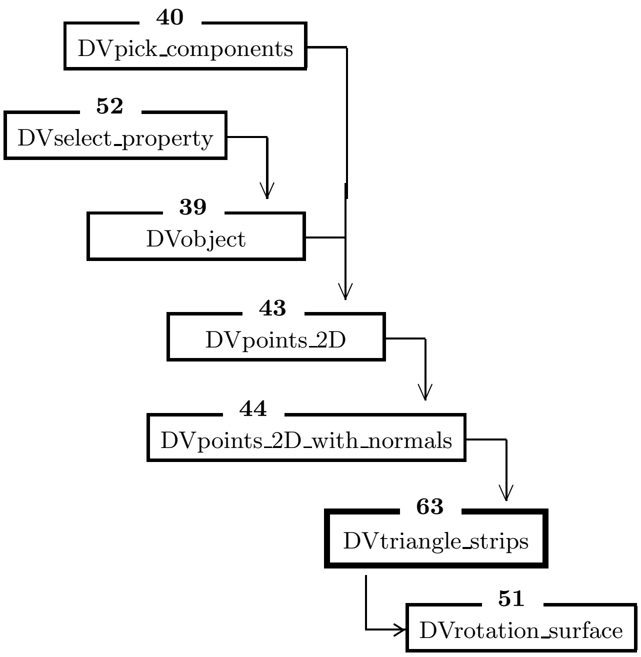
A routine that returns a pointer to a copy of the object created with new

63

class **DVtriangle_strips** : public DVpoints_2D_with_normals

This is a class of triangle strips.

Inheritance



Public Members

DVtriangle_strips ()
Constructor - no triangle strips

DVtriangle_strips (LCVMarray_2D<float> &nodes)
Points given in the array nodes, must be rectangular

63.1 **DVtriangle_strips** (LCVMarray_2D<float> &nodes, LCVMarray_2D<float> &nv)
Points given in the array nodes, must be rectangular. 224

63.2 **DVtriangle_strips** (LCVMarray_3D<float> &nodes)

	print (ostream&)	*A virtual print function*
virtual int	**ReadMyProp** ()	*Methods concerning the scene file format*
	DVtriangle_strips (FileOption a_file_op)	*Methods concerning the scene file format*

Protected Members

virtual DVobject*	**copy** ()	*A routine that returns a pointer to a copy of the object created with new.*

This is a class of triangle strips. It is an example of what can be done with the DVpoints_2D_with_normals class.

63.1

DVtriangle_strips (LCVMarray_2D<float> &nodes, LCVMarray_2D<float> &nv)

Points given in the array nodes, must be rectangular.

Points given in the array nodes, must be rectangular. Normals given in the array nv, must have the same leading size as nodes (one normal vector for each point). Each vector must have length 3.

63.2

DVtriangle_strips (LCVMarray_3D<float> &nodes)

Array of triangle strips given in the array nodes.

Array of triangle strips given in the array nodes. Each nodes[i][j] must must have the same number of coordinates.

63.3

DVtriangle_strips (LCVMarray_3D<float> &nodes, LCVMarray_3D<float> &nv)

array of triangle strips given in the array nodes.

array of triangle strips given in the array nodes. each nodes[i][j] must must have the same number of coordinates. normals given in the array nv, must have the same leading two size as nodes (one normal vector for each point). Each vector must have length 3.

63.4

void **set_data** (LCVMarray_2D<float> &point)

Makes one group of points with components in the array point no normals

Makes one group of points with components in the array point no normals

63.5

void **set_data** (LCVMarray_3D<float> &coords)

Makes an array of groups of points with components in the array "coords",no normals

Makes an array of groups of points with components in the array "coords",no normals

63.6

void **set_data** (LCVMarray_2D<float> &point, LCVMarray_2D<float> &n)

Makes one group of points with components in the array point normals given in n

Makes one group of points with components in the array point normals given in n

63.7

void **set_data** (LCVMarray_3D<float> &coords, LCVMarray_3D<float> &n)

Makes an array of groups of points with components in the array "coords" normals given in n

Makes an array of groups of points with components in the array "coords" normals given in n

Class Graph