Reference Manual

Generated by Doxygen 1.5.6

Mon Jan 19 03:19:32 2009

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 FlowChannel Class Reference

Handles one scalar field of floats defined for each cell.

`#include <FlowChannel.h>`

Public Member Functions

- void setValue (int vtxID, float val)
  sets the value of the given vertex
- void copyValues (float *rawdata, int vtxSize, int offset)
  takes an array containing all attributes for a vertex and copies the j-th attribute to this channel
- float getValue (vec3 pos)
  returns the value at given position in data set coordinates (from 0 to dimX or dimY)
- float getValue (int vtxID)
  returns the value of the given vertex
- float getValueNormPos (vec3 pos)
  returns the value at given position in normalized coordinates for each dimension <0..1>
- float getValueNormPos (float x, float y)
  returns the value at given position in normalized coordinates for each dimension <0..1>
- float normalizeValue (float val)
  scales the value according to the channel minimim and maximum, so that it lies inside of <0,1>
- float getMin ()
  returns the minimum value found in the channel
- float getMax ()
  returns the maximum value found in the channel

- float getRange ()

  returns the range = max - min

Private Attributes

- FlowGeometry * geom

  reference to the geometry structure

- float * values

  channel data storage

- float minimum

  minimum value (of all cells in a single time step)

- float maximum

  maximum value (of all cells in a single time step)

2.1.1 Detailed Description

Handles one scalar field of floats defined for each cell.

More dimensional vectors are split into components. E.g. a 3D velocity vector gets stored in three FlowChannels. A FlowChannel stores data only from one time step, it is not aware of any time related information.

2.1.2 Member Function Documentation

2.1.2.1 void FlowChannel::copyValues (float * *rawdata*, int *vtxSize*, int *offset*)

takes an array containing all attributes for a vertex and copies the j-th attribute to this channel

This methos is used by the loading of data sets.

Parameters:

rawdata data gained directly from the file, without any processing. It contains all channels for all cells. Please note, there is no time information considered here.

vtxSize number of channels per cell (incl. velocity vector size)

offset offset of the parameter loaded into this channel

The documentation for this class was generated from the following files:

- FlowChannel.h
- FlowChannel.cpp

2.2 FlowData Class Reference

class managing the data sets and related stuff like data loading, channels creation etc.

`#include <FlowData.h>`

Public Member Functions

- FlowData ()

  initializes the channel storage

- ~FlowData ()

  destoys all created channels

- bool loadDataset (string filename, bool bigEndian)

  Loads a dataset, returns true if everything successful. You have to specify the byte order used in the data.

- int getNumTimesteps ()

  Returns the number of timesteps.

- int createChannel ()

  creates a new channel and returns it's address in the channels array (line 28)

- void deleteChannel (int i)

  deletes the channel and all it's data at given adress

- FlowChannel * getChannel (int i)

  returns a pointer to the instance of channel at given adress. This is the only way to access the channels storage (at line 28)

- int createChannelGeometry (int dimension)

  creates a new channel containing the geometrical information of the given dimension (x = 0, y = 1). Returns address of the created channel in the channels array (line 28)

- int createChannelVectorLength (int chX, int chY, int chZ=-1)

  creates a new channel containing the vector lengths for the given channels (channels given by IDs). Returns address of the created channel in the channels array (line 28)

- int createChannelVectorLength (FlowChannel *chX, FlowChannel *chY, FlowChannel *chZ=NULL)

  creates a new channel containing the vector lengths for the given channels (channels given by reference). Returns address of the created channel

Private Attributes

- bool loaded

  Is there any data loaded?

- int timesteps

Number of timesteps.

- FlowGeometry geometry

  Stores the underlying geometry.

- bool freeChannel [max_channels]

  is the channel slot free?

- FlowChannel * channels [max_channels]

  stores the values of data channels for one time step. For time-dependent data, the best solution is to create a separate class handling channels in one timestep and to instanciate this class for all timesteps.

2.2.1 Detailed Description

class managing the data sets and related stuff like data loading, channels creation etc.

The documentation for this class was generated from the following files:

- FlowData.h
- FlowData.cpp

2.3 FlowGeometry Class Reference

class for handling the geometry == rectangular grids organized in vertices and cells

`#include <FlowGeometry.h>`

Public Member Functions

- ~FlowGeometry ()
  deletes the allocated geometry storage
- bool getInterpolationAt (vec3 pos, int *vtxID, float *coef)
  Returns true if inside. Stores the vertex indices and interpolation weights for the given position in the arrays.
- bool readFromFile (char *header, FILE *fp, bool bigEndian)
  reads the geometry gris data from a file
- int getDimX ()
  returns the number of vertices in X dimension
- int getDimY ()
  returns the number of vertices in Y dimension
- int getDimZ ()
  returns the number of vertices in Z dimension, is always 1
- float getMinX ()
  returns the minimum in the X dimension
- float getMaxX ()
  returns the maximum in the X dimension
- float getMinY ()
  returns the minimum in the Y dimension
- float getMaxY ()
  returns the maximum in the Y dimension
- int getRightNeigh (int vtxID)
  returns the vertex ID of the neighbour on its right
- int getTopNeigh (int vtxID)
  returns the vertex ID of the neighbour on its top
- int getLeftNeigh (int vtxID)
  returns the vertex ID of the neighbour on its left
- int getBottomNeigh (int vtxID)
  returns the vertex ID of the neighbour on its bottom

- vec3 normalizeCoords (vec3 pos)

  compression of coordinates in each dimension separately, returns values scaled to <0,1>

- vec3 unNormalizeCoords (vec3 pos)

  inverts the compression. From values of <0,1> it restores the real geometrical coordinates

Private Member Functions

- int getVtx (int x, int y)

  returns general vtxID for the vertex array indexes

- int getVtxX (int vtxID)

  returns X index for the general vtxID

- int getVtxY (int vtxID)

  returns Y index for the general vtxID

- int getXYvtx (vec3 pos)

  returns X index of the last vertex lying left to the position x and the Y index of the last vertex lying under the position y

- vec3 getPos (int vtxID)

  returns the position of the vertex

- float getPosX (int vtxID)

  returns the x position of the vertex

- float getPosY (int vtxID)

  returns the y position of the vertex

- int getNearestVtx (vec3 pos)

  a very slow and dumb routine, that finds the nearest vertex to the given position

Private Attributes

- int dim [2]

  resolution of the data for the dimensions X, Y

- vec3 boundaryMin

  minimum boundary values for the dataset geometry sotred as {minX, minY)

- vec3 boundaryMax

  maximum boundary values for the dataset geometry sotred as (maxX, maxY)

- vec3 boundarySize

  boundary sizes for the dataset geometry sotred as (maX - minX, maxY - minY)

- vec3 * geometryData

  Storage for the geometry.

- bool isFlipped

  indicates whether the x and y axes have to be swaped

Friends

- class FlowData

2.3.1 Detailed Description

class for handling the geometry == rectangular grids organized in vertices and cells

2.3.2 Member Function Documentation

2.3.2.1 bool FlowGeometry::getInterpolationAt (vec3 *pos*, int * *vtxID*, float * *coef*)

Returns true if inside. Stores the vertex indices and interpolation weights for the given position in the arrays.

Stores the indexes and weights of vertices surrounding the given position. This information can be used later on for interpolation of channel values.

Parameters:

pos geometrical position for the lookup

vtxID list of surrounding vertices (given by vertex ID)

coef list of surrounding vertex weights (sum == 1.0)

Returns:

true if the given position is inside of the geometrical boundaries

The documentation for this class was generated from the following files:

- FlowGeometry.h
- FlowGeometry.cpp

2.4 GLWidget Class Reference

`#include <glwidget.h>`

Public Slots

- void setBackgroundColors ()
- void setArrowColors ()
- void setStreamlineColors ()
- void setDrawBackground (bool state)
- void setBackgroundColorChannel (int channel)
- void setDrawArrows (bool state)
- void setArrowQuantity (int quantity)
- void setArrowColorChannel (int channel)
- void setDrawStreamlines (bool state)
- void setStreamlineMode (int mode)
- void setStreamlineDtest (int newDtest)
- void setStreamlineDsep (int newDsep)
- void setStreamlineDt (int newDt)
- void setStreamlinePeriod (int Period)
- void setStreamlineLineMode (int mode)
- void setStreamlineColorChannel (int channel)

Public Member Functions

- GLWidget (QWidget *parent=0)
- QSize minimumSizeHint () const
- QSize sizeHint () const
- void setFlowData (FlowData *newFlowData)
- void setBackgroundColorsPtr (QRgb *colorsPtr)
- void setArrowColorsPtr (QRgb *colorsPtr)
- void setStreamlineColorsPtr (QRgb *colorsPtr)

Protected Member Functions

- void initializeGL ()
- void paintGL ()
- void resizeGL (int width, int height)
- void mousePressEvent (QMouseEvent *event)
- void mouseReleaseEvent (QMouseEvent *event)
- void mouseMoveEvent (QMouseEvent *event)
- void wheelEvent (QWheelEvent *event)

Private Member Functions

- void draw ()
- void normalizeAngle (int *angle)
- char * file2string (const std::string &strFilename)
- GLuint loadShader (const std::string &strFilename)
- void printLog (GLuint obj)
- void drawArrowPlot (const int xScaler, const int yScaler)
- void calcEuler (float x, float y)
- void calcRunge (float x, float y)
- void calculateBackground ()
- void drawStreamline (int mode)
- bool testRange (vec3 point)
- bool testCell (int cellx, int celly, vec3 point)
- float getTapering (vec3 point, vector< vec3 > line, unsigned int i)
- float testCellTapering (int cellx, int celly, vec3 point, vector< vec3 > line, unsigned int i)
- bool testSLDist (vec3 point, vector< vec3 > line, unsigned int i)
- void calculateEvenStreamlines ()
- void addPoint2Grid (vec3 point)
- void calcStartCandidates (vec3 pos)
- void initStreamlines ()
- void removeLastPointfromCell (vec3 point)

2.4.1 Detailed Description

Sets up a Qt OpenGl Canvas Widget.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 GLWidget::GLWidget (QWidget * *parent* = 0)

Sets up a Qt OpenGl Canvas.

Parameters:

parent is the parent window of the canvas.

2.4.3 Member Function Documentation

2.4.3.1 QSize GLWidget::minimumSizeHint () const

Returns:

The minimum size for the Widget.

2.4.3.2 QSize GLWidget::sizeHint () const

Returns:

The size hint for the Widget.

2.4.3.3 void GLWidget::setFlowData (FlowData * *newFlowData*)

Sets up the Canvas for a new Flowdata object.

Parameters:

newFlowData is the new Flowdata.

2.4.3.4 void GLWidget::setBackgroundColorsPtr (QRgb * *colorsPtr*)

Initializes the pointer to the background transferfunction array.

Parameters:

colorsPtr is the pointer to the new array.

2.4.3.5 void GLWidget::setArrowColorsPtr (QRgb * *colorsPtr*)

Initializes the pointer to the arrow transferfunction array.

Parameters:

colorsPtr is the pointer to the new array.

2.4.3.6 void GLWidget::setStreamlineColorsPtr (QRgb * *colorsPtr*)

Initializes the pointer to the streamline transferfunction array.

Parameters:

colorsPtr is the pointer to the new array.

2.4.3.7 void GLWidget::setBackgroundColors () [slot]

Notifies that the background transferfunction has changed and redraws the scene.

2.4.3.8 void GLWidget::setArrowColors () [slot]

Notifies that the arrow transferfunction has changed and redraws the scene.

2.4.3.9 void GLWidget::setStreamlineColors () [slot]

Notifies that the streamline transferfunction has changed and redraws the scene.

2.4.3.10 void GLWidget::setDrawBackground (bool *state*) [slot]

Sets the current background Rendermode.

Parameters:

state holds if the backround should be drawn or not.

2.4.3.11 void GLWidget::setBackgroundColorChannel (int *channel*) [slot]

Sets the current FlowChannel for the background transferfunction.

Parameters:

channel the new Channel.

2.4.3.12 void GLWidget::setDrawArrows (bool *state*) [slot]

Sets the current arrow Rendermode.

Parameters:

state holds if the backround should be drawn or not.

2.4.3.13 void GLWidget::setArrowQuantity (int *quantity*) [slot]

Sets the current arrow quantity.

Parameters:

quantity holds how many arrows should be drawn.

2.4.3.14 void GLWidget::setArrowColorChannel (int *channel*) [slot]

Sets the current FlowChannel for the arrow transferfunction.

Parameters:

channel the new Channel.

2.4.3.15 void GLWidget::setDrawStreamlines (bool *state*) [slot]

Sets the current streamline Rendermode.

Parameters:

state holds if the backround should be drawn or not.

2.4.3.16 void GLWidget::setStreamlineMode (int *mode*) [slot]

Sets the current streamline integrationmode.

Parameters:

mode holds the integrationmode.

2.4.3.17 void GLWidget::setStreamlineDtest (int *newDtest*) [slot]

Sets the relation of the dtest to the dsep parameter for the streamline algorithm.

Parameters:

newDtest new dtest relation parameter.

2.4.3.18 void GLWidget::setStreamlineDsep (int *newDsep*) [slot]

Sets the relation of the dsep parameter to the shorter dimension of the dataset.

Parameters:

newDsep new dsep relation parameter.

2.4.3.19 void GLWidget::setStreamlineDt (int *newDt*) [slot]

Sets the relation of the dt to the dtest parameter for the streamline algorithm.

Parameters:

newDt new dt relation parameter.

2.4.3.20 void GLWidget::setStreamlinePeriod (int *newPeriod*) [slot]

Sets length of the period for the streamline texture visualization mode.

Parameters:

newPeriod new period length.

2.4.3.21 void GLWidget::setStreamlineLineMode (int *mode*) [slot]

Sets the current streamline visualization style.

Parameters:

mode holds the streamline visualization style.

2.4.3.22 void GLWidget::setStreamlineColorChannel (int *channel*) [slot]

Sets the current FlowChannel for the streamline transferfunction.

Parameters:

channel the new Channel.

2.4.3.23 void GLWidget::initializeGL () [protected]

Initialzes the OpenGl environment including antialiasing.

2.4.3.24 void GLWidget::paintGL () [protected]

Applys affine transformations and starts the drawing of the frame.

2.4.3.25 void GLWidget::resizeGL (int *width*, int *height*) [protected]

Handles resizing of the OpenGl canvas.

Parameters:

width is the new width of the canvas.

height is the new height of the canvas.

2.4.3.26 void GLWidget::mousePressEvent (QMouseEvent * *event*) [protected]

Reduces the stepsize of the raycastingshader and initialises volume rotations.

Parameters:

event is the mousePressEvent.

2.4.3.27 void GLWidget::mouseReleaseEvent (QMouseEvent * *event*) [protected]

Increases the stepsize of the raycastingshader after finished volume rotations.

Parameters:

event is the mouseReleaseEvent.

2.4.3.28 void GLWidget::mouseMoveEvent (QMouseEvent * *event*) [protected]

Handles mouse dragging and rotation of the volume.

Parameters:

event is the mouseMoveEvent.

2.4.3.29 void GLWidget::wheelEvent (QWheelEvent * *event*) [protected]

Handles mouseWheelEvents.

Parameters:

event is the mouseWheelEvent.

2.4.3.30 void GLWidget::draw () [private]

Draws texture, arrow and streamline layers (if activated).

2.4.3.31 void GLWidget::normalizeAngle (int * *angle*) [private]

Keeps angle within [0 360] interval.

Parameters:

angle is the pointer to the angle.

2.4.3.32 char * GLWidget::file2string (const std::string & *strFilename*) [private]

Loads a binary file into a Charpointer.

Parameters:

strFilename ist the filename of the file to load.

Returns:

The Charpointer with the content of the file.

2.4.3.33 GLuint GLWidget::loadShader (const std::string & *strFilename*) [private]

Handels loading, compiling and attaching of a vertex-, fragmentshader pair to a shader program. * Loading requires 2 glsl shader files with "<filepath>.vert" and "<filepath>.frag" extension to be successful.

Parameters:

strFilename is the filepath of the shaders without extension.

Returns:

The id of the new shaderprogram.

2.4.3.34 void GLWidget::printLog (GLuint *obj*) [private]

Prints the shaderobject debug log.

Parameters:

obj specifies the shaderobject.

2.4.3.35 void GLWidget::drawArrowPlot (const int *xScaler*, const int *yScaler*) [private]

Draws the Arrowlayer.

Parameters:

xScaler density of arrows in x-direction.

yScaler density of arrows in y-direction.

2.4.3.36 void GLWidget::calcEuler (float *x*, float *y*) [private]

Calculates a streamline in both directions of the given point using the Euler integration method.

Parameters:

x component of the startpoint.

y component of the startpoint.

2.4.3.37 void GLWidget::calcRunge (float *x*, float *y*) [private]

Calculates a streamline in both directions of the given point using the Runge-Kutta integration method.

Parameters:

x component of the startpoint.

y component of the startpoint.

2.4.3.38 void GLWidget::calculateBackground () [private]

Creates the background texture layer.

2.4.3.39 void GLWidget::drawStreamline (int *mode*) [private]

Draws the streamline layer with the given visualization mode.

Parameters:

mode line draw mode (0: normal | 1: tapered | 2: glyphs | 3: texture)

2.4.3.40 bool GLWidget::testRange (vec3 *point*) [private]

Tests if the given point is within dtest range to an already existing point.

Parameters:

point point to test.

2.4.3.41 bool GLWidget::testCell (int *cellx*, int *celly*, vec3 *point*) [private]

Tests if the given point is within dtest range to a point in the specified cell.

Parameters:

cellx x-coordinate of the cell.

celly y-coordinate of the cell.

point point to test.

2.4.3.42 float GLWidget::getTapering (vec3 *point*, vector< vec3 > *line*, unsigned int *i*) [private]

Tests if the given point is within dsep range to an already existing point and returns tapering thickness.

Parameters:

point point to test.

line the line containing the testpoint.

i index of the testpoint within the line (to prevent getting hits from neighbours).

2.4.3.43 float GLWidget::testCellTapering (int *cellx*, int *celly*, vec3 *point*, vector< vec3 > *line*, unsigned int *i*) [private]

Tests if the given point is within dsep range to a point in the specified cell and returns distance to the nearest point.

Parameters:

cellx x-coordinate of the cell.

celly y-coordinate of the cell.

point point to test.

line the line containing the testpoint.

i index of the testpoint within the line (to prevent getting hits from neighbours)

2.4.3.44 bool GLWidget::testSLDist (vec3 *point*, vector< vec3 > *line*, unsigned int *i*) [private]

Tests if the given point is member of the given line and at least ceil(dsep/dt) indices away from i.

Parameters:

point point to test.

line the line containing the testpoint.

i index of the testpoint within the line.

2.4.3.45 void GLWidget::calculateEvenStreamlines () [private]

Calculates evenly-spaced streamlines according to the method of Bruno Jobard and Wilfrid Lefer.

2.4.3.46 void GLWidget::addPoint2Grid (vec3 *point*) [private]

Adds the given point to the correct cell of the grid.

Parameters:

point point to add.

2.4.3.47 void GLWidget::calcStartCandidates (vec3 *pos*) [private]

Calculates two new startpoints from the origin, validates and adds them to the startCandidates queue.

Parameters:

pos origin point.

2.4.3.48 void GLWidget::initStreamlines () [private]

Sets up the environment to recalculate the streamlines.

2.4.3.49 void GLWidget::removeLastPointfromCell (vec3 *point*) [private]

Removes the last added point from the cell of the given point.

Parameters:

point reference to the cell.

The documentation for this class was generated from the following files:

- glwidget.h
- glwidget.cpp

2.5 GradientEditor Class Reference

`#include <gradienteditor.h>`

Public Slots

- void pointsUpdated ()
- void colorDialog (int index)
- void moveColorsWithAlpha (int index, qreal newXPos)
- void newColorPoints (int index, qreal xPos)
- void deleteColorPoints (int index)

Signals

- void gradientStopsChanged (const QGradientStops &stops, GradientEditor *editor)
- void setEnabledSignal (bool enabled)

Public Member Functions

- GradientEditor (QWidget *parent)
- void setGradientStops (const QGradientStops &stops)
- QRgb * getColorsPtr ()
- void updateColorTexture ()
- void setGradientEditorRangeLabel (float min=0, float max=1)

Protected Member Functions

- void resizeEvent (QResizeEvent *event)

2.5.1 Detailed Description

The gradienteditor class is a alpha ShadeWidget with a range label.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 GradientEditor::GradientEditor (QWidget * *parent*)

Constructor that creates the color editor.

Returns:

The pointer to the created color editor.

2.5.3 Member Function Documentation

2.5.3.1 void GradientEditor::setGradientStops (const QGradientStops & *stops*)

Sets the gradient stops position and color from the given stops.

Parameters:

stops are the gradient stops which should be set.

2.5.3.2 QRgb * GradientEditor::getColorsPtr ()

Returns:

The pointer to the texture.

2.5.3.3 void GradientEditor::updateColorTexture ()

Updates the texture with the color currently set in ShadeWidgets.

2.5.3.4 void GradientEditor::setGradientEditorRangeLabel (float *min* = `0`, float *max* = `1`)

Sets the left and right label of the color editor.

Parameters:

min is value which should be displayed at the left label.

max is value which should be displayed at the right label.

2.5.3.5 void GradientEditor::pointsUpdated () `[slot]`

Updates the gradient stops position and color from the currently set points. Sets this gradient points at the alpha ShadeWidget and emits a gradientStopsChanged signal.

2.5.3.6 void GradientEditor::colorDialog (int *index*) `[slot]`

Creates a color dialog with which a new color can be set.

Parameters:

index is the index of the point for which the color should be changed.

2.5.3.7 void GradientEditor::moveColorsWithAlpha (int *index*, qreal *newXPos*) `[slot]`

Moves the color with the corresponding alpha point.

Parameters:

index is the index of the color which should be moved.

newXPos is the new x position to which the corresponding color should be moved.

2.5.3.8 void GradientEditor::newColorPoints (int *index*, qreal *xPos*) [slot]

Creates the color for the corresponding alpha point.

Parameters:

index is the index where the color should be saved.

xPos is the x position where the color should be created.

2.5.3.9 void GradientEditor::deleteColorPoints (int *index*) [slot]

Deletes the color of the corresponding alpha point.

Parameters:

index is the index where the color should be deleted.

2.5.3.10 void GradientEditor::gradientStopsChanged (const QGradientStops & *stops*, GradientEditor * *editor*) [signal]

A signal that is send on change of the gradient stops.

Parameters:

stops are the gradient stops which changed.

editor is the color editor for whom the stops changed.

2.5.3.11 void GradientEditor::setEnabledSignal (bool *enabled*) [signal]

A signal that is send on enabling/disabling a draw group.

Parameters:

enabled is the state which should be set for the ShadeWidget.

2.5.3.12 void GradientEditor::resizeEvent (QResizeEvent * *event*) [protected]

Updates the gradient points at each GradientEditor resizing.

The documentation for this class was generated from the following files:

- gradienteditor.h
- gradienteditor.cpp

2.6 HoverPoints Class Reference

`#include <hoverpoints.h>`

Public Types

- enum PointShape
- enum LockType
- enum SortType
- enum ConnectionType

Public Slots

- void setEnabled (bool enabled)
- void setDisabled (bool disabled)

Signals

- void pointsChanged (const QPolygonF &points)
- void chooseColor (int index)
- void alphaChanged (int index, qreal newXPos)
- void alphaPointCreated (int index, qreal xPos)
- void alphaPointDeleted (int index)

Public Member Functions

- HoverPoints (QWidget *widget, PointShape shape)
- bool eventFilter (QObject *object, QEvent *event)
- void paintPoints ()
- QRectF boundingRect () const
- void setBoundingRect (const QRectF &boundingRect)
- QPolygonF points () const
- void setPoints (const QPolygonF &points)
- QSizeF pointSize () const
- void setPointSize (const QSizeF &size)
- SortType sortType () const
- void setSortType (SortType sortType)
- ConnectionType connectionType () const
- void setConnectionType (ConnectionType connectionType)
- void setConnectionPen (const QPen &pen)
- void setShapePen (const QPen &pen)
- void setShapeBrush (const QBrush &brush)
- void setPointLock (int pos, LockType lock)
- void setEditable (bool editable)
- bool editable () const
- void firePointChange ()
- void movePoint (int i, const QPointF &newPos)

Private Slots

- void newPoint ()
- void deletePoint ()
- void fireChooseColor ()

2.6.1 Detailed Description

The hoverpoints class represents the points which are used in the ShadeWidget.

2.6.2 Member Enumeration Documentation

2.6.2.1 enum HoverPoints::PointShape

Enumerates the PointShape options.

2.6.2.2 enum HoverPoints::LockType

Enumerates the LockType options.

2.6.2.3 enum HoverPoints::SortType

Enumerates the SortType options.

2.6.2.4 enum HoverPoints::ConnectionType

Enumerates the ConnectionType options.

2.6.3 Constructor & Destructor Documentation

2.6.3.1 HoverPoints::HoverPoints (QWidget * *widget*, PointShape *shape*)

Constructor that creates the HoverPoints.

Returns:

The pointer to the created HoverPoints.

2.6.4 Member Function Documentation

2.6.4.1 bool HoverPoints::eventFilter (QObject * *object*, QEvent * *event*)

Filters the events and takes appropriate actions.

Parameters:

object is the QObject which caused the QEvent.

event is the QEvent which happend.

Returns:

True if it was a valid object and a valid event, false otherwise.

2.6.4.2 void HoverPoints::paintPoints ()

Paints the points and connections.

2.6.4.3 QRectF HoverPoints::boundingRect () const [inline]

Returns:

The bounding rectangle for this hover points.

2.6.4.4 void HoverPoints::setBoundingRect (const QRectF & *boundingRect*) [inline]

Sets the new bounding rectangle for this hover points.

Parameters:

boundingRect is the new bounding rectangle which should be set.

2.6.4.5 QPolygonF HoverPoints::points () const [inline]

Returns:

The hover points.

2.6.4.6 void HoverPoints::setPoints (const QPolygonF & *points*)

Sets points position from the given points.

Parameters:

points are the points which should be set.

2.6.4.7 QSizeF HoverPoints::pointSize () const [inline]

Returns:

The pixel size of one hover point (for drawing).

2.6.4.8 void HoverPoints::setPointSize (const QSizeF & *size*) [inline]

Sets the new pixel size for this hover points.

Parameters:

size is the new pixel size which should be set.

2.6.4.9 SortType HoverPoints::sortType () const [inline]

Returns:

The sort type, in which the hover points should be sorted.

2.6.4.10 void HoverPoints::setSortType (SortType *sortType*) [inline]

Sets the way, in which the hover points should be sorted.

Parameters:

sortType is the new sortType which should be set.

2.6.4.11 ConnectionType HoverPoints::connectionType () const [inline]

Returns:

The way in which the point connections ar drawn.

2.6.4.12 void HoverPoints::setConnectionType (ConnectionType *connectionType*) [inline]

Sets the way, in which the point connections should be drawn.

Parameters:

connectionType is the new connectionType which should be set.

2.6.4.13 void HoverPoints::setConnectionPen (const QPen & *pen*) [inline]

Sets the pen, which draws the point connections.

Parameters:

pen is the new pen which should be set.

2.6.4.14 void HoverPoints::setShapePen (const QPen & *pen*) [inline]

Sets the pen, which draws the shape of the points.

Parameters:

pen is the new pen which should be set.

2.6.4.15 void HoverPoints::setShapeBrush (const QBrush & *brush*) [inline]

Sets the brush, which fills the shape of the points.

Parameters:

brush is the new brush which should be set.

2.6.4.16 void HoverPoints::setPointLock (int *pos*, LockType *lock*) [inline]

Sets the point lock for a point.

Parameters:

pos is the index of the point from which the lock should be set.

lock is the typ of lock which should be set.

2.6.4.17 void HoverPoints::setEditable (bool *editable*) [inline]

Sets this points editable state.

Parameters:

editable is true if the points are editable and flase otherwise.

2.6.4.18 bool HoverPoints::editable () const [inline]

Returns:

true if the points are editable and flase otherwise.

2.6.4.19 void HoverPoints::setEnabled (bool *enabled*) [slot]

Enables or disables these HoverPints.

Parameters:

enabled is true if the points are enabled and flase otherwise.

2.6.4.20 void HoverPoints::setDisabled (bool *disabled*) [inline, slot]

Enables or disables these HoverPints.

Parameters:

disabled is true if the points are disabled and flase otherwise.

2.6.4.21 void HoverPoints::pointsChanged (const QPolygonF & *points*) [signal]

A signal that is send on change of the points.

Parameters:

points are the new points.

2.6.4.22 void HoverPoints::chooseColor (int *index*) [signal]

A signal that is send to initiate a color dialog.

Parameters:

index is the index of the point for which the color should be changed.

2.6.4.23 void HoverPoints::alphaChanged (int *index*, qreal *newXPos*) [signal]

A signal that is send if an alpha point got moved.

Parameters:

index is the index of the points which should be moved.
newXPos is the new x position to which the corresponding points should be moved.

2.6.4.24 void HoverPoints::alphaPointCreated (int *index*, qreal *xPos*) [signal]

A signal that is send if an alpha point got created.

Parameters:

index is the index where the points should be saved.
xPos is the x position where the points should be created.

2.6.4.25 void HoverPoints::alphaPointDeleted (int *index*) [signal]

A signal that is send if an alpha point got deleted.

Parameters:

index is the index where the points should be deleted.

2.6.4.26 void HoverPoints::firePointChange ()

Sorts the points.

2.6.4.27 void HoverPoints::movePoint (int *index*, const QPointF & *point*)

Moves the given point.

Parameters:

index is the index of the point which should be moved.
point is the point which contains the new position.

2.6.4.28 void HoverPoints::newPoint () [private, slot]

Creates a new point. Emits an alphaPointCreated.

2.6.4.29 void HoverPoints::deletePoint () [private, slot]

Deletes a point. Emits an alphaPointDeleted.

2.6.4.30 void HoverPoints::fireChooseColor () [private, slot]

Emits a chooseColor.

The documentation for this class was generated from the following files:

- hoverpoints.h
- hoverpoints.cpp

2.7 ShadeWidget Class Reference

`#include <shadewidget.h>`

Public Types

- enum ShadeType

Signals

- void colorsChanged ()

Public Member Functions

- ShadeWidget (ShadeType type, QWidget *parent)
- void setGradientStops (const QGradientStops &stops)
- void paintEvent (QPaintEvent *e)
- QSize minimumSizeHint () const
- QSize sizeHint () const
- QPolygonF points () const
- HoverPoints * hoverPoints () const
- QRgb colorAt (int x)

Private Member Functions

- void generateShade ()

2.7.1 Detailed Description

The shadewidget class is a drawable rectangle in which points and a color or alpha gradient can be drawn.

2.7.2 Member Enumeration Documentation

2.7.2.1 enum ShadeWidget::ShadeType

Enumerates the ShateType options.

2.7.3 Constructor & Destructor Documentation

2.7.3.1 ShadeWidget::ShadeWidget (ShadeType *type*, QWidget * *parent*)

Constructor that creates a color or alpha ShadeWidget.

Returns:

The pointer to the created color or alpha ShadeWidget.

2.7.4 Member Function Documentation

2.7.4.1 void ShadeWidget::setGradientStops (const QGradientStops & *stops*)

Sets the gradient stops position and color from the given stops.

Parameters:

stops are the gradient stops which should be set.

2.7.4.2 void ShadeWidget::paintEvent (QPaintEvent * *e*)

Paints the backround, points and the connection lines.

2.7.4.3 QSize ShadeWidget::minimumSizeHint () const

Returns:

The minimum size hint for the Widget.

2.7.4.4 QSize ShadeWidget::sizeHint () const

Returns:

The size hint for the Widget.

2.7.4.5 QPolygonF ShadeWidget::points () const

Returns:

The points of this ShadeWidget.

2.7.4.6 HoverPoints* ShadeWidget::hoverPoints () const [inline]

Returns:

The pointer to the hover points of this ShadeWidget.

2.7.4.7 QRgb ShadeWidget::colorAt (int *x*)

Gets the current color of the ShadeWidet at the specific position.

Parameters:

x is the position from which the color should be taken.

Returns:

The color at the position.

2.7.4.8 void ShadeWidget::colorsChanged () [signal]

A signal that is send on change of the colors.

2.7.4.9 void ShadeWidget::generateShade () [private]

Generates the backround.

The documentation for this class was generated from the following files:

- shadewidget.h
- shadewidget.cpp

2.8 vec3 Class Reference

`#include <vec3.h>`

Public Member Functions

- float & operator[ ] (unsigned)
  returns an element of the vector
- vec3 & operator= (const vec3 &)
  assignment operator
- vec3 & operator+= (const vec3 &)
  adds the input vector to this one
- vec3 & operator-= (const vec3 &)
  subtracts the input vector from this one
- vec3 & operator∗= (float)
  scales this vector with the scalar
- vec3 & operator/= (float)
  scales this vector with the inverse of the scalar
- const vec3 operator+ (const vec3 &) const
  adds two vectors
- const vec3 operator- (const vec3 &) const
  subtracts two vectors
- const vec3 operator∗ (float) const
  scales the vector with the scalar
- const vec3 operator/ (float) const
  scales the vector with the inverse of the scalar
- bool operator== (const vec3 &) const
  returns true if the vectors are equal
- bool operator!= (const vec3 &) const
  returns true if the vectors differ in at least one component
- vec3 & operator- ()
  opposite vector
- float norm () const
  norm of the vector (length∗length)
- float length () const

length of the vector

- vec3 & operator! ()
  normalizes the vector
- float dist2 (const vec3 &)
  norm of the vectors' difference (dist∗dist)
- float dist (const vec3 &)
  length of the vectors' difference
- float operator∗ (const vec3 &) const
  dot product
- const vec3 operator^ (const vec3 &) const
  cross product
- void print ()
  print the vector components

Public Attributes

- float v [3]
  our vector data

2.8.1 Detailed Description

3D vector class.

The documentation for this class was generated from the following files:

- vec3.h
- vec3.cpp

2.9 Window Class Reference

`#include <window.h>`

Signals

- void backgroundColorTextureChanged ()
- void arrowColorTextureChanged ()
- void streamlineColorTextureChanged ()

Public Member Functions

- Window ()

Private Slots

- void setDefaultGradientStops ()
- void setOpenFileName ()
- void saveGradientStops (const QGradientStops &stops, GradientEditor *editor)
- void setBackgroundColorEditorRangeLabel (int channel=-1)
- void setArrowColorEditorRangeLabel (int channel=-1)
- void setStreamlineColorEditorRangeLabel (int channel=-1)

2.9.1 Detailed Description

The window class represents the whole application window.

2.9.2 Constructor & Destructor Documentation

2.9.2.1 Window::Window ()

Constructor that creates the GUI.

2.9.3 Member Function Documentation

2.9.3.1 void Window::setDefaultGradientStops () [private, slot]

Loads and sets the gradient stops for the current FlowData.

2.9.3.2 void Window::setOpenFileName () [private, slot]

Creates a file dialog with which FlowData can be loaded.

2.9.3.3 void Window::saveGradientStops (const QGradientStops & *stops*, GradientEditor * *editor*) [private, slot]

Saves the given gradient stops into a file, calls for a update of the transfer texture and emits a colorTextureChanged signal.

Parameters:

stops are the gradient stops which should be saved.

editor is the color editor for whom the stops should be saved.

2.9.3.4 void Window::setBackgroundColorEditorRangeLabel (int *channel* = -1) [private, slot]

Reads the value range from a specific FlowData channel into the backgroundColorEditor.

Parameters:

channel is the channel from FlowData from which the value range should be read.

2.9.3.5 void Window::setArrowColorEditorRangeLabel (int *channel* = -1) [private, slot]

Reads the value range from a specific FlowData channel into the arrowColorEditor.

Parameters:

channel is the channel from FlowData from which the value range should be read.

2.9.3.6 void Window::setStreamlineColorEditorRangeLabel (int *channel* = -1) [private, slot]

Reads the value range from a specific FlowData channel into the streamlineColorEditor.

Parameters:

channel is the channel from FlowData from which the value range should be read.

2.9.3.7 void Window::backgroundColorTextureChanged () [signal]

A signal that is send on change of the background texture.

2.9.3.8 void Window::arrowColorTextureChanged () [signal]

A signal that is send on change of the arrow texture.

2.9.3.9 void Window::streamlineColorTextureChanged () [signal]

A signal that is send on change of the streamline texture.

The documentation for this class was generated from the following files:

- window.h
- window.cpp

Index