libbiarc

Generated by Doxygen 1.5.6

Mon Feb 8 17:34:06 2010

Contents

Chapter 1

Documentation for libbiarc

1.1 Introduction

The library and tools are intended to be used to manipulate and analyse open and closed curves and knot shapes. A viewer application called `curview` is also included for visualization.

1.2 Mercurial repositories

Clone the `static Mercurial repository` with

```
hg clone static-http://lcvmwww.epfl.ch/libbiarc/
```

1.3 Build

The software has been developed in a linux environment. It has successfully been tested and used on Linux and Apple's OSX, but not in Microsoft Windows.

For the viewer program `Qt 4` and `Coin3D, SoQt` need to be available.

In the cloned directory `libbiarc/` type

```
make
```

to build the library and a subset of the tools.

1.4 Mercurial repositories

There is no system wide installation performed. It is recommended to tell the system where the library and the tools are, for example

```
export LIBBIARC=/path/to/libbiarc
export PATH=$PATH:$LIBBIARC/tools:$LIBBIARC/inventor
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$LIBBIARC/lib
```

A pdf version of the documentation can be found `here`.

A tarball of the latest release can be downloaded `here (tgz)` or `here (tbz2)`.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Directory Hierarchy

3.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 7

Module Documentation

7.1 boxproblem

Classes

- class **Vec2**

  2D Vector class.

- class **Box**

  Square box class.

- class **BoxAnneal**

  N square boxes in a big square.

7.2 fitting

Classes

- class **Vec2**

  2D Vector class.

- class **FittingAnneal**

  Fit a curve through given points.

7.3 annealing

Modules

- **boxproblem**
- **fitting**
- **dotrepulsion**

Classes

- class **BasicMove**

  Basic class for annealing moves.

- class **SimpleFloatMove**

  Change a single float value.

- class **BasicAnneal**

  Base class for annealing problems.

7.4 dotrepulsion

Classes

- class **SampleAnneal**

  Example how to use BaseAnneal.

7.5 OpenInventor

Files

- file **s3viz.cpp**

  Visualize a curve in S^3 in two balls in R^3.

- file **s3viz.cpp**

  Visualize a curve in S^3 in two balls in R^3.

- file **xyzview.cpp**

  Visualize a closed data file.

Classes

- class **MainWindow**

  Main viewer class.

- class **SoKnot**

  *The **SoKnot** (p. 141) class is for rendering tubular curves in the Coin/OpenInventor graphics library.*

- struct **CurveInfo**

  ***Curve** (p. 76) information.*

- class **CurveInterface**

  Interface between curve data and viewer.

7.6 libbiarc

Classes

- class **Biarc< Vector >**

  *The **Biarc** (p. 61) class describes a single biarc.*

- class **Curve< Vector >**

  *The **Curve** (p. 76) class for storing and manipulating a single biarc curve in $\mathcal{R}^3$.*

- class **CurveBundle< Vector >**

  *The **CurveBundle** (p. 102) class for storing and manipulating biarc curves in $\mathcal{R}^3$.*

- class **Matrix3**

  *The **Matrix3** (p. 119) class is a 3x3 dimensional Matrix class with floating point entries.*

- class **Matrix4**

  *The **Matrix4** (p. 126) class is a 4x4 dimensional Matrix class with floating point entries.*

- class **PKFmanip**

  *The **PKFmanip** (p. 133) class for storing and manipulating biarc curves.*

- class **Tube< Vector >**

  *The **Tube** (p. 147) class contains the mesh of a tube around a curve in $\mathcal{R}^3$.*

- class **TubeBundle< Vector >**

  *The **TubeBundle** (p. 152) class contains the mesh of a tube around a curve in $\mathcal{R}^3$.*

- class **Vector3**

  *The **Vector3** (p. 158) class is a 3 dimensional Vector class with floating point coordinates.*

- class **Vector4**

  *The **Vector4** (p. 165) class is a 4 dimensional Vector class with floating point coordinates.*

7.7 Object Directory

Files

- file **aux.h**

  Auxiliary routines that produce particular curves.

- file **bone.cpp**

  Constructs a bone shaped curve.

- file **borromean.cpp**

  Constructs the borromean rings.

- file **circle.cpp**

  Constructs a circle.

- file **circle.cpp**

  Constructs a circle.

- file **crouzy.cpp**

  Constructs a crouzy curve ;).

- file **ellipse.cpp**

  Constructs an ellipse.

- file **helix.cpp**

  Constructs a helical curve.

- file **hopf.cpp**

  Constructs a hopf links.

- file **inf.cpp**

  Constructs an infinity shaped curve.

- file **knottable.cpp**

  Construct a table of curves.

- file **line.cpp**

  Produces a file with a PKF line.

- file **random.cpp**

  Generate a "random" curve.

- file **sinus.cpp**

  Constructs a Sinusoidal PKF curve.

- file **solenoid.cpp**

  Constructs a solenoid like curve.

- file **spiral.cpp**

Constructs a Spiral PKF file.

- file **stadium.cpp**

  Constructs a stadium curve.

- file **torusknot.cpp**

  Constructs a (p,q) torus knot.

- file **torusknot4.cpp**

  Constructs a (p,q) torus knot on a torus in R^4.

- file **trefoil.cpp**

  Approximated trefoil with arcs of circles ...

7.8 PKF curve tools

Files

- file **align.cpp**

  Orient a pkf curve according to a given axes.

- file **angle_condition.cpp**

  Check the angle condition for an ideal knot. input file : s sigma tau ...

- file **angle_principal_normal_contacts.cpp**

  Prints angle between contact chords and principal normal to stdout.

- file **angle_vs_arclength.cpp**

  Prints the arclength and for that arclength the angle between the normal vector and the "2" contact struts.

- file **arcdist.cpp**

  Program for testing purpose only.

- file **arcs2polygonal.cpp**

  Convert an arc curve to a polygonal curve.

- file **biarccli.cpp**

  Biarc *(p. 61) command line interface.*

- file **sigma2inventor.cpp**

  Convert 2D sigma(s) to iv or obj file.

- file **billiard_poly.cpp**

  Construct billiard polygon.

- file **centerandnormalise.cpp**

  Center and normalize a closed PKF curve.

- file **circle_for_f31.cpp**

  Given eps, compute using 3 points the radius and center of circle.

- file **clif_tref_contactset.cpp**

  Compute the contact set and surface (iv file) for the clifford trefoil in S^3.

- file **closest_neighbour.cpp**

  Reconstruct the curve using the nearest point algo.

- file **contacts2inventor.cpp**

  Convert contact chords to obj/iv file.

- file **contacts2sigma.cpp**

  Contact chords to sigma(s) and tau(s).

- file **contacts_s3tor3.cpp**

  Project contact chords from S^3 to R^3.

- file **contactset.cpp**

  Compute the contact chords for a given knot.

- file **convolutionfilter.cpp**

  Smooth a curve using a local gaussian convolution filter.

- file **curvature.cpp**

  Outputs the curvature of a biarc curve.

- file **curvature4.cpp**

  Outputs the curvature of a biarc curve in R^4.

- file **curvesplit.cpp**

  Split a "closed" curve into segments for given arclength values.

- file **extract_sigma.cpp**

  Advancing within a small neighbournood try to extract sigma(s).

- file **extract_sigma_max.cpp**

  Find local max in pt valley.

- file **filter_nodes.cpp**

  Filter a PKF curve. I.e. remove points that are too close to some neighbour.

- file **flatten.cpp**

  Project curve onto 2D plane.

- file **follow_contact.cpp**

  Starting at some point on the curve, follow the contacts to depth max_level.

- file **inertiatensor.cpp**

  Compute the inertia tensor and principal axes.

- file **info.cpp**

  Computes Arc-length, center of mass, thickness (approx), ropelength (aprox) info of a PKF curve.

- file **info4.cpp**

  Computes Arc-length, center of mass, thickness (approx), ropelength (aprox) info of a PKF curve (S^3).

- file **inversion_in_sphere.cpp**

  Inversion in sphere at center c and radius r of a curve.

- file **length.cpp**

  Arc-length of a PKF curve.

- file **link_thickness.cpp**

Compute thickness of a ***CurveBundle*** *(p. 102).*

- file **map.cpp**

  Map one curve to another.

- file **mesh4stokes.cpp**

  Produce a tube mesh from a curve and write that to a file.

- file **nana.cpp**

  Test program.

- file **normal_indicatrix.cpp**

  Unfinished.

- file **perturb.cpp**

  Perturb a curve.

- file **pkf2java.cpp**

  Read data from a PKF file, produce a tubular mesh for each curve in the file and write it as C/C++/Java arrays.

- file **pkf2mesh.cpp**

  Read data from a PKF file, produce a tubular mesh for each curve in the file and write that to a file.

- file **pkf2obj.cpp**

  pkf to obj format. obj file has correct uv texture coords.

- file **pkf2ply.cpp**

  pkf to mesh to stanford ply format.

- file **pkf2ply.cpp**

  pkf to mesh to stanford ply format.

- file **pkf2stl.cpp**

  Read data from a PKF file, produce a tubular mesh for each curve in the file and write that to an STL file.

- file **arcs2polygonal.cpp**

  Convert an arc curve to a polygonal curve.

- file **pkf2xyz.cpp**

  Convert PKF file to XYZ coordinates.

- file **pkfframe.cpp**

  Generate a framing along the curve.

- file **plotslice.cpp**

  Write a pt plot cross section at a given arclength.

- file **plotslice4.cpp**

  Write a pt plot cross section at a given arclength (S^3).

- file **pointat.cpp**

  Writes point/tangent at Curve(s), for s arclength and s in [0,1] to the standart output.

- file **projected_dij.cpp**

  Compute $\sum_{i \neq j} d_{i,j}$ and $\sum_{i \neq j} d_{i,j}^{-1}$ of a given projection.

- file **ptcircles.cpp**

  Resample an open or closed PKF curve.

- file **r3tos3.cpp**

  Project a curve in R^3 to S^3.

- file **redo_tangents.cpp**

  Recompute tangents using only the data points.

- file **refine.cpp**

  Refine/Resample a part of an open or closed PKF curve.

- file **reorder.cpp**

  Reorder the nodes of a curve, so that the first points of the 2 curves are the closest possible.

- file **resample.cpp**

  Resample an open or closed PKF curve.

- file **resample4.cpp**

  Resample an open or closed PKF curve (R^4).

- file **reverse_direction.cpp**

  Change the orientation of the curve.

- file **rotate_curve.cpp**

  Rotate curve about axis v by angle a.

- file **s3tor3.cpp**

  Project a curve on S^3 back to R^3.

- file **s_sigma_angles.cpp**

  Program for testing purpose only.

- file **setcenter.cpp**

  Reset the mass center of PKF curve.

- file **shuffle.cpp**

  Randomize/shuffle a curve.

- file **sigma2contact.cpp**

  From a sigma(s) function, generate the contact struts.

- file **sigma2inventor.cpp**

  Convert 2D sigma(s) to iv or obj file.

- file **sigma_and_tau_contacts.cpp**

  Extract contact functions sigma(s) and tau(s).

- file **sigma_composition.cpp**

  compute $^k(s)$. input file : s sigma tau ...

- file **resample.cpp**

  Resample an open or closed PKF curve.

- file **tangent_indicatrix.cpp**

  Write the tangent indicatrix of a given (open or closed) curve to a PKF file. Tangents for the tangent indicatrix are interpolated, therefore strange points like cusps would not be visible! The default is for an open curve, for closed curves put the -closed switch.

- file **s_sigma_angles.cpp**

  Program for testing purpose only.

- file **s_sigma_angles.cpp**

  Program for testing purpose only.

- file **torsion.cpp**

  Outputs the torsion of a biarc curve.

- file **xyz2pkf.cpp**

  Convert XYZ coordinates to PKF.

Chapter 8

Directory Documentation

8.1 experimental/annealing/ Directory Reference

Files

- file **box_problem.cpp**
- file **fit.cpp**
- file **link_anneal.cpp**
- file **my_anneal.cpp**
- file **torus_trefoil_test.cpp**
- file **trefoil_torus.cpp**

8.2 annealing/ Directory Reference

Directories

- directory **paramtests**

Files

- file **anneal.cpp**
- file **anneal4.cpp**
- file **curvatureflow.cpp**
- file **sono.cpp**

8.3 experimental/curvspeed/ Directory Reference

Files

- file **curvature_parametrization.cpp**

8.4 experimental/ Directory Reference

Directories

- directory **annealing**
- directory **curvspeed**
- directory **mpi**
- directory **pngmanip**
- directory **s3viz**
- directory **thickness**
- directory **valleyflow**

8.5 fourierknots/ Directory Reference

Files

- file **coeff2pkf.cpp**
- file **fcurvature.cpp**
- file **fourier_3_1.cpp**
- file **fourier_3_1.h**
- file **fourier_4_1.cpp**
- file **fourier_4_1.h**
- file **fourier_5_1.cpp**
- file **fourier_5_1.h**
- file **fourier_8_18.cpp**
- file **fourier_8_18.h**
- file **fourier_anneal.cpp**
- file **fourier_anneal_41.cpp**
- file **fourier_anneal_generic.cpp**
- file **fourier_anneal_on_S3.cpp**
- file **fourier_syn.cpp**
- file **fourier_syn.h**
- file **guess_coeffs.cpp**
- file **new_fourier_anneal.cpp**
- file **new_fourier_anneal_fast.cpp**
- file **sym.cpp**
- file **sym41.cpp**
- file **sym818.cpp**
- file **test.cpp**

8.6 gradientflow/ Directory Reference

Files

- file **gradientflow.cpp**

8.7 include/ Directory Reference

Directories

- directory **utils**

Files

- file **Biarc.h**
- file **Curve.h**
- file **CurveBundle.h**
- file **Matrix3.h**
- file **Matrix4.h**
- file **PKFmanip.h**
- file **Tube.h**
- file **TubeBundle.h**
- file **Vector3.h**
- file **Vector4.h**

8.8 experimental/thickness/include/ Directory Reference

Files

- file **curvealgos.h**
- file **minsegdist.h**
- file **segdist.h**

8.9 inventor/ Directory Reference

Files

- file **anim_iv.cpp**
- file **animate.cpp**
- file **curview.cpp**
- file **gui.cpp**
- file **gui.h**
- file **knot4view.cpp**
- file **main.h**
- file **mainwindow.cpp**
- file **mainwindow.h**
- file **plotzoom.cpp**
- file **plotzoom.h**
- file **pp.cpp**
- file **pp.h**
- file **pt.cpp**
- file **pt.h**
- file **qrc_application.cpp**
- file **SoKnot.cpp**
- file **SoKnot.h**
- file **trefoil_builder.cpp**
- file **trefoil_builder.h**
- file **tt.cpp**
- file **tt.h**
- file **utils.cpp**
- file **utils.h**
- file **viewer.cpp**
- file **viewer.h**
- file **xyzview.cpp**

  Visualize a closed data file.

8.10 lib/ Directory Reference

Directories

- directory **utils**

Files

- file **Biarc.cpp**
- file **Curve.cpp**
- file **CurveBundle.cpp**
- file **Matrix3.cpp**
- file **Matrix4.cpp**
- file **PKFmanip.cpp**
- file **Tube.cpp**
- file **TubeBundle.cpp**
- file **Vector3.cpp**
- file **Vector4.cpp**

8.11 experimental/mpi/ Directory Reference

Files

- file **a.cpp**
- file **len.cpp**
- file **len2.cpp**

8.12 objects/ Directory Reference

Files

- file **aux.h**

  Auxiliary routines that produce particular curves.

- file **bone.cpp**

  Constructs a bone shaped curve.

- file **borromean.cpp**

  Constructs the borromean rings.

- file **circle.cpp**

  Constructs a circle.

- file **circle4.cpp**
- file **crouzy.cpp**

  Constructs a crouzy curve ;).

- file **ellipse.cpp**

  Constructs an ellipse.

- file **helix.cpp**

  Constructs a helical curve.

- file **hopf.cpp**

  Constructs a hopf links.

- file **inf.cpp**

  Constructs an infinity shaped curve.

- file **knottable.cpp**

  Construct a table of curves.

- file **line.cpp**

  Produces a file with a PKF line.

- file **random.cpp**

  Generate a "random" curve.

- file **sinus.cpp**

  Constructs a Sinusoidal PKF curve.

- file **solenoid.cpp**

  Constructs a solenoid like curve.

- file **spiral.cpp**

  Constructs a Spiral PKF file.

- file **stadium.cpp**

  Constructs a stadium curve.

- file **torusknot.cpp**

  Constructs a (p,q) torus knot.

- file **torusknot4.cpp**

  Constructs a (p,q) torus knot on a torus in R^4.

- file **trefoil.cpp**

  Approximated trefoil with arcs of circles ...

8.13 annealing/paramtests/ Directory Reference

Files

- file **anneal_param.cpp**

8.14 experimental/pngmanip/ Directory Reference

Files

- file **colors.h**
- file **height2color.cpp**
- file **plot.cpp**
- file **plot_funcs.cpp**
- file **plot_funcs.h**
- file **pngwrite.h**
- file **writhe.cpp**

8.15 experimental/s3viz/ Directory Reference

Files

- file **s2viz.cpp**
- file **s3viz.cpp**

  Visualize a curve in S^3 in two balls in R^3.

8.16 experimental/thickness/ Directory Reference

Directories

- directory **include**

Files

- file **compute_thickness.cpp**
- file **double_critical.cpp**
- file **thick.cpp**

8.17 tools/tmp/ Directory Reference

Files

- file **a.cpp**
- file **b.cpp**

8.18 tools/ Directory Reference

Directories

- directory **tmp**

Files

- file **align.cpp**

  Orient a pkf curve according to a given axes.

- file **angle_condition.cpp**

  Check the angle condition for an ideal knot. input file : s sigma tau ...

- file **angle_principal_normal_contacts.cpp**

  Prints angle between contact chords and principal normal to stdout.

- file **angle_vs_arclength.cpp**

  Prints the arclength and for that arclength the angle between the normal vector and the "2" contact struts.

- file **animate.cpp**
- file **arcdist.cpp**

  Program for testing purpose only.

- file **arcs2polygonal.cpp**

  Convert an arc curve to a polygonal curve.

- file **biarccli.cpp**

  Biarc *(p. 61) command line interface.*

- file **billiard_anim.cpp**
- file **billiard_poly.cpp**

  Construct billiard polygon.

- file **centerandnormalise.cpp**

  Center and normalize a closed PKF curve.

- file **circle_for_f31.cpp**

  Given eps, compute using 3 points the radius and center of circle.

- file **clif_tref_contactset.cpp**

  Compute the contact set and surface (iv file) for the clifford trefoil in S^ 3.

- file **closest_neighbour.cpp**

  Reconstruct the curve using the nearest point algo.

- file **contacts2inventor.cpp**

  Convert contact chords to obj/iv file.

- file **contacts2sigma.cpp**

  Contact chords to sigma(s) and tau(s).

- file **contacts_s3tor3.cpp**

  Project contact chords from S^3 to R^3.

- file **contactset.cpp**

  Compute the contact chords for a given knot.

- file **convolutionfilter.cpp**

  Smooth a curve using a local gaussian convolution filter.

- file **cross_pp.cpp**
- file **curvature.cpp**

  Outputs the curvature of a biarc curve.

- file **curvature4.cpp**

  Outputs the curvature of a biarc curve in R^4.

- file **curvesplit.cpp**

  Split a "closed" curve into segments for given arclength values.

- file **extract_sigma.cpp**

  Advancing within a small neighbournood try to extract sigma(s).

- file **extract_sigma_max.cpp**

  Find local max in pt valley.

- file **extract_sigma_valley_max.cpp**
- file **filter_nodes.cpp**

  Filter a PKF curve. I.e. remove points that are too close to some neighbour.

- file **flatten.cpp**

  Project curve onto 2D plane.

- file **follow_contact.cpp**

  Starting at some point on the curve, follow the contacts to depth max_level.

- file **inertiatensor.cpp**

  Compute the inertia tensor and principal axes.

- file **info.cpp**

  Computes Arc-length, center of mass, thickness (approx), ropelength (aprox) info of a PKF curve.

- file **info4.cpp**

  Computes Arc-length, center of mass, thickness (approx), ropelength (aprox) info of a PKF curve (S^3).

- file **inversion_in_sphere.cpp**

  Inversion in sphere at center c and radius r of a curve.

- file **length.cpp**

  Arc-length of a PKF curve.

- file **link_thickness.cpp**

  Compute thickness of a ***CurveBundle*** *(p. 102).*

- file **map.cpp**

  Map one curve to another.

- file **mesh4stokes.cpp**

  Produce a tube mesh from a curve and write that to a file.

- file **nana.cpp**

  Test program.

- file **normal_indicatrix.cpp**

  Unfinished.

- file **perturb.cpp**

  Perturb a curve.

- file **pkf2java.cpp**

  Read data from a PKF file, produce a tubular mesh for each curve in the file and write it as C/C++/Java arrays.

- file **pkf2mesh.cpp**

  Read data from a PKF file, produce a tubular mesh for each curve in the file and write that to a file.

- file **pkf2obj.cpp**

  pkf to obj format. obj file has correct uv texture coords.

- file **pkf2ply.cpp**

  pkf to mesh to stanford ply format.

- file **pkf2rib.cpp**
- file **pkf2stl.cpp**

  Read data from a PKF file, produce a tubular mesh for each curve in the file and write that to an STL file.

- file **pkf2vect.cpp**
- file **pkf2xyz.cpp**

  Convert PKF file to XYZ coordinates.

- file **pkfframe.cpp**

  Generate a framing along the curve.

- file **plotslice.cpp**

  Write a pt plot cross section at a given arclength.

- file **plotslice4.cpp**

  Write a pt plot cross section at a given arclength (S^3).

- file **pointat.cpp**

  Writes point/tangent at Curve(s), for s arclength and s in [0,1] to the standart output.

- file **process.cpp**
- file **projected_dij.cpp**

  Compute $\sum_{i \neq j} d_{i,j}$ and $\sum_{i \neq j} d_{i,j}^{-1}$ of a given projection.

- file **ptcircles.cpp**

  Resample an open or closed PKF curve.

- file **r3tos3.cpp**

  Project a curve in R^3 to S^3.

- file **redo_tangents.cpp**

  Recompute tangents using only the data points.

- file **refine.cpp**

  Refine/Resample a part of an open or closed PKF curve.

- file **reorder.cpp**

  Reorder the nodes of a curve, so that the first points of the 2 curves are the closest possible.

- file **resample.cpp**

  Resample an open or closed PKF curve.

- file **resample4.cpp**

  Resample an open or closed PKF curve (R^4).

- file **reverse_direction.cpp**

  Change the orientation of the curve.

- file **rotate_curve.cpp**

  Rotate curve about axis v by angle a.

- file **s3tor3.cpp**

  Project a curve on S^3 back to R^3.

- file **s_sigma_angles.cpp**

  Program for testing purpose only.

- file **setcenter.cpp**

  Reset the mass center of PKF curve.

- file **shuffle.cpp**

  Randomize/shuffle a curve.

- file **sigma2contact.cpp**

  From a sigma(s) function, generate the contact struts.

- file **sigma2inventor.cpp**

  Convert 2D sigma(s) to iv or obj file.

- file **sigma_and_tau_contacts.cpp**

  Extract contact functions sigma(s) and tau(s).

- file **sigma_composition.cpp**

  compute $^k(s)$. input file : s sigma tau ...

- file **supercoil.cpp**
- file **tangent_indicatrix.cpp**

  Write the tangent indicatrix of a given (open or closed) curve to a PKF file. Tangents for the tangent indicatrix are interpolated, therefore strange points like cusps would not be visible! The default is for an open curve, for closed curves put the -closed switch.

- file **test.cpp**
- file **torsion.cpp**

  Outputs the torsion of a biarc curve.

- file **vect2pkf.cpp**
- file **xyz2pkf.cpp**

  Convert XYZ coordinates to PKF.

8.19 lib/utils/ Directory Reference

Files

- file **qr.cpp**

8.20 include/utils/ Directory Reference

Files

- file **qr.h**
- file **timer.h**

8.21 experimental/valleyflow/ Directory Reference

Files

- file **valleyflow.cpp**

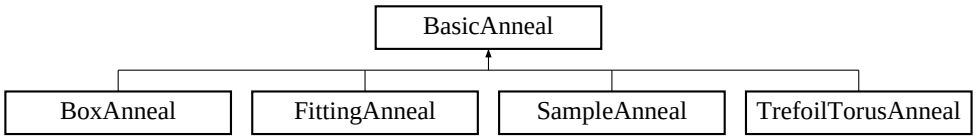
Chapter 9

Class Documentation

9.1 BasicAnneal Class Reference

Base class for annealing problems.

Inheritance diagram for BasicAnneal::



Public Member Functions

- **BasicAnneal** (const char *params="")
- virtual ~**BasicAnneal** ()
- void **std_init** (const char *params="")
- virtual ostream & **show_config** (ostream &out)
- virtual void **accept_curr_moves** ()
- virtual void **reject_curr_moves** ()
- virtual void **wiggle** ()
- virtual bool **stop** ()
- virtual float **energy** ()
- virtual void **best_found** ()
- virtual void **update_minmax_step** ()
- virtual ostream & **logline** (ostream &out)
- virtual void **log** ()
- virtual void **do_anneal** ()

Public Attributes

- float **min_step**
- float **max_step**
- int **min_move**

- int **max_move**
- int **energy_precision**
- float **Temp**
- float **Cooling**
- float **best_energy**
- float **curr_energy**
- float **candidate_energy**
- float **log_optimum**
- vector< **BasicMove** * > **curr_moves**
- vector< **BasicMove** * > **possible_moves**
- int **success**
- int **trys**
- int **log_counter**
- int **log_freq**
- int **last_best_log**
- string **best_filename**

9.1.1 Detailed Description

Base class for annealing problems.

Basic annealing class. Derive other annealing classes from this base class.

9.1.2 Constructor & Destructor Documentation

9.1.2.1 BasicAnneal::BasicAnneal (const char * *params* = "") [inline]

Constructor.

params - T - Temperature C - Cooling factor logfreq - Log frequency log_optimum energy_precision - how many digits precision best_filename

log_optimum is an "external" optimum Energy value. The logging system prints then the difference between this value and the current Emin.

References std_init().

9.1.2.2 virtual BasicAnneal::~BasicAnneal () [inline, virtual]

Virtual destructor.

9.1.3 Member Function Documentation

9.1.3.1 void BasicAnneal::std_init (const char * *params* = "") [inline]

Parse params string and initialize defaults.

Reimplemented in **TrefoilTorusAnneal** (p. 145).

Referenced by BasicAnneal(), BoxAnneal::BoxAnneal(), FittingAnneal::FittingAnneal(), SampleAnneal::SampleAnneal(), and TrefoilTorusAnneal::std_init().

9.1.3.2 virtual ostream& BasicAnneal::show_config (ostream & *out*) [inline, virtual]

Show the configuration parameters.

Reimplemented in **FittingAnneal** (p. 116).

Referenced by do_anneal(), and FittingAnneal::show_config().

9.1.3.3 virtual void BasicAnneal::accept_curr_moves () [inline, virtual]

Accept all current moves.

References BasicMove::accept().

Referenced by do_anneal().

9.1.3.4 virtual void BasicAnneal::reject_curr_moves () [inline, virtual]

Reject all current moves.

References BasicMove::reject().

Referenced by do_anneal().

9.1.3.5 virtual void BasicAnneal::wiggle () [inline, virtual]

Make random moves and store them in curr_moves.

Reimplemented in **BoxAnneal** (p. 75).

Referenced by do_anneal().

9.1.3.6 virtual bool BasicAnneal::stop () [inline, virtual]

Returns true when annealing considers to have converged.

Reimplemented in **BoxAnneal** (p. 74), **FittingAnneal** (p. 116), **SampleAnneal** (p. 137), and **TrefoilTorusAnneal** (p. 145).

Referenced by do_anneal().

9.1.3.7 virtual float BasicAnneal::energy () [inline, virtual]

The energy we want to optimize.

Reimplemented in **BoxAnneal** (p. 75), **FittingAnneal** (p. 116), **SampleAnneal** (p. 138), and **TrefoilTorusAnneal** (p. 146).

Referenced by do_anneal().

9.1.3.8 virtual void BasicAnneal::best_found () [inline, virtual]

This gets called each time anneal finds a new best energy.

Reimplemented in **BoxAnneal** (p. 75), **FittingAnneal** (p. 116), **SampleAnneal** (p. 138), and **TrefoilTorusAnneal** (p. 146).

Referenced by TrefoilTorusAnneal::best_found(), SampleAnneal::best_found(), FittingAnneal::best_found(), BoxAnneal::best_found(), and do_anneal().

9.1.3.9 virtual void BasicAnneal::update_minmax_step () [inline, virtual]

Adjust the min_step and max_step values as well as the min_move and max_move index to the corresponding move.

Referenced by FittingAnneal::best_found(), and logline().

9.1.3.10 virtual ostream& BasicAnneal::logline (ostream & *out*) [inline, virtual]

Write annealing information (temp, energy, minimal energy, success rate, min/max stepsize, and last best count) to ostream *out*.

References update_minmax_step().

Referenced by log(), and FittingAnneal::log().

9.1.3.11 virtual void BasicAnneal::log () [inline, virtual]

Perfomr a logging action (uses logline) but can be reimplemented in derived classes.

Reimplemented in **FittingAnneal** (p. 116).

References logline().

Referenced by do_anneal().

9.1.3.12 virtual void BasicAnneal::do_anneal () [inline, virtual]

Start the actual annealing!

References accept_curr_moves(), best_found(), energy(), log(), reject_curr_moves(), show_config(), stop(), and wiggle().

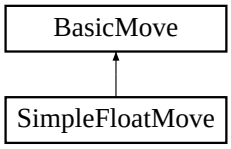
The documentation for this class was generated from the following file:

- experimental/annealing/my_anneal.cpp

9.2 BasicMove Class Reference

Basic class for annealing moves.

Inheritance diagram for BasicMove::



Public Member Functions

- **BasicMove** (float step_size=1e-6, float STEP_CHANGE=0.01)
- virtual ~**BasicMove** ()
- virtual void **move** ()
- virtual void **reject** ()
- virtual void **accept** ()

Public Attributes

- float **step_size**
- float **STEP_CHANGE**

9.2.1 Detailed Description

Basic class for annealing moves.

9.2.2 Constructor & Destructor Documentation

9.2.2.1 BasicMove::BasicMove (float *step_size* = `1e-6`, float *STEP_CHANGE* = `0.01`) `[inline]`

Constructor. Sets step_size and STEP_CHANGE.

9.2.2.2 virtual BasicMove::~BasicMove () `[inline, virtual]`

Virtual destructor.

9.2.3 Member Function Documentation

9.2.3.1 virtual void BasicMove::move () `[inline, virtual]`

Perform the move.

Reimplemented in **SimpleFloatMove** (p. 139).

9.2.3.2 virtual void BasicMove::reject () [inline, virtual]

Reject the move. Decrease step_size. Restore backup. Moves must be rejected in reverse order!

Reimplemented in **SimpleFloatMove** (p. 140).

Referenced by SimpleFloatMove::reject(), and BasicAnneal::reject_curr_moves().

9.2.3.3 virtual void BasicMove::accept () [inline, virtual]

Accept the move. Increase step_size. Discard backup. Moves must be accepted in reverse order!

Reimplemented in **SimpleFloatMove** (p. 140).

Referenced by SimpleFloatMove::accept(), and BasicAnneal::accept_curr_moves().

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

- experimental/annealing/my_anneal.cpp

9.3 Biarc< Vector > Class Template Reference

The **Biarc** (p. 61) class describes a single biarc.

`#include <include/Biarc.h>`

Public Member Functions

- **Biarc** ()
- **Biarc** (const Vector &p, const Vector &t)
- **Biarc** (const **Biarc**< Vector > &b)
- ~**Biarc** ()
- const Vector & **getPoint** () const
- const Vector & **getTangent** () const
- const Vector & **getMidPoint** () const
- const Vector & **getMidTangent** () const
- void **setPoint** (const Vector &p)
- void **setMidPoint** (const Vector &p)
- void **setMidTangent** (const Vector &p)
- void **setTangentUnnormalized** (const Vector &t)
- void **setTangent** (const Vector &t)
- void **get** (Vector &p, Vector &t) const
- void **set** (const Vector &p, const Vector &t)
- void **clear** ()
- void **reverse** ()
- int **isProper** () const
- int **isBiarc** () const
- void **setBiarc** ()
- void **make** (float Gamma)
- float **arclength0** () const
- float **arclength1** () const
- float **biarclength** () const
- float **radius0** () const
- float **radius1** () const
- int **id** () const
- void **setId** (const int i)
- void **setCurve** (**Curve**< Vector > *c)
- const **Curve**< Vector > * **getCurve** ()
- void **setIdAndCurve** (const int i, **Curve**< Vector > *c)
- Vector **pointOnArc0** (float s) const
- Vector **pointOnArc1** (float s) const
- Vector **pointOnBiarc** (float arclength) const
- Vector **tangentOnBiarc** (float arclength) const
- void **getBezierArc0** (Vector &b0, Vector &b1, Vector &b2) const
- void **getBezierArc1** (Vector &b0, Vector &b1, Vector &b2) const
- void **getBezier** (Vector &b0_0, Vector &b1_0, Vector &b2_0, Vector &b0_1, Vector &b1_1, Vector &b2_1) const
- Vector **a0** (float tau) const
- Vector **a1** (float tau) const
- const **Biarc**< Vector > & **getNext** () const

- const **Biarc**< Vector > & **getPrevious** () const
- void **setNext** (const **Biarc**< Vector > &b)
- void **setPrevious** (const **Biarc**< Vector > &b)
- void **setNextNULL** ()
- void **setPreviousNULL** ()
- **Biarc**< Vector > **operator**∗ (const float s) const
- **Biarc**< Vector > **operator**+ (const Vector &v) const
- **Biarc**< Vector > **operator-** (const Vector &v) const
- **Biarc**< Vector > & **operator**= (const **Biarc**< Vector > &b)
- **Biarc**< Vector > & **operator**+= (const Vector &v)
- **Biarc**< Vector > & **operator-**= (const Vector &v)
- **Biarc**< Vector > & **operator**/= (const float d)
- **Biarc**< Vector > & **operator**∗= (const float d)
- void **print** (ostream &out) const
- int **operator**== (const **Biarc**< Vector > &b) const
- int **operator!**= (const **Biarc**< Vector > &b) const

9.3.1 Detailed Description

template<class Vector> class Biarc< Vector >

The **Biarc** (p. 61) class describes a single biarc.

This class is used to store a single data point of an entire space curve in $\mathcal{R}^3$. A biarc object contains a point and the tangent at this particular point. Aligning one biarc after another makes up a biarc approximated space-curve.

The space-curve is a linked-list of **Biarc** (p. 61) elements. Every biarc element in the "curve list" has two neighbours on its left and right side, reachable by **getNext()** (p. 69) and **getPrevious()** (p. 69). If the point is the start or endpoint of an open curve, then it will only have a single neighbour. The other pointer will be set to NULL.

See also:

Curve (p. 76)

9.3.2 Constructor & Destructor Documentation

9.3.2.1 template<class Vector> Biarc< Vector >::Biarc () [inline]

Default constructor. Initializes this instance of a **Biarc** (p. 61) to <0,0,0> for the point and <0,1,0> for the tangent.

Referenced by Biarc< Vector >::operator∗(), Biarc< Vector >::operator+(), and Biarc< Vector >::operator-().

9.3.2.2 template<class Vector> Biarc< Vector >::Biarc (const Vector & *p*, const Vector & *t*) [inline]

Constructor sets the point to p and the tangent to t. Normalizes the tangent.

9.3.2.3 template<class Vector> Biarc< Vector >::Biarc (const Biarc< Vector > & *b*) [inline]

Copy Constructor. Copies only the point and the tangent. Doesn't care if we have an interpolated biarc or not.

9.3.2.4 template<class Vector> Biarc< Vector >::~Biarc () [inline]

Destructor.

9.3.3 Member Function Documentation

9.3.3.1 template<class Vector> const Vector & Biarc< Vector >::getPoint () const [inline]

Returns the point of the biarc as a Vector.

Referenced by Biarc< Vector >::a1(), Biarc< Vector >::isProper(), Biarc< Vector >::pointOnBiarc(), Curve< Vector >::radius_pt(), and Biarc< Vector >::tangentOnBiarc().

9.3.3.2 template<class Vector> const Vector & Biarc< Vector >::getTangent () const [inline]

Returns the tangent of the biarc as a Vector.

Referenced by Biarc< Vector >::isProper(), Curve< Vector >::radius_pt(), and Biarc< Vector >::tangentOnBiarc().

9.3.3.3 template<class Vector> const Vector & Biarc< Vector >::getMidPoint () const [inline]

Returns the matching point of this biarc, if we have interpolated biarcs. Posts a warning message and returns a zero vector if not.

Referenced by Biarc< Vector >::tangentOnBiarc().

9.3.3.4 template<class Vector> const Vector & Biarc< Vector >::getMidTangent () const [inline]

Returns the matching tangent of this biarc, if we have interpolated biarcs. Posts a warning message and returns a zero vector if this is not a valid biarc.

Referenced by Biarc< Vector >::tangentOnBiarc().

9.3.3.5 template<class Vector> void Biarc< Vector >::setPoint (const Vector & *p*) [inline]

Set point to *p*.

9.3.3.6 template<class Vector> void Biarc< Vector >::setMidPoint (const Vector & *p*) [inline]

Set midpoint to p.

9.3.3.7 template<class Vector> void Biarc< Vector >::setMidTangent (const Vector & *p*) [inline]

Set midtangent to p.

References Biarc< Vector >::getNext().

9.3.3.8 template<class Vector> void Biarc< Vector >::setTangentUnnormalized (const Vector & *t*) [inline]

Set tangent of the biarc. No normalization of the tangent is done.

9.3.3.9 template<class Vector> void Biarc< Vector >::setTangent (const Vector & *t*) [inline]

Set tangent of the biarc to t. Tangent automatically normalized.

9.3.3.10 template<class Vector> void Biarc< Vector >::get (Vector & *p*, Vector & *t*) const [inline]

Recover point/tangent data from this biarc and put it into p and t.

9.3.3.11 template<class Vector> void Biarc< Vector >::set (const Vector & *p*, const Vector & *t*) [inline]

Set point/tangent data for this biarc to p and t.

9.3.3.12 template<class Vector> void Biarc< Vector >::clear () [inline]

Sets the biarc data to : point <0,0,0> and tangent <0,1,0>.

9.3.3.13 template<class Vector> void Biarc< Vector >::reverse () [inline]

Sets the biarc point P to -P.

9.3.3.14 template<class Vector> int Biarc< Vector >::isProper () const [inline]

Checks if the **Biarc** (p. 61) is proper. Returns 1 if this is true. Posts a warning message and returns 0 if the biarc is not proper.

A biarc is proper, if the point-tangent data pair $((p_0, t_0), (p_1, t_1))$ satifies

$(p_1 - p_0) \cdot t_0 > 0$, and $(p_1 - p_0) \cdot t_1 > 0$.

This implies that for a **Biarc** (p. 61) b with point p and tangent t and its following biarc (obtained with **Biarc::getNext()** (p. 69)), with point *p'* and *tangent t'*, the previous condition must hold.

See also:

getNext() (p. 69), **getPrevious()** (p. 69).

References Biarc< Vector >::getNext(), Biarc< Vector >::getPoint(), and Biarc< Vector >::getTangent().

Referenced by Biarc< Vector >::make().

9.3.3.15 template<class Vector> int Biarc< Vector >::isBiarc () const [inline]

Returns true (=1) if the current biarc is interpolated, and 0 if not. Here interpolated means, that a valid matching point has been computed using this biarc and its following neighbour.

Referenced by Biarc< Vector >::getBezierArc0(), and Biarc< Vector >::getBezierArc1().

9.3.3.16 template<class Vector> void Biarc< Vector >::setBiarc () [inline]

Sets the "interpolated" flag to be true.

9.3.3.17 template<class Vector> void Biarc< Vector >::make (float *Gamma*) [inline]

This function computes the matching point of this biarc. It uses the point/tangent data of current biarc as starting point and the point/tangent data of the next biarc as the end point.

Giving a value for *Gamma*, it is possible to compute a point (matching point), that links the two data points by two circular arcs of circles. For each arc the control points of a Bezier curve are computed and can be recovered by **getBezier()** (p. 68).

The appropriate rule that computes the matching point is still an open question (since the matching point of a biarc is not unique but located on a particular arc of circle defined by the data points and tangents).

For now, this function takes as input *Gamma*, where *Gamma* is a parameter related to the ratio of the Bezier triangles over the two arcs of circles.

More details can be found in Jana Smutny's PhD Thesis at

`http://lcvmwww.epfl.ch/~lcvm/articles/thesis.html`

See also:

getBezier() (p. 68),**getBezierArc0()** (p. 68),**getBezierArc1()** (p. 68),**Curve::make()** (p. 86)

References Biarc< Vector >::getNext(), and Biarc< Vector >::isProper().

9.3.3.18 template<class Vector> float Biarc< Vector >::arclength0 () const [inline]

Returns the arclength of the first arc of circle.

See also:

arclength1() (p. 66), **biarclength()** (p. 66).

Referenced by Biarc< Vector >::tangentOnBiarc().

9.3.3.19 template<class Vector> float Biarc< Vector >::arclength1 () const [inline]

Returns the arclength of the second arc of circle.

See also:

arclength0() (p. 65), **biarclength()** (p. 66).

Referenced by Biarc< Vector >::tangentOnBiarc().

9.3.3.20 template<class Vector> float Biarc< Vector >::biarclength () const [inline]

Returns the arclength of the current biarc. This is the sum of the arcs one and two.

See also:

arclength0() (p. 65), **arclength1()** (p. 66).

9.3.3.21 template<class Vector> float Biarc< Vector >::radius0 () const [inline]

Compute and return the radius of the first arc.

9.3.3.22 template<class Vector> float Biarc< Vector >::radius1 () const [inline]

Compute and return the radius of the second arc.

9.3.3.23 template<class Vector> int Biarc< Vector >::id () const [inline]

Returns the position if this Biarcs is in a **Curve** (p. 76) object. -1 otherwise.

9.3.3.24 template<class Vector> void Biarc< Vector >::setId (const int *i*) [inline]

Private routine. Sets _BiarcInCurve value.

9.3.3.25 template<class Vector> void Biarc< Vector >::setCurve (Curve< Vector > * *c*) [inline]

Private routine. Sets _Curve value.

9.3.3.26 template<class Vector> const Curve< Vector > * Biarc< Vector >::getCurve () [inline]

Private routine. Get curve pointer.

9.3.3.27 template<class Vector> void Biarc< Vector >::setIdAndCurve (const int *i*, Curve< Vector > * *c*) [inline]

Private routine. Sets _BiarcInCurve and _Curve values.

9.3.3.28 template<class Vector> Vector Biarc< Vector >::pointOnArc0 (float *s*) const [inline]

Returns point $a(s)$ on arc one of the biarc, here s is arc-length parametrization.

We have $s \in [0, l_0]$, where l_0 is the arc-length of the current biarc's first arc.

No check is done if we have a valid biarc interpolation!

See also:

pointOnBiarc() (p. 67),**pointOnArc1()** (p. 67).

References Biarc< Vector >::a0(), Vector3::dot(), Vector3::norm(), and Vector3::normalize().

Referenced by Biarc< Vector >::pointOnBiarc().

9.3.3.29 template<class Vector> Vector Biarc< Vector >::pointOnArc1 (float *s*) const [inline]

Returns point $a(s)$ on arc one of the biarc, here s is arc-length parametrization.

We have $s \in [0, l_1]$, where l_1 is the arc-length of the current biarc's second arc.

No check is done if we have a valid biarc interpolation!

See also:

pointOnBiarc() (p. 67),**pointOnArc0()** (p. 67).

References Biarc< Vector >::a1(), Vector3::dot(), Biarc< Vector >::getNext(), Vector3::norm(), and Vector3::normalize().

Referenced by Biarc< Vector >::pointOnBiarc().

9.3.3.30 template<class Vector> Vector Biarc< Vector >::pointOnBiarc (float *arclength*) const [inline]

Returns the point $a(s)$ on the biarc. Here s is arc-length parametrization.

We have $s \in [0, l]$, where l is the total arc-length of the current biarc.

See also:

pointOnArc0() (p. 67),**pointOnArc1()** (p. 67).

References Biarc< Vector >::getNext(), Biarc< Vector >::getPoint(), Biarc< Vector >::pointOnArc0(), and Biarc< Vector >::pointOnArc1().

Referenced by Biarc< Vector >::tangentOnBiarc().

9.3.3.31 template<class Vector> Vector Biarc< Vector >::tangentOnBiarc (float *arclength*) const [inline]

Returns tangent vector at a(s) on the biarc

References Biarc< Vector >::arclength0(), Biarc< Vector >::arclength1(), Biarc< Vector >::getMidPoint(), Biarc< Vector >::getMidTangent(), Biarc< Vector >::getNext(), Biarc< Vector >::getPoint(), Biarc< Vector >::getTangent(), and Biarc< Vector >::pointOnBiarc().

9.3.3.32 template<class Vector> void Biarc< Vector >::getBezierArc0 (Vector & *b0*, Vector & *b1*, Vector & *b2*) const [inline]

This function is used to extract the Bezier curve control points of the first arc of circle of the biarc and put them into b0,b1,b2.

See also:

getBezier() (p. 68),**getBezierArc1()** (p. 68)

References Biarc< Vector >::isBiarc().

Referenced by Biarc< Vector >::getBezier().

9.3.3.33 template<class Vector> void Biarc< Vector >::getBezierArc1 (Vector & *b0*, Vector & *b1*, Vector & *b2*) const [inline]

This function is used to extract the Bezier curve control points of the second arc of circle of the biarc and put them into b0,b1,b2.

See also:

getBezier() (p. 68),**getBezierArc0()** (p. 68)

References Biarc< Vector >::getNext(), and Biarc< Vector >::isBiarc().

Referenced by Biarc< Vector >::getBezier().

9.3.3.34 template<class Vector> void Biarc< Vector >::getBezier (Vector & *b0_0*, Vector & *b1_0*, Vector & *b2_0*, Vector & *b0_1*, Vector & *b1_1*, Vector & *b2_1*) const [inline]

This method is used to get the Bezier representation of the biarc. The corners of an isosceles triangle over a circular arc are known as the Bezier points.

From a biarc interpolated curve it is easy to compute the Bezier curve control points for each particular arc.

The Bezier points of arc one comes into the Vector objects *b0_0*,b1_0,b2_0 and arc two comes in *b0_1*,b1_1,b2_1.

See also:

getBezierArc0() (p. 68),**getBezierArc1()** (p. 68)

References Biarc< Vector >::getBezierArc0(), and Biarc< Vector >::getBezierArc1().

9.3.3.35 template<class Vector> Vector Biarc< Vector >::a0 (float *tau*) const [inline]

Given a value *tau* between 0 and 1, this function returns the point $a(\tau)$ on the first arc of circle of the biarc. Where $a(\tau)$ is a non-arclength parametrization of an arc of circle.

For arc-length parametrized arcs use one of the pointOn* functions.

See also:

pointOnArc0() (p. 67),**pointOnArc1()** (p. 67),**pointOnBiarc()** (p. 67),**a1()** (p. 69).

Referenced by Biarc< Vector >::pointOnArc0().

9.3.3.36 template<class Vector> Vector Biarc< Vector >::a1 (float *tau*) const [inline]

Given a value *tau* between 0 and 1, this function returns the point $a(\tau)$ on the second arc of circle of the biarc. Where $a(\tau)$ is a non-arclength parametrization of an arc of circle.

For arc-length parametrized arcs use one of the pointOn* functions.

See also:

pointOnArc0() (p. 67),**pointOnArc1()** (p. 67),**pointOnBiarc()** (p. 67),**a0()** (p. 69).

References Biarc< Vector >::getNext(), and Biarc< Vector >::getPoint().

Referenced by Biarc< Vector >::pointOnArc1().

9.3.3.37 template<class Vector> const Biarc< Vector > & Biarc< Vector >::getNext () const [inline]

Returns a pointer to the next biarc. We get NULL if the current biarc is the last point on the curve (i.e. the end of the linked-list).

See also:

getPrevious() (p. 69),**setNext()** (p. 70),**setPrevious()** (p. 70),**setNextNULL()** (p. 70),**setPreviousNULL()** (p. 70).

Referenced by Biarc< Vector >::a1(), Biarc< Vector >::getBezierArc1(), Biarc< Vector >::isProper(), Biarc< Vector >::make(), Biarc< Vector >::pointOnArc1(), Biarc< Vector >::pointOnBiarc(), Biarc< Vector >::setMidTangent(), and Biarc< Vector >::tangentOnBiarc().

9.3.3.38 template<class Vector> const Biarc< Vector > & Biarc< Vector >::getPrevious () const [inline]

Returns a pointer to the previous biarc. We get NULL if the current biarc is the first point of the curve (i.e. the first element of the linked-list).

See also:

getNext() (p. 69),**setNext()** (p. 70),**setPrevious()** (p. 70),**setNextNULL()** (p. 70),**setPreviousNULL()** (p. 70).

9.3.3.39 template<class Vector> void Biarc< Vector >::setNext (const Biarc< Vector > & *b*) [inline]

Sets the next neighbour of this biarc to *b*.

See also:

getNext() (p. 69),**getPrevious()** (p. 69),**setPrevious()** (p. 70),**setNextNULL()** (p. 70),**setPreviousNULL()** (p. 70).

9.3.3.40 template<class Vector> void Biarc< Vector >::setPrevious (const Biarc< Vector > & *b*) [inline]

Sets the previous neighbour of this biarc to *b*.

See also:

getNext() (p. 69),**getPrevious()** (p. 69),**setPrevious()** (p. 70),**setNextNULL()** (p. 70),**setPreviousNULL()** (p. 70).

9.3.3.41 template<class Vector> void Biarc< Vector >::setNextNULL () [inline]

Deletes the link to the next neighbour by setting the pointer to NULL.

See also:

getNext() (p. 69),**getPrevious()** (p. 69),**setNext()** (p. 70),**setPrevious()** (p. 70),**setPreviousNULL()** (p. 70).

9.3.3.42 template<class Vector> void Biarc< Vector >::setPreviousNULL () [inline]

Deletes the link to the previous neighbour by setting the pointer to NULL.

See also:

getNext() (p. 69),**getPrevious()** (p. 69),**setNext()** (p. 70),**setPrevious()** (p. 70),**setNextNULL()** (p. 70).

9.3.3.43 template<class Vector> Biarc< Vector > Biarc< Vector >::operator* (const float *s*) const [inline]

Returns a biarc whose point value is multiplied by *s*.

References Biarc< Vector >::Biarc().

9.3.3.44 template<class Vector> Biarc< Vector > Biarc< Vector >::operator+ (const Vector & *v*) const [inline]

Returns a by *v* translated biarc. Tangent remains unchanged.

References Biarc< Vector >::Biarc().

9.3.3.45 template<class Vector> Biarc< Vector > Biarc< Vector >::operator-(const Vector & *v*) const [inline]

Returns a by -*v* translated biarc. Tangent remains unchanged.

References Biarc< Vector >::Biarc().

9.3.3.46 template<class Vector> Biarc< Vector > & Biarc< Vector >::operator=(const Biarc< Vector > & *b*) [inline]

Assign operator. Copies the values of the point and the tangent and sets the biarc flag _BIARC_ to zero.

References Biarc< Vector >::_BIARC_, Biarc< Vector >::_Curve, Biarc< Vector >::_Point, and Biarc< Vector >::_Tangent.

9.3.3.47 template<class Vector> Biarc< Vector > & Biarc< Vector >::operator+=(const Vector & *v*) [inline]

Adds *v* to the point value of this biarc and returns an instance to itself.

9.3.3.48 template<class Vector> Biarc< Vector > & Biarc< Vector >::operator-=(const Vector & *v*) [inline]

Subtracts *v* from the point value of this biarc and returns an instance to itself.

9.3.3.49 template<class Vector> Biarc< Vector > & Biarc< Vector >::operator/=(const float *d*) [inline]

Divides the point value of this biarc by *d* and returns an instance to itself.

9.3.3.50 template<class Vector> Biarc< Vector > & Biarc< Vector >::operator*=(const float *d*) [inline]

Multiplies the point value of this biarc by *d* and returns an instance to itself.

9.3.3.51 template<class Vector> void Biarc< Vector >::print (ostream & *out*) const [inline]

Prints starting point, tangent, matching point and matching tangent onto the stream *out*. If the biarc is not valid, it only prints the point/tangent data of this biarc and if this biarc data point is the last of the curve, then the tag –NULL– is printed at the end.

9.3.3.52 template<class Vector> int Biarc< Vector >::operator==(const Biarc< Vector > & *b*) const [inline]

Compares this instance to a biarc *b*. Returns 1 if both biarcs agree in point and tangent, 0 otherwise.

References Biarc< Vector >::_Point, and Biarc< Vector >::_Tangent.

9.3.3.53 template<class Vector> int Biarc< Vector >::operator!= (const Biarc< Vector > & *b*) const [inline]

Compares this instance to a biarc *b*. Returns 1 if the biarcs disagree in point or tangent or both, 0 otherwise.

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

- include/Biarc.h
- lib/Biarc.cpp

9.4 Box Class Reference

Square box class.

Public Member Functions

- **Vec2 v0** () const
- **Vec2 v1** () const
- **Vec2 v2** () const
- **Vec2 v3** () const
- **Box** (const **Box** &b)
- void **bbox** (**Vec2** *bl, **Vec2** *ur) const
- int **vecInBox** (const **Vec2** &v) const
- int **overlap** (const **Box** &b) const

Public Attributes

- **Vec2 c**
- float **ang**

Friends

- ostream & **operator<<** (ostream &out, const **Box** &b)

9.4.1 Detailed Description

Square box class.

Unit-boxes with center at x,y and orientation angle ang. Function v0 to v3 give the corner coordinates. bbox is the bounding box, vecInBox returns 1 if vector is inside the box and overlap checks this and another box for overlap.

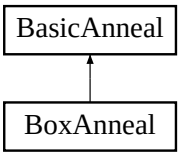
The documentation for this class was generated from the following file:

- experimental/annealing/box_problem.cpp

9.5 BoxAnneal Class Reference

N square boxes in a big square.

Inheritance diagram for BoxAnneal::



Public Member Functions

- **BoxAnneal** (const char *params="")
- bool **stop** ()
- int **check_for_overlap** ()
- void **wiggle** ()
- void **best_found** ()
- float **energy** ()

Public Attributes

- int **no_of_nodes**
- int **N**
- vector< **Box** > **nodes**
- vector< **Box** > **best_nodes**

9.5.1 Detailed Description

N square boxes in a big square.

This annealing class tries to put N square boxes in the smallest square containing them.

9.5.2 Constructor & Destructor Documentation

9.5.2.1 BoxAnneal::BoxAnneal (const char * *params* = "") [inline]

params - like **BasicAnneal** (p. 55). Addionally N - Number of Boxes

References Box::ang, Box::c, and BasicAnneal::std_init().

9.5.3 Member Function Documentation

9.5.3.1 bool BoxAnneal::stop () [inline, virtual]

Reimplemented stop criterion.

Reimplemented from **BasicAnneal** (p. 57).

9.5.3.2 int BoxAnneal::check_for_overlap () [inline]

Check wheter our boxes overlap.

Referenced by wiggle().

9.5.3.3 void BoxAnneal::wiggle () [inline, virtual]

Move and rotate a single box.

Reimplemented from **BasicAnneal** (p. 57).

References check_for_overlap().

9.5.3.4 void BoxAnneal::best_found () [inline, virtual]

This gets called each time anneal finds a new best energy.

Reimplemented from **BasicAnneal** (p. 57).

References BasicAnneal::best_found().

9.5.3.5 float BoxAnneal::energy () [inline, virtual]

Compute square surrounding the current boxes and use the edge size as energy.

Reimplemented from **BasicAnneal** (p. 57).

References Vec2::x, and Vec2::y.

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

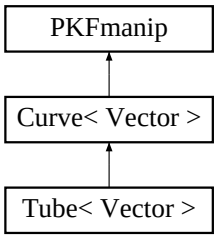
- experimental/annealing/box_problem.cpp

9.6 Curve< Vector > Class Template Reference

The **Curve** (p. 76) class for storing and manipulating a single biarc curve in $\mathcal{R}^3$.

`#include <include/Curve.h>`

Inheritance diagram for Curve< Vector >::



Public Member Functions

- **Curve** ()
- **Curve** (const char *filename)
- **Curve** (istream &in)
- **Curve** (const **Curve**< Vector > &curve)
- **Curve** & **operator**= (const **Curve**< Vector > &c)
- ~**Curve** ()
- biarc_ref **operator[]** (int c)
- biarc_constref **operator[]** (int c) const
- void **push** (const **Biarc**< Vector > &b)
- void **push** (const Vector &p, const Vector &t)
- void **append** (const **Biarc**< Vector > &b)
- void **append** (const Vector &p, const Vector &t)
- void **insert** (int loc, const **Biarc**< Vector > &b)
- void **insert** (int loc, const Vector &p, const Vector &t)
- void **remove** (int loc)
- void **remove** (biarc_it b)
- void **flush_all** ()
- biarc_constref **getNext** (int i) const
- biarc_constref **getPrevious** (int i)
- void **setNext** (int i, const **Biarc**< Vector > &b)
- void **setPrevious** (int i, const **Biarc**< Vector > &b)
- biarc_it **begin** ()
- biarc_it **end** ()
- int **isClosed** () const
- void **link** ()
- void **unlink** ()
- void **changeDirection** ()
- void **make** (float f)
- void **makeMidpointRule** ()
- void **make** (int from_N, int to_N, float f)
- void **make** (**Biarc**< Vector > *from, **Biarc**< Vector > *to, float f)
- void **makeMidpointRule** (int from_N, int to_N)

- void **makeMidpointRule** (**Biarc**< Vector > *from, **Biarc**< Vector > *to)
- void **resample** (int NewNoNodes)
- void **refine** (int from_N, int to_N, int NewNoNodes)
- void **refine** (biarc_it from, biarc_it to, int NewNoNodes)
- float **radius_pt** (int from, int to)
- float **radius_pt** (biarc_it from, biarc_it to)
- float **radius_pt** (const **Biarc**< Vector > &from, const **Biarc**< Vector > &to)
- float **radius_pt** (const Vector &p0, const Vector &t0, const Vector &p1) const
- float **radius_pt** (const float s, const float t) const
- float **pp** (int from, int to) const
- float **pp** (float s, float t) const
- float **radius_global** (**Biarc**< Vector > &at)
- float **thickness_fast** ()
- float **thickness** (Vector *from=NULL, Vector *to=NULL)
- void **get_hint** (int *i, int *j) const
- void **set_hint** (const int i, const int j)
- float **minSegDistance** ()
- float **maxSegDistance** ()
- float **span** () const
- float **distEnergy** ()
- float **curvature** (int n)
- float **curvature** (biarc_it b)
- Vector **normalVector** (int n)
- Vector **normalVector** (biarc_it b)
- float **torsion** (int n, int a)
- float **torsion2** (int n)
- void **inertiaTensor** (**Matrix3** &mat)
- void **principalAxis** (**Matrix3** &mat)
- void **computeTangents** ()
- void **polygonalToArcs** ()
- void **arcsToPolygonal** ()
- int **nodes** () const
- float **length** () const
- Vector **pointAt** (float s) const
- Vector **tangentAt** (float s) const
- biarc_it **biarcAt** (float s)
- int **biarcPos** (float s)
- **Curve** & **rotAroundAxis** (float angle, Vector axis)
- **Curve** & **operator+=** (const Vector &v)
- **Curve** & **operator-=** (const Vector &v)
- **Curve operator+** (const **Curve** &c) const
- **Curve operator-** (const **Curve** &c) const
- **Curve operator*** (const float s) const
- **Curve** & **apply** (**Matrix3** &m)
- Vector **getCenter** ()
- void **center** ()
- void **normalize** ()
- void **scale** (float s)
- void **check_tangents** ()
- int **readPKF** (const char *filename)

- int **readPKF** (istream &in)
- int **writePKF** (const char *filename, int Header=1)
- int **writePKF** (ostream &out, int Header=1)
- int **readXYZ** (const char *filename)
- int **readXYZ** (istream &in)
- int **readData** (const char *filename, const char *delimiter)
- int **readData** (istream &in, const char *delimiter)
- int **writeData** (const char *filename, const char *delimiter, int tangents_flag)
- int **writeData** (ostream &out, const char *delimiter, int tangents_flag)

Protected Member Functions

- biarc_constit **accessBiarc** (int i) const
- biarc_it **accessBiarc** (int i)
- int **readSinglePKF** (istream &in)
- ostream & **writeSinglePKF** (ostream &out)
- int **readSingleData** (istream &in, const char *delimiter)
- ostream & **writeSingleData** (ostream &out, const char *delimiter, int tangents_flag)
- int **readSingleXYZ** (istream &in)

Protected Attributes

- vector< **Biarc**< Vector > > **_Biarcs**
- int **_Closed**

9.6.1 Detailed Description

template<class Vector> class Curve< Vector >

The **Curve** (p. 76) class for storing and manipulating a single biarc curve in $\mathcal{R}^3$.

This class is used to store and manipulate a set of point/tangent data. This data can be interpolated to a biarc curve. The class is for open and closed curves, but this must be specified (how to do that is explained later in this text).

The space-curve is a linked-list of **Biarc** (p. 61) elements. Every biarc element in the "curve list" has two neighbours. If the point is the start or endpoint of an open curve, then it will only have a single neighbour.

In order to change data in a curve, the functions **push()** (p. 82), pop(), **append()** (p. 83), **insert()** (p. 83) or **remove()** (p. 84) can be used.

Here comes an example that reads in a PKF file describing a closed curve, interpolates the curve and resamples it :

```
#include "../include/Curve.h"

int main(int argc,char** argv) {

  // Read the data
  Curve c("data.pkf");

  // Close the curve
  c.link();
```

```
  // Interpolate the curve by biarcs
  c.make_default();

  // Resample the curve with 100 points.
  c.resample(100);

  // Write new curve to a file
  c.writePKF("resampled.pkf");

  return 0;
}
```

The next example illustrates how to build a circle with radius one and 10 data points :

```
#include "../include/Curve.h"

int main(int argc,char** argv) {

  // Init empty curve
  Curve<Vector3> circle;

  // Point and Tangent
  Vector3 p,t;

  // Construct PKF header
  circle.header("Circle","Author","","");

  // Build the circle with 10 nodes
  for (float i=0;i<10;i+=1.0) {

    // Compute point and tangent of the circle
    p = Vector3(sin(i*M_PI/5.0), cos(i*M_PI/5.0),0);
    t = Vector3(cos(i*M_PI/5.0),-sin(i*M_PI/5.0),0);

    // Normalize the tangent
    t.normalize();

    // Add point to the curve
    circle.append(p,t);
  }

  ...

  return 0;

}
```

See also:

Biarc (p. 61)

9.6.2 Constructor & Destructor Documentation

9.6.2.1 template<class Vector> Curve< Vector >::Curve () [inline]

Constructs an empty curve and sets the header to "No name","","","",""

9.6.2.2 template<class Vector> Curve< Vector >::Curve (const char * *filename*) [inline]

Constructs a curve by reading it from the istream *in*.

References Curve< Vector >::readPKF().

9.6.2.3 template<class Vector> Curve< Vector >::Curve (istream & *in*) [inline]

Read the curve data from a stream *in*. Does not read a PKF header. Call the **readHeader()** (p. 136) method for that purpose.

See also:

PKFmanip (p. 133)

References Curve< Vector >::readSinglePKF().

9.6.2.4 template<class Vector> Curve< Vector >::Curve (const Curve< Vector > & *curve*) [inline]

Copy constructor.

See also:

operator=

9.6.2.5 template<class Vector> Curve< Vector >::~Curve () [inline]

Destructor. Flushes all the elements of the curve object.

References Curve< Vector >::flush_all().

9.6.3 Member Function Documentation

9.6.3.1 template<class Vector> biarc_constit Curve< Vector >::accessBiarc (int *i*) const [inline, protected]

Private access function to the biarcs in the curve. Returns an STL iterator to a biarc

Referenced by Curve< Vector >::curvature(), Curve< Vector >::normalVector(), and Curve< Vector >::refine().

9.6.3.2 template<class Vector> biarc_it Curve< Vector >::accessBiarc (int *i*) [inline, protected]

Private access function to the biarcs in the curve. Returns an STL iterator to a biarc

9.6.3.3 template<class Vector> int Curve< Vector >::readSinglePKF (istream & *in*) [inline, protected]

Read in only the curve data, this is from the COMP tag to the curve's END tag!

See also:

readPKF() (p. 97)

References Curve< Vector >::append().

Referenced by Curve< Vector >::Curve(), and Curve< Vector >::readPKF().

9.6.3.4 template<class Vector> ostream & Curve< Vector >::writeSinglePKF (ostream & *out*) [inline, protected]

Write the current curve object to a stream *out*. Start with sending "COMP \#C" to the stream, where #C are the number of nodes currently stored in the **Curve** (p. 76) object. Then follow the NODE tags with the data point coordinates.

See also:

writePKF() (p. 98)

References Curve< Vector >::nodes().

Referenced by Curve< Vector >::writePKF().

9.6.3.5 template<class Vector> int Curve< Vector >::readSingleData (istream & *in*, const char * *delimiter*) [inline, protected]

Read in the curve data for a single curve given in VECT format! Only one component is supported. Skips the first two lines. Reads the number of vertices from the 2nd column of line three. Skips 3 more lines and reads in the coordinates.

See also:

readData() (p. 99)

References Curve< Vector >::append().

Referenced by Curve< Vector >::readData().

9.6.3.6 template<class Vector> ostream & Curve< Vector >::writeSingleData (ostream & *out*, const char * *delimiter*, int *tangents_flag*) [inline, protected]

Writes the current curve to a a stream. A custom delimiter may be specified and the tangents are also written or not depending on the value of the *tangents_flag*. 1 = please write the tangents to the file, 0 = please don't.

See also:

writeData() (p. 100)

References Curve< Vector >::nodes().

Referenced by Curve< Vector >::writeData().

9.6.3.7 template<class Vector> int Curve< Vector >::readSingleXYZ (istream & *in*) [inline, protected]

Read in xyz data. First line is the number of data points. Then the x y z coordinates. Tangents are interpolated using these points.

See also:

readPKF() (p. 97), **readData()** (p. 99)

References Curve< Vector >::append(), and Curve< Vector >::computeTangents().

Referenced by Curve< Vector >::readXYZ().

9.6.3.8 template<class Vector> Curve< Vector > & Curve< Vector >::operator= (const Curve< Vector > & *c*) [inline]

Assign operator. Copies all the point/tangent data. Closes the curve if *c* is closed.

References Curve< Vector >::_Biarcs, Curve< Vector >::_hint_i, Curve< Vector >::_hint_j, Curve< Vector >::begin(), Curve< Vector >::end(), Curve< Vector >::flush_all(), Curve< Vector >::isClosed(), Curve< Vector >::link(), and Curve< Vector >::nodes().

9.6.3.9]

template<class Vector> biarc_ref **Curve**< Vector >::operator[] (int *c*) [inline]

Returns biarc number *c* of the curve. Index starts off with zero (C style).

9.6.3.10]

template<class Vector> biarc_constref **Curve**< Vector >::operator[] (int *c*) const [inline]

Returns biarc number *c* of the curve. Index starts off with zero (C style).

9.6.3.11 template<class Vector> void Curve< Vector >::push (const Biarc< Vector > & *b*) [inline]

Adds a point/tangent data stored in biarc *b* at the front of the curve.

See also:

append() (p. 83),**insert()** (p. 83),pop(),**remove()** (p. 84)

Referenced by Curve< Vector >::push().

9.6.3.12 template<class Vector> void Curve< Vector >::push (const Vector & *p*, const Vector & *t*) [inline]

Adds a point *p*, tangent *t* data at the front of the curve.

See also:

append() (p. 83),**insert()** (p. 83),pop(),**remove()** (p. 84)

References Curve< Vector >::push().

9.6.3.13 template<class Vector> void Curve< Vector >::append (const Biarc< Vector > & *b*) [inline]

Adds a point/tangent data stored in biarc *b* at the end of the curve.

See also:

push() (p. 82),**insert()** (p. 83),pop(),**remove()** (p. 84)

Referenced by Curve< Vector >::append(), Curve< Vector >::arcsToPolygonal(), TrefoilTorusAnneal::energy(), Curve< Vector >::operator+(), Curve< Vector >::operator-(), Curve< Vector >::polygonalToArcs(), Curve< Vector >::readSingleData(), Curve< Vector >::readSinglePKF(), Curve< Vector >::readSingleXYZ(), CurveBundle< Vector >::readVECT(), and Curve< Vector >::resample().

9.6.3.14 template<class Vector> void Curve< Vector >::append (const Vector & *p*, const Vector & *t*) [inline]

Adds a point *p*, tangent *t* data at the end of the curve.

See also:

push() (p. 82),**insert()** (p. 83),pop(),**remove()** (p. 84)

References Curve< Vector >::append().

9.6.3.15 template<class Vector> void Curve< Vector >::insert (int *loc*, const Biarc< Vector > & *b*) [inline]

Insert a biarc *b* at position *loc* on the curve and push all the elements after *loc* one position to the right. The index starts with zero. If *loc* = nodes on the current curve, then the new element is added at the end of the curve.

See also:

push() (p. 82),**append()** (p. 83),pop(),**remove()** (p. 84)

Referenced by Curve< Vector >::insert().

9.6.3.16 template<class Vector> void Curve< Vector >::insert (int *loc*, const Vector & *p*, const Vector & *t*) [inline]

Insert a point *p*, tangent *t* data at position *loc* on the curve and push all the elements after *loc* one position to the right. The index starts with zero. If *loc* = nodes on the current curve, then the new element is added at the end of the curve.

See also:

push() (p. 82),**append()** (p. 83),pop(),**remove()** (p. 84)

References Curve< Vector >::insert().

9.6.3.17 template<class Vector> void Curve< Vector >::remove (int *loc*) [inline]

Remove biarc at position *loc*

Referenced by Curve< Vector >::resample().

9.6.3.18 template<class Vector> void Curve< Vector >::remove (biarc_it *b*) [inline]

Remove biarc at position *loc*

9.6.3.19 template<class Vector> void Curve< Vector >::flush_all () [inline]

Remove all the nodes on the curve.

Referenced by Tube< Vector >::operator=(), Curve< Vector >::operator=(), CurveBundle< Vector >::readVECT(), and Curve< Vector >::~Curve().

9.6.3.20 template<class Vector> biarc_constref Curve< Vector >::getNext (int *i*) const [inline]

Get a reference to the **Biarc** (p. 61) at position i+1 No check is made wheter i+1 exists! However it wraps around for closed curves!

References Curve< Vector >::_Closed, and Curve< Vector >::nodes().

9.6.3.21 template<class Vector> biarc_constref Curve< Vector >::getPrevious (int *i*) [inline]

Get a reference to the **Biarc** (p. 61) at position i-1 No check is made wheter i-1 exists for open curves! However it wraps around for closed curves!

References Curve< Vector >::_Closed.

9.6.3.22 template<class Vector> void Curve< Vector >::setNext (int *i*, const Biarc< Vector > & *b*) [inline]

Set the **Biarc** (p. 61) at position i+1 to be **Biarc** (p. 61) b

References Curve< Vector >::_Closed, and Curve< Vector >::nodes().

9.6.3.23 template<class Vector> void Curve< Vector >::setPrevious (int *i*, const Biarc< Vector > & *b*) [inline]

Set the **Biarc** (p. 61) at position i-1 to be **Biarc** (p. 61) b

References Curve< Vector >::_Closed, and Curve< Vector >::nodes().

9.6.3.24 template<class Vector> biarc_it Curve< Vector >::begin () [inline]

Return a STL iterator pointing to the first biarc

Referenced by Curve< Vector >::changeDirection(), Curve< Vector >::check_tangents(), Curve< Vector >::curvature(), Curve< Vector >::distEnergy(), Curve< Vector >::inertiaTensor(), Tube< Vector >::makeMesh(), Curve< Vector >::maxSegDistance(), Curve< Vector >::minSegDistance(), Curve< Vector >::normalVector(), Tube< Vector >::operator=(), Curve< Vector >::operator=(), Curve< Vector >::radius_global(), Curve< Vector >::refine(), Tube< Vector >::scaleTubeRadius(), Curve< Vector >::thickness(), and Curve< Vector >::thickness_fast().

9.6.3.25 template<class Vector> biarc_it Curve< Vector >::end () [inline]

Return a STL iterator pointing to the last biarc

Referenced by Curve< Vector >::changeDirection(), Curve< Vector >::check_tangents(), Curve< Vector >::curvature(), Curve< Vector >::distEnergy(), Curve< Vector >::inertiaTensor(), Curve< Vector >::maxSegDistance(), Curve< Vector >::minSegDistance(), Curve< Vector >::normalVector(), Tube< Vector >::operator=(), Curve< Vector >::operator=(), Curve< Vector >::refine(), Tube< Vector >::scaleTubeRadius(), Curve< Vector >::thickness(), and Curve< Vector >::thickness_fast().

9.6.3.26 template<class Vector> int Curve< Vector >::isClosed () const [inline]

Returns 1 if the curve is closed (i.e. the **link()** (p. 85) function had been used), and 0 otherwise.

References Curve< Vector >::_Closed.

Referenced by Curve< Vector >::biarcAt(), SoKnot::generatePrimitives(), SoKnot::GLRender(), Tube< Vector >::operator=(), Curve< Vector >::operator=(), Curve< Vector >::pointAt(), Curve< Vector >::polygonalToArcs(), Curve< Vector >::resample(), and Curve< Vector >::tangentAt().

9.6.3.27 template<class Vector> void Curve< Vector >::link () [inline]

This is the way to close a curve. It links the last element in the list to the first. If the curve is already linked, nothing happens.

References Curve< Vector >::_Closed.

Referenced by TrefoilTorusAnneal::energy(), Tube< Vector >::makeMesh(), Tube< Vector >::operator=(), Curve< Vector >::operator=(), CurveBundle< Vector >::readVECT(), and Curve< Vector >::resample().

9.6.3.28 template<class Vector> void Curve< Vector >::unlink () [inline]

Opens a closed curve. This cancels the link between the last and the first element in the curve. If the curve is already open, nothing happens.

References Curve< Vector >::_Closed.

Referenced by Tube< Vector >::makeMesh().

9.6.3.29 template<class Vector> void Curve< Vector >::changeDirection () [inline]

Change the direction of the curve. This flips the tangents and reorders the points, so that we run through the curve in the opposite direction.

References Curve< Vector >::begin(), Curve< Vector >::end(), and Curve< Vector >::nodes().

9.6.3.30 template<class Vector> void Curve< Vector >::make (float *f*) [inline]

Interpolate the point/tangent data actually stored in this **Curve** (p. 76) object. This computes all the matching points along the curve for a given value of f in [0,1].

A "good" matching rule is still a unknown, usually f=0.5 seems not to be a good guess. Some people use the matching point that is equidistant between the two data points to be interpolated.

See also:

Biarc::make() (p. 65),**CurveBundle::make()** (p. 105)

References Curve< Vector >::_Closed.

9.6.3.31 template<class Vector> void Curve< Vector >::makeMidpointRule () [inline]

Interpolate the point/tangent data actually stored in this **Curve** (p. 76) object. The computed matching points will be equidistant from their neighboring data points. The correct formulas are kindly provided by Antonio Trovato.

See also:

make() (p. 86), **CurveBundle::makeMidpointRule()** (p. 106)

References Curve< Vector >::_Closed, and Curve< Vector >::nodes().

9.6.3.32 template<class Vector> void Curve< Vector >::make (int *from_N*, int *to_N*, float *f*) [inline]

TODO DOC

9.6.3.33 template<class Vector> void Curve< Vector >::make (Biarc< Vector > * *from*, Biarc< Vector > * *to*, float *f*) [inline]

TODO DOC

9.6.3.34 template<class Vector> void Curve< Vector >::makeMidpointRule (int *from_N*, int *to_N*) [inline]

TODO DOC

9.6.3.35 template<class Vector> void Curve< Vector >::makeMidpointRule (Biarc< Vector > * *from*, Biarc< Vector > * *to*) [inline]

TODO DOC

9.6.3.36 template<class Vector> void Curve< Vector >::resample (int *NewNoNodes*) [inline]

This function resamples the current **Curve** (p. 76) object with *NewNoNodes* nodes on the curve. The whole curve is given as a sequence of biarcs and resampling by biarcs is therefore straightforward.

The resampling can only be done if the curve is biarc interpolated (meaning a call to one of the **make()** (p. 86) methods).

See also:

refine() (p. 87),**make()** (p. 86),**CurveBundle::resample()** (p. 106)

References Curve< Vector >::_Closed, Curve< Vector >::append(), Curve< Vector >::isClosed(), Curve< Vector >::length(), Curve< Vector >::link(), Curve< Vector >::nodes(), and Curve< Vector >::remove().

Referenced by Tube< Vector >::makeMesh().

9.6.3.37 template<class Vector> void Curve< Vector >::refine (int *from_N*, int *to_N*, int *NewNoNodes*) [inline]

This function puts *NewNoNodes* nodes between curve node *from_N* and node *to_N*. The count of the number of new nodes includes start and endpoint of that part of the curve.

The resampling can only be done if the curve is biarc interpolated (meaning a call to one of the **make()** (p. 86) methods).

So Far, periodic refinement is not yet implemented. I.e. it is not possible to refine the curve between node 'nodes-5' and node '10' !!!!

See also:

resample() (p. 87),**make()** (p. 86).

References Curve< Vector >::accessBiarc().

9.6.3.38 template<class Vector> void Curve< Vector >::refine (biarc_it *from*, biarc_it *to*, int *NewNoNodes*) [inline]

This function puts *NewNoNodes* nodes between pointer Biarc* from and Biarc* to. The count of the number of new nodes includes start and endpoint of that part of the curve.

The resampling can only be done if the curve is biarc interpolated (meaning a call to one of the **make()** (p. 86) methods).

So Far, periodic refinement is not yet implemented. I.e. it is not possible to refine the curve between node 'nodes-5' and node '10' !!!! We do not know which new node is the "start" of the new curve, that's why!

See also:

resample() (p. 87),**make()** (p. 86).

References Curve< Vector >::_Closed, Curve< Vector >::begin(), Curve< Vector >::end(), and Curve< Vector >::normalize().

9.6.3.39 template<class Vector> float Curve< Vector >::radius_pt (int *from*, int *to*) [inline]

This function returns the radius of a circle that goes through point *from* and is tangent to the curve at point *to*. This function is commonly called $\rho_{pt}(s,t)$. The arguments are the positions of the datapoints.

See also:

thickness() (p. 90), **thickness_fast()** (p. 89)

Referenced by Curve< Vector >::radius_global(), Curve< Vector >::radius_pt(), and Curve< Vector >::thickness_fast().

9.6.3.40 template<class Vector> float Curve< Vector >::radius_pt (biarc_it *from*, biarc_it *to*) [inline]

This function returns the radius of a circle that goes through point *from* and is tangent to the curve at point *to*. This function is commonly called $\rho_{pt}(s,t)$. The arguments are two pointers to biarcs.

See also:

thickness() (p. 90), **thickness_fast()** (p. 89)

References Curve< Vector >::radius_pt().

9.6.3.41 template<class Vector> float Curve< Vector >::radius_pt (const Biarc< Vector > & *from*, const Biarc< Vector > & *to*) [inline]

This function returns the radius of a circle that goes through point *from* and is tangent to the curve at point *to*. This function is commonly called $\rho_{pt}(s,t)$. The arguments are two pointers to biarcs.

See also:

thickness() (p. 90), **thickness_fast()** (p. 89)

References Biarc< Vector >::getPoint(), Biarc< Vector >::getTangent(), and Curve< Vector >::radius_pt().

9.6.3.42 template<class Vector> float Curve< Vector >::radius_pt (const Vector & *p0*, const Vector & *t0*, const Vector & *p1*) const [inline]

This function returns the radius of a circle that goes through point *p0* and its tangent t0 to the curve at point *p1*. This function is commonly called $\rho_{pt}(s,t)$. The arguments are thre Vectors p0,t0,p1.

See also:

thickness() (p. 90), **thickness_fast()** (p. 89)

9.6.3.43 template<class Vector> float Curve< Vector >::radius_pt (const float *s*, const float *t*) const [inline]

This function returns the radius of a circle that goes through point curve(s) and is tangent to the curve at point curve(t). This function is commonly called $\rho_{pt}(s,t)$. The arguments s and t are arclength parameters.

See also:

thickness() (p. 90), **thickness_fast()** (p. 89)

References Curve< Vector >::pointAt(), Curve< Vector >::radius_pt(), and Curve< Vector >::tangentAt().

9.6.3.44 template<class Vector> float Curve< Vector >::pp (int *from*, int *to*) const [inline]

Compute the pp (euclidean distance) function between two nodes

9.6.3.45 template<class Vector> float Curve< Vector >::pp (float *s*, float *t*) const [inline]

Compute the pp (euclidean distance) function between $\gamma(s)$ and $\gamma(t)$

References Curve< Vector >::pointAt().

9.6.3.46 template<class Vector> float Curve< Vector >::radius_global (Biarc< Vector > & *at*) [inline]

This function returns the radius of a circle that goes through the 3 points x,y,z. This function is commonly called $\rho_{ppp}(s,t)$.

See also:

thickness() (p. 90), **thickness_fast()** (p. 89)

References Curve< Vector >::begin(), and Curve< Vector >::radius_pt().

9.6.3.47 template<class Vector> float Curve< Vector >::thickness_fast () [inline]

Returns the diameter of the fattest possible tube around the biarc curve. This is the inaccurate/fast version where we only look for the smallest local radius of curvature or the smallest rho_pt.

See also:

radius_pt() (p. 88)

References Curve< Vector >::_Closed, Curve< Vector >::begin(), Curve< Vector >::end(), and Curve< Vector >::radius_pt().

9.6.3.48 template<class Vector> float Curve< Vector >::thickness (Vector * *from* = NULL, Vector * *to* = NULL) [inline]

Returns the diameter of the thickest possible tube around the biarc curve without self intersection or crossing. This function implements the subdivision scheme as proposed in the thesis of Jana Smutny.

The arguments from and to give the exact position of the contact corresponding to thickness (if not NULL).

See also:

radius_pt() (p. 88), **thickness_fast()** (p. 89)

References Curve< Vector >::begin(), and Curve< Vector >::end().

Referenced by TrefoilTorusAnneal::energy().

9.6.3.49 template<class Vector> void Curve< Vector >::get_hint (int * *i*, int * *j*) const [inline]

Get the thickness hint. This can speed up the thickness computation if executed several times with changing only a little the curve. As for example in annealing.

See also:

set_hint() (p. 90)

9.6.3.50 template<class Vector> void Curve< Vector >::set_hint (const int *i*, const int *j*) [inline]

Set the thickness hint. This can speed up the thickness computation if executed several times with changing only a little the curve. As for example in annealing.

See also:

get_hint() (p. 90)

9.6.3.51 template<class Vector> float Curve< Vector >::minSegDistance () [inline]

Returns the length of the smallest biarc in the current curve.

See also:

maxSegDistance() (p. 91)

References Curve< Vector >::_Closed, Curve< Vector >::begin(), and Curve< Vector >::end().

9.6.3.52 template<class Vector> float Curve< Vector >::maxSegDistance () [inline]

Returns the length of the longest biarc in the current curve.

See also:

minSegDistance() (p. 90)

References Curve< Vector >::_Closed, Curve< Vector >::begin(), and Curve< Vector >::end().

9.6.3.53 template<class Vector> float Curve< Vector >::span () const [inline]

Return distance between the 2 most distant points.

9.6.3.54 template<class Vector> float Curve< Vector >::distEnergy () [inline]

Returns the Distance Energy in the current curve.

See also:

maxSegDistance() (p. 91), **minSegDistance()** (p. 90)

References Curve< Vector >::_Closed, Curve< Vector >::begin(), Curve< Vector >::end(), Curve< Vector >::length(), and Curve< Vector >::nodes().

9.6.3.55 template<class Vector> float Curve< Vector >::curvature (int *n*) [inline]

Computes the curvature at biarc number *n*.

This curvature function is independant of the interpolation it takes only data points/tangents and computes the curvature approximation as given in Smutny's Thesis pp.60

Caution : No inflection points test, so far.

See also:

torsion() (p. 92),**torsion2()** (p. 92)

References Curve< Vector >::accessBiarc().

9.6.3.56 template<class Vector> float Curve< Vector >::curvature (biarc_it *b*) [inline]

Computes the curvature at biarc *b*.

This curvature function is independant of the interpolation it takes only data points/tangents and computes the curvature approximation as given in Smutny's Thesis pp.60

Caution : No inflection points test, so far.

See also:

torsion() (p. 92),**torsion2()** (p. 92)

References Curve< Vector >::_Closed, Curve< Vector >::begin(), and Curve< Vector >::end().

9.6.3.57 template<class Vector> Vector Curve< Vector >::normalVector (int *n*) [inline]

Returns the normal vector at biarc number *n*.

References Curve< Vector >::accessBiarc().

9.6.3.58 template<class Vector> Vector Curve< Vector >::normalVector (biarc_it *b*) [inline]

Returns the normal vector at biarc *b*.

References Curve< Vector >::begin(), Vector3::dot(), and Curve< Vector >::end().

9.6.3.59 template<class Vector> float Curve< Vector >::torsion (int *n*, int *a*) [inline]

Returns the 3 sin(angle)/arclength, where angle is the angle between the planes in which 2 consequent arcs lie.

Parameter a is 0 for arc 1 and 1 for the second arc

The torsion computation as given in Smutny's Thesis p.64 it is also only based on point/tangent data.

Caution : No inflection points test, so far.

See also:

torsion2() (p. 92),**curvature()** (p. 91)

References Curve< Vector >::_Closed, and Curve< Vector >::nodes().

9.6.3.60 template<class Vector> float Curve< Vector >::torsion2 (int *n*) [inline]

Returns the torsion at biarc number *n*.

Slightly modified version of the **torsion()** (p. 92) computation. This time we consider a biarc's previous and next midpoint to compute the torsion at the current biarc.

Interpolated biarcs are necessary!

Caution : No inflection points test, so far. The torsion at the beginning and the end of an open curve is clamped to zero.

See also:

torsion() (p. 92),**curvature()** (p. 91)

References Curve< Vector >::_Closed, and Curve< Vector >::nodes().

9.6.3.61 template<class Vector> void Curve< Vector >::inertiaTensor (Matrix3 & *mat*) [inline]

Compute the inertia tensor of this curve. We consider uniform density along the knot centerline.

Returns a **Matrix3** (p. 119) object.

References Curve< Vector >::begin(), Curve< Vector >::end(), and Matrix3::zero().

Referenced by Curve< Vector >::principalAxis().

9.6.3.62 template<class Vector> void Curve< Vector >::principalAxis (Matrix3 & *mat*) [inline]

Compute the principal axis of the curve. Return them as colons of a **Matrix3** (p. 119)

References Matrix3::id(), and Curve< Vector >::inertiaTensor().

9.6.3.63 template<class Vector> void Curve< Vector >::computeTangents () [inline]

Computes the tangents for a set of points only. The tangent t_i at the point p_i is set to

$t_i = \frac{p_{i+1} - p_{i-1}}{|p_{i+1} - p_{i-1}|}$

If the curve is open the first and the last point get the tangent

$t_0 = \frac{p_1 - p_0}{|p_1 - p_0|}$ $t_{N-1} = \frac{p_{N-1} - p_{N-2}}{|p_{N-1} - p_{N-2}|}$

References Curve< Vector >::_Closed.

Referenced by TrefoilTorusAnneal::energy(), Curve< Vector >::readSingleXYZ(), and CurveBundle< Vector >::readVECT().

9.6.3.64 template<class Vector> void Curve< Vector >::polygonalToArcs () [inline]

This function converts a polygonal curve into a curve made of arcs of circles.

The current data points (supposed to describe a polygonal curve) stored in **Curve** (p. 76) c are transformed in the following way

$p_i^{new} = \frac{p_i^{old} + p_{i+1}^{old}}{2}$

the corresponding tangents at each of these points p_i if given by

$t_i = \frac{p_i^{old} + p_{i+1}^{old}}{2\,|p_i^{old} - p_{i+1}^{old}|}$

See also:

arcsTolPolygonal()

References Curve< Vector >::append(), and Curve< Vector >::isClosed().

9.6.3.65 template<class Vector> void Curve< Vector >::arcsToPolygonal () [inline]

This function converts a biarc curve into a polygonal curve.

The vertices for the polygonal curve are the Bezier control points of the biarc interpolated curve, by taking only the control point at the tip of the triangle.

The tangent data for the polygonal knot are set to <0,1,0>.

Caution : a curve with N biarcs yields a polygonal curve with 2N vertices. This is true for an open and for a closed curve, since for an open curve the endpoints of the original biarc curve are also included.

See also:

arcsToPolygonal() (p. 93)

References Curve< Vector >::_Closed, Curve< Vector >::append(), and Curve< Vector >::nodes().

9.6.3.66 template<class Vector> int Curve< Vector >::nodes () const [inline]

Returns the number of nodes on the curve. A node is a point/tangent data.

See also:

Tube::radius() (p. 150),**Tube::segments()** (p. 150)

Referenced by Curve< Vector >::arcsToPolygonal(), Curve< Vector >::changeDirection(), Curve< Vector >::distEnergy(), TrefoilTorusAnneal::energy(), Curve< Vector >::getCenter(), Curve< Vector >::getNext(), SoKnot::GLRender(), Tube< Vector >::makeMesh(), Curve< Vector >::makeMidpointRule(), Curve< Vector >::operator+(), Curve< Vector >::operator-(), Tube< Vector >::operator=(), Curve< Vector >::operator=(), Curve< Vector >::resample(), Tube< Vector >::scaleTubeRadius(), SoKnot::setKnot(), Curve< Vector >::setNext(), Curve< Vector >::setPrevious(), Curve< Vector >::torsion(), Curve< Vector >::torsion2(), SoKnot::updateMesh(), Curve< Vector >::writeSingleData(), and Curve< Vector >::writeSinglePKF().

9.6.3.67 template<class Vector> float Curve< Vector >::length () const [inline]

Returns the arc-length of the whole curve, making the correct difference between an open or a closed curve.

References Curve< Vector >::_Closed.

Referenced by Curve< Vector >::biarcAt(), Curve< Vector >::distEnergy(), TrefoilTorusAnneal::energy(), Curve< Vector >::normalize(), Curve< Vector >::pointAt(), Curve< Vector >::resample(), and Curve< Vector >::tangentAt().

9.6.3.68 template<class Vector> Vector Curve< Vector >::pointAt (float *s*) const [inline]

Return Point at **Curve** (p. 76) position Curve(s), where s is in [0,curvelength]

References Curve< Vector >::isClosed(), and Curve< Vector >::length().

Referenced by Curve< Vector >::pp(), and Curve< Vector >::radius_pt().

9.6.3.69 template<class Vector> Vector Curve< Vector >::tangentAt (float *s*) const [inline]

Return Tangent at **Curve** (p. 76) position Curve(s), where s is in [0,curvelength]

References Curve< Vector >::isClosed(), and Curve< Vector >::length().

Referenced by Curve< Vector >::radius_pt().

9.6.3.70 template<class Vector> biarc_it Curve< Vector >::biarcAt (float *s*) [inline]

Returns an STL iterator to the biarc on which the point Curve(s) is. s is in [0,curvelength]

References Curve< Vector >::isClosed(), and Curve< Vector >::length().

Referenced by Curve< Vector >::biarcPos().

9.6.3.71 template<class Vector> int Curve< Vector >::biarcPos (float *s*) [inline]

Returns the position number of the biarc at Curve(s). s is in [0,curvelength]

References Curve< Vector >::biarcAt().

9.6.3.72 template<class Vector> Curve< Vector > & Curve< Vector >::rotAroundAxis (float *angle*, Vector *axis*) [inline]

Rotate curve about some given axis. The rotation angle is given in radians!!!

References Curve< Vector >::apply().

Referenced by TrefoilTorusAnneal::energy().

9.6.3.73 template<class Vector> Curve< Vector > & Curve< Vector >::operator+= (const Vector & *v*) [inline]

Translates all the curve points by *v*.

Redo the interpolation after this operation if the initial curve was biarc interpolated, since the matching points and bezier points are no longer correct.

See also:

center() (p. 96).

9.6.3.74 template<class Vector> Curve< Vector > & Curve< Vector >::operator-= (const Vector & *v*) [inline]

Translates all the curve points by -*v*.

Redo the interpolation after this operation if the initial curve was biarc interpolated, since the matching points and bezier points are no longer correct.

See also:

center() (p. 96).

9.6.3.75 template<class Vector> Curve< Vector > Curve< Vector >::operator+ (const Curve< Vector > & *c*) const [inline]

Add a curve to another. Checks wheter the two curves have the same number of nodes.

References Curve< Vector >::_Biarcs, Curve< Vector >::append(), PKFmanip::getCite(), PKFmanip::getEtic(), PKFmanip::getHistory(), PKFmanip::getName(), PKFmanip::header(), and Curve< Vector >::nodes().

9.6.3.76 template<class Vector> Curve< Vector > Curve< Vector >::operator- (const Curve< Vector > & *c*) const [inline]

Remove a curve from another. Checks wheter the two curves have the same number of nodes.

References Curve< Vector >::_Biarcs, Curve< Vector >::append(), PKFmanip::getCite(), PKFmanip::getEtic(), PKFmanip::getHistory(), PKFmanip::getName(), PKFmanip::header(), and Curve< Vector >::nodes().

9.6.3.77 template<class Vector> Curve< Vector > Curve< Vector >::operator* (const float *s*) const [inline]

Returns a copy of the curve scaled by a factor of s.

References Curve< Vector >::scale().

9.6.3.78 template<class Vector> Curve< Vector > & Curve< Vector >::apply (Matrix3 & *m*) [inline]

Applies the rotation specified by a rotation matrix m to the curve. No check is done for m, the user must know what matrix he wants to apply.

This is not the standart 4x4 transformation matrix approach known from homogeneous coordinates stuff.

Referenced by Curve< Vector >::rotAroundAxis().

9.6.3.79 template<class Vector> Vector Curve< Vector >::getCenter () [inline]

Returns the curve's center of mass.

See also:

center() (p. 96)

Reimplemented in **Tube< Vector >** (p. 151), and **Tube< Vector3 >** (p. 151).

References Curve< Vector >::nodes().

Referenced by Curve< Vector >::center().

9.6.3.80 template<class Vector> void Curve< Vector >::center () [inline]

Centers the curve's mass center to <0,0,0>.

See also:

getCenter() (p. 96)

References Curve< Vector >::getCenter().

9.6.3.81 template<class Vector> void Curve< Vector >::normalize () [inline]

Normalize the length of the curve. The resulting curve will have arclenth one.

An interpolated curve is necessary to compute the length of it.

See also:

scale() (p. 97),**CurveBundle::normalize()** (p. 106)

References Curve< Vector >::length(), and Curve< Vector >::scale().

Referenced by Curve< Vector >::refine().

9.6.3.82 template<class Vector> void Curve< Vector >::scale (float *s*) [inline]

Scales the length of the curve by *s*. The curve needs not to be interpolated, since only the data points are changed.

See also:

normalize() (p. 97),**CurveBundle::scale()** (p. 108)

Referenced by Curve< Vector >::normalize(), and Curve< Vector >::operator*().

9.6.3.83 template<class Vector> void Curve< Vector >::check_tangents () [inline]

Prints **Biarc** (p. 61) and Tangent norm, if abs(norm-1)>1e-4

References Curve< Vector >::begin(), and Curve< Vector >::end().

9.6.3.84 template<class Vector> int Curve< Vector >::readPKF (const char * *filename*) [inline]

Read a PKF curve from the file *filename*.

See also:

writePKF() (p. 98)

Referenced by Curve< Vector >::Curve().

9.6.3.85 template<class Vector> int Curve< Vector >::readPKF (istream & *in*) [inline]

This function reads the curve data from a file *filename*, known as a portable knot format file (PKF). The PKF comes from the links-knots library written by Ben Laurie and Myk Soar.

An initial header gives details about the knot or link, such as the number of components (curves), the name of the current knot and more. The tags ETIC,CITE and HIST store a string for copyright purpose. The same tags ended with *L give the length of the string. The header manipulation is done in the **PKFmanip** (p. 133) class.

Each curve centerline is given as a list of point/tangent data.

The **Curve** (p. 76) class can only handle a single curve. If the number of components in the file is larger than 1, a warning message is posted to the output, saying that a **CurveBundle** (p. 102) object is necessary to process all the curves in the file.

It follows an example of a PKF file :

```
PKF 0.2
BIARC_KNOT "Knot Name"
ETICL 14
ETIC "Please cite me"
END
CITEL 14
CITE "I have to cite"
END
HISTL 7
HIST "Remarks"
END
NCMP <Number of components>
COMP <Number of nodes for component 1>
NODE px py pz tx ty tz
NODE 1.1363 0.2903 0.1548 0.2936 1.2251 0.2837
NODE 0.7187 0.8510 0.0271 -0.6460 0.7450 0.1103
...
END
COMP <Number of nodes for component 2>
...
END
```

Returns 1 if all went well, zero otherwise.

See also:

writePKF() (p. 98),**CurveBundle** (p. 102),**PKFmanip** (p. 133)

References PKFmanip::readHeader(), and Curve< Vector >::readSinglePKF().

9.6.3.86 template<class Vector> int Curve< Vector >::writePKF (const char * *filename*, int *Header* = 1) [inline]

Writes the current curve object to a portable PKF file *filename*. For more than 1 component use the **CurveBundle** (p. 102) class.

Returns 1 if all went well, zero otherwise.

See also:

readPKF() (p. 97),**CurveBundle** (p. 102)

Referenced by TrefoilTorusAnneal::energy().

9.6.3.87 template<class Vector> int Curve< Vector >::writePKF (ostream & *out*, int *Header* = 1) [inline]

Writes the current curve object to a stream *out*. For more than 1 component use the **CurveBundle** (p. 102) class.

Returns 1 if all went well, zero otherwise.

See also:

readPKF() (p. 97),**CurveBundle** (p. 102)

References PKFmanip::writeHeader(), and Curve< Vector >::writeSinglePKF().

9.6.3.88 template<class Vector> int Curve< Vector >::readXYZ (const char * *filename*) [inline]

Read the curve data (only x,y,z coordinates) from a file *filename*, where the delimiter can be any char*. The delimiter is a space.

Referenced by TubeBundle< Vector >::readXYZ(), and CurveBundle< Vector >::readXYZ().

9.6.3.89 template<class Vector> int Curve< Vector >::readXYZ (istream & *in*) [inline]

This function reads the curve data from a file *infile*. The file structure is a list of x,y,z coordinates. Delimiter is space " "

Returns 1 if all went well, zero otherwise.

See also:

readPKF() (p. 97), **readData()** (p. 99)

References Curve< Vector >::readSingleXYZ().

9.6.3.90 template<class Vector> int Curve< Vector >::readData (const char * *filename*, const char * *delimiter*) [inline]

Read the curve data (only x,y,z coordinates) from a file *filename*, where the delimiter can be any char*. The default delimiter is a space.

Referenced by TubeBundle< Vector >::readData(), and CurveBundle< Vector >::readData().

9.6.3.91 template<class Vector> int Curve< Vector >::readData (istream & *in*, const char * *delimiter*) [inline]

This function reads the curve data from a file *infile*. The file structure is a list of x,y,z coordinates. The default for the delimiter is " ", but can be changed with the second argument of the function.

The first line gives the number of nodes in the following format {**nodes** (p. 94)}. Then the coordinates are read in. The *delimiter* argument is any string that separates the coordinate values from each other, the default value is a space ' ' delimiter.

Returns 1 if all went well, zero otherwise.

See also:

computeTangents() (p. 93),**polygonalToArcs()** (p. 93),**arcsToPolygonal()** (p. 93)

References Curve< Vector >::readSingleData().

9.6.3.92 template<class Vector> int Curve< Vector >::writeData (const char * *filename*, const char * *delimiter*, int *tangents_flag*) [inline]

Write the curve to a file. The format is : the number of curves at the first line, then the number of data points (the format is {#N}) and then the list of x,y,z coordinates for each data point.

See also:

writeSingleData() (p. 81),**readData()** (p. 99)

References Curve< Vector >::writeSingleData().

9.6.3.93 template<class Vector> int Curve< Vector >::writeData (ostream & *out*, const char * *delimiter*, int *tangents_flag*) [inline]

Writes the curve to a data file. First line is the number of points and then follows a list of x,y,z coordinates. If the *tangents_flag* is set to 1, the tangents of the points are also dropped (Default is 0).

In fact we call only the **readSingleData()** (p. 81) function.

See also:

writeSingleData() (p. 81),**readData()** (p. 99)

References Curve< Vector >::writeSingleData().

9.6.4 Member Data Documentation

9.6.4.1 template<class Vector> int Curve< Vector >::_Closed [protected]

Flag is set to 1 if the curve is closed, 0 otherwise.

See also:

isClosed() (p. 85),**link()** (p. 85),**unlink()** (p. 85).

Referenced by Curve< Vector >::arcsToPolygonal(), Curve< Vector >::computeTangents(), Curve< Vector >::curvature(), Curve< Vector >::distEnergy(), Curve< Vector >::getNext(), Curve< Vector >::getPrevious(), Curve< Vector >::isClosed(), Curve< Vector >::length(), Curve< Vector >::link(), Curve< Vector >::make(), Tube< Vector >::makeMesh(), Curve< Vector >::makeMidpointRule(), Curve< Vector >::maxSegDistance(), Curve< Vector >::minSegDistance(), Curve< Vector >::refine(), Curve< Vector >::resample(), Tube< Vector >::scaleTubeRadius(), Curve< Vector >::setNext(), Curve< Vector >::setPrevious(), Curve< Vector >::thickness_fast(), Curve< Vector >::torsion(), Curve< Vector >::torsion2(), and Curve< Vector >::unlink().

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

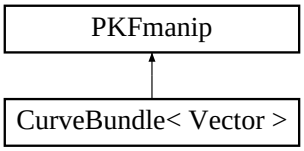
- include/Curve.h
- lib/Curve.cpp

9.7 CurveBundle< Vector > Class Template Reference

The **CurveBundle** (p. 102) class for storing and manipulating biarc curves in $\mathcal{R}^3$.

`#include <include/CurveBundle.h>`

Inheritance diagram for CurveBundle< Vector >::



Public Member Functions

- int **curves** () const
- int **nodes** () const
- void **link** ()
- void **unlink** ()
- void **newCurve** (**Curve**< Vector > &c)
- void **newCurve** (istream &in)
- void **newCurve** (**Curve**< Vector > *c)
- **Curve**< Vector > & **operator[ ]** (int c)
- **CurveBundle** ()
- **CurveBundle** (const char *filename)
- **CurveBundle** (const **CurveBundle** &cb)
- **CurveBundle** & **operator**= (const **CurveBundle** &cb)
- ~**CurveBundle** ()
- void **make** (float f)
- void **makeMidpointRule** ()
- void **resample** (int NewNoNodes)
- void **changeDirection** ()
- void **normalize** ()
- float **length** ()
- float **thickness** ()
- float **thickness_fast** ()
- **CurveBundle** & **operator**+= (const Vector &vec)
- **CurveBundle** & **operator-**= (const Vector &vec)
- void **center** ()
- Vector **getCenter** ()
- **CurveBundle** & **scale** (float s)
- **CurveBundle** & **rotate** (**Matrix3** &m)
- int **readPKF** (const char *infile)
- int **readPKF** (istream &in)
- int **writePKF** (const char *outfile)
- int **writePKF** (ostream &out)
- int **readXYZ** (const char *infile)
- int **readData** (const char *infile, const char *delimiter=" ")
- int **writeData** (const char *outfile, const char *delimiter=" ", int tangents_flag=0)

- int **readVECT** (const char *infile)
- int **writeVECT** (const char *outfile)
- void **computeTangents** ()
- void **polygonalToArcs** ()
- void **arcsToPolygonal** ()

9.7.1 Detailed Description

template<class Vector> class CurveBundle< Vector >

The **CurveBundle** (p. 102) class for storing and manipulating biarc curves in $\mathcal{R}^3$.

This class is used to store and manipulate a set of curves. This data can be interpolated to a biarc curve. The class is for open and closed curves, but this must be specified (how to do that is explained later in this text).

```
#include "../include/CurveBundle.h"

int main() {
  int N = 100;

  CurveBundle<Vector3> borromean;
  Curve<Vector3> c[3];

  ... construct curves (see objects/borromean.cpp) ...

  for (int i=0;i<3;i++)
    borromean.newCurve(c[i]);

  return 0;
}
```

See also:

Curve (p. 76),**Biarc** (p. 61)

9.7.2 Constructor & Destructor Documentation

9.7.2.1 template<class Vector> CurveBundle< Vector >::CurveBundle () [inline]

Constructs an empty curve and sets the header to "No name","","",""

9.7.2.2 template<class Vector> CurveBundle< Vector >::CurveBundle (const char * *filename*) [inline]

Constructs a **CurveBundle** (p. 102) object from a portable knot format data file (PKF file).

See also:

readPKF() (p. 108).

References CurveBundle< Vector >::readPKF().

9.7.2.3 template<class Vector> CurveBundle< Vector >::CurveBundle (const CurveBundle< Vector > & *cb*) [inline]

Copy constructor.

See also:

operator=

9.7.2.4 template<class Vector> CurveBundle< Vector >::~CurveBundle () [inline]

Destructor.

9.7.3 Member Function Documentation

9.7.3.1 template<class Vector> int CurveBundle< Vector >::curves () const [inline]

Returns the number of curves currently stored in the bundle.

Referenced by CurveBundle< Vector >::arcsToPolygonal(), CurveBundle< Vector >::changeDirection(), CurveBundle< Vector >::computeTangents(), CurveBundle< Vector >::getCenter(), CurveBundle< Vector >::length(), CurveBundle< Vector >::link(), CurveInterface::load(), CurveBundle< Vector >::make(), CurveBundle< Vector >::makeMidpointRule(), CurveBundle< Vector >::nodes(), CurveBundle< Vector >::normalize(), CurveBundle< Vector >::operator+=(), CurveBundle< Vector >::operator-=(), CurveBundle< Vector >::operator=(), CurveBundle< Vector >::polygonalToArcs(), CurveBundle< Vector >::resample(), CurveBundle< Vector >::rotate(), CurveBundle< Vector >::scale(), CurveBundle< Vector >::thickness_fast(), TubeBundle< Vector >::TubeBundle(), CurveBundle< Vector >::unlink(), CurveBundle< Vector >::writeData(), CurveBundle< Vector >::writePKF(), and CurveBundle< Vector >::writeVECT().

9.7.3.2 template<class Vector> int CurveBundle< Vector >::nodes () const [inline]

Returns the total number of nodes of the Bundle (i.e. the sum of the number of nodes of each curve)

References CurveBundle< Vector >::curves().

Referenced by CurveBundle< Vector >::writeVECT().

9.7.3.3 template<class Vector> void CurveBundle< Vector >::link () [inline]

This is the way to close all the curves at once.

See also:

unlink() (p. 105), **Curve::link()** (p. 85), **Curve::unlink()** (p. 85)

References CurveBundle< Vector >::curves().

9.7.3.4 template<class Vector> void CurveBundle< Vector >::unlink () [inline]

Opens all curves in bundle.

See also:

link() (p. 104), **Curve::link()** (p. 85), **Curve::unlink()** (p. 85)

References CurveBundle< Vector >::curves().

9.7.3.5 template<class Vector> void CurveBundle< Vector >::newCurve (Curve< Vector > & *c*) [inline]

Add a new curve *c* to the bundle.

Referenced by CurveBundle< Vector >::newCurve(), CurveBundle< Vector >::readData(), CurveBundle< Vector >::readPKF(), CurveBundle< Vector >::readVECT(), and CurveBundle< Vector >::readXYZ().

9.7.3.6 template<class Vector> void CurveBundle< Vector >::newCurve (istream & *in*) [inline]

Read a new curve from the stream *in* and add it to the bundle.

References CurveBundle< Vector >::newCurve().

9.7.3.7 template<class Vector> void CurveBundle< Vector >::newCurve (Curve< Vector > * *c*) [inline]

Takes an already valid curve object pointer and adds this as a component to the current Bundle

References CurveBundle< Vector >::newCurve().

9.7.3.8]

template<class Vector> **Curve**< Vector > & **CurveBundle**< Vector >::operator[] (int *c*) [inline]

Return a reference to the curve number *c*.

9.7.3.9 template<class Vector> CurveBundle< Vector > & CurveBundle< Vector >::operator= (const CurveBundle< Vector > & *cb*) [inline]

Assign operator. Copies the header structure from *c*. Copies all the curves in this bundle (i.e. all the point/tangent data and closes the curve if necessary).

References CurveBundle< Vector >::bundle, and CurveBundle< Vector >::curves().

9.7.3.10 template<class Vector> void CurveBundle< Vector >::make (float *f*) [inline]

Interpolate the curves in the bundle with biarcs. This function uses the same Γ value for all curves!

See also:

Curve::make() (p. 86), **makeMidpointRule()** (p. 106)

References CurveBundle< Vector >::curves().

9.7.3.11 template<class Vector> void CurveBundle< Vector >::makeMidpointRule () [inline]

Interpolate all the curves in the bundle with the midpoint matching rule.

See also:

Curve::make() (p. 86), **make()** (p. 105)

References CurveBundle< Vector >::curves().

9.7.3.12 template<class Vector> void CurveBundle< Vector >::resample (int *NewNoNodes*) [inline]

This function resamples all curves in the bundle with *NewNoNodes* nodes.

resample() (p. 106) can only be done if we have an interpolated set of curves!

See also:

refine(),**make()** (p. 105).

References CurveBundle< Vector >::curves().

9.7.3.13 template<class Vector> void CurveBundle< Vector >::changeDirection () [inline]

Change the orientation of all the curves in the bundle. This means flipping the tangents and reordering the points.

References CurveBundle< Vector >::curves().

9.7.3.14 template<class Vector> void CurveBundle< Vector >::normalize () [inline]

Normalize the length of the sum of curve lengths! This means that for 10 curves with length 1 in the bundle, the length will be 1/10 after normalisation!

An interpolated curve is necessary to compute the length of it.

See also:

scale() (p. 108), **Curve::normalize()** (p. 97), **Curve::scale()** (p. 97)

References CurveBundle< Vector >::curves(), CurveBundle< Vector >::length(), and CurveBundle< Vector >::scale().

9.7.3.15 template<class Vector> float CurveBundle< Vector >::length () [inline]

Returns the total length of the bundle. This is the sum of the arc-length of all the stored curves.

References CurveBundle< Vector >::curves().

Referenced by CurveBundle< Vector >::normalize().

9.7.3.16 template<class Vector> float CurveBundle< Vector >::thickness () [inline]

Returns the thickness (biggest possible tube radius without self-intersection) of the curve bundle

9.7.3.17 template<class Vector> float CurveBundle< Vector >::thickness_fast () [inline]

Returns the "fast" thickness (pt radii only) of the curve bundle.

References CurveBundle< Vector >::curves().

9.7.3.18 template<class Vector> CurveBundle< Vector > & CurveBundle< Vector >::operator+= (const Vector & *v*) [inline]

Translates all the curves by *v*.

Redo the interpolation after this operation if the initial curve was biarc interpolated, since the matching points and bezier points are no longer correct.

See also:

center() (p. 107), **operator-=()** (p. 107)

References CurveBundle< Vector >::curves().

9.7.3.19 template<class Vector> CurveBundle< Vector > & CurveBundle< Vector >::operator-= (const Vector & *v*) [inline]

Translate all the curves by -*v*.

Redo the interpolation after this operation if the initial curve was biarc interpolated, since the matching points and bezier points are no longer correct.

See also:

operator+=() (p. 107)

References CurveBundle< Vector >::curves().

9.7.3.20 template<class Vector> void CurveBundle< Vector >::center () [inline]

This function shifts the center of mass of the bundle to <0,0,0>. This is different from : centering each particular the curve in the bundle to <0,0,0>!!!

See also:

getCenter() (p. 108)

References CurveBundle< Vector >::getCenter().

9.7.3.21 template<class Vector> Vector CurveBundle< Vector >::getCenter () [inline]

Returns the bundles's center of mass.

See also:

center() (p. 107)

References CurveBundle< Vector >::curves().

Referenced by CurveBundle< Vector >::center().

9.7.3.22 template<class Vector> CurveBundle< Vector > & CurveBundle< Vector >::scale (float *s*) [inline]

Scales the length of the curves by s. The curves need not to be interpolated, since only the data points are changed. Returns a reference to itself.

See also:

normalize() (p. 106)

References CurveBundle< Vector >::curves().

Referenced by CurveBundle< Vector >::normalize().

9.7.3.23 template<class Vector> CurveBundle< Vector > & CurveBundle< Vector >::rotate (Matrix3 & *m*) [inline]

Applies the rotation specified by a rotation matrix m to each curve in the bundle. No check is done for m, the user must know what matrix he wants to apply.

This is not the standart 4x4 transformation matrix approach known from homogeneous coordinates stuff.

References CurveBundle< Vector >::curves().

9.7.3.24 template<class Vector> int CurveBundle< Vector >::readPKF (const char * *infile*) [inline]

Read data from a PKF file *infile*. More details are in the class **Curve** (p. 76) documentation. This class can store more than 1 curve!

Returns 1 if all went well, zero otherwise.

See also:

writePKF() (p. 109),**Curve::readPKF()** (p. 97),**Curve::writePKF()** (p. 98)

Referenced by CurveBundle< Vector >::CurveBundle().

9.7.3.25 template<class Vector> int CurveBundle< Vector >::readPKF (istream & *in*) [inline]

Read the curves from a stream *in*. More details about the PKF data format are given in the **Curve** (p. 76) class documentation.

See also:

writePKF() (p. 109),**Curve::readPKF()** (p. 97)

References CurveBundle< Vector >::newCurve(), and PKFmanip::readHeader().

9.7.3.26 template<class Vector> int CurveBundle< Vector >::writePKF (const char * *outfile*) [inline]

Writes the current curves to a PKF file *outfile*. This goes through all components and writes them to the file.

Returns 1 if all went well, zero otherwise.

See also:

readPKF() (p. 108)

Referenced by CurveBundle< Vector >::writePKF().

9.7.3.27 template<class Vector> int CurveBundle< Vector >::writePKF (ostream & *out*) [inline]

Writes the current **CurveBundle** (p. 102) instance to the stream *out*. (I.e. header, number of curves and the curve data).

See also:

readPKF() (p. 108)

References CurveBundle< Vector >::curves(), PKFmanip::writeHeader(), and CurveBundle< Vector >::writePKF().

9.7.3.28 template<class Vector> int CurveBundle< Vector >::readXYZ (const char * *infile*) [inline]

Read data from a XYZ file *infile*. More details are in the class **Curve** (p. 76) documentation. This class can store more than 1 curve!

Returns 1 if all went well, zero otherwise.

See also:

writePKF() (p. 109),**Curve::readPKF()** (p. 97),**Curve::writePKF()** (p. 98)

References CurveBundle< Vector >::newCurve(), and Curve< Vector >::readXYZ().

9.7.3.29 template<class Vector> int CurveBundle< Vector >::readData (const char * *infile*, const char * *delimiter* = " ") [inline]

This function reads the curve data from a file *infile*. The file structure is a list of x,y,z coordinates. The default for the delimiter is " ", but can be changed with the second argument of the function.

The first line gives the number of nodes in the following format {**nodes** (p. 104)}. Then the coordinates are read in. The *delimiter* argument is any string that separates the coordinate values from each other, the default value is a space ' ' delimiter.

Returns 1 if all went well, zero otherwise.

See also:

computeTangents() (p. 111),**polygonalToArcs()** (p. 111),**arcsToPolygonal()** (p. 111)

TODO : not ready, change DOC!!!

References CurveBundle< Vector >::newCurve(), and Curve< Vector >::readData().

9.7.3.30 template<class Vector> int CurveBundle< Vector >::writeData (const char * *outfile*, const char * *delimiter* = " ", int *tangents_flag* = 0) [inline]

Writes the curve to a data file. First line is the number of points and then follows a list of x,y,z coordinates. If the *tangents_flag* is set to 1, the tangents of the points are also dropped (Default is 0).

See also:

readData() (p. 110) TODO : not ready, change doc!!! TODO : not ready!!!

References CurveBundle< Vector >::curves().

9.7.3.31 template<class Vector> int CurveBundle< Vector >::readVECT (const char * *infile*) [inline]

Read a file in VECT format. Infos about the format at

`http://www.geomview.org/docs/html/geomview_42.html`

Returns 1 if all went well, zero otherwise. Does not support the whole VECT spec.

See also:

computeTangents() (p. 111),**polygonalToArcs()** (p. 111),**arcsToPolygonal()** (p. 111),**writeVECT()** (p. 110)

References Curve< Vector >::append(), Curve< Vector >::computeTangents(), Curve< Vector >::flush_all(), Curve< Vector >::link(), and CurveBundle< Vector >::newCurve().

Referenced by CurveInterface::load().

9.7.3.32 template<class Vector> int CurveBundle< Vector >::writeVECT (const char * *outfile*) [inline]

Not implemented

References CurveBundle< Vector >::curves(), and CurveBundle< Vector >::nodes().

9.7.3.33 template<class Vector> void CurveBundle< Vector >::computeTangents () [inline]

Comptes the tangents on each curve in the bundle.

See also:

Curve::computeTangents() (p. 93)

References CurveBundle< Vector >::curves().

Referenced by CurveInterface::load().

9.7.3.34 template<class Vector> void CurveBundle< Vector >::polygonalToArcs () [inline]

This function converts all polygonal curves into curves made of arcs of circles.

See also:

arcsTolPolygonal(), **Curve::polygonalToArcs()** (p. 93)

References CurveBundle< Vector >::curves().

9.7.3.35 template<class Vector> void CurveBundle< Vector >::arcsToPolygonal () [inline]

This function converts all the biarc curves into polygonal curves.

See also:

polygonalToArcs() (p. 111),**Curve::arcsToPolygonal** (p. 93)

References CurveBundle< Vector >::curves().

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

- include/CurveBundle.h
- lib/CurveBundle.cpp

9.8 CurveInfo Struct Reference

Curve (p. 76) information.

`#include <utils.h>`

Public Attributes

- **TubeBundle**< **Vector3** > * **Knot**
- int **N**
- int **S**
- float **R**
- float **Tol**
- int **Closed**
- QStringList **filenames**
- uint **TEXTURES**
- QString **texture_file**

9.8.1 Detailed Description

Curve (p. 76) information.

Knot class interface for viewer, containing info shared among viewer, interaction callbacks, gui routines.

The documentation for this struct was generated from the following file:

- inventor/utils.h

9.9 CurveInterface Class Reference

Interface between curve data and viewer.

`#include <utils.h>`

Public Slots

- int **save** ()
- int **exportIV** ()
- int **exportRIB** ()
- int **exportPOV** ()
- void **decreaseTransparency** ()
- void **increaseTransparency** ()
- void **increaseRadius** ()
- void **decreaseRadius** ()
- void **increaseSegments** ()
- void **decreaseSegments** ()
- void **setNumberOfNodes** (int N)
- SoSeparator * **frame** (int FRAME=0)

Public Member Functions

- void **clear** ()
- void **makeMesh** ()
- SoSeparator * **load** ()
- void **dumpInfo** ()
- SoSeparator * **curveSeparator** ()

Public Attributes

- struct **CurveInfo info**
- SoSeparator ** **knot_node**
- **SoKnot** ** **knot_shape**
- SoMaterialBinding ** **material_bindings**
- SoMaterial ** **materials**
- SoTexture2 * **knot_texture**
- SoSeparator * **graph_node**
- SoSeparator * **frame_node**

9.9.1 Detailed Description

Interface between curve data and viewer.

9.9.2 Member Function Documentation

9.9.2.1 void CurveInterface::makeMesh () [inline]

Construct the meshes for all pkf curves currently loaded. Close them if requested.

References CurveInfo::Closed, CurveInfo::Knot, TubeBundle< Vector >::makeMesh(), CurveInfo::N, CurveInfo::R, CurveInfo::S, CurveInfo::Tol, and TubeBundle< Vector >::tubes().

Referenced by load().

9.9.2.2 SoSeparator* CurveInterface::load () [inline]

Load a set of pkf curves. We expect the filenames list in info to contain the files to be loaded.

References CurveBundle< Vector >::computeTangents(), CurveBundle< Vector >::curves(), CurveInfo::filenames, CurveInfo::Knot, makeMesh(), CurveInfo::N, TubeBundle< Vector >::newTube(), TubeBundle< Vector >::readPKF(), CurveBundle< Vector >::readVECT(), and TubeBundle< Vector >::readXYZ().

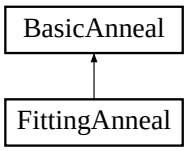
The documentation for this class was generated from the following file:

- inventor/utils.h

9.10 FittingAnneal Class Reference

Fit a curve through given points.

Inheritance diagram for FittingAnneal::



Public Member Functions

- float **interpolate** (float s)
- void **dumpVectors** ()
- ostream & **show_config** (ostream &out)
- void **log** ()
- **FittingAnneal** (const char *filename, const char *params="")
- bool **stop** ()
- void **best_found** ()
- float **energy** ()

Public Attributes

- string **resume_from**
- int **resample**
- int **no_of_nodes**
- vector< **Vec2** > **orig**
- vector< **Vec2** > **nodes**
- vector< **Vec2** > **best_nodes**

9.10.1 Detailed Description

Fit a curve through given points.

Here we try to fit a monotonic function through a given set of points with periodic boundary conditions at both sides (x and y).

9.10.2 Constructor & Destructor Documentation

9.10.2.1 FittingAnneal::FittingAnneal (const char * *filename*, const char * *params* = "") [inline]

params - like **BasicAnneal** (p. 55). Addionally filename - curve to be fitted no_of_nodes - Number of points resume_from - File containing a previous run. resample - If we resume from a file. Resample it with that num of points.

References BasicAnneal::std_init().

9.10.3 Member Function Documentation

9.10.3.1 ostream& FittingAnneal::show_config (ostream & *out*) [inline, virtual]

Show the configuration parameters.

Reimplemented from **BasicAnneal** (p. 57).

References BasicAnneal::show_config().

9.10.3.2 void FittingAnneal::log () [inline, virtual]

Perfomr a logging action (uses logline) but can be reimplemented in derived classes.

Reimplemented from **BasicAnneal** (p. 58).

References BasicAnneal::logline().

9.10.3.3 bool FittingAnneal::stop () [inline, virtual]

Returns true when annealing considers to have converged.

Reimplemented from **BasicAnneal** (p. 57).

9.10.3.4 void FittingAnneal::best_found () [inline, virtual]

This gets called each time anneal finds a new best energy.

Reimplemented from **BasicAnneal** (p. 57).

References BasicAnneal::best_found(), and BasicAnneal::update_minmax_step().

9.10.3.5 float FittingAnneal::energy () [inline, virtual]

The energy is $\sum_{i=1}^{N}(y_i - inter(f(x_i)))^2$, where y_i are the sample points, inter is a linear interpolation of the fitted points we currently have.

Reimplemented from **BasicAnneal** (p. 57).

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

- experimental/annealing/fit.cpp

9.11 MainWindow Class Reference

Main viewer class.

`#include <inventor/mainwindow.h>`

Public Slots

- void **setFraming** (int FRAME)
- void **plotWindow** (int w)

Signals

- void **changed** ()

Public Member Functions

- **MainWindow** (QWidget *parent=NULL, const char *name=NULL, SbBool embed=TRUE, SoQtFullViewer::BuildFlag flag=BUILD_ALL, SoQtViewer::Type=BROWSER)
- void **swap_view** ()
- void **emitChanged** ()

Public Attributes

- VIEW_MODE **view_mode**
- **ViewerInfo** * **vi**
- **CurveInterface** * **ci**
- void(* **gradient**)(RGB *, float, float, float)
- bool **PRESSED**
- vector< **Biarc**< **Vector3** > >::iterator **picked_biarc**
- SbPlaneProjector **spp**
- SbVec3f **UpVector**
- SbVec3f **LeftVector**
- SbVec3f **delta**
- float **AspectratioX**
- float **AspectratioY**
- int **EditTangent**
- QFileDialog * **fileDialog**
- PPPlotWindow * **pp_win**
- PTPlotWindow * **pt_win**
- TTPlotWindow * **tt_win**
- SoSeparator * **root**
- SoSeparator * **circles**
- SoSeparator * **interaction**
- SoSwitch * **scene**

Protected Member Functions

- void **closeEvent** (QCloseEvent *event)

9.11.1 Detailed Description

Main viewer class.

The main viewer contains information about the curve objects and the current scene. It inherits SoQtExaminerViewer and add custom menu and toolbars. Various plots can be generated and inventor scene files $*$.iv can be loaded and included into the scene as well.

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

- inventor/mainwindow.h
- inventor/mainwindow.cpp

9.12 Matrix3 Class Reference

The **Matrix3** (p. 119) class is a 3x3 dimensional Matrix class with floating point entries.

`#include <include/Matrix3.h>`

Public Member Functions

- **Matrix3** ()
- **Matrix3** (const **Vector3** &v0, const **Vector3** &v1, const **Vector3** &v2)
- **Matrix3** (const float &x00, const float &x01, const float &x02, const float &x10, const float &x11, const float &x12, const float &x20, const float &x21, const float &x22)
- **Matrix3** & **zero** ()
- **Vector3** & **operator[]** (int n)
- const **Vector3** & **operator[]** (int n) const
- void **setOne** (const int c, const **Vector3** &v)
- void **setOne** (const int c, const float v1, const float v2, const float v3)
- void **setAll** (const **Vector3** &v1, const **Vector3** &v2, const **Vector3** &v3)
- void **setAll** (const **Vector3** *v)
- **Vector3** **getOne** (const int c)
- void **getAll** (**Vector3** &v1, **Vector3** &v2, **Vector3** &v3)
- **Matrix3** & **id** ()
- **Matrix3** & **transpose** ()
- float **det** ()
- **Matrix3** & **inv** ()
- **Matrix3** & **outer** (const **Vector3** &a, const **Vector3** &b)
- **Matrix3** & **vecCross** (const **Vector3** &v)
- **Matrix3** & **cay** (const **Vector3** &v)
- **Matrix3** & **rotAround** (const **Vector3** &v, float angle)
- **Matrix3** **operator*** (const **Matrix3** &m)
- **Vector3** **operator*** (const **Vector3** &v)
- **Matrix3** **operator+** (const **Matrix3** &m) const
- **Matrix3** **operator-** (const **Matrix3** &m) const
- **Matrix3** **operator-** () const
- **Matrix3** & **operator=** (const **Matrix3** &m)
- **Matrix3** & **operator+=** (const **Matrix3** &m)
- **Matrix3** & **operator-=** (const **Matrix3** &m)
- **Matrix3** & **operator*=** (const float s)
- **Matrix3** & **operator/=** (const float s)
- int **operator==** (const **Matrix3** &m) const
- int **operator!=** (const **Matrix3** &m) const
- void **print** (ostream &out) const

Friends

- **Matrix3** **operator*** (const **Matrix3** &m, float d)
- **Matrix3** **operator*** (float d, const **Matrix3** &m)
- **Matrix3** **operator/** (const **Matrix3** &m, float d)
- ostream & **operator<<** (ostream &out, const **Matrix3** &m)

9.12.1 Detailed Description

The **Matrix3** (p. 119) class is a 3x3 dimensional Matrix class with floating point entries.

This class provides storage for a 3x3 floating point table, representing a matrix. It is possible to compute determinant, inverse, transpose and other standart matrix operations.

A matrix is given by 3 **Vector3** (p. 158) column vectors v_0, v_1, v_2 and has the form $M = \begin{pmatrix} v_{00} & v_{10} & v_{20} \\ v_{01} & v_{11} & v_{21} \\ v_{02} & v_{12} & v_{22} \end{pmatrix}$

Caution : When accessing a matrix element by M[i][j], i is the column index, and j the row index! In the linear algebra literature most of the time this convention is the inverse. The structure is chosen like this such that a change of basis from a coordinate system A to a coordinate system B is given by the matrix M, composed by the basis column vectors of the final coordinate system.

9.12.2 Constructor & Destructor Documentation

9.12.2.1 Matrix3::Matrix3 ()

The default constructor initializes all matrix elements to zero.

References Vector3::zero().

Referenced by operator+(), and operator-().

9.12.2.2 Matrix3::Matrix3 (const Vector3 & *v0*, const Vector3 & *v1*, const Vector3 & *v2*)

Constructs a **Matrix3** (p. 119) given three row vectors *v0*, *v1* and *v2*.

9.12.2.3 Matrix3::Matrix3 (const float & *x00*, const float & *x01*, const float & *x02*, const float & *x10*, const float & *x11*, const float & *x12*, const float & *x20*, const float & *x21*, const float & *x22*)

Constructs a **Matrix3** (p. 119) from 9 floats. Given in row-first order.

9.12.3 Member Function Documentation

9.12.3.1 Matrix3 & Matrix3::zero ()

Set all matrix elements to zero. Returns an instance to itself.

References Vector3::zero().

Referenced by Curve< Vector >::inertiaTensor().

9.12.3.2]

Vector3 & Matrix3::operator[] (int *n*) `[inline]`

Index operator. Returns one of the three matrix columns as a **Vector3** (p. 158) instance.

See also:

getOne() (p. 121), **getAll()** (p. 121), **setOne()** (p. 121), **setAll()** (p. 121).

9.12.3.3]

const **Vector3** & Matrix3::operator[] (int *n*) const [inline]

Index operator. Returns one of the three matrix columns as a **Vector3** (p. 158) instance.

See also:

getOne() (p. 121), **getAll()** (p. 121), **setOne()** (p. 121), **setAll()** (p. 121).

9.12.3.4 void Matrix3::setOne (const int *c*, const Vector3 & *v*)

Fill matrix column *c* with the components of the vector *v*.

9.12.3.5 void Matrix3::setOne (const int *c*, const float *v1*, const float *v2*, const float *v3*)

Fill matrix column *c* from top to bottom with the values *v1* to *v3*.

9.12.3.6 void Matrix3::setAll (const Vector3 & *v1*, const Vector3 & *v2*, const Vector3 & *v3*)

Reset the matrix by columns *v1* to *v3*.

Referenced by id().

9.12.3.7 void Matrix3::setAll (const Vector3 * *v*)

Fill the matrix with the array of **Vector3** (p. 158) *v*.

9.12.3.8 Vector3 Matrix3::getOne (const int *c*)

Returns matrix column *c* as a **Vector3** (p. 158) instance.

9.12.3.9 void Matrix3::getAll (Vector3 & *v1*, Vector3 & *v2*, Vector3 & *v3*)

Put the matrix columns into the vectors *v1* to *v3*.

9.12.3.10 Matrix3 & Matrix3::id ()

Set the current matrix to the identity and return a reference to itself.

References setAll().

Referenced by cay(), and Curve< Vector >::principalAxis().

9.12.3.11 Matrix3 & Matrix3::transpose ()

Set the current matrix to its transpose and return a reference to itself.

Columns become rows and rows become the columns. I.e, the transpose of a matrix $M =$ is

$$M^t = \begin{pmatrix} v_{00} & v_{01} & v_{02} \\ v_{10} & v_{11} & v_{12} \\ v_{20} & v_{21} & v_{22} \end{pmatrix}$$

9.12.3.12 float Matrix3::det ()

Computes and returns the determinant of the **Matrix3** (p. 119).

Referenced by Matrix4::adjoint(), Matrix4::det(), and inv().

9.12.3.13 Matrix3 & Matrix3::inv ()

Sets the current matrix to its inverse and return a reference to itself.

Given the matrix A, its inverse is the matrix B, satisfying the relation

$A \cdot B = id$,

where id is the identity matrix.

If the matrix is hardly invertible, i.e. its determinant is close to zero, then a warning is posted to the error channel and the matrix itself is not changed.

References det().

9.12.3.14 Matrix3 & Matrix3::outer (const Vector3 & *a*, const Vector3 & *b*)

Computes the outer product between a vector a and a vector b and stores the result in the current **Matrix3** (p. 119) object.

Formally we have the product between a column vector a with a row vector b.

$$M = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} \cdot \begin{pmatrix} b_0 & b_1 & b_2 \end{pmatrix}$$

Referenced by Vector3::reflect().

9.12.3.15 Matrix3 & Matrix3::vecCross (const Vector3 & *v*)

Constructs an skew-symmetric matrix from a vector v of the form

$$M = \begin{pmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{pmatrix}$$

Applied to a vector, this matrix acts like a cross product between v and some other vector w. Therefore

$M \cdot w = v \times w$

Referenced by cay().

9.12.3.16 Matrix3 & Matrix3::cay (const Vector3 & *v*)

Construct a Cayleigh rotation matrix, defined by

Cay Rotation Matrix "Cay(v)", where v are the components of the rot axis and |v| is magnitude of rotation

References id(), Vector3::norm2(), and vecCross().

Referenced by Tube< Vector >::makeMesh().

9.12.3.17 Matrix3 & Matrix3::rotAround (const Vector3 & *v*, float *alpha*)

Sets the current matrix to a rotation matrix. Where the rotation is in the positive trigonometric direction about the axis v by an angle alpha.

References Vector3::normalize().

9.12.3.18 Matrix3 Matrix3::operator* (const Matrix3 & *m*)

Multiplies the current matrix on the right hand side by m and returns the result as a new **Matrix3** (p. 119) instance.

9.12.3.19 Vector3 Matrix3::operator* (const Vector3 & *v*)

Applies the current matrix on the left side of a column vector v and returns a vector with the result.

9.12.3.20 Matrix3 Matrix3::operator+ (const Matrix3 & *m*) const

Matrix summation. Returns a **Matrix3** (p. 119) instance.

References Matrix3().

9.12.3.21 Matrix3 Matrix3::operator- (const Matrix3 & *m*) const

Matrix substraction. Returns a **Matrix3** (p. 119) instance.

References Matrix3().

9.12.3.22 Matrix3 Matrix3::operator- () const

Matrix negation. Returns a **Matrix3** (p. 119) instance where all the elements have now switched their sign.

References Matrix3().

9.12.3.23 Matrix3 & Matrix3::operator= (const Matrix3 & *m*)

Assign operator. Makes a copy of m. Returns an instance to itself.

9.12.3.24 Matrix3 & Matrix3::operator+= (const Matrix3 & *m*)

Adds the matrix *m* to the current one. Returns an instance to itself.

9.12.3.25 Matrix3 & Matrix3::operator-= (const Matrix3 & *m*)

Substracts the matrix *m* from the current one. Returns an instance to itself.

9.12.3.26 Matrix3 & Matrix3::operator*= (const float *s*)

Multiplies all the elements of the current matrix by *s* and returns\ an instance to itself.

9.12.3.27 Matrix3 & Matrix3::operator/= (const float *s*)

Divides all the elements of the current matrix by *s* and returns\ an instance to itself.

9.12.3.28 int Matrix3::operator== (const Matrix3 & *m*) const

Compare the current matrix to *m*. If all the elements are the same, the function return 1, 0 otherwise.

9.12.3.29 int Matrix3::operator!= (const Matrix3 & *m*) const

Compare the current matrix to *m*. Returns 1 if they are different. 0 if they are the same.

9.12.3.30 void Matrix3::print (ostream & *out*) const

Prints the components of this matrix in a formated way onto the stream *out*.

9.12.4 Friends And Related Function Documentation

9.12.4.1 Matrix3 operator* (const Matrix3 & *m*, float *d*) [friend]

Multiplies every element of the matrix *m* by a factor of *d*. and returns a **Matrix3** (p. 119) object.

9.12.4.2 Matrix3 operator* (float *d*, const Matrix3 & *m*) [friend]

Multiplies every element of the matrix *m* by a factor of *d*. and returns a **Matrix3** (p. 119) object.

9.12.4.3 Matrix3 operator/ (const Matrix3 & *m*, float *d*) [friend]

Divides every element of the matrix *m* by *d*. and returns a **Matrix3** (p. 119) object.

9.12.4.4 ostream & Matrix3::operator<< (ostream & *out*, const Matrix3 & *m*) [friend]

Overloaded left shift operator. Returns the **Matrix3** (p. 119) *m* as an ostream object that can be written to a file or to standart output.

See also:

print() (p. 124)

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

- include/Matrix3.h
- lib/Matrix3.cpp

9.13 Matrix4 Class Reference

The **Matrix4** (p. 126) class is a 4x4 dimensional Matrix class with floating point entries.

`#include <include/Matrix4.h>`

Public Member Functions

- **Matrix4** ()
- **Matrix4** (const **Vector4** &v0, const **Vector4** &v1, const **Vector4** &v2, const **Vector4** &v3)
- **Matrix4** & **zero** ()
- **Vector4** & **operator[]** (int n)
- const **Vector4** & **operator[]** (int n) const
- void **setOne** (const int c, const **Vector4** &v)
- void **setOne** (const int c, const float v1, const float v2, const float v3, const float v4)
- void **setAll** (const **Vector4** &v1, const **Vector4** &v2, const **Vector4** &v3, const **Vector4** &v4)
- void **setAll** (const **Vector4** *v)
- **Vector4** **getOne** (const int c)
- void **getAll** (**Vector4** &v1, **Vector4** &v2, **Vector4** &v3, **Vector4** &v4)
- **Matrix4** & **id** ()
- **Matrix4** & **transpose** ()
- **Matrix3** **sub** (const int r, const int c)
- float **det** ()
- **Matrix4** & **inv** ()
- **Matrix4** & **adjoint** ()
- **Matrix4** & **outer** (const **Vector4** &a, const **Vector4** &b)
- **Matrix4** **operator*** (const **Matrix4** &m)
- **Vector4** **operator*** (const **Vector4** &v)
- **Matrix4** **operator+** (const **Matrix4** &m) const
- **Matrix4** **operator-** (const **Matrix4** &m) const
- **Matrix4** **operator-** () const
- **Matrix4** & **operator=** (const **Matrix4** &m)
- **Matrix4** & **operator+=** (const **Matrix4** &m)
- **Matrix4** & **operator-=** (const **Matrix4** &m)
- **Matrix4** & **operator*=** (const float s)
- **Matrix4** & **operator/=** (const float s)
- int **operator==** (const **Matrix4** &m) const
- int **operator!=** (const **Matrix4** &m) const
- void **print** (ostream &out) const

Friends

- **Matrix4** **operator*** (const **Matrix4** &m, float d)
- **Matrix4** **operator*** (float d, const **Matrix4** &m)
- **Matrix4** **operator/** (const **Matrix4** &m, float d)
- ostream & **operator<<** (ostream &out, const **Matrix4** &m)

9.13.1 Detailed Description

The **Matrix4** (p. 126) class is a 4x4 dimensional Matrix class with floating point entries.

This class provides storage for a 4x4 floating point table, representing a matrix. It is possible to compute determinant, inverse, transpose and other matrix operations in 4-dimensions.

A matrix is given by 4 **Vector4** (p. 165) column vectors v_0, v_1, v_2, v_3 and has the form $M = \begin{pmatrix} v_{00} & v_{10} & v_{20} & v_{30} \\ v_{01} & v_{11} & v_{21} & v_{31} \\ v_{02} & v_{12} & v_{22} & v_{32} \\ v_{03} & v_{13} & v_{23} & v_{33} \end{pmatrix}$

Caution : When accessing a matrix element by M[i][j], i is the column index, and j the row index! In the linear algebra literature most of the time this convention is the inverse. The structure is chosen like this such that a change of basis from a coordinate system A to a coordinate system B is given by the matrix M, composed by the basis column vectors of the final coordinate system.

See also:

class **Vector4** (p. 165)

9.13.2 Constructor & Destructor Documentation

9.13.2.1 Matrix4::Matrix4 ()

The default constructor initializes all matrix elements to zero.

References Vector4::zero().

Referenced by operator+(), and operator-().

9.13.2.2 Matrix4::Matrix4 (const Vector4 & *v0*, const Vector4 & *v1*, const Vector4 & *v2*, const Vector4 & *v3*)

Constructs a **Matrix4** (p. 126) given three row vectors *v0*, *v1*, *v2* and *v3*.

9.13.3 Member Function Documentation

9.13.3.1 Matrix4 & Matrix4::zero ()

Set all matrix elements to zero. Returns an instance to itself.

References Vector4::zero().

9.13.3.2]

Vector4 & Matrix4::operator[] (int *n*) `[inline]`

Index operator. Returns one of the three matrix columns as a **Vector4** (p. 165) instance.

See also:

getOne() (p. 128), **getAll()** (p. 128), **setOne()** (p. 128), **setAll()** (p. 128).

9.13.3.3]

const **Vector4** & Matrix4::operator[] (int *n*) const `[inline]`

Index operator. Returns one of the three matrix columns as a **Vector4** (p. 165) instance.

See also:

getOne() (p. 128), **getAll()** (p. 128), **setOne()** (p. 128), **setAll()** (p. 128).

9.13.3.4 void Matrix4::setOne (const int *c*, const Vector4 & *v*)

Fill matrix column *c* with the components of the vector *v*.

9.13.3.5 void Matrix4::setOne (const int *c*, const float *v1*, const float *v2*, const float *v3*, const float *v4*)

Fill matrix column *c* from top to bottom with the values *v1* to *v4*.

9.13.3.6 void Matrix4::setAll (const Vector4 & *v1*, const Vector4 & *v2*, const Vector4 & *v3*, const Vector4 & *v4*)

Reset the matrix by columns *v1* to *v4*.

Referenced by id().

9.13.3.7 void Matrix4::setAll (const Vector4 * *v*)

Fill the matrix with the array of **Vector4** (p. 165) *v*.

9.13.3.8 Vector4 Matrix4::getOne (const int *c*)

Returns matrix column *c* as a **Vector4** (p. 165) instance.

9.13.3.9 void Matrix4::getAll (Vector4 & *v1*, Vector4 & *v2*, Vector4 & *v3*, Vector4 & *v4*)

Put the matrix columns into the vectors *v1* to *v4*.

9.13.3.10 Matrix4 & Matrix4::id ()

Set the current matrix to the identity and return a reference to itself.

References setAll().

9.13.3.11 Matrix4 & Matrix4::transpose ()

Set the current matrix to its transpose and return a reference to itself.

Columns become rows and rows become the columns. I.e, the transpose of a matrix $M =$ is

$$M^t = \begin{pmatrix} v_{00} & v_{01} & v_{02} & v_{03} \\ v_{10} & v_{11} & v_{12} & v_{13} \\ v_{20} & v_{21} & v_{22} & v_{23} \\ v_{30} & v_{31} & v_{32} & v_{33} \end{pmatrix}$$

9.13.3.12 Matrix3 Matrix4::sub (const int r, const int c)

Returns a **Matrix3** (p. 119) object that is the current **Matrix4** (p. 126) without line r and column c. Let's take as an example a matrix M. A call to M.sub(2,1), where the first argument is the line 3 and the second one the column 2 (since we start counting at 0), yields the following **Matrix3** (p. 119) :

$$M^{4x4} = \begin{pmatrix} v_{00} & v_{10} & v_{20} & v_{30} \\ v_{01} & v_{11} & v_{21} & v_{31} \\ v_{02} & v_{12} & v_{22} & v_{32} \\ v_{03} & v_{13} & v_{23} & v_{33} \end{pmatrix} \quad \Rightarrow \quad M^{3x3} = \begin{pmatrix} v_{00} & v_{20} & v_{30} \\ v_{01} & v_{21} & v_{31} \\ v_{03} & v_{23} & v_{33} \end{pmatrix}$$

Referenced by adjoint(), and det().

9.13.3.13 float Matrix4::det ()

Computes and returns the determinant of the **Matrix4** (p. 126).

See also:

sub() (p. 129),**inv()** (p. 129),**adjoint()** (p. 129)

References Matrix3::det(), and sub().

Referenced by inv().

9.13.3.14 Matrix4 & Matrix4::inv ()

Sets the current matrix to its inverse and return a reference to itself.

Given the matrix A, its inverse is the matrix B, satisfying the relation

$A \cdot B = id$,

where id is the identity matrix.

If the matrix is hardly invertible, i.e. its determinant is close to zero, then a warning is posted to the error channel and the matrix itself is not changed.

See also:

adjoint() (p. 129),**sub()** (p. 129),**det()** (p. 129),**Matrix3::inv()** (p. 122)

References adjoint(), and det().

9.13.3.15 Matrix4 & Matrix4::adjoint ()

Computes the adjoint matrix A. Replaces the currently stored matrix by A and returns a reference to it.

The transposition of the cofactor matrix is the adjoint matrix. A cofactor is given by :

$cof_{ij} = (-1)^{i+j} det\, M_{ij}$,

where M_{ij} is the matrix obtained by erasing line i and column j.

See also:

inv() (p. 129), **det()** (p. 129), **sub()** (p. 129)

References Matrix3::det(), and sub().

Referenced by inv().

9.13.3.16 Matrix4 & Matrix4::outer (const Vector4 & *a*, const Vector4 & *b*)

Computes the outer product between a vector a and a vector b and stores the result in the current **Matrix4** (p. 126) object.

Formally we have the product between a column vector a with a row vector b.

$$M = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \cdot \begin{pmatrix} b_0 & b_1 & b_2 & b_3 \end{pmatrix}$$

Referenced by Vector4::reflect().

9.13.3.17 Matrix4 Matrix4::operator∗ (const Matrix4 & *m*)

Multiplies the current matrix on the right hand side by m and returns the result as a new **Matrix4** (p. 126) instance.

9.13.3.18 Vector4 Matrix4::operator∗ (const Vector4 & *v*)

Applies the current matrix on the left side of a column vector v and returns a vector with the result.

9.13.3.19 Matrix4 Matrix4::operator+ (const Matrix4 & *m*) const

Matrix summation. Returns a **Matrix4** (p. 126) instance.

References Matrix4().

9.13.3.20 Matrix4 Matrix4::operator- (const Matrix4 & *m*) const

Matrix substraction. Returns a **Matrix4** (p. 126) instance.

References Matrix4().

9.13.3.21 Matrix4 Matrix4::operator- () const

Matrix negation. Returns a **Matrix4** (p. 126) instance where all the elements have now switched their sign.

References Matrix4().

9.13.3.22 Matrix4 & Matrix4::operator= (const Matrix4 & *m*)

Assign operator. Makes a copy of *m*. Returns an instance to itself.

9.13.3.23 Matrix4 & Matrix4::operator+= (const Matrix4 & *m*)

Adds the matrix *m* to the current one. Returns an instance to itself.

9.13.3.24 Matrix4 & Matrix4::operator-= (const Matrix4 & *m*)

Substracts the matrix *m* from the current one. Returns an instance to itself.

9.13.3.25 Matrix4 & Matrix4::operator*= (const float *s*)

Multiplies all the elements of the current matrix by *s* and returns\ an instance to itself.

9.13.3.26 Matrix4 & Matrix4::operator/= (const float *s*)

Divides all the elements of the current matrix by *s* and returns\ an instance to itself.

9.13.3.27 int Matrix4::operator== (const Matrix4 & *m*) const

Compare the current matrix to *m*. If all the elements are the same, the function return 1, 0 otherwise.

9.13.3.28 int Matrix4::operator!= (const Matrix4 & *m*) const

Compare the current matrix to *m*. Returns 1 if they are different. 0 if they are the same.

9.13.3.29 void Matrix4::print (ostream & *out*) const

Prints the components of this matrix in a formated way onto the stream *out*.

9.13.4 Friends And Related Function Documentation

9.13.4.1 Matrix4 operator* (const Matrix4 & *m*, float *d*) [friend]

Multiplies every element of the matrix *m* by a factor of *d*. and returns a **Matrix4** (p. 126) object.

9.13.4.2 Matrix4 operator* (float *d*, const Matrix4 & *m*) [friend]

Multiplies every element of the matrix *m* by a factor of *d*. and returns a **Matrix4** (p. 126) object.

9.13.4.3 Matrix4 operator/ (const Matrix4 & *m*, float *d*) [friend]

Divides every element of the matrix *m* by *d*. and returns a **Matrix4** (p. 126) object.

9.13.4.4 ostream & Matrix4::operator<< (ostream & *out*, const Matrix4 & *m*) [friend]

Overloaded left shift operator. Returns the **Matrix4** (p. 126) *m* as an ostream object that can be written to a file or to standart output.

See also:

print() (p. 131)

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

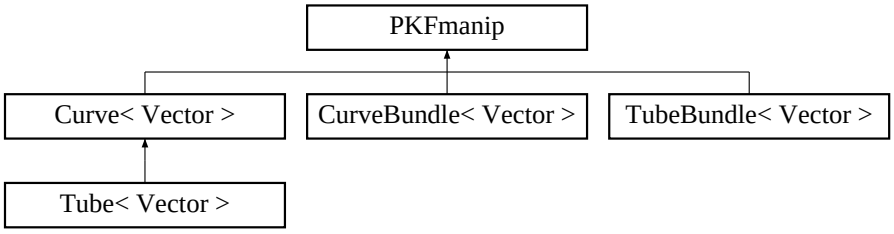
- include/Matrix4.h
- lib/Matrix4.cpp

9.14 PKFmanip Class Reference

The **PKFmanip** (p. 133) class for storing and manipulating biarc curves.

`#include <include/PKFmanip.h>`

Inheritance diagram for PKFmanip::



Public Member Functions

- **PKFmanip** ()
- **PKFmanip** (const **PKFmanip** &h)
- **PKFmanip** & **operator**= (const **PKFmanip** &h)
- ~**PKFmanip** ()
- void **header** (const char *name="No name", const char *etic="", const char *cite="", const char *history="")
- void **setName** (const char *name)
- void **setEtic** (const char *etic)
- void **setCite** (const char *cite)
- void **setHistory** (const char *history)
- const char * **getName** () const
- const char * **getEtic** () const
- const char * **getCite** () const
- const char * **getHistory** () const
- int **readHeader** (istream &in)
- int **writeHeader** (ostream &out)

Friends

- ostream & **operator**<< (ostream &out, **PKFmanip** &c)

9.14.1 Detailed Description

The **PKFmanip** (p. 133) class for storing and manipulating biarc curves.

9.14.2 Constructor & Destructor Documentation

9.14.2.1 PKFmanip::PKFmanip ()

Constructs an empty curve and sets the header to "No name","","",""

References header().

9.14.2.2 PKFmanip::PKFmanip (const PKFmanip & *h*)

Copy constructor.

See also:

operator=

9.14.2.3 PKFmanip::~PKFmanip ()

Delete the header string and destroy the **PKFmanip** (p. 133) instance.

9.14.3 Member Function Documentation

9.14.3.1 PKFmanip & PKFmanip::operator= (const PKFmanip & *h*)

Assign operator. Copies the header strings from *h*.

References getCite(), getEtic(), getHistory(), getName(), and header().

9.14.3.2 void PKFmanip::header (const char * *name* = "No name", const char * *etic* = "", const char * *cite* = "", const char * *history* = "")

Change the current header. *name* is the name of the curve. *etic* is citation of people at the origin of this particular curve. *cite* is the people to cite if this curve is used further. *history*, comments or other infos about the curve.

Default values : "No name","","",""

See also:

setName() (p. 134),**setEtic()** (p. 134),**setCite()** (p. 135),**setHistory()** (p. 135)

References setCite(), setEtic(), setHistory(), and setName().

Referenced by Curve< Vector >::operator+(), Curve< Vector >::operator-(), operator=(), and PKFmanip().

9.14.3.3 void PKFmanip::setName (const char * *name*)

Set the name of the curve.

See also:

setEtic() (p. 134),**setCite()** (p. 135),**setHistory()** (p. 135)

Referenced by header().

9.14.3.4 void PKFmanip::setEtic (const char * *etic*)

Set the etic string of the curve.

See also:

setName() (p. 134),**setCite()** (p. 135),**setHistory()** (p. 135)

Referenced by header().

9.14.3.5 void PKFmanip::setCite (const char * *cite*)

Set the cite string of the curve.

See also:

setName() (p. 134),**setEtic()** (p. 134),**setHistory()** (p. 135)

Referenced by header().

9.14.3.6 void PKFmanip::setHistory (const char * *history*)

Set the history string of the curve.

See also:

setName() (p. 134),**setEtic()** (p. 134),**setCite()** (p. 135)

Referenced by header().

9.14.3.7 const char * PKFmanip::getName () const

Returns a pointer to the name of the curve.

See also:

getEtic() (p. 135),**getCite()** (p. 135),**getHistory()** (p. 136)

Referenced by Curve< Vector >::operator+(), Curve< Vector >::operator-(), and operator=().

9.14.3.8 const char * PKFmanip::getEtic () const

Returns a pointer to the etic string.

See also:

getName() (p. 135),**getCite()** (p. 135),**getHistory()** (p. 136)

Referenced by Curve< Vector >::operator+(), Curve< Vector >::operator-(), and operator=().

9.14.3.9 const char * PKFmanip::getCite () const

Returns a pointer to the cite string.

See also:

getName() (p. 135),**getEtic()** (p. 135),**getHistory()** (p. 136)

Referenced by Curve< Vector >::operator+(), Curve< Vector >::operator-(), and operator=().

9.14.3.10 const char * PKFmanip::getHistory () const

Returns a pointer to the history string.

See also:

getName() (p. 135),**getEtic()** (p. 135),**getCite()** (p. 135)

Referenced by Curve< Vector >::operator+(), Curve< Vector >::operator-(), and operator=().

9.14.3.11 int PKFmanip::readHeader (istream & *in*)

Read the PKF header from a stream object *in*. Returns 1 if all is ok, 0 otherwise. More details about the PKF format are given in classes that inherit the **PKFmanip** (p. 133) class. I.e. **Curve** (p. 76), **CurveBundle** (p. 102) ...

See also:

writeHeader() (p. 136)

Referenced by TubeBundle< Vector >::readPKF(), CurveBundle< Vector >::readPKF(), and Curve< Vector >::readPKF().

9.14.3.12 int PKFmanip::writeHeader (ostream & *out*)

Writes the PKF header to an ostream object *out*. Returns 1 if all went well, zero otherwise.

See also:

readPKF()

Referenced by CurveBundle< Vector >::writePKF(), and Curve< Vector >::writePKF().

9.14.4 Friends And Related Function Documentation

9.14.4.1 ostream & PKFmanip::operator<< (ostream & *out*, PKFmanip & *c*) [friend]

Overloaded left shift operator. Writes the current **PKFmanip** (p. 133) object *c* to the ostream object *out*. If there is an interpolated curve, this function prints point/tangent,matching point/tangent of all the biarcs of the curve. For non valid biarcs only the point/tangent data is written to the stream.

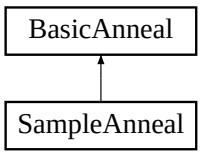
The documentation for this class was generated from the following files:

- include/PKFmanip.h
- lib/PKFmanip.cpp

9.15 SampleAnneal Class Reference

Example how to use BaseAnneal.

Inheritance diagram for SampleAnneal::



Public Member Functions

- **SampleAnneal** (const char *params="")
- bool **stop** ()
- void **best_found** ()
- float **energy** ()

Public Attributes

- int **no_of_nodes**
- int **N**
- vector< float > **nodes**
- vector< float > **best_nodes**

9.15.1 Detailed Description

Example how to use BaseAnneal.

An example annealing class that inherits **BasicAnneal** (p. 55). For N points fx_if in 2D, we want to minimize the energy $E = \sum_{i,j=1}^{N}(2 - dist(x_i, x_j))^2$, where dist is the euclidean distance between two points.

9.15.2 Constructor & Destructor Documentation

9.15.2.1 SampleAnneal::SampleAnneal (const char * *params* = "") [inline]

params - like **BasicAnneal** (p. 55). Addionally N - Number of points

References BasicAnneal::std_init().

9.15.3 Member Function Documentation

9.15.3.1 bool SampleAnneal::stop () [inline, virtual]

Stop criterion.

Reimplemented from **BasicAnneal** (p. 57).

9.15.3.2 void SampleAnneal::best_found () [inline, virtual]

This is called when a smaller Energy is found. Writes current coordinates to file best_filename.

Reimplemented from **BasicAnneal** (p. 57).

References BasicAnneal::best_found().

9.15.3.3 float SampleAnneal::energy () [inline, virtual]

The energy for this problem. We want the euclidean distance (dist) between N points to be 2! Which means that the energy is $E = \sum_{i,j=1}^{N}(2 - dist(x_i, x_j))^2$.

Reimplemented from **BasicAnneal** (p. 57).

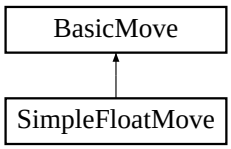
The documentation for this class was generated from the following file:

- experimental/annealing/my_anneal.cpp

9.16 SimpleFloatMove Class Reference

Change a single float value.

Inheritance diagram for SimpleFloatMove::



Public Member Functions

- **SimpleFloatMove** (float *node, float step_size=1e-6, float STEP_CHANGE=0.01)
- virtual ~**SimpleFloatMove** ()
- void **move** ()
- void **reject** ()
- void **accept** ()

Public Attributes

- float * **node**
- float **old_value**

9.16.1 Detailed Description

Change a single float value.

Change the float entry of anneal.nodes[addr] by a random number within (-step_size, +step_size).

9.16.2 Constructor & Destructor Documentation

9.16.2.1 SimpleFloatMove::SimpleFloatMove (float * *node*, float *step_size* = 1e-6, float *STEP_CHANGE* = 0.01) [inline]

node - address of the node in anneal.

9.16.2.2 virtual SimpleFloatMove::~SimpleFloatMove () [inline, virtual]

Virtual destructor

9.16.3 Member Function Documentation

9.16.3.1 void SimpleFloatMove::move () [inline, virtual]

Perform a simple move. Change the value of a node.

Reimplemented from **BasicMove** (p. 59).

9.16.3.2 void SimpleFloatMove::reject () [inline, virtual]

Reject a move and reset to old value.

Reimplemented from **BasicMove** (p. 60).

References BasicMove::reject().

9.16.3.3 void SimpleFloatMove::accept () [inline, virtual]

Accept move and backup new value.

Reimplemented from **BasicMove** (p. 60).

References BasicMove::accept().

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

- experimental/annealing/my_anneal.cpp

9.17 SoKnot Class Reference

The **SoKnot** (p. 141) class is for rendering tubular curves in the Coin/OpenInventor graphics library.

`#include <SoKnot.h>`

Public Member Functions

- **SoKnot** (void)
- void **setKnot** (**Tube**< **Vector3** > *t, const float Tol)
- void **updateMesh** (const float Tol)
- **Tube**< **Vector3** > * **getKnot** ()
- void **reset** ()
- virtual void **GLRender** (SoGLRenderAction *action)
- virtual void **getPrimitiveCount** (SoGetPrimitiveCountAction *action)

Static Public Member Functions

- static void **initClass** (void)

Public Attributes

- SoSFFloat **radius**
- SoSFInt32 **nodes**
- SoSFInt32 **segments**

Protected Member Functions

- virtual ~**SoKnot** ()
- virtual void **generatePrimitives** (SoAction *action)
- virtual void **computeBBox** (SoAction *action, SbBox3f &box, SbVec3f &center)

9.17.1 Detailed Description

The **SoKnot** (p. 141) class is for rendering tubular curves in the Coin/OpenInventor graphics library.

For details about the Coin3D API and some more precise explanations about the structure of Coin see

`http://www.coin3d.org/doc/`

The **SoKnot** (p. 141) class is a ready made node for a Coin3D/OpenInventor program that generates a tubular shape and can then be inserted in a scenegraph. The shape is rendered with the current material. The paramters for rendering a knot are the radius, the number of nodes on the centerline and the number of segments on the cross-section.

put example how to use this ...

9.17.2 Constructor & Destructor Documentation

9.17.2.1 SoKnot::SoKnot (void)

Constructor.

References nodes, radius, and segments.

9.17.2.2 SoKnot::~SoKnot () [protected, virtual]

Destructor.

9.17.3 Member Function Documentation

9.17.3.1 void SoKnot::initClass (void) [static]

Static function that initializes the **SoKnot** (p. 141) class in Coin.

9.17.3.2 void SoKnot::setKnot (Tube< Vector3 > * *t*, const float *Tol*)

Initializes the **SoKnot** (p. 141) instance, given a **Tube** (p. 147) object t and a tolerance *Tol* for the mesh generation routine.

The original **Tube** (p. 147) t will not be changed in any way. Every time an action on the **Tube** (p. 147) object is necessary, the work is carried out on a copy, internally of the **SoKnot** (p. 141) instance.

See also:

getKnot() (p. 142)

References Tube< Vector >::makeMesh(), Curve< Vector >::nodes(), Tube< Vector >::radius(), and Tube< Vector >::segments().

9.17.3.3 void SoKnot::updateMesh (const float *Tol*)

Recreates the mesh using the params currently stored in knot. Used for example by refine to update the internal mesh and number of nodes.

References Tube< Vector >::makeMesh(), nodes, Curve< Vector >::nodes(), radius, Tube< Vector >::radius(), segments, and Tube< Vector >::segments().

9.17.3.4 Tube< Vector3 > * SoKnot::getKnot ()

Returns a pointer to the current **Tube** (p. 147) object stored internally in the **SoKnot** (p. 141) object.

See also:

setKnot() (p. 142)

9.17.3.5 void SoKnot::reset ()

Reset current knot to original.

9.17.3.6 void SoKnot::GLRender (SoGLRenderAction * *action*) [virtual]

Action method for the SoGLRenderAction. This is called during rendering traversals. Reimplemented from SoShape.

Note : This comes from the Coin documentation

References Curve< Vector >::isClosed(), nodes, Curve< Vector >::nodes(), radius, segments, and Tube< Vector >::segments().

9.17.3.7 void SoKnot::getPrimitiveCount (SoGetPrimitiveCountAction * *action*) [virtual]

Action method for the SoGetPrimitiveCountAction.

Calculates the number of triangle, line segment and point primitives for the node and adds these to the counters of the action.

Nodes influencing how geometry nodes calculates their primitive count also overrides this method to change the relevant state variables.

Reimplemented from SoShape.

References nodes, and segments.

9.17.3.8 void SoKnot::generatePrimitives (SoAction * *action*) [protected, virtual]

The method implements action behavior for shape nodes for SoCallbackAction. It is invoked from SoShape::callback(). (Subclasses should not override SoNode::callback().)

The subclass implementations uses the convenience methods SoShape::beginShape(), SoShape::shapeVertex(), and SoShape::endShape(), with SoDetail instances, to pass the primitives making up the shape back to the caller.

Implements SoShape.

References Curve< Vector >::isClosed(), Tube< Vector >::meshNormal(), Tube< Vector >::meshPoint(), nodes, and segments.

9.17.3.9 void SoKnot::computeBBox (SoAction * *action*, SbBox3f & *box*, SbVec3f & *center*) [protected, virtual]

Implemented by SoShape subclasses to let the SoShape superclass know the exact size and weighted center point of the shape's bounding box.

The bounding box and center point should be calculated and returned in the local coordinate system.

The method implements action behavior for shape nodes for SoGetBoundingBoxAction. It is invoked from SoShape::getBoundingBox(). (Subclasses should not override SoNode::getBoundingBox().)

The box parameter sent in is guaranteed to be an empty box, while center is undefined upon function entry.

Implements SoShape.

References Tube< Vector >::getBoundingBox(), and Tube< Vector >::getCenter().

9.17.4 Member Data Documentation

9.17.4.1 SoSFFloat SoKnot::radius

Tube (p. 147) radius of the **SoKnot** (p. 141) instance.

Referenced by GLRender(), SoKnot(), and updateMesh().

9.17.4.2 SoSFInt32 SoKnot::nodes

Number of nodes (number of data points) along the curve.

Referenced by generatePrimitives(), getPrimitiveCount(), GLRender(), SoKnot(), and updateMesh().

9.17.4.3 SoSFInt32 SoKnot::segments

Number of segments on the circular cross section of the tube.

Referenced by generatePrimitives(), getPrimitiveCount(), GLRender(), SoKnot(), and updateMesh().

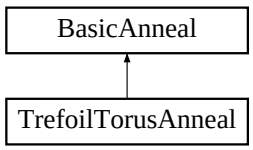
The documentation for this class was generated from the following files:

- inventor/SoKnot.h
- inventor/SoKnot.cpp

9.18 TrefoilTorusAnneal Class Reference

Anneal a trefoil running on 3 torii.

Inheritance diagram for TrefoilTorusAnneal::



Public Member Functions

- void **std_init** (const char *params="")
- **TrefoilTorusAnneal** (const char *params="")
- bool **stop** ()
- void **best_found** ()
- float **energy** ()

Public Attributes

- int **N**
- float **xoffset**
- float **tilt**
- vector< float > **angles**

9.18.1 Detailed Description

Anneal a trefoil running on 3 torii.

9.18.2 Member Function Documentation

9.18.2.1 void TrefoilTorusAnneal::std_init (const char * *params* = "") [inline]

params - like **BasicAnneal** (p. 55). Addionally N - Number of angles tilt - Torus tilt angle xoffset - Distance from z-Axis in x direction

Reimplemented from **BasicAnneal** (p. 56).

References BasicAnneal::std_init().

9.18.2.2 bool TrefoilTorusAnneal::stop () [inline, virtual]

Reimplemented stop criterion.

Reimplemented from **BasicAnneal** (p. 57).

9.18.2.3 void TrefoilTorusAnneal::best_found () [inline, virtual]

This gets called each time anneal finds a new best energy.

Reimplemented from **BasicAnneal** (p. 57).

References BasicAnneal::best_found().

9.18.2.4 float TrefoilTorusAnneal::energy () [inline, virtual]

Build PKF curve and compute its thickness with the parameters for the 3 torii.

Reimplemented from **BasicAnneal** (p. 57).

References Curve< Vector >::append(), Curve< Vector >::computeTangents(), Curve< Vector >::length(), Curve< Vector >::link(), Curve< Vector >::nodes(), Curve< Vector >::rotAroundAxis(), Curve< Vector >::thickness(), and Curve< Vector >::writePKF().

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

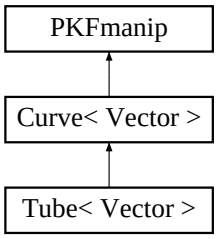
- experimental/annealing/trefoil_torus.cpp

9.19 Tube< Vector > Class Template Reference

The **Tube** (p. 147) class contains the mesh of a tube around a curve in $\mathcal{R}^3$.

`#include <include/Tube.h>`

Inheritance diagram for Tube< Vector >::



Public Member Functions

- **Tube** ()
- **Tube** (istream &in)
- **Tube** (const **Curve**< Vector > &curve)
- **Tube** (const **Tube**< Vector > &tube)
- **Tube** & **operator**= (const **Tube**< Vector > &t)
- ~**Tube** ()
- void **init** ()
- void **clear_tube** ()
- Vector & **meshPoint** (int i)
- Vector & **meshNormal** (int i)
- int **segments** ()
- float **radius** ()
- void **makeMesh** (int N, int S, float R, float Tol=-1.0)
- void **getBoundingBox** (Vector &BBox_BL, Vector &BBox_UR)
- Vector & **getCenter** ()
- void **scaleTubeRadius** (float NewRadius)

9.19.1 Detailed Description

template<class Vector> class Tube< Vector >

The **Tube** (p. 147) class contains the mesh of a tube around a curve in $\mathcal{R}^3$.

This class is used to store the points and normals of a tubular mesh around a space-curve.

The usage of **Tube** (p. 147) objects are shown in the following example :

```
#include "../include/Tube.h"

main(int argc, char **argv) {

  // Read curve data from a file
  Tube t("curve.pkf");

  // Close the curve
```

```
    t.link();

    // Number of nodes on the curve
    int N = t.nodes();

    // We want 10 segments on the cross section
    int S = 10;

    // The radius is set to 1.0
    float R = 1.0;

    // We match up the mesh with this tolerance value
    float Tol = 1e-3;

    // Generate mesh points and normals
    t.makeMesh(N,S,R,Tol);

    ...

    // Write the output to standart out
    cout << t << endl;

    ...

    return 0;
  }
```

See also:

Curve (p. 76), **Biarc** (p. 61)

9.19.2 Constructor & Destructor Documentation

9.19.2.1 template<class Vector> Tube< Vector >::Tube () [inline]

Default constructor. Creates an empty curve with no mesh points.

References Tube< Vector >::init().

9.19.2.2 template<class Vector> Tube< Vector >::Tube (istream & *in*) [inline]

Constructs a new **Tube** (p. 147) object, reading the curve data from a stream *in*. Header is automatically set to the default values and is NOT expected to on the stream!

See also:

makeMesh() (p. 150)

References Tube< Vector >::init().

9.19.2.3 template<class Vector> Tube< Vector >::Tube (const Curve< Vector > & *curve*) [inline]

Constructor copies a **Curve** (p. 76) object *curve*. No interpolation and no mesh generation is done. To do that **makeMesh()** (p. 150) must be called.

See also:

makeMesh() (p. 150)

References Tube< Vector >::init().

9.19.2.4 template<class Vector> Tube< Vector >::Tube (const Tube< Vector > & *tube*) [inline]

Constructor copies a **Tube** (p. 147) object *tube*. Only the curve data points are copied and the curve is closed if necessary. No interpolation and no mesh generation is done in the new **Tube** (p. 147) instance. To do that **makeMesh()** (p. 150) must be called.

Maybe this will be changed in the future to copy all the mesh points if the *tube* object has mesh points.

See also:

makeMesh() (p. 150)

References Tube< Vector >::init().

9.19.2.5 template<class Vector> Tube< Vector >::~Tube () [inline]

Destructor. Clears the memory used by the mesh points and normals.

References Tube< Vector >::clear_tube().

9.19.3 Member Function Documentation

9.19.3.1 template<class Vector> Tube< Vector > & Tube< Vector >::operator= (const Tube< Vector > & *c*) [inline]

Assign operator. Copies the curve data and calls **link()** (p. 85) if **Tube** (p. 147) *c* is closed.

References Curve< Vector >::_Biarcs, Curve< Vector >::begin(), Curve< Vector >::end(), Curve< Vector >::flush_all(), Curve< Vector >::isClosed(), Curve< Vector >::link(), and Curve< Vector >::nodes().

9.19.3.2 template<class Vector> void Tube< Vector >::init () [inline]

Initialize **Tube** (p. 147) object. Number of segments and the radius are set to 0.0. The mesh point table and the mesh normal table are set to NULL.

Reimplemented from **Curve< Vector >** (p. 76).

Referenced by Tube< Vector >::Tube().

9.19.3.3 template<class Vector> void Tube< Vector >::clear_tube () [inline]

Deletes the mesh points and the mesh normals.

Referenced by Tube< Vector >::makeMesh(), and Tube< Vector >::~Tube().

9.19.3.4 template<class Vector> Vector & Tube< Vector >::meshPoint (int *i*) [inline]

Returns mesh point number i. There are a total of $(segments + 1) \cdot (nodes + 1)$ mesh points.

Referenced by SoKnot::generatePrimitives().

9.19.3.5 template<class Vector> Vector & Tube< Vector >::meshNormal (int *i*) [inline]

Returns mesh normal number i. There are a total of $(segments + 1) \cdot (nodes + 1)$ mesh normals.

Referenced by SoKnot::generatePrimitives().

9.19.3.6 template<class Vector> int Tube< Vector >::segments () [inline]

Returns the number of segments on the tubular cross section.

See also:

Curve::nodes() (p. 94),**radius()** (p. 150)

Referenced by SoKnot::GLRender(), SoKnot::setKnot(), and SoKnot::updateMesh().

9.19.3.7 template<class Vector> float Tube< Vector >::radius () [inline]

Returns the current radius of the tubular curve.

See also:

Curve::nodes() (p. 94),**segments()** (p. 150)

Referenced by SoKnot::setKnot(), and SoKnot::updateMesh().

9.19.3.8 template<class Vector> void Tube< Vector >::makeMesh (int *N*, int *S*, float *R*, float *Tol* = -1.0) [inline]

Creates a tubular mesh with radius R around the curve with N nodes and S segments on each cross sectional circle. If the actully stored curve has not exactly N nodes, the **Curve::resample()** (p. 87) routine is called automatically.

The parameter *Tol* is the precision used to math up the mesh of a closed curve. *Tol* is the difference in radians of the angle between the first and the last normal of the sequence of curve frames.

Use *Tol* = -1 (default) for no torsion adjustment and open curves.

Computes also the mesh normals at each point where the normal is an average of the neighbouring face normals. (The normals are used for Gouraud shading in visualisation programs).

At the end the center of mass and the bounding box are computed.

The interpolation in case of a resampling is done with Gamma = 0.5.

References Curve< Vector >::_Closed, Curve< Vector >::begin(), Matrix3::cay(), Tube< Vector >::clear_tube(), Vector3::cross(), Vector3::dot(), Curve< Vector >::link(), Curve< Vector >::nodes(), Vector3::normalize(), Curve< Vector >::resample(), Vector3::rotPtAroundAxis(), and Curve< Vector >::unlink().

Referenced by SoKnot::setKnot(), and SoKnot::updateMesh().

9.19.3.9 template<class Vector> void Tube< Vector >::getBoundingBox (Vector & *BBox_Min*, Vector & *BBox_Max*) [inline]

Puts the position of the minimum and the maximum corner of a box containing the whole tubular object into the vectors *BBox_Min* and *BBox_Max*.

Referenced by SoKnot::computeBBox().

9.19.3.10 template<class Vector> Vector & Tube< Vector >::getCenter () [inline]

Returns the center of mass of the tubular mesh.

Reimplemented from **Curve< Vector >** (p. 96).

Referenced by SoKnot::computeBBox().

9.19.3.11 template<class Vector> void Tube< Vector >::scaleTubeRadius (float *NewRadius*) [inline]

Inflates or deflates the existing mesh according to the new radius given. All the cross sectional circles are changed to have radius *NewRadius*. This function drops an error message if there are no valid mesh points.

The bounding box values are updated at the end of the process.

References Curve< Vector >::_Closed, Curve< Vector >::begin(), Curve< Vector >::end(), and Curve< Vector >::nodes().

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

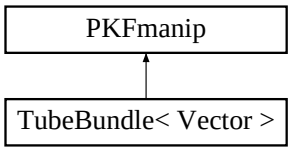
- include/Tube.h
- lib/Tube.cpp

9.20 TubeBundle< Vector > Class Template Reference

The **TubeBundle** (p. 152) class contains the mesh of a tube around a curve in $\mathcal{R}^3$.

`#include <include/TubeBundle.h>`

Inheritance diagram for TubeBundle< Vector >::



Public Member Functions

- void **clear_tb** ()
- **TubeBundle** ()
- **TubeBundle** (const char *filename)
- **TubeBundle** (const **TubeBundle**< Vector > &tb)
- **TubeBundle** (const **CurveBundle**< Vector > &cb)
- **TubeBundle**< Vector > & **operator**= (const **TubeBundle**< Vector > &tb)
- ~**TubeBundle** ()
- int **tubes** () const
- void **newTube** (const **Tube**< Vector > &t)
- void **newTube** (istream &in)
- **Tube**< Vector > & **operator[]** (int c)
- void **makeMesh** (int N, int S, float R, float Tol=-1.0)
- void **getBoundingBox** (Vector &BBox_Min, Vector &BBox_Max)
- Vector & **getCenter** ()
- void **scaleTubeRadius** (float NewRadius)
- int **readPKF** (const char *infile)
- int **readXYZ** (const char *infile)
- int **readPKF** (istream &in)
- int **readData** (const char *infile, const char *delimiter=" ")

9.20.1 Detailed Description

template<class Vector> class TubeBundle< Vector >

The **TubeBundle** (p. 152) class contains the mesh of a tube around a curve in $\mathcal{R}^3$.

This class is used to store the points and normals of a tubular mesh around a space-curve.

The usage of **TubeBundle** (p. 152) objects are shown in the following example :

```
#include "../include/TubeBundle.h"

main(int argc, char **argv) {

  // Read curve data from a file
  TubeBundle t("curve.pkf");
```

```
    // Close the curve
    t.link();

    // Number of nodes on the curve
    int N = t.nodes();

    // We want 10 segments on the cross section
    int S = 10;

    // The radius is set to 1.0
    float R = 1.0;

    // We match up the mesh with this tolerance value
    float Tol = 1e-3;

    // Generate mesh points and normals
    t.makeMesh(N,S,R,Tol);

    ...

    // Write the output to standart out
    cout << t << endl;

    ...

    return 0;
  }
```

See also:

Curve (p. 76), **Biarc** (p. 61)

9.20.2 Constructor & Destructor Documentation

9.20.2.1 template<class Vector> TubeBundle< Vector >::TubeBundle () [inline]

Default constructor. Creates an empty tube bundle.

9.20.2.2 template<class Vector> TubeBundle< Vector >::TubeBundle (const char * *filename*) [inline]

Constructor reads the curves data from *infile* and stores it in a **Tube** (p. 147) container. No interpolation is done, and no mesh points are generated. A call to **makeMesh()** (p. 155) is necessary to generate the mesh points for all the curves.

See also:

makeMesh() (p. 155),**Tube::makeMesh()** (p. 150)

References TubeBundle< Vector >::readPKF().

9.20.2.3 template<class Vector> TubeBundle< Vector >::TubeBundle (const TubeBundle< Vector > & *tb*) [inline]

Constructor copies a **TubeBundle** (p. 152) object *tube*. Only the curve data points are copied and the curve is closed if necessary. No interpolation and no mesh generation is done in the new **TubeBundle** (p. 152) instance. To do that **makeMesh()** (p. 155) must be called.

Maybe this will be changed in the future to copy all the mesh points if the *tube* object has mesh points.

See also:

makeMesh() (p. 155)

9.20.2.4 template<class Vector> TubeBundle< Vector >::TubeBundle (const CurveBundle< Vector > & *cb*) [inline]

Constructor copies a **Curve** (p. 76) object *curve*. No interpolation and no mesh generation is done. To do that **makeMesh()** (p. 155) must be called.

See also:

makeMesh() (p. 155)

References CurveBundle< Vector >::curves(), and TubeBundle< Vector >::newTube().

9.20.2.5 template<class Vector> TubeBundle< Vector >::~TubeBundle () [inline]

Destructor. Clears the memory used by the mesh points and normals.

9.20.3 Member Function Documentation

9.20.3.1 template<class Vector> TubeBundle< Vector > & TubeBundle< Vector >::operator= (const TubeBundle< Vector > & *tb*) [inline]

Assign operator. Copies the curve data and calls link() if **TubeBundle** (p. 152) *c* is closed.

References TubeBundle< Vector >::container, and TubeBundle< Vector >::tubes().

9.20.3.2 template<class Vector> int TubeBundle< Vector >::tubes () const [inline]

Returns the number of tubes int this bundle.

Referenced by TubeBundle< Vector >::makeMesh(), CurveInterface::makeMesh(), TubeBundle< Vector >::operator=(), and TubeBundle< Vector >::scaleTubeRadius().

9.20.3.3 template<class Vector> void TubeBundle< Vector >::newTube (const Tube< Vector > & *t*) [inline]

Add a tube *t* to the bundle.

Referenced by CurveInterface::load(), TubeBundle< Vector >::newTube(), TubeBundle< Vector >::readData(), TubeBundle< Vector >::readPKF(), TubeBundle< Vector >::readXYZ(), and TubeBundle< Vector >::TubeBundle().

9.20.3.4 template<class Vector> void TubeBundle< Vector >::newTube (istream & *in*) [inline]

Read the curve data from a stream *in*. The curve is converted into a **Tube** (p. 147) object and added to the **TubeBundle** (p. 152).

References TubeBundle< Vector >::newTube().

9.20.3.5]

template<class Vector> **Tube**< Vector > & **TubeBundle**< Vector >::operator[] (int *c*) [inline]

Return a reference to the tube number *c* in the **TubeBundle** (p. 152) container.

9.20.3.6 template<class Vector> void TubeBundle< Vector >::makeMesh (int *N*, int *S*, float *R*, float *Tol* = -1.0) [inline]

Creates a tubular mesh with radius *R* around the curve with N nodes and S segments on each cross sectional circle. If the actually stored curve has not exactly N nodes, the **Curve::resample()** (p. 87) routine is called automatically.

The parameter *Tol* is the precision used to math up the mesh of a closed curve. *Tol* is the difference in radians of the angle between the first and the last normal of the sequence of curve frames.

Use *Tol* = -1 (default) for no torsion adjustment and open curves.

Computes also the mesh normals at each point where the normal is an average of the neighbouring face normals. (The normals are used for Gouraud shading in visualisation programs).

At the end the center of mass and the bounding box are computed.

The interpolation in case of a resampling is done with Gamma = 0.5.

References TubeBundle< Vector >::tubes().

Referenced by CurveInterface::makeMesh().

9.20.3.7 template<class Vector> void TubeBundle< Vector >::getBoundingBox (Vector & *BBox_Min*, Vector & *BBox_Max*) [inline]

Puts the position of the minimum and the maximum corner of a box containing the whole tubular object into the vectors *BBox_Min* and *BBox_Max*.

9.20.3.8 template<class Vector> Vector & TubeBundle< Vector >::getCenter () [inline]

Returns the center of mass of the tubular mesh.

9.20.3.9 template<class Vector> void TubeBundle< Vector >::scaleTubeRadius (float *NewRadius*) [inline]

Inflates or deflates the existing mesh according to the new radius given. All the cross sectional circles are changed to have radius *NewRadius*. This function drops an error message if there are no valid mesh points.

The bounding box values are updated at the end of the process.

References TubeBundle< Vector >::tubes().

9.20.3.10 template<class Vector> int TubeBundle< Vector >::readPKF (const char * *infile*) [inline]

Read a PKF file with more than one curve from *infile*.

Referenced by CurveInterface::load(), and TubeBundle< Vector >::TubeBundle().

9.20.3.11 template<class Vector> int TubeBundle< Vector >::readXYZ (const char * *infile*) [inline]

Read XYZ file.

See also:

Curve::readXYZ() (p. 99)

References TubeBundle< Vector >::newTube(), and Curve< Vector >::readXYZ().

Referenced by CurveInterface::load().

9.20.3.12 template<class Vector> int TubeBundle< Vector >::readPKF (istream & *in*) [inline]

Read a PKF file with more than one curve from istream *in*.

References TubeBundle< Vector >::newTube(), and PKFmanip::readHeader().

9.20.3.13 template<class Vector> int TubeBundle< Vector >::readData (const char * *infile*, const char * *delimiter* = " ") [inline]

Read a data file. This routine has been used for different file formats. If needed, adapt it ;)

References TubeBundle< Vector >::newTube(), and Curve< Vector >::readData().

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

- include/TubeBundle.h
- lib/TubeBundle.cpp

9.21 Vec2 Class Reference

2D Vector class.

Public Member Functions

- **Vec2** (const float _x, const float _y)
- **Vec2** (const **Vec2** &v)
- **Vec2 operator**+ (const **Vec2** &v) const
- **Vec2 operator-** (const **Vec2** &v) const
- **Vec2 rot** (const float ang) const
- **Vec2** (const float _x, const float _y)
- **Vec2** (const **Vec2** &v)
- **Vec2 operator**+ (const **Vec2** &v) const
- **Vec2 operator-** (const **Vec2** &v) const
- bool **operator**< (const **Vec2** &v) const
- **Vec2 rot** (const float ang) const

Public Attributes

- float **x**
- float **y**

Friends

- ostream & **operator**<< (ostream &out, const **Vec2** &v)
- ostream & **operator**<< (ostream &out, const **Vec2** &v)

9.21.1 Detailed Description

2D Vector class.

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

- experimental/annealing/box_problem.cpp
- experimental/annealing/fit.cpp

9.22 Vector3 Class Reference

The **Vector3** (p. 158) class is a 3 dimensional Vector class with floating point coordinates.

`#include <include/Vector3.h>`

Public Member Functions

- **Vector3** ()
- **Vector3** (const float x, const float y, const float z)
- **Vector3** (const float v[3])
- **Vector3** (const **Vector3** &v)
- ~**Vector3** ()
- float & **operator[]** (const int c)
- const float & **operator[]** (const int c) const
- **Vector3** & **zero** ()
- void **getValues** (float &X, float &Y, float &Z) const
- **Vector3** & **setValues** (const float X, const float Y, const float Z)
- **Vector3** & **setValues** (const float v[3])
- float **dot** (const **Vector3** &v) const
- **Vector3 cross** (const **Vector3** &v) const
- float **norm** () const
- float **norm2** () const
- **Vector3** & **normalize** ()
- float **max** ()
- float **min** ()
- **Vector3 reflect** (const **Vector3** &v) const
- **Vector3 rotPtAroundAxis** (float angle, **Vector3** axis) const
- **Vector3 operator*** (const **Vector3** &v) const
- **Vector3 operator+** (const **Vector3** &v) const
- **Vector3 operator-** (const **Vector3** &v) const
- **Vector3 operator-** () const
- **Vector3** & **operator+=** (const **Vector3** &v)
- **Vector3** & **operator-=** (const **Vector3** &v)
- **Vector3** & **operator*=** (const float s)
- **Vector3** & **operator/=** (const float s)
- int **operator==** (const **Vector3** &v) const
- int **operator!=** (const **Vector3** &v) const
- void **print** (ostream &out) const

Static Public Attributes

- static const unsigned int **type** = 3

Friends

- **Vector3 operator*** (const **Vector3** &v, float d)
- **Vector3 operator*** (float d, const **Vector3** &v)
- **Vector3 operator/** (const **Vector3** &v, float d)
- ostream & **operator<<** (ostream &out, const **Vector3** &v)
- istream & **operator>>** (istream &in, **Vector3** &v)

9.22.1 Detailed Description

The **Vector3** (p. 158) class is a 3 dimensional Vector class with floating point coordinates.

This class provides storage for a 3 dimensional vector aswell as arithmetic operations on vectors, like the dot product, the cross product, the length of the vector.

9.22.2 Constructor & Destructor Documentation

9.22.2.1 Vector3::Vector3 ()

Constructs an empty vector initialized to zero.

See also:

zero() (p. 160).

Referenced by cross(), operator∗(), operator+(), and operator-().

9.22.2.2 Vector3::Vector3 (const float *x*, const float *y*, const float *z*)

Constructs a **Vector3** (p. 158) instance from x, y and z.

9.22.2.3 Vector3::Vector3 (const float *v*[3])

Constructs a **Vector3** (p. 158) with initial values from v.

9.22.2.4 Vector3::Vector3 (const Vector3 & *v*)

Constructs a copy of v.

9.22.2.5 Vector3::∼Vector3 ()

The destructor has nothing to do

9.22.3 Member Function Documentation

9.22.3.1]

float & Vector3::operator[] (const int *c*) `[inline]`

Index operator. Returns modifiable x, y or z coordinate of the vector.

See also:

getValue() and setValue().

9.22.3.2]

float Vector3::operator[] (const int *c*) const [inline]

Index operator. Returns x, y or z coordinate of vector.

See also:

getValue() and setValue().

9.22.3.3 Vector3 & Vector3::zero ()

Sets all vector components to zero

Referenced by Matrix3::Matrix3(), and Matrix3::zero().

9.22.3.4 void Vector3::getValues (float & *X*, float & *Y*, float & *Z*) const

Recover the vector components and put them into X, Y and Z.

9.22.3.5 Vector3 & Vector3::setValues (const float *X*, const float *Y*, const float *Z*)

Set the vector to <X,Y,Z>. Returns a reference to itself.

9.22.3.6 Vector3 & Vector3::setValues (const float *v*[3])

Set new coordinates from the given array v. Returns a reference to itself.

9.22.3.7 float Vector3::dot (const Vector3 & *v*) const

Returns the dot product between the current vector and v.

References _v.

Referenced by Tube< Vector >::makeMesh(), Curve< Vector >::normalVector(), Biarc< Vector >::pointOnArc0(), Biarc< Vector >::pointOnArc1(), and rotPtAroundAxis().

9.22.3.8 Vector3 Vector3::cross (const Vector3 & *v*) const

Returns a vector w, that is the result of the cross product between the actual vector *and* a vector v.

returns $w = a \times v$

References _v, and Vector3().

Referenced by Tube< Vector >::makeMesh().

9.22.3.9 float Vector3::norm () const

Computes and returns the length of the vector.

$|v| = \sqrt{x^2 + y^2 + z^2}$.

See also:

norm2() (p. 161), **normalize()** (p. 161).

Referenced by normalize(), Biarc< Vector >::pointOnArc0(), and Biarc< Vector >::pointOnArc1().

9.22.3.10 float Vector3::norm2 () const

Computes and returns the squared norm of the vector.

$|v|^2 = x^2 + y^2 + z^2$.

See also:

norm() (p. 160), **normalize()** (p. 161).

Referenced by Matrix3::cay().

9.22.3.11 Vector3 & Vector3::normalize ()

Normalizes the current vector and returns a reference to itself

If the vector has zero norm, a warning message is posted to the error output and the **Vector3** (p. 158) is Not modified.

See also:

norm() (p. 160), **norm2()** (p. 161).

References norm().

Referenced by Tube< Vector >::makeMesh(), Biarc< Vector >::pointOnArc0(), Biarc< Vector >::pointOnArc1(), reflect(), and Matrix3::rotAround().

9.22.3.12 float Vector3::max ()

Returns the largest component of the current vector.

9.22.3.13 float Vector3::min ()

Returns the smallest component of the current vector.

9.22.3.14 Vector3 Vector3::reflect (const Vector3 & *ref_ax*) const

Returns the mirror vector according to the given axe *ref_ax*. The reference axe needs not to be normalized, since this is done automatically.

References normalize(), and Matrix3::outer().

9.22.3.15 Vector3 Vector3::rotPtAroundAxis (float *angle*, Vector3 *axis*) const

Rotate the current point at <x,y,z> around the given rotational axes *axis* and an angle *angle*. The value for the angle is given in radians.

References dot().

Referenced by Tube< Vector >::makeMesh().

9.22.3.16 Vector3 Vector3::operator* (const Vector3 & *v*) const

Returns a **Vector3** (p. 158) that is the dot product of this vector and *v*.

References _v, and Vector3().

9.22.3.17 Vector3 Vector3::operator+ (const Vector3 & *v*) const

Sum of the vector with *v*.

References Vector3().

9.22.3.18 Vector3 Vector3::operator- (const Vector3 & *v*) const

Returns the difference between this vector and *v*.

References Vector3().

9.22.3.19 Vector3 Vector3::operator- () const

Returns a new **Vector3** (p. 158) that points in the opposite direction.

References Vector3().

9.22.3.20 Vector3 & Vector3::operator+= (const Vector3 & *v*)

Adds the vector *v* to this vector and returns an instance to itself.

9.22.3.21 Vector3 & Vector3::operator-= (const Vector3 & *v*)

Substracts the vector *v* from this vector and returns an instance to itself.

9.22.3.22 Vector3 & Vector3::operator*= (const float *s*)

Assign operator. Does the same as the copy constructor.

Multiplies this vector's components by *s* and returns an instance to itself.

9.22.3.23 Vector3 & Vector3::operator/= (const float *s*)

Divides this vector's components by *s* and returns an instance to itself.

9.22.3.24 int Vector3::operator== (const Vector3 & *v*) const

Comparison operator. Returns 1 if the current vector and v are the same. Returns 0 otherwise.

9.22.3.25 int Vector3::operator!= (const Vector3 & *v*) const

If the current vector and v are not equal, this function returns 1, and 0 otherwise.

9.22.3.26 void Vector3::print (ostream & *out*) const

Prints the components of this vector in a formatted way onto the stream *out*.

9.22.4 Friends And Related Function Documentation

9.22.4.1 Vector3 operator* (const Vector3 & *v*, float *d*) [friend]

Returns a **Vector3** (p. 158) that is this vector scaled by a scalar s.

Returns a **Vector3** (p. 158) that is the vector v scaled by a scalar d.

9.22.4.2 Vector3 operator* (float *d*, const Vector3 & *v*) [friend]

Returns a **Vector3** (p. 158), that is the vector v scaled by a scalar s.

9.22.4.3 Vector3 operator/ (const Vector3 & *v*, float *d*) [friend]

Returns a **Vector3** (p. 158) whose components are divided by d.

Friend function that returns the vector v whose components are scaled by d.

9.22.4.4 ostream & Vector3::operator<< (ostream & *out*, const Vector3 & *v*) [friend]

Overloaded left shift operator. Returns the Vector v as an ostream that can be written to a file or to standart output.

See also:

print() (p. 163) **operator>>()** (p. 163)

9.22.4.5 ostream & Vector3::operator>> (istream & *out*, Vector3 & *v*) [friend]

Overloaded right shift operator. Initialize vector v with istream *out*.

See also:

print() (p. 163) **operator<<()** (p. 163)

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

- include/Vector3.h
- lib/Vector3.cpp

9.23 Vector4 Class Reference

The **Vector4** (p. 165) class is a 4 dimensional Vector class with floating point coordinates.

`#include <include/Vector4.h>`

Public Member Functions

- **Vector4** ()
- **Vector4** (const float x, const float y, const float z, const float w)
- **Vector4** (const float v[4])
- **Vector4** (const **Vector4** &v)
- **Vector4** (const **Vector3** &v, const float f)
- ~**Vector4** ()
- float & **operator[ ]** (const int c)
- const float & **operator[ ]** (const int c) const
- **Vector4** & **zero** ()
- void **getValues** (float &X, float &Y, float &Z, float &W) const
- **Vector4** & **setValues** (const float X, const float Y, const float Z, const float W)
- **Vector4** & **setValues** (const float v[4])
- float **dot** (const **Vector4** &v) const
- float **norm** () const
- float **norm2** () const
- **Vector4** & **normalize** ()
- float **max** ()
- float **min** ()
- **Vector4 reflect** (const **Vector4** &v) const
- **Vector4 operator**∗ (const **Vector4** &v) const
- **Vector4 operator**+ (const **Vector4** &v) const
- **Vector4 operator-** (const **Vector4** &v) const
- **Vector4 operator-** () const
- **Vector4** & **operator**+= (const **Vector4** &v)
- **Vector4** & **operator-**= (const **Vector4** &v)
- **Vector4** & **operator**∗= (const float s)
- **Vector4** & **operator**/= (const float s)
- int **operator**== (const **Vector4** &v) const
- int **operator!**= (const **Vector4** &v) const
- void **print** (ostream &out) const

Static Public Attributes

- static const unsigned int **type** = 4

Friends

- **Vector4 operator**∗ (const **Vector4** &v, float d)
- **Vector4 operator**∗ (float d, const **Vector4** &v)
- **Vector4 operator**/ (const **Vector4** &v, float d)
- ostream & **operator**<< (ostream &out, const **Vector4** &v)
- istream & **operator**>> (istream &in, **Vector4** &v)

9.23.1 Detailed Description

The **Vector4** (p. 165) class is a 4 dimensional Vector class with floating point coordinates.

This class provides storage for a 4 dimensional vector aswell as arithmetic operations on vectors, like the dot product, the cross product, the length of the vector in 4-dimensions.

See also:

class **Vector3** (p. 158)

9.23.2 Constructor & Destructor Documentation

9.23.2.1 Vector4::Vector4 ()

Constructs an empty vector initialized to zero.

See also:

zero() (p. 167).

Referenced by operator∗(), operator+(), and operator-().

9.23.2.2 Vector4::Vector4 (const float *x*, const float *y*, const float *z*, const float *w*)

Constructs a **Vector4** (p. 165) instance from x, y, z and w.

9.23.2.3 Vector4::Vector4 (const float *v*[4])

Constructs a **Vector4** (p. 165) with initial values from v.

9.23.2.4 Vector4::Vector4 (const Vector4 & *v*)

Constructs a copy of v.

9.23.2.5 Vector4::Vector4 (const Vector3 & *v*, const float *f*)

Construct a **Vector4** (p. 165) object from a **Vector3** (p. 158) v and a float f.

9.23.2.6 Vector4::∼Vector4 ()

The destructor has nothing to do

9.23.3 Member Function Documentation

9.23.3.1]

float & Vector4::operator[] (const int *c*) `[inline]`

Index operator. Returns modifiable x, y or z coordinate of the vector.

See also:

getValue() and setValue().

9.23.3.2]

float Vector4::operator[] (const int *c*) const `[inline]`

Index operator. Returns x, y or z coordinate of vector.

See also:

getValue() and setValue().

9.23.3.3 Vector4 & Vector4::zero ()

Sets all vector components to zero

Referenced by Matrix4::Matrix4(), and Matrix4::zero().

9.23.3.4 void Vector4::getValues (float & *X*, float & *Y*, float & *Z*, float & *W*) const

Recover the vector components and put them into X, Y, Z and W.

9.23.3.5 Vector4 & Vector4::setValues (const float *X*, const float *Y*, const float *Z*, const float *W*)

Set the vector to <X,Y,Z,W>. Returns a reference to itself.

9.23.3.6 Vector4 & Vector4::setValues (const float *v*[4])

Set new coordinates from the given array v. Returns a reference to itself.

9.23.3.7 float Vector4::dot (const Vector4 & *v*) const

Returns the dot product between the current vector and v.

References _v.

9.23.3.8 float Vector4::norm () const

Computes and returns the length of the vector.
$|v| = \sqrt{x^2 + y^2 + z^2 + w^2}$.

See also:

norm2() (p. 168), **normalize()** (p. 168).

Referenced by normalize().

9.23.3.9 float Vector4::norm2 () const

Computes and returns the squared norm of the vector.

$|v|^2 = x^2 + y^2 + z^2 + w^2$.

See also:

norm() (p. 167), **normalize()** (p. 168).

9.23.3.10 Vector4 & Vector4::normalize ()

Normalizes the current vector and returns a reference to itself

If the vector has zero norm, a warning message is posted to the error output and the **Vector4** (p. 165) is Not modified.

See also:

norm() (p. 167), **norm2()** (p. 168).

References norm().

Referenced by reflect().

9.23.3.11 float Vector4::max ()

Returns the largest component of the current vector.

9.23.3.12 float Vector4::min ()

Returns the smallest component of the current vector.

9.23.3.13 Vector4 Vector4::reflect (const Vector4 & *ref_ax*) const

Returns the mirror vector according to the given axe *ref_ax*. The reference axe needs not to be normalized, since this is done automatically.

References normalize(), and Matrix4::outer().

9.23.3.14 Vector4 Vector4::operator∗ (const Vector4 & *v*) const

Returns a **Vector4** (p. 165) that is the dot product of this vector and v.

References _v, and Vector4().

9.23.3.15 Vector4 Vector4::operator+ (const Vector4 & *v*) const

Sum of the vector with v.

References Vector4().

9.23.3.16 Vector4 Vector4::operator- (const Vector4 & *v*) const

Returns the difference between this vector and *v*.

References Vector4().

9.23.3.17 Vector4 Vector4::operator- () const

Returns a new **Vector4** (p. 165) that points in the opposite direction.

References Vector4().

9.23.3.18 Vector4 & Vector4::operator+= (const Vector4 & *v*)

Adds the vector *v* to this vector and returns an instance to itself.

9.23.3.19 Vector4 & Vector4::operator-= (const Vector4 & *v*)

Substracts the vector *v* from this vector and returns an instance to itself.

9.23.3.20 Vector4 & Vector4::operator*= (const float *s*)

Assign operator. Does the same as the copy constructor.

Multiplies this vector's components by *s* and returns an instance to itself.

9.23.3.21 Vector4 & Vector4::operator/= (const float *s*)

Divides this vector's components by *s* and returns an instance to itself.

9.23.3.22 int Vector4::operator== (const Vector4 & *v*) const

Comparison operator. Returns 1 if the current vector and *v* are the same. Returns 0 otherwise.

9.23.3.23 int Vector4::operator!= (const Vector4 & *v*) const

If the current vector and *v* are not equal, this function returns 1, and 0 otherwise.

9.23.3.24 void Vector4::print (ostream & *out*) const

Prints the components of this vector in a formatted way onto the stream *out*.

9.23.4 Friends And Related Function Documentation

9.23.4.1 Vector4 operator* (const Vector4 & *v*, float *d*) [friend]

Returns a **Vector4** (p. 165) that is this vector scaled by a scalar *s*.

Returns a **Vector4** (p. 165) that is the vector *v* scaled by a scalar *d*.

9.23.4.2 Vector4 operator∗ (float *d*, const Vector4 & *v*) [friend]

Returns a **Vector4** (p. 165), that is the vector v scaled by a scalar s.

9.23.4.3 Vector4 operator/ (const Vector4 & *v*, float *d*) [friend]

Returns a **Vector4** (p. 165) whose components are divided by d.

Friend function that returns the vector v whose components are scaled by d.

9.23.4.4 ostream & Vector4::operator<< (ostream & *out*, const Vector4 & *v*) [friend]

Overloaded left shift operator. Returns the Vector v as an ostream that can be written to a file or to standart output.

See also:

print() (p. 169)

9.23.4.5 ostream & Vector4::operator>> (istream & *out*, Vector4 & *v*) [friend]

Overloaded right shift operator. Initialize vector v with istream *out*.

See also:

print() (p. 169) **operator<<()** (p. 170)

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

- include/Vector4.h
- lib/Vector4.cpp

9.24 ViewerInfo Struct Reference

`#include <main.h>`

Public Attributes

- int **ResamplePartFlag**
- int **FirstPoint**
- int **BackGroundFlag**
- vector< **Biarc**< **Vector3** > >::iterator **FirstBiarc**
- uint **TEXTURES**
- uint **IV_SCENE**
- QString **ptplot_file**
- QString **texture_file**
- QString **iv_file**

9.24.1 Detailed Description

Global variables, configuration and viewer setings structure.

The documentation for this struct was generated from the following file:

- inventor/main.h

Chapter 10

File Documentation

10.1 experimental/s3viz/s3viz.cpp File Reference

Visualize a curve in S^3 in two balls in R^3.

10.1.1 Detailed Description

Visualize a curve in S^3 in two balls in R^3.

10.2 inventor/xyzview.cpp File Reference

Visualize a closed data file.

10.2.1 Detailed Description

Visualize a closed data file.

10.3 objects/aux.h File Reference

Auxiliary routines that produce particular curves.

10.3.1 Detailed Description

Auxiliary routines that produce particular curves.

The routines contained in this file are convenience functions needed for generating f.ex circle, ellipse, and more. Read the source for more info.

10.4 objects/bone.cpp File Reference

Constructs a bone shaped curve.

10.4.1 Detailed Description

Constructs a bone shaped curve.

10.5 objects/borromean.cpp File Reference

Constructs the borromean rings.

10.5.1 Detailed Description

Constructs the borromean rings.

10.6 objects/circle.cpp File Reference

Constructs a circle.

10.6.1 Detailed Description

Constructs a circle.

10.7 objects/crouzy.cpp File Reference

Constructs a crouzy curve ;).

10.7.1 Detailed Description

Constructs a crouzy curve ;).

10.8 objects/ellipse.cpp File Reference

Constructs an ellipse.

10.8.1 Detailed Description

Constructs an ellipse.

10.9 objects/helix.cpp File Reference

Constructs a helical curve.

10.9.1 Detailed Description

Constructs a helical curve.

10.10 objects/hopf.cpp File Reference

Constructs a hopf links.

10.10.1 Detailed Description

Constructs a hopf links.

10.11 objects/inf.cpp File Reference

Constructs an infinity shaped curve.

10.11.1 Detailed Description

Constructs an infinity shaped curve.

10.12 objects/knottable.cpp File Reference

Construct a table of curves.

10.12.1 Detailed Description

Construct a table of curves.

10.13 objects/line.cpp File Reference

Produces a file with a PKF line.

10.13.1 Detailed Description

Produces a file with a PKF line.

10.14 objects/random.cpp File Reference

Generate a "random" curve.

10.14.1 Detailed Description

Generate a "random" curve.

10.15 objects/sinus.cpp File Reference

Constructs a Sinusoidal PKF curve.

10.15.1 Detailed Description

Constructs a Sinusoidal PKF curve.

10.16 objects/solenoid.cpp File Reference

Constructs a solenoid like curve.

10.16.1 Detailed Description

Constructs a solenoid like curve.

10.17 objects/spiral.cpp File Reference

Constructs a Spiral PKF file.

10.17.1 Detailed Description

Constructs a Spiral PKF file.

10.18 objects/stadium.cpp File Reference

Constructs a stadium curve.

10.18.1 Detailed Description

Constructs a stadium curve.

10.19 objects/torusknot.cpp File Reference

Constructs a (p,q) torus knot.

10.19.1 Detailed Description

Constructs a (p,q) torus knot.

10.20 objects/torusknot4.cpp File Reference

Constructs a (p,q) torus knot on a torus in R^4.

10.20.1 Detailed Description

Constructs a (p,q) torus knot on a torus in R^4.

10.21 objects/trefoil.cpp File Reference

Approximated trefoil with arcs of circles ...

10.21.1 Detailed Description

Approximated trefoil with arcs of circles ...

An ideal trefoil is known not to be made of arcs of circles.

10.22 tools/align.cpp File Reference

Orient a pkf curve according to a given axes.

10.22.1 Detailed Description

Orient a pkf curve according to a given axes.

This tool reads a PKF curve from a file, then 3 arguments for the alignement axis (X,Y,Z) are given. The curve is reoriented and then written to an output PKF file.

10.23 tools/angle_condition.cpp File Reference

Check the angle condition for an ideal knot. input file : s sigma tau ...

10.23.1 Detailed Description

Check the angle condition for an ideal knot. input file : s sigma tau ...

10.24 tools/angle_principal_normal_contacts.cpp File Reference

Prints angle between contact chords and principal normal to stdout.

10.24.1 Detailed Description

Prints angle between contact chords and principal normal to stdout.

10.25 tools/angle_vs_arclength.cpp File Reference

Prints the arclength and for that arclength the angle between the normal vector and the "2" contact struts.

10.25.1 Detailed Description

Prints the arclength and for that arclength the angle between the normal vector and the "2" contact struts.

10.26 tools/arcdist.cpp File Reference

Program for testing purpose only.

10.26.1 Detailed Description

Program for testing purpose only.

10.27 tools/arcs2polygonal.cpp File Reference

Convert an arc curve to a polygonal curve.

10.27.1 Detailed Description

Convert an arc curve to a polygonal curve.

The curve is taken to be closed.

10.28 tools/biarccli.cpp File Reference

Biarc (p. 61) command line interface.

10.28.1 Detailed Description

Biarc (p. 61) command line interface.

10.29 tools/billiard_poly.cpp File Reference

Construct billiard polygon.

10.29.1 Detailed Description

Construct billiard polygon.

Given the contacts for the trefoil billiard (from follow_contact tool) write the vertices of the actual polygon of the billiard to stdout.

10.30 tools/centerandnormalise.cpp File Reference

Center and normalize a closed PKF curve.

10.30.1 Detailed Description

Center and normalize a closed PKF curve.

10.31 tools/circle_for_f31.cpp File Reference

Given eps, compute using 3 points the radius and center of circle.

10.31.1 Detailed Description

Given eps, compute using 3 points the radius and center of circle.

10.32 tools/clif_tref_contactset.cpp File Reference

Compute the contact set and surface (iv file) for the clifford trefoil in S^3.

10.32.1 Detailed Description

Compute the contact set and surface (iv file) for the clifford trefoil in S^3.

Example for an optimal g-Trefoil ./contactset clif_tref_s3_100.pkf 0.0001 2>contacts.txt >surface.iv

10.33 tools/closest_neighbour.cpp File Reference

Reconstruct the curve using the nearest point algo.

10.33.1 Detailed Description

Reconstruct the curve using the nearest point algo.

10.34 tools/contacts2inventor.cpp File Reference

Convert contact chords to obj/iv file.

10.34.1 Detailed Description

Convert contact chords to obj/iv file.

10.35 tools/contacts2sigma.cpp File Reference

Contact chords to sigma(s) and tau(s).

10.35.1 Detailed Description

Contact chords to sigma(s) and tau(s).

10.36 tools/contacts_s3tor3.cpp File Reference

Project contact chords from S^3 to R^3.

10.36.1 Detailed Description

Project contact chords from S^3 to R^3.

10.37 tools/contactset.cpp File Reference

Compute the contact chords for a given knot.

10.37.1 Detailed Description

Compute the contact chords for a given knot.

Jana Smutny's rhopt contact computation algorithm. Given a pkf curve and a tolerance, write an inventor file with the contacts as a lineset to stdout.

Example : Get Jana's best trefoil contactset with ./contactset j3.1.pkf 0.00002

10.38 tools/convolutionfilter.cpp File Reference

Smooth a curve using a local gaussian convolution filter.

10.38.1 Detailed Description

Smooth a curve using a local gaussian convolution filter.

10.39 tools/curvature.cpp File Reference

Outputs the curvature of a biarc curve.

10.39.1 Detailed Description

Outputs the curvature of a biarc curve.

10.40 tools/curvature4.cpp File Reference

Outputs the curvature of a biarc curve in R^4.

10.40.1 Detailed Description

Outputs the curvature of a biarc curve in R^4.

10.41 tools/curvesplit.cpp File Reference

Split a "closed" curve into segments for given arclength values.

10.41.1 Detailed Description

Split a "closed" curve into segments for given arclength values.

Splits a closed curve c into segments given the values s0, s1, etc. Initially written to be used with billiard_sigmas.py.

Example : ./curvesplit f3.1.pkf 'python billiard_sigmas.py'

Resulting curves are of the form XXXX.pkf

10.42 tools/extract_sigma.cpp File Reference

Advancing within a small neighbournood try to extract sigma(s).

10.42.1 Detailed Description

Advancing within a small neighbournood try to extract sigma(s).

10.43 tools/extract_sigma_max.cpp File Reference

Find local max in pt valley.

10.43.1 Detailed Description

Find local max in pt valley.

10.44 tools/filter_nodes.cpp File Reference

Filter a PKF curve. I.e. remove points that are too close to some neighbour.

10.44.1 Detailed Description

Filter a PKF curve. I.e. remove points that are too close to some neighbour.

10.45 tools/flatten.cpp File Reference

Project curve onto 2D plane.

10.45.1 Detailed Description

Project curve onto 2D plane.

10.46 tools/follow_contact.cpp File Reference

Starting at some point on the curve, follow the contacts to depth max_level.

10.46.1 Detailed Description

Starting at some point on the curve, follow the contacts to depth max_level.

10.47 tools/inertiatensor.cpp File Reference

Compute the inertia tensor and principal axes.

10.47.1 Detailed Description

Compute the inertia tensor and principal axes.

Compute the inertia tensor of the curve (assuming uniform mass distribution of the curve). Then it prints out the principal axes of the given curve.

10.48 tools/info.cpp File Reference

Computes Arc-length, center of mass, thickness (approx), ropelength (aprox) info of a PKF curve.

10.48.1 Detailed Description

Computes Arc-length, center of mass, thickness (approx), ropelength (aprox) info of a PKF curve.

10.49 tools/info4.cpp File Reference

Computes Arc-length, center of mass, thickness (approx), ropelength (aprox) info of a PKF curve (S^3).

10.49.1 Detailed Description

Computes Arc-length, center of mass, thickness (approx), ropelength (aprox) info of a PKF curve (S^3).

10.50 tools/inversion_in_sphere.cpp File Reference

Inversion in sphere at center c and radius r of a curve.

10.50.1 Detailed Description

Inversion in sphere at center c and radius r of a curve.

10.51 tools/length.cpp File Reference

Arc-length of a PKF curve.

10.51.1 Detailed Description

Arc-length of a PKF curve.

10.52 tools/link_thickness.cpp File Reference

Compute thickness of a **CurveBundle** (p. 102).

10.52.1 Detailed Description

Compute thickness of a **CurveBundle** (p. 102).

10.53 tools/map.cpp File Reference

Map one curve to another.

10.53.1 Detailed Description

Map one curve to another.

The tool maps a given curve to another given a parameter t between zero and one. The curves must have the same number of nodes. Corresponding nodes are interpolated linearely (using t) and the resulting curve is written to the specified output file.

10.54 tools/mesh4stokes.cpp File Reference

Produce a tube mesh from a curve and write that to a file.

10.54.1 Detailed Description

Produce a tube mesh from a curve and write that to a file.

10.55 tools/nana.cpp File Reference

Test program.

10.55.1 Detailed Description

Test program.

10.56 tools/normal_indicatrix.cpp File Reference

Unfinished.

10.56.1 Detailed Description

Unfinished.

10.57 tools/perturb.cpp File Reference

Perturb a curve.

10.57.1 Detailed Description

Perturb a curve.

This tool perturbs all the points of a given "closed" PKF curve. The perturbation takes place in the plane that is normal to the tangent vector at the point to be displaced. The angle is in the interval [0,2 pi] and the magnitude is randomly chosen according to the scaling factor given as an input.

10.58 tools/pkf2java.cpp File Reference

Read data from a PKF file, produce a tubular mesh for each curve in the file and write it as C/C++/Java arrays.

10.58.1 Detailed Description

Read data from a PKF file, produce a tubular mesh for each curve in the file and write it as C/C++/Java arrays.

10.59 tools/pkf2mesh.cpp File Reference

Read data from a PKF file, produce a tubular mesh for each curve in the file and write that to a file.

10.59.1 Detailed Description

Read data from a PKF file, produce a tubular mesh for each curve in the file and write that to a file.

Warning : Here all the tubes will have the same number of nodes,segments, same radius and if tolerance, same tolerance

10.60 tools/pkf2obj.cpp File Reference

pkf to obj format. obj file has correct uv texture coords.

10.60.1 Detailed Description

pkf to obj format. obj file has correct uv texture coords.

10.61 tools/pkf2ply.cpp File Reference

pkf to mesh to stanford ply format.

10.61.1 Detailed Description

pkf to mesh to stanford ply format.

10.62 tools/pkf2stl.cpp File Reference

Read data from a PKF file, produce a tubular mesh for each curve in the file and write that to an STL file.

10.62.1 Detailed Description

Read data from a PKF file, produce a tubular mesh for each curve in the file and write that to an STL file.

10.63 tools/pkf2xyz.cpp File Reference

Convert PKF file to XYZ coordinates.

10.63.1 Detailed Description

Convert PKF file to XYZ coordinates.

10.64 tools/pkfframe.cpp File Reference

Generate a framing along the curve.

10.64.1 Detailed Description

Generate a framing along the curve.

10.65 tools/plotslice.cpp File Reference

Write a pt plot cross section at a given arclength.

10.65.1 Detailed Description

Write a pt plot cross section at a given arclength.

10.66 tools/plotslice4.cpp File Reference

Write a pt plot cross section at a given arclength (S^3).

10.66.1 Detailed Description

Write a pt plot cross section at a given arclength (S^3).

10.67 tools/pointat.cpp File Reference

Writes point/tangent at Curve(s), for s arclength and s in [0,1] to the standart output.

10.67.1 Detailed Description

Writes point/tangent at Curve(s), for s arclength and s in [0,1] to the standart output.

10.68 tools/projected_dij.cpp File Reference

Compute $\sum_{i \neq j} d_{i,j}$ and $\sum_{i \neq j} d_{i,j}^{-1}$ of a given projection.

10.68.1 Detailed Description

Compute $\sum_{i \neq j} d_{i,j}$ and $\sum_{i \neq j} d_{i,j}^{-1}$ of a given projection.

This tool aligns the pkf curve along an axis given as an input to the program. The pkf filename is expected to be the last argument on the command line. Then the reoriented curve is projected onto the XY-plane and resampled, such that the curve has a uniform distribution in the sense of arc-length. Then the following quantity is computed :

$\frac{1}{N \cdot (N-1)} \sum_{i \neq j} d_{i,j} and \frac{1}{N \cdot (N-1)} \sum_{i \neq j} \frac{1}{d_{i,j}}$, where N is the number of points on the curve.

The values are written to the standart output. There is a result with the sum of the distances and a result for the sum of the inverse of each distance.

Points that are closer than some tolerance are rejected and do not contribute to the sum.

10.69 tools/ptcircles.cpp File Reference

Resample an open or closed PKF curve.

10.69.1 Detailed Description

Resample an open or closed PKF curve.

10.70 tools/r3tos3.cpp File Reference

Project a curve in R^3 to S^3.

10.70.1 Detailed Description

Project a curve in R^3 to S^3.

10.71 tools/redo_tangents.cpp File Reference

Recompute tangents using only the data points.

10.71.1 Detailed Description

Recompute tangents using only the data points.

10.72 tools/refine.cpp File Reference

Refine/Resample a part of an open or closed PKF curve.

10.72.1 Detailed Description

Refine/Resample a part of an open or closed PKF curve.

10.73 tools/reorder.cpp File Reference

Reorder the nodes of a curve, so that the first points of the 2 curves are the closest possible.

10.73.1 Detailed Description

Reorder the nodes of a curve, so that the first points of the 2 curves are the closest possible.

10.74 tools/resample.cpp File Reference

Resample an open or closed PKF curve.

10.74.1 Detailed Description

Resample an open or closed PKF curve.

10.75 tools/resample4.cpp File Reference

Resample an open or closed PKF curve (R^4).

10.75.1 Detailed Description

Resample an open or closed PKF curve (R^4).

10.76 tools/reverse_direction.cpp File Reference

Change the orientation of the curve.

10.76.1 Detailed Description

Change the orientation of the curve.

We change the orientation of the curve by flipping the tangents and reordering the points.

10.77 tools/rotate_curve.cpp File Reference

Rotate curve about axis v by angle a.

10.77.1 Detailed Description

Rotate curve about axis v by angle a.

10.78 tools/s3tor3.cpp File Reference

Project a curve on S^3 back to R^3.

10.78.1 Detailed Description

Project a curve on S^3 back to R^3.

10.79 tools/s_sigma_angles.cpp File Reference

Program for testing purpose only.

10.79.1 Detailed Description

Program for testing purpose only.

10.80 tools/setcenter.cpp File Reference

Reset the mass center of PKF curve.

10.80.1 Detailed Description

Reset the mass center of PKF curve.

10.81 tools/shuffle.cpp File Reference

Randomize/shuffle a curve.

10.81.1 Detailed Description

Randomize/shuffle a curve.

10.82 tools/sigma2contact.cpp File Reference

From a sigma(s) function, generate the contact struts.

10.82.1 Detailed Description

From a sigma(s) function, generate the contact struts.

Takes the function sigma(s) and a pkf knot and writes the corresponding struts (start,mid,endpoint) to stdout.

10.83 tools/sigma2inventor.cpp File Reference

Convert 2D sigma(s) to iv or obj file.

10.83.1 Detailed Description

Convert 2D sigma(s) to iv or obj file.

10.84 tools/sigma_and_tau_contacts.cpp File Reference

Extract contact functions sigma(s) and tau(s).

10.84.1 Detailed Description

Extract contact functions sigma(s) and tau(s).

10.85 tools/sigma_composition.cpp File Reference

compute $^k(s)$. input file : s sigma tau ...

10.85.1 Detailed Description

compute $^k(s)$. input file : s sigma tau ...

10.86 tools/tangent_indicatrix.cpp File Reference

Write the tangent indicatrix of a given (open or closed) curve to a PKF file. Tangents for the tangent indicatrix are interpolated, therefore strange points like cusps would not be visible! The default is for an open curve, for closed curves put the -closed switch.

10.86.1 Detailed Description

Write the tangent indicatrix of a given (open or closed) curve to a PKF file. Tangents for the tangent indicatrix are interpolated, therefore strange points like cusps would not be visible! The default is for an open curve, for closed curves put the -closed switch.

10.87 tools/torsion.cpp File Reference

Outputs the torsion of a biarc curve.

10.87.1 Detailed Description

Outputs the torsion of a biarc curve.

10.88 tools/xyz2pkf.cpp File Reference

Convert XYZ coordinates to PKF.

10.88.1 Detailed Description

Convert XYZ coordinates to PKF.

Index